Ari Pikkarainen

Vedia CO2 -laskin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

29.6.2022

Tiivistelmä

Tekijä:	Ari Pikkarainen
Otsikko:	Vedia CO2 -laskin
Sivumäärä:	20 sivua
Aika:	29.6.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine:	Automaatiotekniikka
Ohjaajat:	Projektijohtaja Lasse Nykänen
	Yliopettaja Erkki Räsänen

Insinöörityössä selitetään kaikki tarvittava tieto, jota tarvitaan karttasovellukseen missä, lasketaan matkamuotojen päästöjä. Kaikki palvelut ovat joko ilmaisia tai niistä on ilmainen kokeilu, joka tukee kaikkia maksullisen version ominaisuuksia.
Työn luettua saa hyvän käsitteen karttasovelluksen rakenteesta ja siihen käytetyistä moduuleista. Löysimme useita rajapintoja, joilla pystyttiin lisäämään toiminnallisuutta. Teimme tutkimuksen koneoppimisen hyödyntämisestä laskimen tarkkuuden parantamisesta ja palvelun rakentamisen strategiasta.

Avainsanat: Päästöt, Kartta, Reititys

Abstract

Author:	Ari Pikkarainen
Title:	Vedia CO2 -calculator
Number of Pages:	20 pages
Date:	29 June 2022
Degree:	Bachelor of Engineering
Degree Programme:	Electrical and Automation Engineering
Professional Major:	Automation Engineering
Supervisors:	Lasse Nykänen, Project Director
	Erkki Räsänen Principal Lecturer

This document explains all necessary tools and knowledge, which is required to create a interactive map application where you calculate emissions based on different modes of transportation. Every service used in this project are either free or have free trials that support every feature that a premium service has to offer. After reading this thesis, you'll be well acquainted with the map applications structure and modules in its inner workings. We found and gathered plenty of useful APIs which we we're able to add to the app to improve its functionality. We documented the research into machine learning and how it could be beneficial to the calculators estimation accuracy and provided services strategy.

Keywords: Emission, Map, Routing

Sisällys

Lyhenteet

ORS: *OpenRouteServices*. Rajapinta, jota käytettiin reittien määrittämiseen.

POC: *Proof Of Concept*. Työllä pyritään esittämään, että se on tehtävissä.

CVM: *Clean Vehicle Mobile.* Vedian oma mobiilisovellus, jolla seurataan ajoneuvojen päästöjä.

CVW: Clean Vehicles Wizard. Vedian oma sovellus ajoneuvokaluston päästöjen ja energian kulutuksen seurantaan ja hallintaan.

1 Johdanto

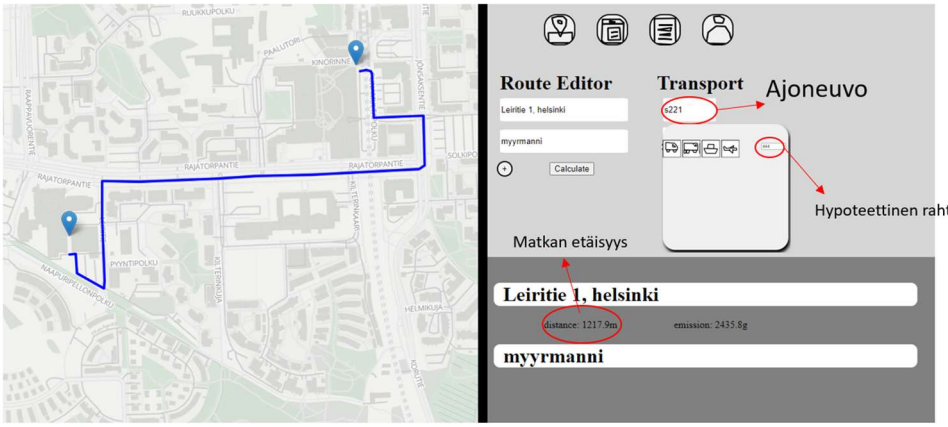
Vedia on rakentanut oman CVW-sovelluksen ja sitä tukevan CVM-mobiilisovelluksen, jolla voidaan seurata ajoneuvojen päästöjä. [1.] Nyt Vedia on ryhtynyt jatkokehittämään palveluaan myös kuljetusketjun päästöjen seurantaan. Tätä varten Vedia integroi VTT:n ylläpitämän Lipasto-päästötietokannan (Lipasto) omaan järjestelmään. Tässä työssä tästä integraatiosta luodaan POC-päästölaskennan käyttöliittymä. VTT on ilmoittanut lopettavansa Lipaston ylläpidon.

