Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Amir Ingher

Operaatio Robottien internet

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.4.2020

Tekijä Otsikko Sivumäärä Aika	Amir Ingher Operaatio Robottien internet 36 sivua 10.4.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Software Engineering
Ohjaajat	Tutkintovastaava Janne Salonen

Tämän insinöörityön tarkoituksena oli rakentaa IoT-robotti ja sille toimiva ohjelmisto. Robotin komponentit eivät kommunikoi toistensa kanssa. Näin ollen logiikka ulkoistetaan yhdelle tai usealle eri palvelimelle. Luodun ympäristön tarkoituksena on lähettää robotista kuvadataa verkon kautta palvelimelle, jossa palvelimella pyörivä sovellus päättää, mitä kuvalla tehdään. Päätöksen jälkeen lähetetään robotille käsky siitä, kuinka toimitaan. Kehitetyllä ympäristöllä demonstroidaan skaalautuvaa ja helposti integroitavissa olevaa systeemiä.

Insinöörityössä käydään läpi robotin kokoamiseen käytettyjä komponentteja ja niiden käyttöönottoa. Työssä perehdytään ohjelmistoihin, joilla luodaan robotin osien kommunikaatio, sekä myös eri kommunikaatioprotokolliin. Robotille luodaan myös Node.Js-sovellus ja Python-ohjelmointikielellä tekoäly, joka toimii robotin toimintalogiikkana. Sovelluksen avulla pystytään etäohjaamaan robottia WiFi-verkon kautta, ja tekoäly on yksinkertainen konvoluutioneuroverkko, joka tunnistaa yksinkertaiset esteet. Työssä pohdittiin robotin jatkokehitystä.

Työ tehtiin oppimisprojektina. Tarkoituksena on saada syvempi ymmärrys verkoista, eri komponenteista ja käytetyistä työkaluista.

Insinöörityön päätavoitteissa onnistuttiin. Luotiin toimiva ympäristö, jossa robottia voidaan ohjata saadun datan vuorovaikutuksesta. Joka tapauksessa älykkäämmän tekoälyn kehittäminen jäi toteuttamatta ajallisten haasteiden vuoksi.

Avainsanat	Robottien internet, tekoäly, Arduino

Author	Amir Ingher
Title	Operation Internet of Robotic Things
Number of Pages	36 pages
Date	10 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Principal Lecturer

The main goal of the thesis was to build an IoT robot and create a working environment for it. The IoT robot's components do not communicate with each other. Instead the components communicate with a server, thus making the logic externalized to a cloud. The idea is that the robot sends image data from one of the components to server socket. The server then interacts with the data and decides what to do with it. After a decision has been made, it will send a command to the robot on how to act based on the image data. This project demonstrates the scalability and effortless integration of the system.

The thesis introduces the various components used in the robot. Also, the thesis explains how ready-made and created software are used to establish working communication between components. In addition to the working environment, this project also includes two self-made applications. One of them enables manual control of the robot through WiFi and other one uses a simple convolutional neural network for recognizing obstacles. Lastly, the thesis discusses the created systems and the robot's future development.

This project was a learning experience. The purpose was to get a deeper understanding about networks, electric components and software tools. In addition, the robot's purpose is to work as a platform for developing artificial intelligence.

The main goals of the thesis were reached by successfully creating an environment where the user could interact with the image stream and remotely control the robot. However, the robot's artificial intelligence remained unfinished.

Keywords	Internet of Robotic Things, Machine Learning, Arduino

Sisällys

Lyhenteet

JPEG	Joint Photographic Experts Group. Digitaalisten kuvien häviöllistä pakkausta hyödyntävä bittikarttagrafiikan tallennusformaatti.
TTL	Transistor-Transistor Logic. Logiikkapiiriperhe.
GPIO	General Purpose I/O. Nasta mikrokontrollereissa. Sen avulla voidaan ottaa sisään tai ulos signaaleja.
URL	Uniform Resource Locator. WWW-sivujen osoite. Merkkijonon avulla osoitetaan tietyn tiedon paikka sekä protokolla.
HTML	Hypertext Markup Language. Hypertekstin merkintäkieli on standardi, jolla voidaan ilmaista tekstin rakennetta ja kuvata hyperlinkkejä. Tunnettu internetsivujen kirjoituskielenä.
TCP	Transmission Control Protocol. Luotettava tiedonsiirronprotokolla.
TCP/IP	Transmission Control Protocol / Internet Protocol. Protokolla, jota käytetään tiedon liikennöinnissä. IP vastaa pakettien reitittimestä.
Base64	Base64. Tapa miten binääridata muunnetaan ASCII-merkkijono formaattiin.
API	Application programming interface. Ohjelmointirajapinta, jonka avulla ohjelmat voivat välittää dataa keskenään.
CPU	Central Processing Unit. Suoritetaan tietokoneen sisäisiä laskelmointeja ja käskyjä.
GPU	Graphics Processing Unit. Suorittaa grafiikan renderointiä tietokoneessa.
SSID	Service Set Identifier. Verkon nimi.

JMS	Java Message Service. Rajapinta, jonka avulla voidaan lähettää viestejä kahden tai useamman asiakkaan välillä.

1 Johdanto

Esineiden internet on ollut läsnä arjessamme jo pitkään. Matkapuhelimien yleistyttyä alkoi myös verkon käyttö yleistyä. Suureen verkkoon alettiin samalla yhdistää muita laitteita, kuten älylamppuja ja älyjääkaappeja. Näin luoden älykoteja. Sensoreiden yhdistäminen verkkoon on edelleen jatkuvassa kasvussa. Esineiden internetiin on kehittynyt alakategorioita, joista yksi on robotiikan internet. Tehtaiden ja eri tuotantojen robotteja ollaan kytkemässä myös verkkoon. Tällöin robotit ovat vuorovaikutuksessa sensoreista saatavan datan kanssa.

Insinöörityöni tavoite on luoda pienemmässä mittakaavassa robotti, jonka logiikka on ulkoistettu palvelimelle. Robotti lähettää dataa WiFi-verkon kautta palvelimelle, jossa tekoäly tai sovellus päättää, mitä tiedolla tehdään. Saadun datan perusteella robotille lähetään komento siitä, kuinka tämän tulisi toimia.

Työn alussa käydään läpi, mitä hyötyjä robottien kytkemisestä verkkoon on. Samassa luvussa ideoidaan insinöörityöhön kehitettävän robotin rakennetta ja funktionaalista tarkoitusta. Kolmannessa luvussa tutkitaan komponentteja, jotka valittiin robotin toteuttamiseen. Neljännessä luvussa tutustutaan tarkemmin robotin ohjelmistoratkaisuun eli siihen, kuinka robotti lähettää dataa ja vastaanottaa käskyjä. Lisäksi luvussa syvennytään toteutukseen tarvittaviin ohjelmiin ja niiden käyttöönottoon. Viidennessä luvussa pyritään luomaan sovellusta, jonka avulla demonstroidaan robotin toiminnallisuutta. Kuudennessa luvussa pohditaan robotin tulevaisuutta ja kehityssuuntaa. Viimeisenä on yhteenveto, jossa tutkitaan insinöörityön saavutusta.

Insinöörityö luotiin oppimistarkoitusta varten. Tarkoituksena on saada syvempi ymmärrys verkoista, eri komponenteista ja käytetyistä työkaluista. Robotti tulee myös toimimaan tekoälyn kehitysalustana tulevissa projekteissa.

2 Esineiden internet ja robotiikka

IoRT (Internet of Robotic Things) eli robotiikan internet on alaluokka esineiden internetistä (Internet of Things). Robotiikan internet on digitaalisen ja fyysisen risteytys, eli konsepti, jossa laitteella on monitorointi- ja tunnistamiskyky, mutta samanaikaisesti ominaisuus hakea dataa monista eri lähteistä. Laite pystyy analysoimaan monitoroituja tapahtumia, sekä tekemään päätöksiä tapahtumien perusteella ja toimimaan päätöksen mukaisesti. Toiminta voi olla fyysisen objektiin kontrollointi tai manipulointi fyysisessä ympäristössä. [1.]

Esineiden internet ei pelkästään koske teollista aluetta, vaikkakin tämänhetkiset vaikutukset näkyvät pitkälti teollisuusalalla. Vaikutus tulee näkymään myös laajalti terveydenhuollossa, kodinkoneissa ja ajoneuvoissa. Markkinoiden kasvu on ennustettu saavuttavan 21,44 biljoonaan dollarin kokoluokan vuoteen 2022 mennessä. [2.]

Robotin yhdistäminen verkkoon tarjoaa monia etuja. Näitä hyötyjä ovat esimerkiksi

- laskelmointi tehon skaalautuvuus
- lähes loputon tallennustila
- uuden ohjelmiston nopea integrointi
- tiedon jakelu laitteiden kesken.

Yhdistäminen tuo mukanaan myös haittapuolia esimerkiksi

- verkko-ongelmat tai viiveitä
- korkeampi todennäköisyys kohdistua hakkeroinnille
- robotin liiallinen riippuvuus verkkoon.

Insinöörityössä tavoitellaan ympäristöä, jossa fyysiset toiminnot perustuvat verkon välityksen kautta saatuun sensori- tai kuvadataan. Työllä halutaan esitellä ympäristön skaalautuvuutta, eli monen eri ohjelmiston kykyä vaikuttaa toimintoihin sekä uuden päivitetyn ohjelmiston vaivatonta integraatiota. Tavoitteena on luoda jonkinlainen

etäohjattavuus ja tekoäly robotille. Tekoälylaskentoja voi olla useita erilaisia robotissa, esimerkiksi kyky navigoida ja tunnistaa hahmoja samanaikaisesti.

3 Komponentit

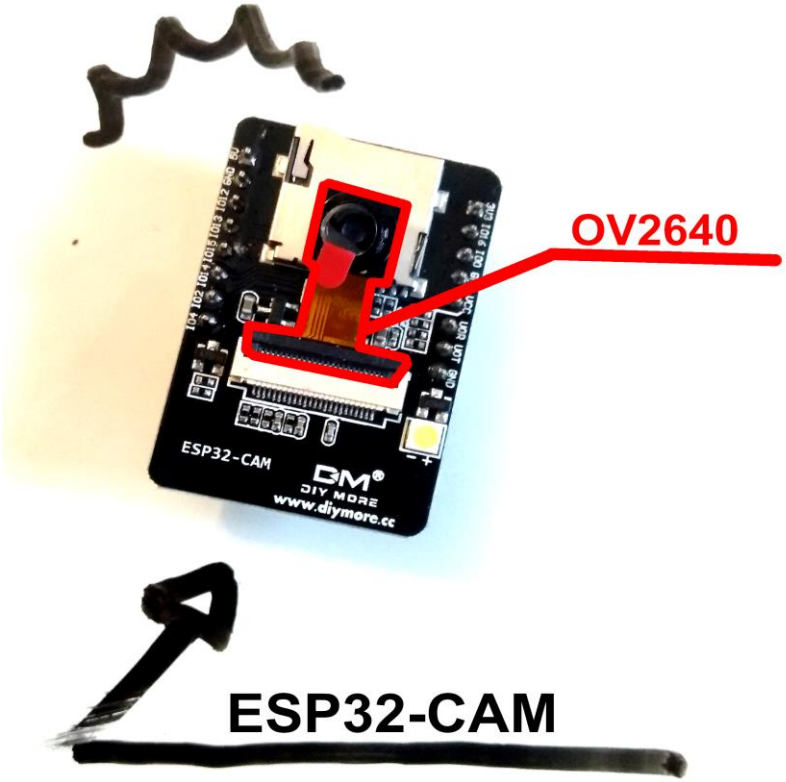
Tässä luvussa perehdytään tilattuihin komponentteihin ja niiden käyttötarkoitukseen robotissa.

Tilatuista komponenteista suurin osa saapui Kiinasta sekä Iso-Britanniasta. Osat on tilattu verkkokauppayhtiö eBay:n kautta, joten suurin osa tilatuista osista on halpaversioita alkuperäisten tuottajien komponenteista.