Lipasto ottaa yksikköpäästötiedot pois käytöstä 1.8.2022 vanhenemisen vuoksi. [2.] Vedia pyrkii tekemään uuden tietokannan, jonka he jatkokehittävät ja laajentavat omalla CVM-sovelluksellaan sekä muilla päästötietokannoilla.

Laskurilla voidaan arvioida pakettikuljetusten ja yksittäisten matkustajien päästöjä merkitsemällä laskuriin heidän tulevat kohteet ja lähtöpaikat sekä matkustustapa. Siitä laskin laskee reitin ja tämän ajoneuvon tuottamat päästöt.

2 Laskin

Laskimen oli täytettävä monet vaatimukset voidakseen laskea CO2-päästöjä. Laskimen on kyettävä muuttamaan katuosoitteet koordinaateiksi, joita sitten ajetaan reititysrajapinnan lävitse, jotta saadaan matkan etäisyys ja kesto. Näitä tietoja käytettiin kuljetun matkan laskemiseen, joka kerrotaan ajoneuvon kertoimella ja saadaan matkan päästöt. Nämä tiedot ovat peräisin Lipaston verkkosivuilta ladatusta yksikköpäästöjen Excelistä. Se korvataan Vedian omilla tiedoilla, kun ne ovat valmiita.

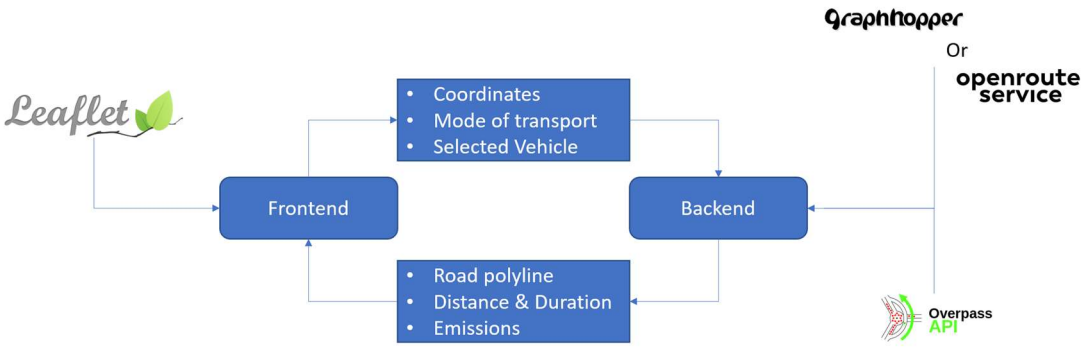


$e_x = e_a + ((e_b - e_a) / (m_b - m_a)) \times (m_x - m_a)$

e_x = Päästö ajoneuvokilometriä kohden autolla, jonka kokonaismassa on x [g/km]
e_b = Päästö autolla, jonka kokonaismassa on b [g/km]
e_a = Päästö autolla, jonka kokonaismassa on a [g/km]
m_x = Auton x kokonaismassa [t]
m_b = Auton b kokonaismassa [t]
m_a = Auton a kokonaismassa [t]
Päästöt [g] = e_x x km

Kuva 1. Päästöjen laskuesimerkki. Auton massa lasketaan ajoneuvomallin perusteella, johon summataan hypoteettinen rahtipaino. [3.]

Reittirajapinnan antama lopputulos täytyi esittää laskimen käyttöliittymässä piirtämällä sen sinisellä viivalla maailman kartassa. Tämä oli olennaista, koska reittiä pystyi muokkaamaan painamalla karttaan uusia pisteitä, joiden läpi reititysrajapinnan täytyi laskea uudet reitit.



Kuva 2. Laskinsovelluksen arkkitehtuuri.

2.1 Backend

Backend-toteutus tehtiin Pythonissa Flask-moduulilla, jolla haettiin käyttöliittymälle tarvittavat tiedot eri rajapinnoista. Backend välitti myös käyttäjän tallentamat tiedot käyttöliittymään.

Aina kun käyttöliittymä halusi vastauksia, sitä ei laskettu frontendissä, vaan sen tiedot lähetettiin backendiin käsiteltäväksi, tämä lähetti vastauksen ja reitin takaisin frontendille.

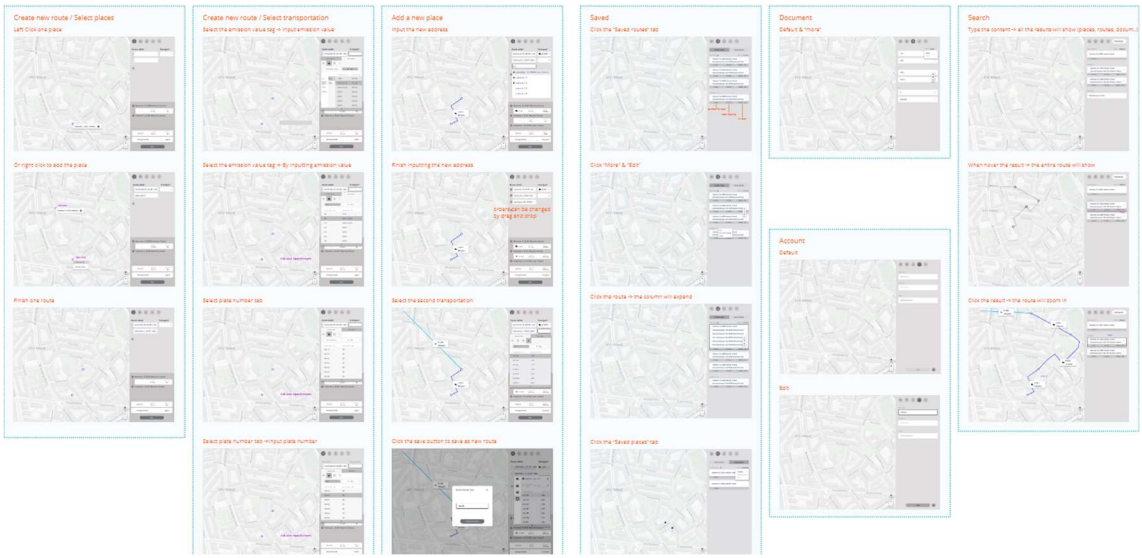
Backend vastasi tietojen käsittelystä ja tallentamisesta. Jos frontendissä ollut reitti ja lasketut päästöt haluttiin tallentaa, niin se lähetettiin backendille, joka tallensi sen tietokantaan käyttäjän avaimella. Toisessa istunnossa käyttäjä voi noutaa saman reitin ja lasketut arvot taustajärjestelmästä, jotta hän voi jatkaa sitä ja tallentaa toisen muunnelman.

2.2 Frontend

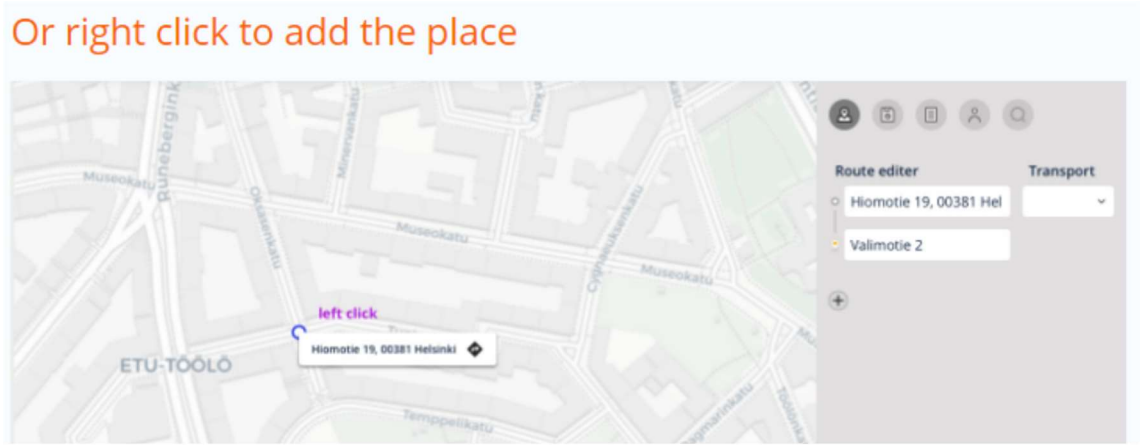
Frontendille jätettiin tehtäväksi ottaa talteen käyttäjän asettamat arvot. Näille arvoille määritettiin lokero, johon asetetaan kannettu lasti kilogrammoina ja millä ajoneuvolla kulkee minne.