Robotin motoriikka voisi esiintyä monessa eri muodossa, mutta tätä insinöörityötä varten on luotu kaksipyöräinen robotti, jolla pystytään liikkumaan eri suuntiin. Data, jonka kanssa ollaan vuorovaikutuksessa, on JPEG-tietovirtaa. Syy juuri tämän mekaanisen ratkaisun valintaan oli kokoamisen helppous.

3.1 ESP32 Kamera

ESP32 on suosiota saanut ohjelmoitava piirilevy, johon on integroitu WiFi-moduuli sekä Bluetooth-yhteys. Se sisältää 520 kB sisäistä muistia ja 4 MB ulkoista muistia, lisäksi siinä on microSD-kortille oma paikka. Tilattu malli on kuitenkin ESP32-cam (kuva 1), ja tämä poikkeaa ESP32:sta sen piirilevyyn sulautetulla kameraliittimellä.



Kuva 1. ESP32-cam ja punaisella rajattu OVA2640.

ESP32-cam-mallin mukana tulee erillinen kameramoduuli OV2640 (kuva 1). Kameramoduulilla kyetään ottamaan JPEG-kuvia. Komponentin tarkoitus on yhdistää WiFi-verkon kautta sokettiin ja lähettää JPEG-kuvia sokettia tarjoavalle palvelimelle. [3.]

3.2 FTDI FT232RL Ohjelmoija

FTDI FT232RL (kuva 2) on USB TTL -sarjasovitin. Komponentti kääntää USB:sta saapuvan tiedon TTL-muotoon.

Kuva 2. FT232RL ohjelmoija.

FT232RL:n avulla ladataan ohjelma USB:n kautta ESP32-cam:iin. Sarjasovittimella voidaan myös antaa 5V tai 3,3V virtaa USB-johdon kautta. [4.]

3.3 ESP8266 NodeMCU (ESP-12E)

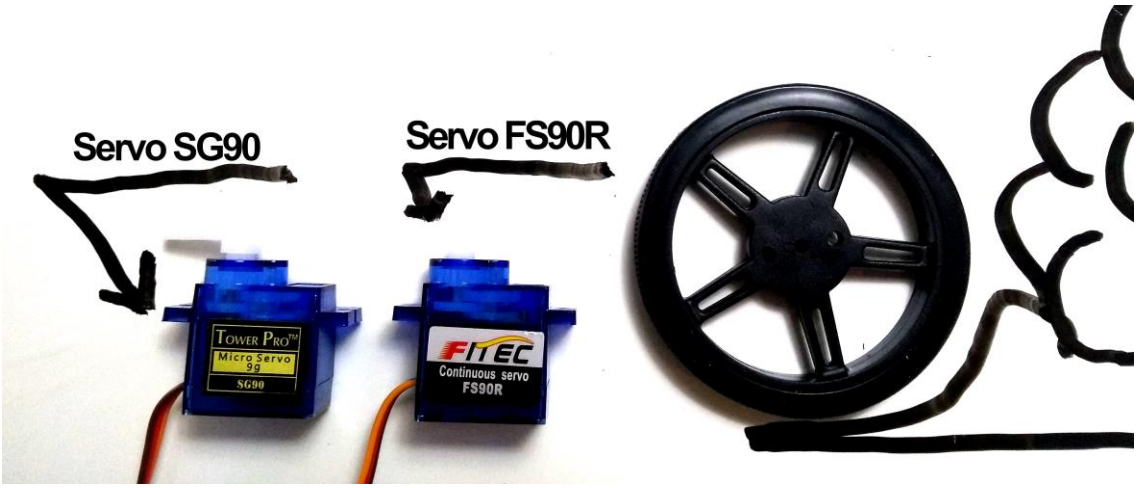
NodeMCU (kuva 3) on IoT-kehityksessä yleistynyt malli, joka tarjoaa ESP-12E-WiFi-moduulin, johon on liitetty ESP8266-siru, joka pyörittää Tensilica Xtensa® 32-bit LX106 -mikroprosessoria. Mallista löytyy 128 KB sisäistä RAM-muistia sekä 4 MB ulkoista Flash-muistia.

Kuva 3. ESP8266 NodeMCU ESP-12E ohjelmoitava piirilevy. Sisältää WiFi-moduulin

Esp8266 NodeMCU:sta tekee erikoisen sen, että se tarjoaa 17 GPIO-nastaa. Tämä tarkoittaa, että voimme hallinnoida montaa eri moottoria samanaikaisesti tai vastaanottaa sensoridataa yhden ESP8266 NodeMCU:n avulla. Tämä on käytännöllinen ominaisuus monissa IoT-projekteissa. Tässä työssä ESP8266 NodeMCU yhdistetään WiFi-verkon kautta palvelimeen, joka vastaanottaa palvelimesta saapuvia viestejä. Tämän jälkeen saapunut viesti käännetään käskyksi kolmelle servomoottorille. [5.]

3.4 Servomoottorit

Työssä käytetään kahta eri servomallia. FS90R Servoa (kuva 4) [6] käytetään robotin liikuttamiseen, koska moottorissa on jatkuvan rotaatioliikkeen ominaisuus.



Kuva 4. Servomoottorit, jossa vasemman puolimmainen moottori on SG90 ja keskellä oleva on FS90R. Oikealla on rengas.

Toinen servomoottori on malliltaan SG90 (kuva 4) [7], joka tarjoaa 180 asteen kääntymisrajoitteen. Tätä hyödynnetään kamerassa ylös ja alas katsomiseen, koska valmis rajoitus estää kameraa pyörähtämästä ympäri.

4 Robotin ohjelmistojen suunnittelu ja käyttöönotto

Osiossa keskitytään toteuttamaan IoT-robotin oleellinen ohjelmisto, jota tarvitaan WiFiverkkoon yhdistämiseen, kuvan striimaamiseen, viestien lähettämiseen sekä robotin osien ohjaamiseen. Perehdymme komponenttien kytkemiseen sekä siihen, miten ESP-komponentit konfiguroidaan Arduino IDE:ssä, kun niihin aiotaan tallentaa ajettavaa koodia.

Lisäksi luvussa tutustutaan erilaisiin ohjelmointimenetelmiin ja vaihtoehtoihin sekä käydään läpi ongelmia, joita suunnitellun toiminnallisuuden toteuttamisessa tuli vastaan.

Ohjelmointikielenä käytetään Arduino-kielen hyödyntämää C/C++-kieltä piirilevyjen ohjelmoimisen. Java-ohjelmointikielellä käsitellään sokettien ja liikkuvan datan hallinnoimista. Lisäksi käytämme ActiveMQ:ta, joka sisältää monia eri viestiprotokollia.

4.1 ESP32-cam-ohjelmointi

ESP32-cam toimii niin sanotusti robotin silmänä, joka lähettää konfiguroidun ajan välein JPEG-kuvia WiFi-verkon kautta tietokoneeseen. Malliin itse ei ole sulautettu minkäänlaista USB-porttia, jonka vuoksi ESP32-cam tarvitsee FT232RL-ohjelmoijan, jonka voi suoraan yhdistää miniUSB:n avulla tietokoneeseen. FT232RL-ohjelmoija pitää kytkeä ESP32-cam:n GPIO-nastoihin. Taulukossa 1 listataan, miten ESP32-cam:n GPIO-nastat kuuluu yhdistää FT232RL-ohjelmoijaan, jotta ESP32-cam on ohjelmoitavissa. [8.]

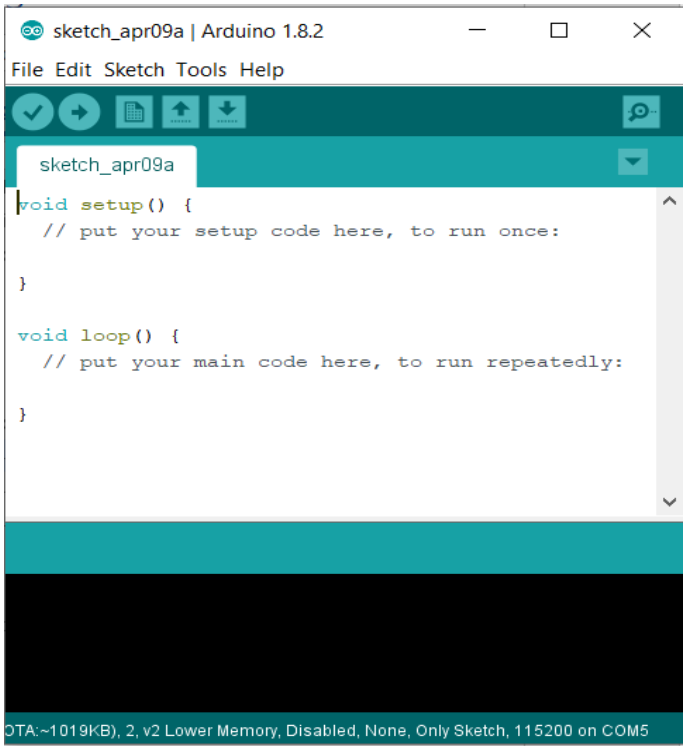
Taulukko 1. ESP32-cam komponentin GPIO-yhdistäminen FT232RL.

ESP32-cam (GPIO)	**FT232RL (GPIO)**	**ESP32-cam (GPIO)**
5V	VCC	
GND	GND	
U0R	TX0	
U0T	RX1	
IO0		GND

Kun nastoja yhdistetään hyppylangan avulla toisiinsa, on tärkeätä varmistaa, että johdot on kunnolla yhdistetty. Nimittäin työssä käytettiin noin viisi tuntia ylimääräistä aikaa huolettomasti kytkettyjen johtojen takia. [8.]

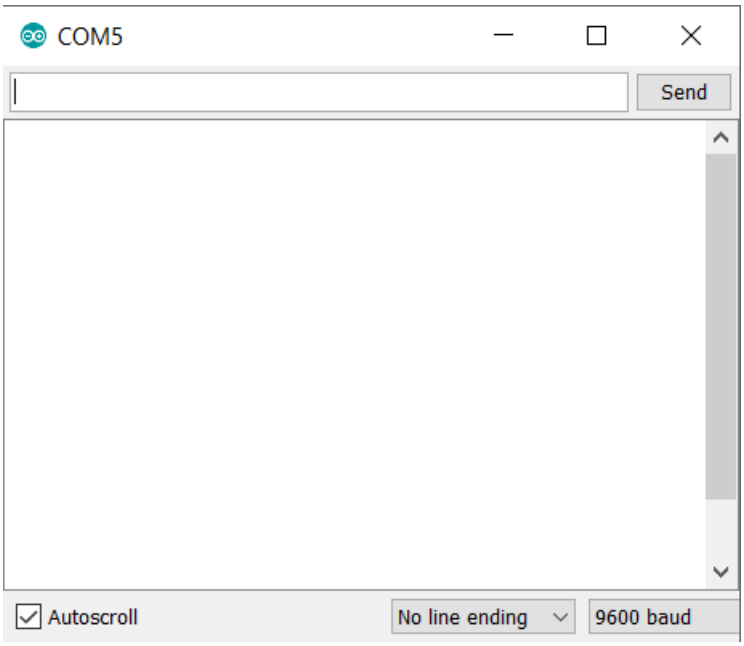
Arduino IDE

Insinöörityön komponenttien ohjelmointiympäristönä käytetään Arduino IDE:tä, jolla voimme ohjelmoida Arduino-yhteensopiville piirilevyille. Tämän avulla voidaan helposti kääntää ja puskea kehitetty koodi piirilevylle. Arduino IDE tukee C/C++-ohjelmointikieliä, ja sen mukana tulee kattava määrä ohjelmakirjastoja. [9.]



Kuva 5. Arduino IDE:stä sketch-näkymä, joka sisältää setup- ja loop-funktiot.