Frontendillä oli myös tämän lisäksi pystyttävä kirjautumaan joksikin käyttäjäksi, joka sitten pyysi tämän käyttäjän tallentamia tietoja, että se voi esittää ne jossain käyttöliittymässä.

Ulkonäön määräsivät asiakkaan laatimat mockup-kuvat, joiden perusteella kaikki funktiot oli myös määritelty.



Kuva 3. Kaikki asiakkaan mockup-kuvat.



Kuva 4. Frontendin toivottu ulkonäkö.

Asiakas halusi kuvien piirteet alustavasti tähän projektiin. Projektin aikana todettiin, että on vielä tarve UX designerille ja suunnitellaan käyttöliittymästä paljon käyttäjäystävällisempi.

3 Rajapinnat

Tässä työssä testattiin erilaisia rajapintoja, joita käytettiin eri tavoin. Reittien luominen oli tehtävä, joka vaati ulkoisen rajapinnan käyttöä, jonka tehtävänä oli laskea lyhin reitti pisteiden A ja B välillä. Sen piti myös osata muuntaa osoite kartan pisteiksi ja tulostaa sen koordinaatit.

Meidän piti selvittää, miten vesiliikenne toimii, joten otimme käyttöön myös tähän tehtävään rajapinnan, jolla voitiin tehdä kyselyitä kaikista maailman laiva-terminaaleista ja lentokentistä. Näiden tietojen avulla voitiin piirtää ja laskea matkaa vesiväyliä pitkin maasta toiseen.

Käyttöliittymän koordinaattorina toimi karttarajapinta, jolla esiteltiin käyttäjäominaisuuksia, kuten painallusta, nimen selvennystä ja reitin piirtämistä kartalle, jolla pääsee pisteestä A paikkaan B nopeimmin ja vähiten päästöjä tuottaen.

3.1 Reititysrajapinnat

Testasimme kahta eri reititysrajapintaa ja selvitimme, kumpi soveltuu paremmin laskimen tarpeiden toteutukseen. Katsoimme näiden dokumentoinnin saatavuutta ja sitä, kuinka helppoa näiden sisäistäminen tähän projektiin olisi.

3.1.1 Openrouteservices

OpenRouteServices tarjoaa käyttöönsä oman python-moduulin, mikä helpottaa sen lisäämistä backendiin. ORS tarjoaa laajat parametrit reittinsä optimointiin siltojen ja matalien tunneleiden välttämiseksi.

ORS ei pysty esittämään vesireittejä. Tämän seikan kiertämiseksi tarvitaan oma moottori, joka lukee kaikki mahdolliset vesiväylät pisteiden A ja B välillä ja valitsee parhaan mahdollisen reitin ja ohjaa ORS:n saapuvalle reitille, joka päättyy moottorin antamaan vesiväylän lähtölaituriin ja tekee toisen kyselyn ORS:lle, joka jatkuu määränpäähän johtavan vesiväylän päätylaiturista. [4.]

3.1.2 Graphhopper

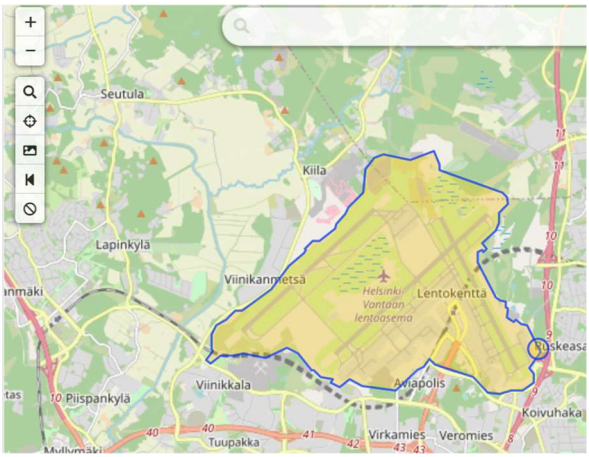
Graphhopper-rajapinta on hyvin dokumentoitu ja tälle löytyi python-moduuli, mutta se ei ole ajan tasalla. Tämän käyttö toimi suoraan sen POST/GET-pyynnöillä. Näille voi määrittää omat funktiot, jotka saa asiat hoidettua yhtä hyvin kuin ORS-moduuli.

Graphhopperin reittien optimointi on ORS:iin verrattuna paljon ketterämpi, sillä se voi määrittää oman mallin, jota Graphhopperin reittilaskuri joutuu noudattamaan. Voidaan käskeä sen vain ottamaan vesireittejä, jolloin matkan pituus tulee olemaan pidempi, mutta päästöt eivät ole mitään. Mikäli tämä ei ole mieleen, Graphhopper tarjoaa myös käyttöön omia profiilejaan, jotka toteuttavat peruskäsitteitä, joita ovat esimerkiksi rekkakuljetus, pyöräily ja kävely jne. Nämä sitten valikoivasti kulkevat itselleen sopivia reittejä. [5.]

3.2 Overpass

Overpass on OpenStreetMappiin tehty ilmainen tietokanta, josta voi kysyä kaikki halutut tiedot korallialueista vesilähteisiin kaupungeissa. Tänne voi lähettää kyselyn käyttäen näiden omaa tag-systeemiä, johon voi laittaa maailman rajaksi tai yksittäisen valtion ja kysyä sen sisältä kaikkia asioita, mitkä putoavat tagin ”Route” alle. Tämä tulostaa silloin kaikki kulkureitit ja ajotiet. Vastaavasti voi kysyä ”Amenity”, niin se kertoo meille kaikki R-kioskit ja baarit. Tämä on helppokäyttönen tehokas rajapinta.

```
  node["aeroway"="aerodrome"]({{bbox}});
  way["aeroway"="aerodrome"]({{bbox}});
(._;>;);
out;
```



Kuva 5. Esimerkkikysely. https://overpass-turbo.eu/.

Sieltä pystyttiin kysymään kaikkien maailman lentokenttien sijainnit ja näiden nimet, jotta voitiin näiden kautta ohjata lentokonerahteja halutuin parametrein. Se komplementoi reititysrajapintaa siten, että lentokoneella lentäminen maan halki oli vain viiva kahden pisteen välillä. Nyt pystyimme tämän avulla rakentamaan

oman moottorin siltä varalta, että reitin varressa löytyy lähin lentokenttä, josta voi ottaa lennon määränpään lähimpään lentokenttään.

Vastaavasti vesireitit toteutettiin. Merkattiin haluavamme kulkea maasta toiseen, joten moottori etsi lähimmän satamaterminaalin, josta lähtee matka määränpäähän. Sitten tehdään kaksi kyselyä reittirajapinnalle: toinen vie terminaalille ja toinen alkaa terminaalilta määränpäähän.

Overpass noudattaa geoJSON-standardia kohteiden tulostamisen nähden. Standardi on struktuuri, joka on tuettu kaikissa muissa kartta-arkistoissa. Näiden kesken voi sitten vaihdella tarvittaessa ilman, että tulee yhteensopivuushäiriöitä. [6.]

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Esimerkkikoodi 1. geoJSON-vastaustruktuuri.

3.3 Leaflet

Leaflet on ilmainen interaktiivinen karttapalvelu, jonka voi vapaasti lisätä omiin sovelluksiin. Leaflet tuo sovellukseen toimivan kartan ja paljon ominaisuuksia sen mukana. Kartalle voi lisätä eri merkintöjä ja tästä voidaan kerätä tietoa kuten koordinaatteja klikkaamalla.

Tässä työssä leaflet toimii yhtenä kulmakivenä, jonka avulla voitiin todentaa annettujen tietojen olevan ne oikeat. Voitiin visualisoida se data ja lasketut arvot piirtämällä karttaan reitti ja pisteet, jota kautta ajoneuvot kulkevat.

Leafletin lisääminen sovellukseen on tehty todella helpoksi. Täytyy vain noudattaa paria askelta, se toimii. [7.]