Avatessa Arduino IDE:n avautuu näkymä ohjelmakoodista. Kuvan 5 näkymää kutsutaan Sketchiksi. Tämä sisältää kaksi funktiota: setupin ja loopin. Setup ajetaan aina ensimmäisenä kerran, jonka jälkeen siirrytään loop-funktioon. Loop-funktiota ajetaan ohjelmassa jatkuvasti. Parametrien konfigurointi tapahtuu setup-funktiossa ja ohjelman logiikka useasti taas loop-funktiossa. [9.]



Kuva 6. Serial Monitorin ikkuna. Oikealla alakulmassa on asetettu bpd-arvo 9600.

Arduino IDE sisältää myös Serial Monitor -ohjelman (kuva 6), jonka avulla voidaan viestitellä piirilevyn ja tietokoneen välillä. Tämä on välttämätön työkalu, kun halutaan saada selville, miten piirilevyssä pyörivä ohjelma käyttäytyy. Kommunikaatio luodaan Sketchin setup-funktiossa alla olevalla komennolla, mutta jos kommunikaatioon ei ole tarvetta, voidaan alla oleva pätkä unohtaa.

```
Serial.begin(115200);
```

Esimerkkikoodi 1. Arduinon oma kirjasto Serial, joka aloittaa kommunikaation funktiolla begin. Metodille asetetaan parametri 115200 bpd.

Esimerkkikoodi yhden arvo, joka asetetaan “begin” metodin parametriksi, on bpd (bits per secound) eli tiedonsiirron nopeus. Serial Monitorista voi vaihtaa tätä arvoa oikean alakulman valikkopalkista. Joten kun halutaan vastaanottaa tai lähettää selkokielisiä viestejä, on tärkeätä asettaa begin-metodille sama arvo kuin Serial Monitorille. [10.]

Jotta ESP32-cam:iin voidaan ohjelmoida yhtään mitään, täytyy Arduino IDE:n kautta määritellä tarkat konfigurointiarvot. Ensimmäiseksi avataan Preferences-ikkuna.

Sisällöstä löytyy Additional Board Manager URLs, joka mahdollistaa muiden piirilevyvalmistajien tekemien ohjelmien ja kirjastojen hyödyntämisen. Otsikon vierellä on tyhjäsyöttökenttä, johon syötetään seuraavat URL-osoitteet:

https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Tämä URL määrittää, mistä ESP-komponenteille tarvittavan ohjelmistopaketin voi löytää. Heti, kun URL-pääte on asetettu syöttökenttään, avataan Tools-valikosta “Board” > “Boards Manager...”. Ikkunalle pitäisi ilmestyä Boards Manager -näkymä. Oikealta yläkulmasta löytyy hakukenttä, johon syötetään “esp32”, jonka seurauksena ikkunaan ilmestyy “esp32 by Espressif Systems” -niminen paketti. Tämä pitää asentaa, jotta koodi voidaan kääntää esp32-camille sopivaan muotoon. Paketin mukana tulee myös hyödyllisiä ohjelmistokirjastoja sekä esimerkki-sketchejä. [8.]

```
const char* ssid = "Sinun WiFi-verkon nimi";
const char* password =  "WiFi-verkon salasana";
```

Esimerkkikoodi 2. Ssid- ja password-parametrille asetetaan arvot.

Testatakseen ESP32-kameran toimivuuden avataan Arduino IDE:n kautta “Examples” > “ESP32” > “Camera” > “CameraWebServer”. Kyseinen ohjelma käynnistää web-serverin ja striimaa reaaliaikaista videota sekä kykenee tunnistamaan kasvoja. Tämä kaikki toiminnallisuus tapahtuu ESP32-camin sisällä, joka jo itsestään kertoo, kuinka tehokas tämä komponentti on. Ennen kuin ohjelma voidaan ajaa, täytyy koodin parametrit konfiguroida oikein, koska esp32 pyrkii liittymään WiFi-verkkoon. Jotta web-palvelimeen voidaan jakaa sisäverkon välityksellä, täytyy yllä oleville parametrille “ssid” ja “password” asettaa oikeat arvot. Ssid on verkon nimi, johon piirilevy aikoo yhdistää ja password on mahdollisen WiFi-verkon salasana. [8.]

```
// Select camera model
#define CAMERA_MODEL_WROVER_KIT
```

```
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER
```

Esimerkkikoodi 3. Valittu kameramalli.

Toinen parametri, joka on pakko muuttaa, on kameramalli. Parametri löytyy kommentoidun “Select camera modelin” alapuolelta. Kohdassa on lista erilaisia kameramalleja. Koska ostettu malli on halpa kopio alkuperäisestä mallista, valitaan “CAMERA_MODEL_WROVER_KIT”, vaikka verkkoaineisto [8] kehottaa valitsemaan toisin. Valitseminen tapahtuu poistamalla kaksi kenoviivaa rivin alusta. Ohjelma ladataan piirilevylle painamalla nuolta, joka osoittaa oikealle. Ohjelma alkaa pyöriä heti, kun lataus on valmis. [8.]

Avataan Serial Monitor ikkuna, joka syöttää tekstiä piirilevystä ikkunaan. Tekstin seasta löytyy URL-linkki, joka on ESP32:lle asetettu IP-osoite. Linkin kautta voi päästä laitteen palvelemaan web-palvelimeen. [8.]

CameraWebServer-ohjelmaa muokaten luotiin sopivampi versio ohjelmasta, joka lähettää ottamansa kuvan haluttuun osoitteeseen sekä porttiin. Suurin osa CameraWebServer-ohjelmasta oli tarpeetonta. Työssä säästettiin vain parametrien konfiguraatio eli kaikki parametrit ja setup-funktio. Loop-funktion sisältö poistettiin, koska siihen haluttiin työhön sopivampi logiikka. Sen sijaan, että avataan web-palvelin, josta voidaan GET API -kyselyn avulla hakea viimeisin kuva, käytetään projektissa client-write-metodia. Kyseinen metodi lähettää kuvan client-parametrissa asetettuun osoitteeseen.

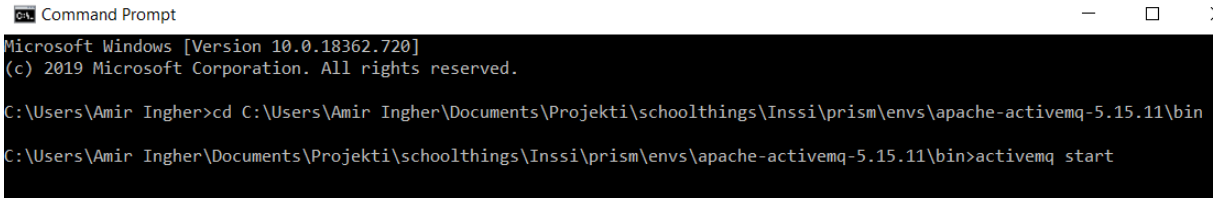
Ohjelman toimivuutta on hyvä kokeilla. Työssä toteutettiin kokeellinen python-ohjelma, joka avaa tcp-soketin, johon voidaan ottaa yhteys laitteesta. Python-koodia ajaessa Windows 10 kysyi, saako ohjelmalle antaa tarvittavat oikeudet, jolloin on tärkeätä vastata kyllä eli “Allow”, jotta palomuuri ei estä kommunikaatiota laitteen kanssa. Tämä on hyvä muistaa aina, kun avataan kuuntelevia soketteja.

4.2 Apache ActiveMQ

Tässä luvussa käydään lyhyesti läpi ActiveMQ ja sen tarjoamat hyödyt tälle projektille. Insinöörityössä hyödynnettiin ActiveMQ:n tukemaa MQTT-viestintäprotokollaa.

ActiveMQ on avoimeen lähdekoodiin perustuva viestintäpalvelu. Ohjelma on kirjoitettu Javalla, ja se tukee kokonaan JMS clientia. Viestintäpalvelusta löytyy erilaisia viestintäprotokollia muun muassa OpenWire, STOMP, MQTT, AMQP, REST ja WebSocket. Työssä hyödynnetään MQTT-protokollaa. Lisäksi ActiveMQ tarjoaa monille tunnetuille ohjelmistokielille client-kirjaston eli istunnon luomista ei tarvitse tehdä pelkästään Javalla. [11, s. 12.]

ActiveMQ käynnistetään kuvassa 7 näkyvällä komennolla.



Kuva 7. Kuvan ensimmäisellä komennolla suunnistetaan ActiveMQ:n bin-kansioon, jonka jälkeen ActiveMQ käynnistetään alimmaisen rivin komennolla.

ActiveMQ:n mukana tulee myös web-konsoli, jonka avulla voidaan monitoroida aiheita, kuluttajia ja yhteyksiä. Viestien tuottaminen suoraan aiheisiin ja niiden kuluttaminen onnistuu myös konsolin kautta. Web-konsoli käynnistyy heti, kun ActiveMQ käynnistetään yllä olevalla komennolla. Konsoliin pääsee käsiksi selaimen kautta seuraavasta osoitteesta: http//:localhost:8161/admin osoitteen kautta. [12.]

MQTT

MQTT eli MQ Telemetry Transport on kevyt viestintäprotokolla, joka jakaa viestejä laitteiden välillä. Tätä käytetään useasti IoT-laitteiden viestinnässä. Viestintäprotokolla pohjautuu TCP/IP-protokollaan. [13.]

Protokolla toimii “publish-subscribe”-periaatteen mukaisesti eli julkaise ja tilaa. Termillä tarkoitetaan jonkin aiheen luontia ja sen tilaamista. Ideana on, että luodaan “topic” eli aihe, johon julkaistaan viestejä. Näitä viestejä taas kuuntelee “subscriber” eli tilaaja. Terminä voidaan myös käyttää “producer-consumeriä”, jossa “producer” on tuottaja ja “consumer” on kuluttaja. MQTT-välitin käynnistetään automaattisesti, kun ActiveMQ käynnistyy. Välittimeen saadaan yhteys portista 1883. [13.]

MQTT on saanut paljon suosiota maailmalla: muun muassa HSL:n julkiset kulkuvälineet hyödyntävät MQTT-protokollaa ilmaisemalla sijaintinsa kerran sekunnissa. [14.]

4.3 Prism

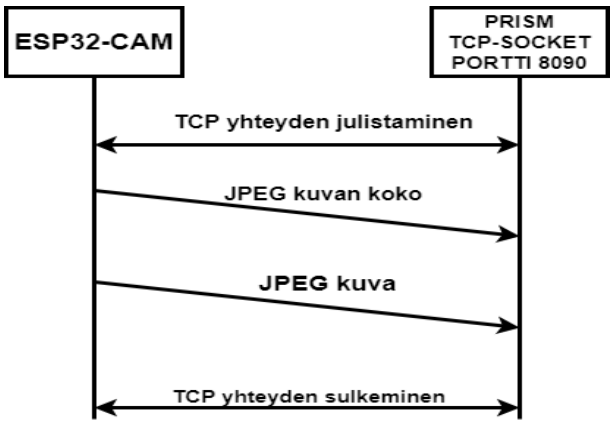
Insinöörityössä tuli vastaan paljon erilaisia ongelmia, jotka johtivat Prism-ohjelman kehittämiseen. Prism on tarkoitettu isokokoisten datamäärien jakeluun eli toimii hyvin samantyyppisesti kuin mikä tahansa “publish-subscribe”. Suurimpana ongelmana oli se, että ActiveMQ oli lopettanut tiedostopalvelimen tukemisen version 5.1.1 jälkeen. Tiedostopalvelinta tarvitaan isokokoisten tiedostojen jakeluun [15]. Toinen ongelma oli myös dokumentaation vaihtelevuus ja ratkaistujen ongelmatilanteiden vähäinen materiaali.

Prism-ohjelma koodattiin Javalla, koska Java-kieltä on dokumentoitu pitkään eli materiaalin ja ratkaisun saanti ongelmakohtiin on huomattavasti helpompaa. Tämän lisäksi Java pyörii myös monessa eri ympäristössä.