4 Rajaukset ja tekninen toteutus

Tässä osassa koitan avata backendin ja frontendin toiminnan logiikkaa. Löysin muutaman keinon, jotka helpottivat työhön tehtäviä äkillisiä muutoksia.

4.1 Frontend

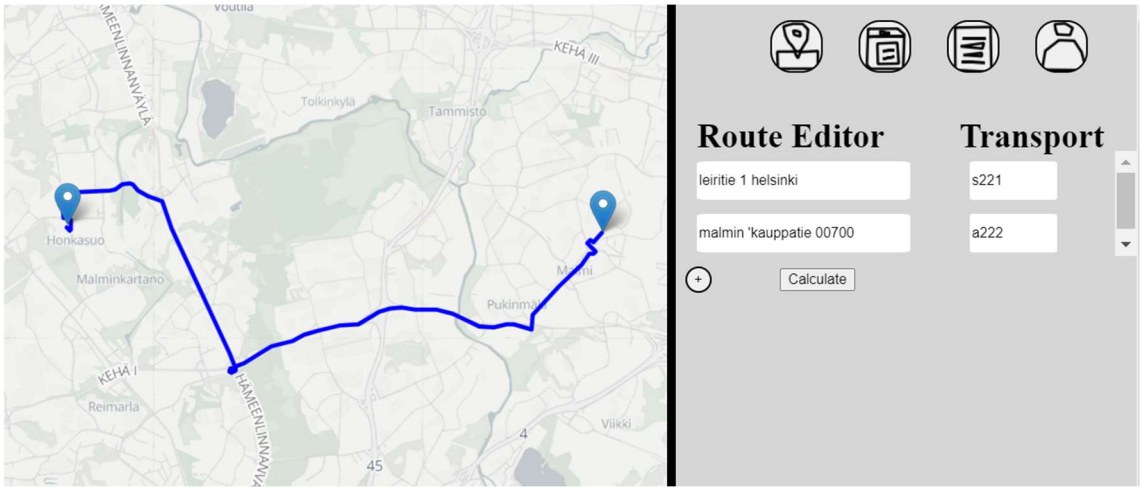
Leafletin lisättyä frontendiin saadaan käyttöön karttaobjektin, johon tehdään layereitä ja johon lisätään kaikki reitit ja pisteet. Reitin tallentamisen voi toteuttaa yhdistämällä nämä layerit yhdeksi ja lähettämällä sen backendille säilöttäväksi. Sen voi sitten noutaa tarvittaessa tai käyttäjän pyydöstä.

Markereiden poisto oli helpointa luomalla näille oma layer näiden nimissä, joka sitten pystyttiin sitomaan siihen elementtiin, mikä edusti tätä sijaintia. Täten kaadettiin kaksi lintua yhdellä kivellä.

```
//Markerin lisääminen
let marker=L.marker(loc);
map.addLayer(marker);
parentNode.marker=marker;

//Markerin poisto
map.removeLayer(parentNode.marker);
lista.removeChild(parentNode);
```

Esimerkkikoodi 2. Markereiden lisääminen ja poisto. Vaikka poistetaan marker-muuttuja, niin map layer marker silti jatkaa olemistaan. Se täytyy poistaa myös ennen muuttujan poistamista.



Kuva 6. Markerit ovat tien alkupäässä ja loppupäässä olevat heittomerkit.

4.2 Backend

Tässä työssä backend teki raskaat siirrot. Frontendin koodi pysyi siistinä, sillä sen tehtävä ei ollut muuta kuin välittää se tieto, mitä käyttäjä on asettanut taakse backendille. Siellä laskettiin reitit, päästöt, matkat ja lähimmät asemat jne. Python oli juuri täydellinen tähän tehtävään, kun listojen käsittely on sen kotikenttä.

Tämä ratkaisu oli hyvä, sillä kun tarvitsi vaihtaa ajoneuvoa kesken matkan, niin täytyi muodostaa uusi kysely. Huomattiin nopeasti, että meillä on useita listoja, joita pitää laskea yhteen, että voi saada koko kuvan selville. Python hoiti homman upeasti, koska matkojen koordinaattien summaaminen tapahtuu vain yhdellä rivillä koodia. Tämä säästää huikeasti aikaa.

Se myös helpottaa tehtävässä, kun tekee overpass-pynnön ja tallentaa kaiken olennaisen etukäteen tietokantaan.