Prism on kuvanstriimausohjelma, joka toteutettiin tätä insinöörityötä varten lähinnä oppimiskokemuksen vuoksi. Ohjelmaa käytetään tässä projektissa JPEG-kuvien välittämiseen muille kuunteleville laitteille tai ohjelmille.

Sokettiprotokolla

Jotta minkäänlainen dataliikenne voisi alkaa, on avattava soketti, joka ottaa vastaan dataliikennettä. Prism:ssa toimii kaksi sokettia. Ensimmäinen, joka ottaa vastaan raakaa binääridataa, ja toinen, joka lähettää datan kaikille kuunteleville laitteille. Vastaanottava soketti on läsnä portilla 8090 ja lähettävä soketti taas portilla 8096.

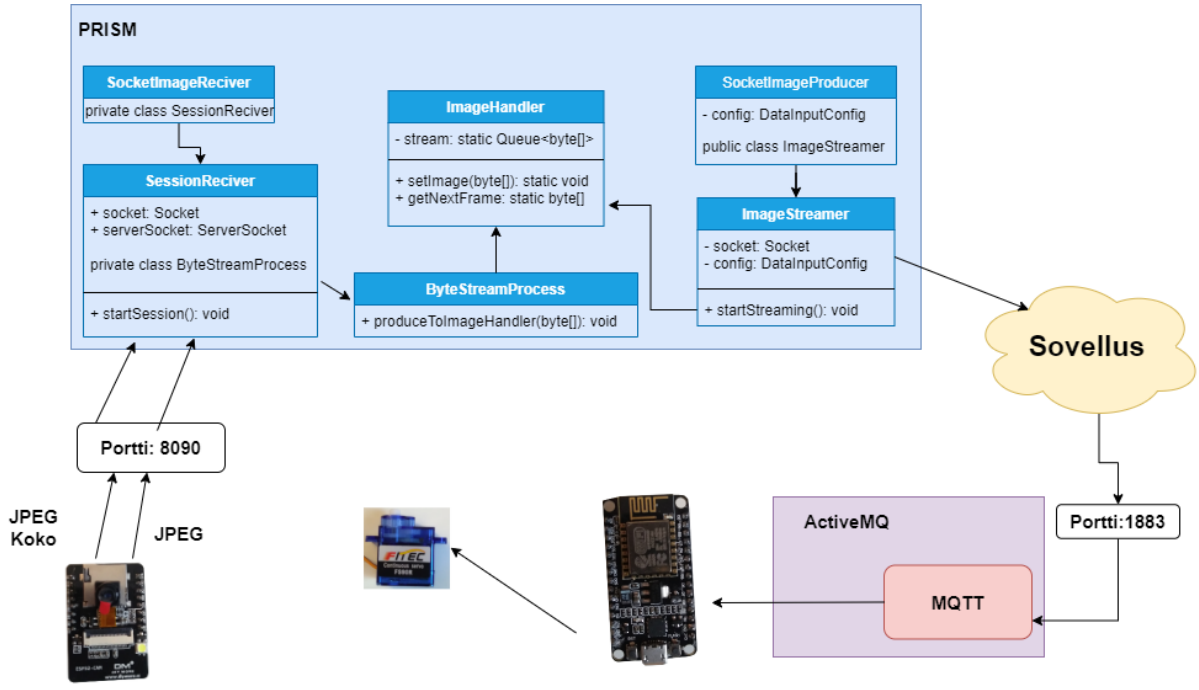


Kuva 8. Kuvataan protokollaa, jota Prism käyttää. Kuvassa ESP32-cam luo TCP-yhteyden, jonka jälkeen lähetetään kuvan koko ja kuva Prism:n TCP-sokettiin.

Laitteen ja vastaanottavan soketin välille oli luotava kommunikaatioprotokolla, jotta molemmat osapuolet kykenisivät käymään keskustelua keskenään. Prism:ssa käytetty sokettiprotokolla käyttää TCP-protokollaa alustana. Protokolla on hyvin yksinkertainen. Laitteesta otetaan ensiksi TCP/IP-yhteys porttiin 8090. Tämän jälkeen laite lähettää tiedoston koon, jonka jälkeen itse tiedoston. Syynä tiedoston koon lähettämiseen oli se, että vastaanottava soketti tietää, milloin tiedosto on kokonaan saapunut ja milloin voidaan varautua toisen tiedoston saapumiseen. Tällä tekniikalla voidaan varmistaa, että tiedostot saapuvat ja lähtevät ehjinä, eikä minkäänlaisena epämääräisenä bittilistana. Protokolla kykenee myös kestämään häiriötekijät, esimerkiksi bittien katoamisen tai heikon verkkoyhteyden. Jos kommunikaatiossa ilmenisi ongelmia, Prism-sokettiprotokolla hylkää odotettavan tiedoston saapumisen ja odottaa niin pitkään, kunnes laite lähettää uuden tiedoston koon. Kuvassa 8 havainnollistetaan Prism:n ja ESP32-cam:n välistä keskustelua.

Arkkitehtuuri

Prism sisältää kolme tärkeätä luokkaa, joista kahta käytiin läpi tässä luvussa jo aikaisemmin. Nämä kaksi luokkaa ovat nimeltään SocketImageReciver ja SocketImageProducer, jotka molemmat ylläpitävät sokettipalvelinta. SocketImageReciver on vastaanottava palvelin, ja SocketImageProducer on taas dataa jakava palvelin. Näiden kahden välillä toimii toinen luokka nimeltä Handler, joka vastaa datan säilyttämisestä jonomuodossa. DataInput-luokka ottaa vastaan bittiejä ja kokoaa ne yhteen, minkä jälkeen se tallentaa ne objektiin nimeltä Keeper, joka lisätään ImageHandler-luokan parametriin stream-jonoon. SocketImageProducer-luokka lukee tietyn konfiguroidun ajan välein jonoa, jolloin viimeiseksi asetettu arvo haetaan ja poistetaan ImageHandler-jonosta ja välitetään kaikille kuuntelijoille.



Kuva 9. Kuvassa visualisoidaan tarpeellisen ympäristön kommunikaatioriippuvuudet. Kuva sisältää myös UML-kartan Prism:sta.

Molemmat luokat SocketImageProducer ja SocketImageReciver käynnistetään säieprosesseina main-luokassa. Jokiselle palvelimeen liittyvälle asiakkaalle luodaan myös oma säieprosessi.

Ohjelma sisältää myös erillisen konfigurointiluokan nimeltä DataInputConfig, jonka avulla voidaan muun muassa säätää datan lähettämisnopeutta niin kuin edellä jo mainittiin ja ImageHandler-luokan jonoparametrin kokoa, eli kuinka pitkäksi jono saa kasvaa, kunnes aletaan tallentamaan edellisen datan päälle. Myös kuvien tallentaminen haluttuun kansioon onnistuu DataInputConfig-luokan avulla.

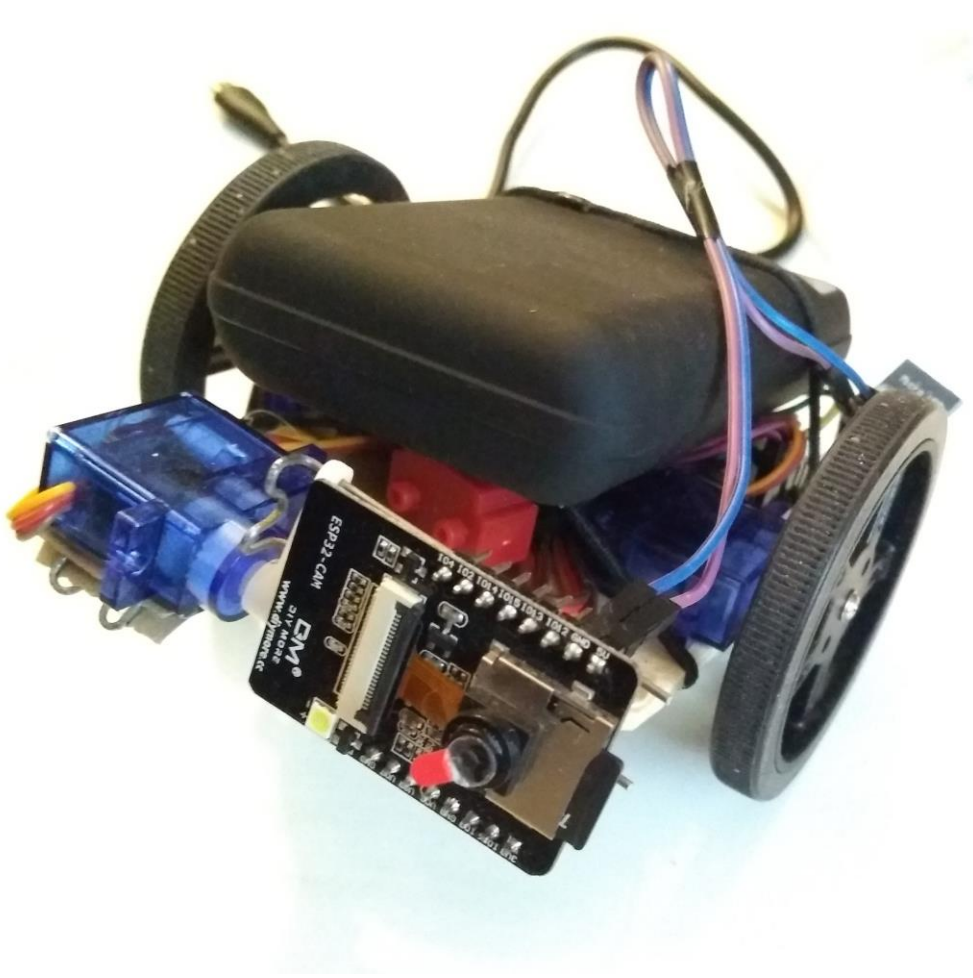
ESP32-cam pitää koodata yhdistämään WiFi-verkkoon, jonka jälkeen se yhdistää itsensä Prism:n vastaanottavaan sokettiin. Tämän prosessin jälkeen istunto on luotu, ja datan lähettäminen voidaan aloittaa. ESP32:een piti ohjelmoida protokollaa käyttävä ohjelma, joka ottaa valokuvan ja lähettää ensiksi otetun JPEG-kuvan koon, jonka jälkeen itse kuvan.

Lyhyesti Prism-ohjelmasta

Prism on kaikin puolin vielä keskeneräinen projekti, joka vaatii vielä paljon kehittämistä, jotta sen voisi julkaista muiden käyttöön. Joka tapauksessa insinöörityöhön se on tarpeeksi toimiva, jotta sillä kyettäisiin demonstroimaan IoT-robotin toimintaa.

4.4 NodeMCU:n ja servomoottorien ohjelmointi

Tässä luvussa syvennytään tarkemmin robotin motoriikkaan ja sen logiikkaan. Näemme, kuinka servomoottorit yhdistetään NodeMCU:hun ja kuinka niiden fyysinen ohjaaminen tapahtuu.



Kuva 10. Ensimmäinen toimiva prototyyppi robotista.

Ensimmäinen prototyyppi on hyvin yksinkertainen. Robotti koostuu kolmesta moottorista, joista kaksi on liimattu kuumaliimalla pahvialustaan renkaiksi ja yksi kameran pitimeksi kuten kuvasta 10 ilmenee. Renkaiden moottorina toimii Servo FS90R, eli jatkuvan rotaatioliikkeen servomoottori. Kameran tukimoottorina käytetään Servo SG90:ta.

Taulukko 2. NodeMCU:n yhdistäminen kahteen Servo FS90R:ään ja yhteen Servo SG90:aan. Lisäksi taulukkoon on lisätty virtalähde.

PowerBank	**NodeMCU**	**Servo FS90R**	**Servo SG90**
5v	microUSB		
	GND	Power (ruskea)	Power (ruskea)
	Vin	Ground (punainen)	Ground (punainen)

	D1 / GPIO 5	Signal (keltainen)	
	D2 / GPIO 4	Signal (keltainen)	
	D4 / GPIO 2		Signal (keltainen)

Taulukko 2 näyttää, miten GPIO-nastat on yhdistetty servomoottorien ja ESP8266 NodeMCU:n välille.

Konfigurointi

Arduino IDE:n konfiguraation asetuksia pitää vaihtaa, koska kyseessä on ESP8266 NodeMCU -piirilevy eli toinen komponentti. Tämä tapahtuu samalla tavalla kuin ESP32-cam:n konfiguroinnissa. Luvussa asetettiin linkki, joka mahdollistaa esp8266-tarkoitettujen ohjelmakirjastojen löytämisen. [5.]

Arduino IDE:stä tulee avata “Boards Manager” ja ladataan paketti nimellä “esp8266 by ESP8266 Community”. Tämän jälkeen “Board”-valikosta etsitään malli nimeltä “NodeMCU 1.0 (ESP-12E Module)”. [5.]

Ohjelmointi

Ennen varsinaisen ohjelman kehittämistä ja lataamista NodeMCU:hun on hyvä kokeilla yhdistettyjen moottorien toimivuus. Insinöörityössä hyödynnettiin internetistä löydettyä valmista koodipohjaa [16]. Koodi sisälsi valmiiksi moottorin seuraavat liikesuunnat: eteenpäin, taaksepäin, vasemmalle ja oikealle. Valmista koodia muokattiin tähän insinöörityöhön sopivaksi. Tarkoituksena oli lähettää viestejä tietokoneesta Serial Monitorista USB-johdon kautta NodeMCU:hun, joka kääntää viestit käskyiksi servomoottorille, näin simuloiden robotin ohjausta.

Käskyviesti rakentuu kahdesta numeroarvosta, jotka on eritelty kaksoispisteellä kuten seuraavalla esimerkkirakenteella havainnollistetaan: {type1}:{type2}.

Ensimmäinen numero “type1” kertoo käskyn tyypistä, esimerkiksi ajetaanko eteenpäin, vasemmalle vai lisätäänkö nopeutta. Toinen numeroarvo “type2” riippuu ensimmäisen

arvon käskytyypistä. Esimerkiksi jos käskytyyppinä on ajaminen taaksepäin niin toinen arvo “type2” kertoo, kuinka monta millisekuntia moottorit pyörivät, mutta jos käsky tyyppinä on nopeuden säätäminen, silloin toinen numeroarvo tarkoittaa itse nopeutta. Taulukossa 3 nähdään käskytyyppien arvo ja niiden tarkoitus.

Taulukko 3. Käskyviestin rakenne ja tarkoitus.

Tyyppi 1	**Tyyppi 2**	**Käskyn Merkkijono / Tarkoitus**
1	200 (ms)	“1:200” / Aja eteenpäin 200 millisekunnin ajan
2	200 (ms)	“2:200” / Aja taaksepäin 200 millisekunnin ajan
3	200 (ms)	“3:200” / Käänny vasemalle 200 millisekunnin ajan
4	200 (ms)	“4:200” / Käänny oikealle 200 millisekunnin ajan
5	80 (nopeus)	”5:80” / Aseta nopeus 80. Nopeus voi olla 0 ja 100 väliltä
8	1	”8:1” / Katso alas. Tyyppi 2 voi olla mitä tahansa.
9	1	”9:1” / Katso ylös. Tyyppi 2 voi olla mitä tahansa.
0	0	”0:0” / Pysähdy. Tyyppi 2 voi olla mitä tahansa.
	-1	Jatkuva liike

Taulukossa 3 huomataan otsikon “type2” kohdalla arvo -1, jossa mainitaan “Jatkuva liike”. Tämä tarkoittaa, että jos “type1” arvo olisi 3, mutta “type2” arvo on -1, niin robotti kääntyy vasemmalle niin pitkään, kunnes toisin käsketään.

4.5 Robotin ohjaus hyödyntäen MQTT

Luvussa kerrotaan, miten tietokoneen ja robotin välinen USB-ohjaus korvataan langattomalla viestinnällä hyödyntäen MQTT:tä.

ActiveMQ kykenee palvelemaan asiakasohjelmaa MQTT-protokollan avulla, mutta NodeMCU:n pitää kyetä keskustelemaan samalla kielellä. NodeMCU tarvitsee MQTT asiakasohjelman, jotta kykenisi vastaanottamaan viestejä ActiveMQ:sta.

Tästä syystä työssä käytettiin valmista asiakaskirjastoa MQTT-keskustelua varte. Kirjastoja voidaan löytää ja ladata Arduino IDE:ssä sijaitsevan Library Manager -työkalun kautta. Työkalun löytää Arduino IDE:stä “sketch” > “Include Library” > “Manage Libraries”.

Kirjaston löytää nimellä EspMQTTClient. Kirjasto on Patrick Lapointen kehittämä, ja se valittiin työhön sen helppokäyttöisyyden vuoksi. Kirjasto hoitaa automaattisesti yhteydenoton WiFi-verkkoon ja MQTT-välittäjään. [17.]

Seuraten kirjaston README.md [17] -ohjeita onnistuttiin kirjasto implementoimaan NodeMCU:n aikaisemman ohjaamiskoodin kanssa. ActiveMQ:n käynnistyessä voitiin web-konsolin avulla lähettää viesti aiheeseen, jota NodeMCU kuuntelee. Viestin saapuminen NodeMCU:hun varmistetaan Serial Monitorin avulla. Tämän viritelmän avulla kyettiin ohjaamaan robottia langattomasti.

5 Applikaatiot ja tekoäly

5.1 Puppeteer

Insinöörityössä luotiin kaksi työkalua, joista ensimmäinen oli Prism ja toinen Puppeteer.

Robotti saatiin siihen vaiheeseen, että se kykenee lähettämään tietovirtaa ja vastaanottamaan käskyjä, jotka käännetään moottoritoiminnoiksi. Enää puuttuu niin sanotut aivot eli jokin äly, joka päättää saadun tietovirran perusteella mitä tehdä. Tätä varten kehitettiin Puppeteer, joka on web-applikaatio, jolla tarkkaillaan ja ohjataan robottia selaimen avulla. Puppeteerin avulla voidaan nähdä reaaliaikaista kuvalähetystä ja lähettää käskyjä nappuloiden avulla sekä muokata käskyjen arvoja syötekentässä.

Applikaation ohjelmoinissa on käytetty Node.js:ää, koska se on nopea ja tuttu ympäristö. Ohjelmointikielinä käytetiin JavaScriptiä back-endiä varten, ja HTML-kieltä front-endissä. Node.js-ympäristöstä tekee käteväksi sen, että komentorivin kautta voidaan ladata tarvittavia paketteja **npm:n** eli *Node package manager*in avulla. Seuraavalla komennolla voidaan ladata tarvittavia paketteja: *npm install* {kirjaston nimi}.

Seuraavassa esimerkissä käydään läpi applikaatiossa käytettyjä kirjastoja.

socket.io

socket.io mahdollistaa reaaliaikaisen kommunikaation selaimen ja serverin välilä. [18.]

mqtt

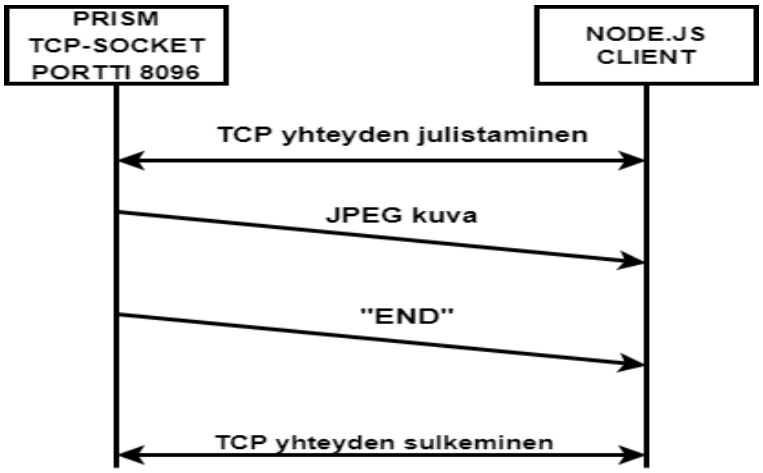
mqtt on asiakas kirjasto MQTT protokolla kommunikaatiota varten. [19.]

express

express on web applikaatiota varten kehitetty palvelin framework, jonka avulla voidaan käsitellä monenlaisia pyyntöjä esimerkiksi GET, PUT, POST ja DELETE pyyntö tyyppejä. [20.]

Ohjelmointi

Sovelluksen ohjelmoinnissa hyödynnettiin muitakin kirjastoja, jotka löytyivät jo valmiiksi Node.js:stä. Projekti aloitettiin ottamalla yhteys Prism:aan lähettävään sokettiporttiin 8096. Alkuun yhteyttä ei kyetty muodostamaan. Syynä tähän oli väärän protokollan käyttö. Nimittäin yhteys yritettiin luoda socket.io-kirjaston avulla, mutta socket.io käyttää web-sokettia eikä tcp-pohjaista protokollaa. Tämä korjattiin Node.js:n mukana tulevan net kirjaston [21] avulla. Net mahdollistaa asynkronisen API-rajapinnan TCP-palvelimelle ja asiakkaalle. Kirjaston avulla voidaan luoda asiakkaasta istunto Prism:n sokettiin ja vastaanottaa dataa, jota lähetetään ESP32-cam-laitteesta.



Kuva 11. Prism-ohjelman ja Node.JS-asiakkaan välinen kommunikaatioprotokollaratkaisu.

Net-kirjaston kanssa ilmeni ongelma, kun Prism:sta saapuneet kuvan bitit yritettiin pakata yhteen. Jostakin syystä kirjasto vastaanottaa 2 kb dataa, jonka jälkeen se ottaa loput saapuvasta datasta [22]. Eli kun haluttiin tallentaa JPEG-kuva, tallentui se kahdesti. Tätä varten Prism-lähetysprotokollaa piti muokata. Prism:n lähetysprotokolla muokattiin siten, että jokaisen kuva lähetyksen lopuksi lähetetään viesti “END” Prisma:sta. Node.js osaa “END”-viestin pituuden perusteella filtteröidä ja pakata bitit JPEG-kuvaksi. Kuvassa 11 havainnollistetaan datansiirtoa Prism:aan ja kuuntelevan asiakkaan välillä.