5 Laskenta

Työssä jouduttiin käyttää joitain menetelmiä, jotta saatiin ongelmia ratkottua. Polylinestringillä avattiin kompressoidut koordinaatit apin vastauksesta. As the crow flies -menetelmä näytti meille näiden koordinaattien väliset etäisyydet, jos

näitä ei ollut reitissä valmiiksi ja tämän avulla saatiin laskettua sen reitin pääs-töt.

5.1 Polylinestring

Tämä nerokas keksintö säästää aikaa. Sen sijaan, että rajapinta antaa miljoona longitude latitude -pistettä, saadaan sen sijaan pitkä stringi. Tämä on helppo muuttaa sitten backendissä koordinaateiksi ja laskea, joka lähetetään fronten-dille esitettäväksi. Esimerkkikoodi 3 pythonfunktio tekee juuri tämän, eli muuttaa polylinestringin koordinaattiin arrayksi.

```
def decode_polyline(polyline, is3d=False):
    points = []
    index = lat = lng = z = 0

    while index < len(polyline):
        result = 1
        shift = 0
        while True:
            b = ord(polyline[index]) - 63 - 1
            index += 1
            result += b << shift
            shift += 5
            if b < 0x1F:
                break
        lat += (~result >> 1) if (result & 1) != 0 else (result >> 1)

        result = 1
        shift = 0
        while True:
            b = ord(polyline[index]) - 63 - 1
            index += 1
            result += b << shift
            shift += 5
            if b < 0x1F:
                break
        lng += ~(result >> 1) if (result & 1) != 0 else (result >> 1)

        if is3d:
            result = 1
            shift = 0
            while True:
                b = ord(polyline[index]) - 63 - 1
                index += 1
                result += b << shift
                shift += 5
                if b < 0x1F:
                    break
            if (result & 1) != 0:
                z += ~(result >> 1)
            else:
                z += result >> 1

            points.append(
                [
                    round(lng * 1e-5, 6),
                    round(lat * 1e-5, 6),
                    round(z * 1e-2, 1),
                ]
            )

        else:
            points.append([round(lng * 1e-5, 6), round(lat * 1e-5, 6)])

    geojson = {u"type": u"LineString", u"coordinates": points}

    return geojson
```

Esimerkkikoodi 3. Polylinestring decoder. Rajapinnat palauttavat reitin polylinestringinä, jotta säästävät aikaa sitä välittäessä. Eli koordinaatit on kompressoitu. [8.]

5.2 As the crow flies

Reittien etäisyydet on aina saatavilla rajapinnoille tehtyjen kyselyjen seassa, mutta vesireitit ja lentokonematkat on tehty omalla moottorilla. Täten tarvitaan näiden etäisyyksien laskennassa As The Crow Flies -menetelmää. Tämä mittaa pisteiden A ja B välisen etäisyyden näiden longitude- ja latitude-arvoilla.

```
Import math
matka=0
r=6371
for i in range(len(pisteet)-1):
    lat1=pisteet[i][0]
    lat2=pisteet[i+1][0]
    lng1=pisteet[i][1]
    lng2=pisteet[i+1][1]

    dlat=math.radians(lat2-lat1)
    dlng=math.radians(lng2-lng1)
    a=math.sin(dlat/2)**2+math.cos(math.radi-
ans(lat1))*math.cos(math.radians(lat2))*math.sin(dlng/2)**2
    c=2*math.atan2(math.sqrt(a),math.sqrt(1-a))
    matka+=r*c
```

Esimerkkikoodi 4. As the crow flies. Tällä koodinpätkällä voi mitata polyline-pisteiden pituuden. [9.]

5.3 Päästöt

Päästöjen laskemiseen oli käytetty lipastontietokantaa, josta ladattiin kaikki henkilöajoneuvojen, lentokoneiden ja rahtialuksien kilometripäästöt. Tämä sitten kerrottiin reitin pituudella, jotta saatiin melko hyvä arvio matkustusmuotoon nähden.

Matkassa voi käyttää useata eri kulkumuotoa, joten lopuksi näiden matkojen lasketut päästöt summataan.

Laskuissa voi myös käyttää lipaston yhtälöä, johon lasketaan ajoneuvon paino ilman lastia ja täysi lasti ja tämän ajoneuvokerroin, niin voitiin laskea päästöjä tietyn kokoisille tavaralastille. [3.]