Kuvan bittien pakattua palvelimessa ne käännetään Base64:n avulla ja lähetetään front-endiin käyttäen socket.io:ta. Front-endin pitää myös ottaa käyttöön socket.io asiakasohjelma, jotta web-sokettiviestejä voidaan saada back-endistä. Esimerkkikoodi 4 HTML-elementin avulla voidaan käyttää ulkoista socket.io-kirjastoa.

```
<script src="https://cdnjs.cloud-
flare.com/ajax/libs/socket.io/2.3.0/socket.io.js"></script>
```

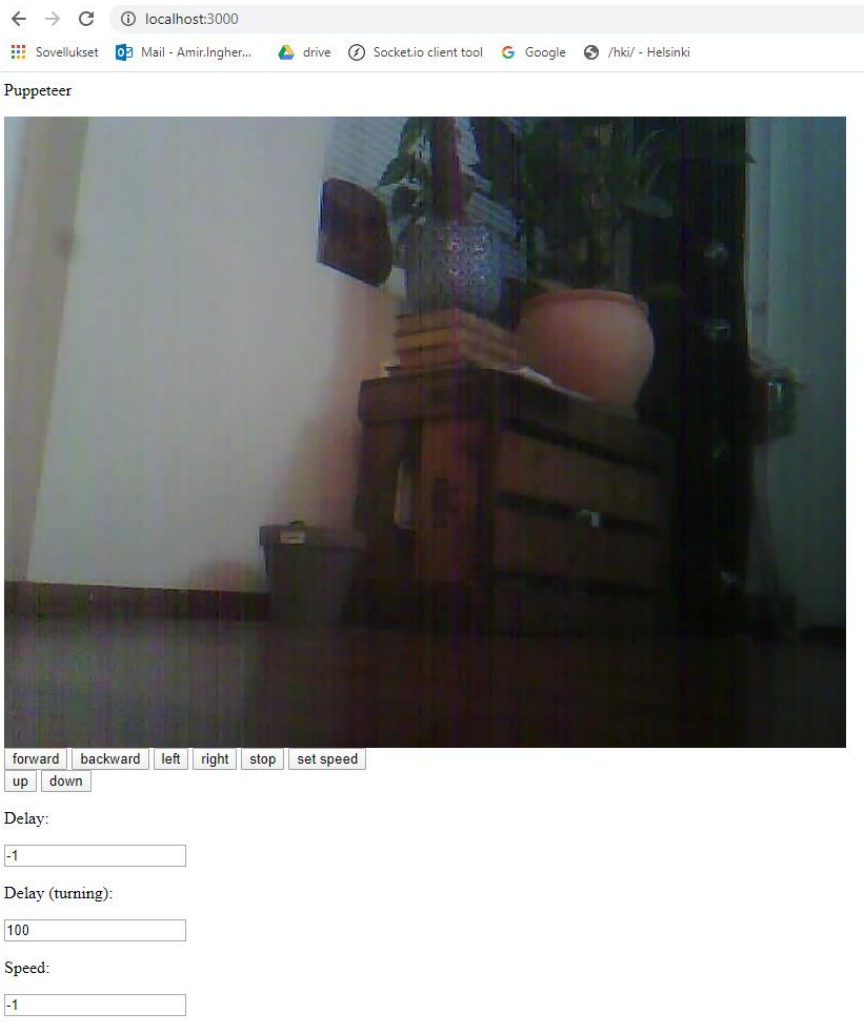
Esimerkkikoodi 4. Socket.io-kirjaston käyttöönotto HTML-sivulla.

Base64-kuva julkaistaan selaimen ikkunaan esimerkkikoodin 5 JavaScript-koodin avulla. Kyseinen JavaScript-koodi päivittää kuvaelementin, joka kerta kun uusi web-sokettiviesti saapuu.

```
const socket = io.connect("http://localhost:3000");

socket.on('image',(data) => {

      console.log("data");

      const imageElm =document.getElementById('image');

      imageElm.setAttribute("src",""+data.buffer);
});
```

Esimerkkikoodi 5. Socket.io-kirjaston avulla vastaanotettujen kuvien striimaaminen HTML-sivun *image* id:tä kantavan elementin näkymään.

Yllä asetetulle elementille imageElm asetetaan attribuutti “src”, jonka arvoksi asetetaan "". Tämä kertoo, että kyseistä kuvaa pitää kohdella Base64-käännettynä JPEG-kuvana.



Kuva 12. Puppeteer HTML -sivun näkymä osoitteessa *http://localhost:3000*. Sivussa nähdään reaalikuvaa robotista ja vaihtoehtoisia ohjaustoimintoja sekä konfiguroitavissa olevia syötekentän arvoja.

Front-end sisältää kahdeksan nappula ja kolme syöttökenttää, kuten kuvassa 12 nähdään.

Express-kirjaston avulla luodaan palvelin, joka pyörittää API-rajapintaa. Tämän avulla voidaan lähettää komentoja. Komennot käännetään viesteiksi mqtt-kirjaston avulla ja lähetetään ActiveMQ-välittäjään. Rajapintaan lähetetään GET-pyyntöjä seuraavan esimerkkiosoitteen mukaan:

http://localhost:3000/command?first={type1}&secound={type2}.

Parametriin “first” asetetaan käskytyyppi 1 (type1) ja parametriin “secound” laitetaan käskytyyppi 2 (type2). Aikaisemman luvun taulukossa 3 voidaan havainnoida käskytyyppien merkitykset. Parametrien arvot sisällytetään MQTT-viestiin, joka lähetetään esimerkkikoodin 6 avulla.

```
client.subscribe(topic, function (err) {
      if (!err) {
            client.publish(topic, first+':'+secound)
      }else{
            console.log(err);
      }
})
```

Esimerkkikoodi 6. Node.JS API-rajapinnan kautta ajettava komento, joka lähettää saadut parametrit MQTT-välittäjälle.

Puppeteer-sovelluksen avulla kyetään toimimaan väliaikaisena älynä, joka näyttää kuvavirtaa, jonka perusteella ohjataan robottia. Ohjelma mahdollistaa myös kuvien tallentamisen paikalliseen muistiin.

5.2 Esteiden väistäminen kuvien perusteella

Tässä luvussa kehitetään robotille simppeli tekoäly esineiden ja seinien väistämiseen. Tekoälyn tulee kyetä ymmärtämään milloin voi ajaa ja milloin pitää väistää. Luvussa tutustutaan Anaconda-ohjelmaan ja tekoälyn kouluttamiseen kerätyllä kuvadatalla. Työssä toteutettu tekoäly on kokeellinen ratkaisu esteiden väistämiseen, joten ei ole varmaa toimiiko se kaikissa tilanteissa oikein.

Anaconda

Tekoälyn kouluttamiseen käytettiin Anaconda-työkalua. Anaconda on koneoppimiselle ja datatieteelle tarkoitettu applikaatio, joka sisältää lukuisan määrän työkaluja eri tarkoituksiin. Ohjelman mukana tulee myös paketinhallintaa varten kehitetty Conda, joka hallinnoi riippuvuuksia, kirjastoja ja ympäristöä. [23.]

Jupyter Notebook

Jupyter Notebook on avoimen lähdekoodin web-sovellus, joka tulee Anacondan mukana. Sovelluksen avulla voidaan kirjoittaa python-koodia selaimen kautta. Koodin pätkiä voidaan ajaa erissä ilman, että tarvitsee koko ohjelmaa ajaa aina uudestaan. Sovellus kykenee myös narratiivisen tekstin ja datan visualisointiin sekä paljon muutakin. Jupyter Notebook on datatieteilijöille tarkoitettu IDE. [24.]

TenserFlow

TenserFlow on Google Brain -ryhmän kehittämä avoimen lähdekoodin kirjasto koneoppimiselle. Kirjaston avulla kyetään luomaan ja kouluttamaan koneoppimismalleja. [25.]

Keras

Keras on korkean tason neuroverkko API, joka on kirjoitettu pythonilla. Rajapinnan fokuksena on mahdollistaa nopeaa kokeilua. Kerasia käytetään, kun halutaan syväoppimiskirjasto, jolla

- helposti ja nopeasti kehittää prototyyppejä
- luodaan konvoluutioneuroverkkoa tai takaisinkytketty neuroverkkoa tai molempia
- prosessointi voidaan tehdä CPU:ssa tai GPU:ssa. [26.]

5.2.1 Datan kerääminen

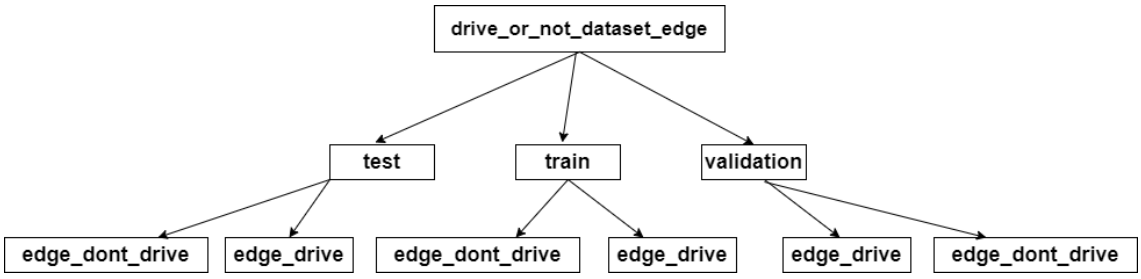
Tekoälyn oppimista varten tarvitaan dataa, josta oppia. Tätä dataa pitää kerätä ja luokitella, koska tarkoituksena on kehittää hyvin yksinkertainen tekoäly, joka päättää, voiko ajaa vai ei. Eli JPEG-kuvadatan perusteella luokitellaan kuvat kahteen luokkaan.

Puppeteer-ohjelman avulla voidaan tallentaa kuvia kouluttamista varten. Robotilla ajettiin ympäri taloa ja tallennettiin kuvia Puppeteer-ohjelman sisällä olevaan kansioon nimeltä ”img”. Kuvia tallennettiin 2 267 kappaletta *img*-kansioon. Dataa on yleensä hyvä olla paljon, jotta malli kykenee ennustamaan tarkemmalla todennäköisyydellä.

Kerätyt kuvat luokiteltiin kahteen kansioon manuaalisesti. Ensimmäisen kansion nimi on ”drive”, johon kerätään kuvat, joiden perusteella voidaan ajaa eteenpäin. Toiseen kansioon nimeltä ”dont_drive” tallennettiin kuvat, joiden perusteella ei voitu ajaa eteenpäin.

5.2.2 Tekoälyn sovittaminen

Kouluttamista varten käytetään Anaconda-sovelluksen mukana tulevia ohjelmia. Yksi näistä on Jupyter Notebook, jota käytetään työssä koodin kirjoittamiseen sekä sen raportoimiseen. Ennen koulutusprosessia pitää kuvat järjestää kuvan 13 mukaiseen kansiorakenteeseen. Tätä prosessia helpottaakseen käytettiin yksinkertaista python-ohjelmaa, joka jakaa kuvat annetun prosenttiluvun perusteella sekä satunnaisesti valitsee kuvia näihin kansioihin.



Kuva 13. Tekoälyn kouluttamista varten rakennettu kansiorakenne.

Kansiot jaetaan kolmeen osaan nimeltä ”test”, ”train” ja ”validation”. Train-kansio sisältää itse koulutukseen tarkoitetut kuvat ja test-kansio sisältää kuvia, joita käytetään koulutetun mallin ennustuskyvyn testaamiseen. Viimeinen kansio on validation-kansio. Validoinnin avulla voidaan arvioida, kuinka hyvin malli oppii koulutuksen aikana ja jälkeen. Voimme myös nähdä, jos malli ylisovittaa tai alisovittaa. 50 % kuvista on tallennettu train-kansioon, 25 % test-kansioon ja loput 25 % validation-kansioon.

Työssä käytettiin neuroverkkoa luokittelemaan kuvien perusteella, voiko kuvassa ajaa eteenpäin vai ei. Neuroverkkoalgoritmi käy hyvin, kun koulutusdatana on kuvat tai moniulotteiset matriisit. Robotti tulee liikkumaan lattiatasolla, kuten kuvassa 14 voidaan huomata.

Kuva 14. Robotin näkymä. Tästä perspektiivistä robotti näkee.

Ideaalitilanteessa koulutettu malli kykenee hahmottamaan kuvan horisontin perusteella eli voiko ajaa eteenpäin vai ei. Jos horisonttia leikataan millään tavalla, niin tiedetään, että edessä on este. Insinöörityössä käytettyä neuroverkkomallin rakennetta voidaan havainnollistaa esimerkkikoodissa 7 olevan python-ohjelmakoodin perusteella.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(im-
age_x_size, image_y_size, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Esimerkkikoodi 7. Python-kielellä rakennettu konvoluutioneuroverkon rakenne.

Esimerkkikoodin 7 mallin rakenne sisältää kolme konvoluutiokerrosta eli Conv2D. Konvoluutiokerros pilkkoo kuvan pienemmiksi paloiksi ja kertoo ne painoarvon kanssa. Yllä oleva mallin toisessa rivissä määritellään 32 filtteriä, eli kuva pilkotaan 32 osaan. Pilkottujen koko on 3x3. Rivissä myös määritellään input_shape paremetrissä kuvan koko. Tässä työssä käytettiin 800x600x3, jossa 800 on leveys ja 600 korkeus. Arvo 3 kertoo kuvan syvyydestä, koska kuva on värillinen eli RGB-kuva, niin se sisältää kolme arvoa, jos arvo olisi 1, niin kuvaa käsiteltäisiin mustavalko kuvana. [27.]

ReLu (rectified linear unit) toimii kerroksen aktivointifunktiona. Alla oleva yhtälö kuvaa aktivointifunktion luonnetta:

$$f(x) = \max(0, \mathrm{x}) \quad (1)$$

Jos x arvo on negatiivinen, arvo on 0. [28.]

MaxPooling2D on metodi, jolla pienennetään konvoluutiokerroksen kokoa valitsemalla tietyn kokoiselta kaksulotteisesta alueesta suurin pixeliarvo. Mallissa kerrokset muutetaan kokoon 2x2. [29.]

Flatten [30] kerroksella muutetaan konvoluutiokerroksen moniulotteinen matriisi lineaariseksi vektorilistaksi. Tätä tarvitaan aina, kun halutaan siirtyä konvoluutiokerroksen käytöstä klassisempaan kokonaan yhdistettyyn verkostoon eli Densen-kerrokseen [30]. Dense-kerros sisältää 512 neuronia, eli se kykenee ottamaan vastaan 512 arvoa. Ensimmäisen Dense-verkon jälkeen seuraa Dropout [30], joka satunnaisesti lakkauttaa osan neuroneista verkossa. Tässä mallissa Dropout on 0,2 eli 20 % kerroksen neuroneista lakkautetaan. Viimeinen rivi sisältää Dense-kerroksen, joka ottaa vastaan yhden arvon. Tämän arvon aktivointifunktiona käytetään sigmoidia. Arvo voi olla jotakin 0 ja 1 välitä. Sigmoid-yhtälöä kuvaa kaava:

$$f(x) = 1/(1 + \mathrm{e}\text{^} - \mathrm{x}) \quad (2)$$

Koulutetun mallin kautta saadaan arvo, joka asettuu skaalalle 0-1. Jos arvo on 0, eli false, niin se tarkoittaa, että ei saa ajaa. Arvon ollessa 1, eli true, saa ajaa. Mikäli arvo

on jotakin näiden väliltä, on liikkuminen riippuvainen ohjelmassa asetetusta kynnysarvosta. Useassa tilanteessa kynnyksenä käytetään 0,5. [31.]

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch = 20,
      verbose = 1,
      epochs = 25,
      validation_data = validation_generator,
      validation_steps = 20)
```

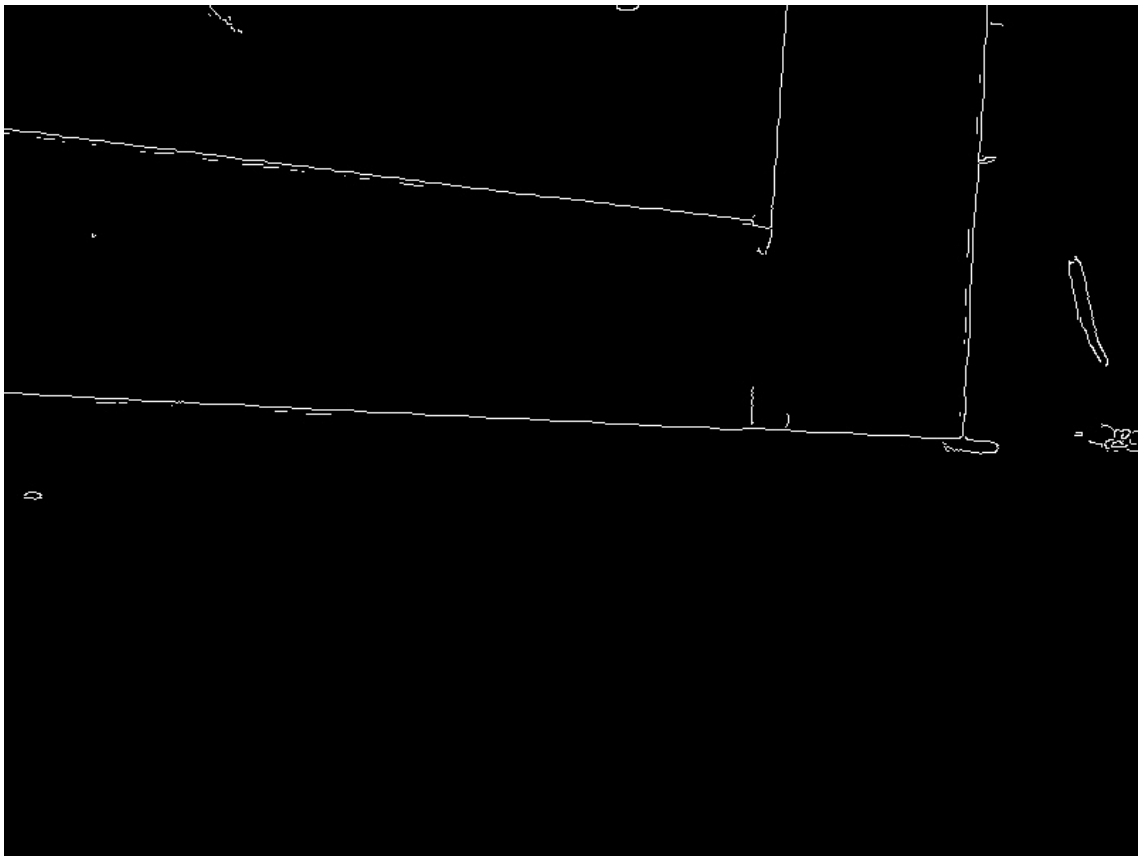
Esimerkkikoodi 8. Mallin kouluttaminen aloitetaan.

Mallin kouluttaminen aloitetaan yllä näkyvällä koodilla. Parametri

- train_generator sisältää koulutettavat kuvat
- steps_per_epoch on kokonaismäärä askeleita (batches of samples) ennen yhden erän (epoch) valmistumista ja siirtymistä toiseen erään
- verbose määrittää näytetäänkö koulutusprosessin kulku lokissa
- epochs kertoo erien määrästä, joita käytetään mallin kouluttamiseen
- validation_data sisältää validointiin käytettävät kuvat
- validation_steps on sama kuin steps_per_epoch:s, mutta koulutuskuvien sijaan käytetään validointiin tarkoitettuja kuvia. [32.]

Mallin rakennetta on hyvä vaihdella ja kokeilla erilaisia yhdistelmiä neuroverkon kerroksien kanssa. Mallin kouluttamiseen meni noin 4 tuntia. Muita rakenteita ei kokeiltu tässä insinöörityössä.

Ensimmäisen koulutetun mallin ennustustulokset eivät olleet hyvät. Malli ennusti oikein noin 46 % dont_drive-kansion ja 32 % drive-kansion testikuvista. Tämä tarkoittaa, että malli ei ole lainkaan hyödyllinen. Kuvia tarvittaisiin huomattavan paljon enemmän tai selkeämpi ympäristö, jotta malli oppisi paremmin.



Kuva 15. Robotin näkymä kuvankäsittelyn jälkeen. Kuvaa on muokattu lisäämällä Canny Edge -tunnistus.

On olemassa toinenkin vaihtoehto, jolla mallin suorituskykyä voidaan parantaa. Vaihtoehto on kuvankäsittely. Kuvaa käsitellään sillä tavalla, että se korostaa niitä tekijöitä, joihin halutaan neuroverkon keskittyvän. Työssä käytetyn mallin halutaan keskittyvän horisontti alueelle. Tätä varten työssä käytetään Canny Edge -tunnistamista. Kuvankäsittelymenetelmä korostaa reuna-alueita eli kaikki kuvassa näkyvä pikseli jatkumo korostetaan, mutta kaikki muu poistetaan. Käsittelyn jälkeen kuva sisältää valkoisella korostettuja reunoja mustalla taustalla. Kuvassa 15 huomataan, että horisontti korostuu. Kuva 14 on kuvan 15 käsittelyn jälkeen.

Kuvankäsittelyä varten kehitettiin ohjelma, joka käsittelee kaikkia 2 267 kuvaa Canny Edge -tunnistamisalgoritmin avulla. Tätä varten ladattiin kirjasto nimeltä OpenCV, joka sisältää laajan määrän kuvankäsittelyä varten kehitettyjä metodeja. Kuvia käsitellään esimerkkikoodin 9 avulla.

```
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

imgGray = cv2.bilateralFilter(imgGray,9,30,30)

gray = cv2.Canny(imgGray, 20, 30)
```

Esimerkkikoodi 9. Kuvankäsittelyprosessi, jossa kuva muutetaan ensimmäisessä rivissä mustavalkoiseksi. Toisessa rivissä kuvaa sumennetaan ja kolmannessa kuvaan lisätään Canny Edge -tunnistaminen.

Sama mallirakenne uudelleen koulutettiin käsitellyillä kuvilla. Uusi malli ennusti oikein dont_drive-kansion testikuvista 66 %:n todennäköisyydellä ja drive-kansion testikuvista se ennusti 98 %:n todennäköisyydellä oikein. Sovitettu malli oli tarpeeksi hyvä käyttöönottoon.

```
# Save the model
model.save('../ai_model_v1/drive_or_not_model_edge.h5')
```

Esimerkkikoodi 10. Koulutettu malli tallennetaan paikalliseen kansioon osoitteen mukaisesti

Testailuprosessin päätteeksi malli tallennetaan yllä olevalla komennolla.

5.3 Mallin käyttöönotto

Koulutetun mallin ympärille piti rakentaa yksinkertainen logiikka, joka ohjaa robottia mallin ennusteen perusteella. Logiikka ohjelmoitiin myös pythonilla, koska valmiiksi koulutettu malli oli helpompi ottaa käyttöön. Mallia otettiin käyttöön alla olevalla komennolla.

```
model = load_model('../ai_model_v1/drive_or_not_model_edge.h5')
```

Esimerkkikoodi 11. Koulutetun mallin käyttöönotto tallennetusta osoitteesta.

Päätöksen kynnyksenä käytettiin arvoa 0,9. Aina kun malli ennusti enemmän kuin 0,9 niin robotille lähetettiin käsky ajaa eteenpäin 400 millisekunnin ajan, mutta mallin ennustaessa alle 0,9 lähetetään kääntymiskäsky robotille.

```
    img = cv2.bilateralFilter(img,9,30,30) #sumenna kuvaa, jotta päästään
kohinasta eroon
    img = cv2.Canny(img, 20, 30)             # canny edge -tunnistaminen

    result = predict(img) # ennustetaan saako ajaa vai ei

    print(result) #tulostetaan tulos

    if(result < 0.9): #verrataan ennustusta kynnysarvoon
        mqtt.publish('mytopic/test', '3:400') #käänny vasemmalle 400ms ajan
    else:
        mqtt.publish('mytopic/test', '1:400') #aja eteenpäin 400ms ajan
```

Esimerkkikoodi 12. Saapuvia kuvia käsitellään Canny Edge -tunnistamismetodilla, jonka jälkeen metodissa predict koulutettu malli ennustaa arvolla, joka voi olla nollan tai yhden välillä. Metodista saadun arvon perusteella lähetetään käsky MQTT-välittäjälle.

Robotilla ei ole varsinaista päämäärää, vaikka se olisikin tehtävissä. Sen sijaan robotti pyrkii väistämään esteitä ja ajamaan niin pitkään eteenpäin kuin voi.

Kuvat haetaan prism:sta suoraan, samalla logiikalla, kuin Node.js-ympäristössä. *Net*-kirjaston sijaan käytettiin *socket*-kirjastoa [33]. Kehitetty ohjelma oli helppo integroida käyttöön joka kerta, kun ohjelmaa muokattiin. Integroinnin aikana robotti vain pysähtyi odottamaan uusia käskyjä. Ohjelman tulokseksi saatiin robotti, joka kimpoili ympäri huonetta.

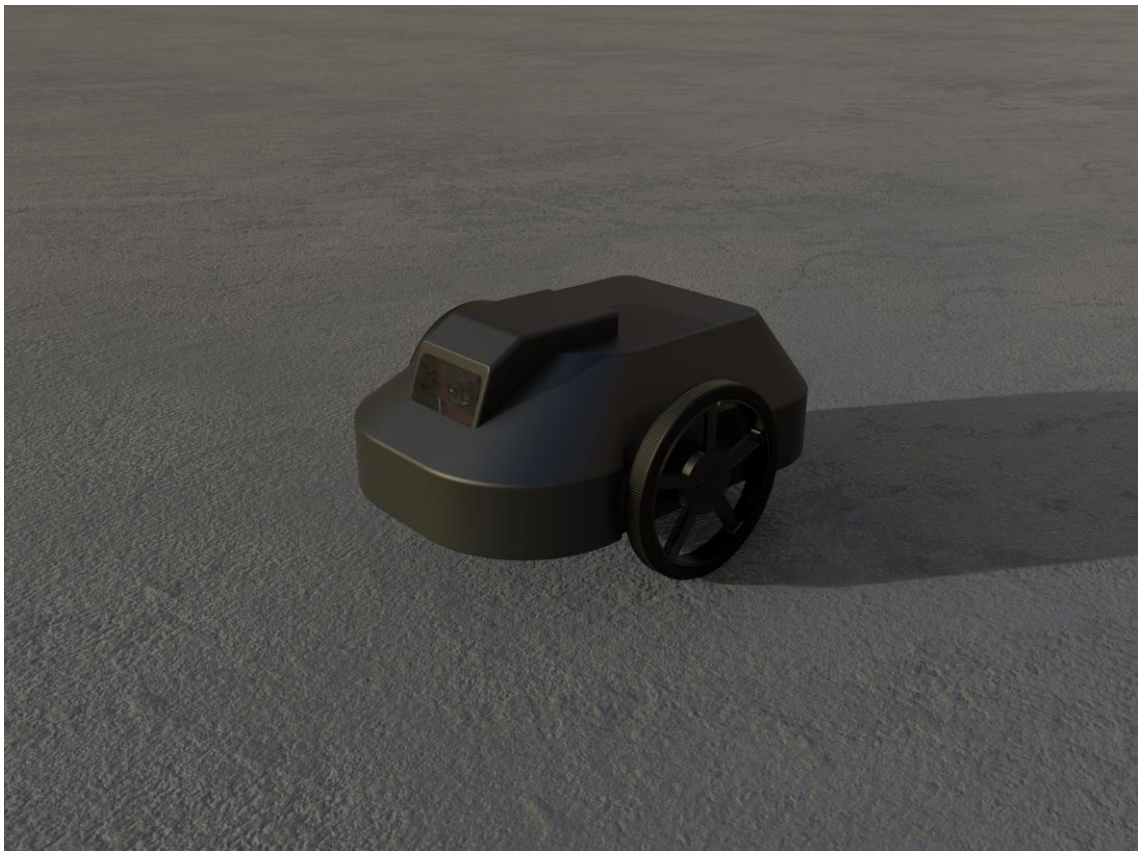
6 Robotin tulevaisuus

IoT-robotti on vielä keskeneräinen monella tapaa. Kehittämistyötä tarvitaan vielä paljon. Insinöörityössä saatiin toimiva demo, jolla kyettiin demonstroimaan ideaa. Projekti ei ole vielä tuotantovalmis. Luvussa pohditaan robotin ja prism:n mahdollista tulevaisuutta.

Robotti

Robotti on kehitetty lähinnä oppimistarkoitukseen. Sen avulla on helppo kokeilla esimerkiksi tekoälyä, kun hyödynnetään robotista lähtevää kuvadataa. Tulevaisuudessa robotille on tarkoitus kehittää tekoälyä, joka kykenee navigointiin, kommunikaatioon ja hahmojen tunnistamiseen. Pääasiassa robotti toimii eräänlaisena hiekkalaatikkona,

johon voi kokeilla erilaisia sovelluksia helposti ilman minkäänlaista viivettä. Riittää kun konfiguraatio on hoidettu oikein.



Kuva 16. Aaro Sariolan suunnittelema design robotille.

Robotille on suunniteltu insinöörityön aikana myös kotelot. Tarkoituksena on kehittää kotelot, jotka olisivat vaivattomia koota ja helppo 3d-printata. Kokoamisvaiheessa halutaan välttää kuumaliiman ja ruuvien käyttöä. Kotelon suunnittelija on Aaro Sariola. Kuvassa 16 voidaan nähdä ensimmäisen version design.

Prism

Prism on tällä hetkellä välttämätön ohjelma robotin toimivuuden kannalta. Prism-ohjelman kehittämistä varten tarvitaan isoja määriä kehitystunteja, jotta sen julkaisu olisi mahdollista. Tulevaisuudessa se halutaan, jos mahdollista, korvata jollakin paremmin soveltuvalla ohjelmalla, jolle on kehitetty kirjasto tai kattava dokumentaatio ESP-malleja varten.

Korvattavan ohjelman etsinnän aikana ohjelmaa vielä kehitetään toistaiseksi. Tällä hetkellä Prism kykenee palvelemaan yhtä robottia kerralla. Jatkokehitystä varten Prism:an palvelukaistaa pyritään laajentamaan, jotta ohjelma voisi ottaa dataa vastaan monelta robotilta samanaikaisesti.

7 Yhteenveto

Insinöörityössä kehitettiin IoT-robotti, jota kyetään ohjaamaan etänä. Lisäksi toteutettiin yksinkertainen tekoäly käyttäen neuroverkkoalgoritmia esteiden väistämiseen. Tuloksena saatiin rakennettujen ja olemassa olevien ohjelmien avulla toimiva ympäristö demonstroimaan IoT-robotin toiminnallisuutta. Demonstroinnin tarkoituksena oli havainnollistaa IoT-robottien skaalautuvuutta sekä vaivatonta sovelluksen integrointia.

Robotille ei keritty kehittämään ohjelmaa, jonka avulla robotille oltaisiin luotu syvempi tarkoitus. Myös insinöörityöhön suunnitellut apuohjelmistot, kuten Prism, jäivät keskeneräisiksi, myöskään robottia varten suunnitellut kotelot eivät ehtineet käyttöönottoon. Roboti saatiin toimimaan, vaikka kaikkia yksityiskohtia ei ehditty työssä toteuttamaan. Robotin kehitys jatkuu insinöörityöstä huolimatta.

Lähteet

1 The Internet of Robotic Things. Verkkoaineisto. ResearchGate. <https://www.researchgate.net/publication/323418535_The_Internet_of_Robotic_Things_A_review_of_concept_added_value_and_applications>. Luettu 9.4.2020.

2 IoRT: definition, market and examples. Verkkoaineisto. I-SCOOP. <https://www.i-scoop.eu/internet-of-things-guide/internet-robotic-things-iort/>. Luettu 9.4.2020.

3 ESP32-CAM Review. Verkkoaineisto. HackSpace. <https://hackspace.raspberrypi.org/articles/esp32-cam-review>. Luettu 11.4.2020.

4 FT232RL. Verkkoaineisto. kuongshun. < http://fi.szks-kuongshun.com/uno/uno-board-shield/ft232rl-ft232-usb-to-ttl-download-cable-to-serial.html >. Luettu 11.4.2020.

5 Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE. Verkkoaineisto. Last Minute Engineers. <https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/> Luettu 11.4.2020.

6 FEETECH FS90R Micro Continuous Rotation Servo. Verkkoaineisto. Pololu. <https://www.pololu.com/product/2820>. Luettu 11.4.2020.

7 Servo Motor SG-90. Verkkoaineisto. Components101. <https://components101.com/servo-motor-basics-pinout-datasheet> Luettu 11.4.2020.

8 ESP32-CAM Video Streaming and Face Recognition with Arduino IDE. Verkkoaineisto. Random Nerd Tutorials. <https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/> Luettu 11.4.2020.

9 Arduino IDE. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Arduino_IDE> Luettu 11.4.2020.

10 Arduino - Begin. Verkkoaineisto. Arduino. <https://www.arduino.cc/en/Serial.Begin>. Luettu 11.4.2020.

11 Real-time Report System for Online Gaming Platform. Verkkoaineisto. Theseus. <https://www.theseus.fi/bitstream/handle/10024/167104/Meng_Yang.pdf?sequence=2&isAllowed=y>. Luettu 11.4.2020.

12 Running the Web Console on ActiveMQ 5.0 or later. Verkkoaineisto. ActiveMQ. <https://activemq.apache.org/web-console.html>. Luettu 11.4.2020.

13 MQTT Version 5.0. Verkkoaineisto. OASIS. <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> Luettu 11.4.2020.

14 High-frequency positioining. Verkkoaineisto. Digitransit. <https://digi-transit.fi/en/developers/apis/4-realtime-api/vehicle-positions/>. Luettu 11.4.2020.

15 Fileserver webapps doesn't works out of the box. Verkkoaineisto. Apache. <https://issues.apache.org/jira/browse/AMQ-6195>. Luettu 11.4.2020.

16 Arduino Controlled Servo Robot (SERB) - Simple Routines. Verkkoaineisto. Instructables. <https://cdn.instructables.com/ORIG/FKK/R8UT/FNNJYJIS/FKKR8UTFNNJYJIS.txt>. Luettu 11.4.2020.

17 MQTT and Wifi handling for ESP8266 and ESP32. Verkkoaineisto. GitHub. <https://github.com/plapointe6/EspMQTTClient/blob/master/README.md>. Luettu 11.4.2020.

18 What Socket.IO is. Verkkoaineisto. Socket.io. <https://socket.io/docs/>. Luettu 11.4.2020.

19 MQTT.Js. Verkkoaineisto. GitHub. <https://github.com/mqttjs/MQTT.js#readme>. Luettu 11.4.2020.

20 Node.js Express FrameWork Tutorial - Learn in 10 Minutes. Verkkoaineisto. Guru99. <https://www.guru99.com/node-js-express.html>. Luettu 11.4.2020.

21 Node.js v13.12.0 Documentation. Verkkoaineisto. Node.js. <https://nodejs.org/api/net.html>. Luettu 11.4.2020.

22 Sending files through net.socket. Verkkoaineisto. Stack Overflow. <https://stackoverflow.com/questions/57516023/sending-files-through-net-socket>. Luettu 11.4.2020.

23 Anaconda Individual Edition. Verkkoaineisto. Anaconda. <https://www.anaconda.com/distribution/>. Luettu 11.4.2020.

24 Project Jupyter. Verkkoaineisto. Jupyter. <https://jupyter.org/>. Luettu 11.4.2020.

25 What is TensorFlow? The machine learning library explained. Verkkoaineisto. Infoworld. <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>. Luettu 11.4.2020.

26 Keras Documentation. Verkkoaineisto. Keras. <https://keras.io/>. Luettu 11.4.2020.

27 Convolutional Layers - Keras Documentation. Verkkoaineisto. Keras. <https://keras.io/layers/convolutional/>. Luettu 11.4.2020.

28 Rectifier. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Rectifier_(neural_networks)>. Luettu 11.4.2020.

29 Pooling Layers - Keras Documentation. Verkkoaineisto. Keras. <https://keras.io/layers/pooling/>. Luettu 11.4.2020.

30 Core Layers - Keras Documentation. Verkkoaineisto. Keras. <https://keras.io/layers/core/>. Luettu 11.4.2020.

31 Sigmoid Function. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Sigmoid_function>. Luettu 11.4.2020.

32 Model - Keras Documentation. Verkkoaineisto. Keras. <https://keras.io/models/model/>. Luettu 11.4.2020.

33 17.2. socket — Low-level networking interface. Verkkoaineisto. Python. <https://docs.python.org/2/library/socket.html>. Luettu 11.4.2020