5.4 Koneoppiminen

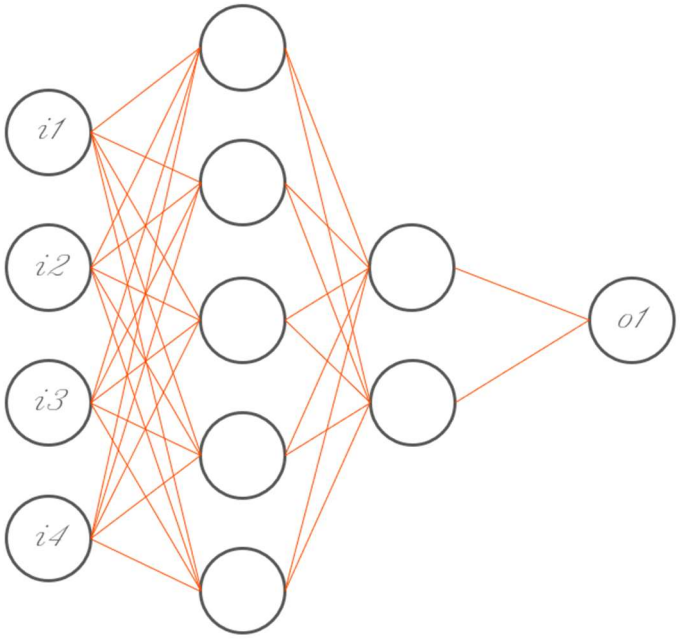
Ajoneuvojen tuottamia päästöjä on hankala arvioida, koska operaatio ei ole missään nimessä lineaarinen. Ajoneuvon päästöihin on monta tekijää, joten koneoppiminen voi olla seuraava pykälä projektin edetessä. Koneoppimisella voidaan luoda digitaalinen kaksonen tai regressiomalli. Jokaisesta auton malleista demo ajaa näillä reittejä ja täten saa paremman arvon päästöistä.

Mallien luominen on kevyt operaatio ensimmäisen toimivan mallin valmistumisen jälkeen. Tässä kouluttamisessa voi käyttää hyödyksi transfer learning -menetelmää, jossa uudelleenkoulutetaan valmiiksi koulutettu malli suorittamaan tehtävä, joka vastaa tämän alkuperäistä tehtävää. [§10.] Täten voidaan vain työntää uusi data mallille opittavaksi ja tästä tulee toimiva kaksonen toiselle automallille lyhyessä ajassa. Operaation voi toistaa jokaiselle erilliselle automallille ja näin kouluttaa usea malli jokaista vastaavaa autoa varten.

5.4.1 Mallin laatiminen

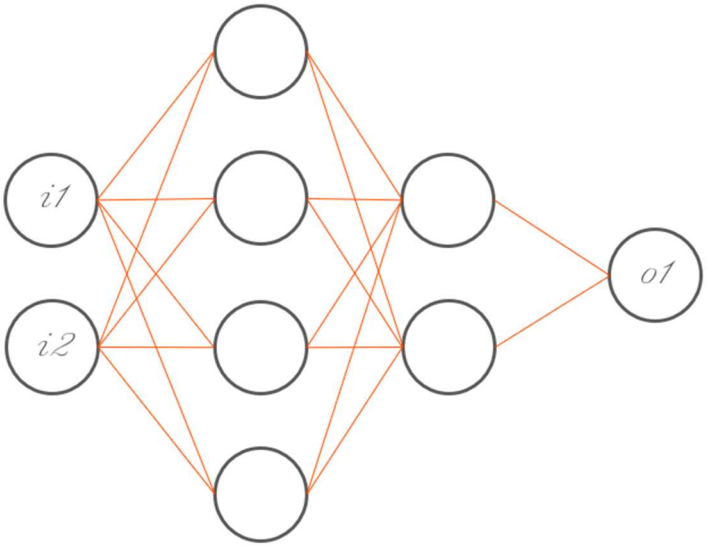
Mallia luodessa huomioidaan transfer learning -edut. Meidän täytyy luoda malli jokaiselle erilliselle matkustusmuodolle. Ajamisella olisi yksi malli, laivalle toinen malli ja lentokoneille kolmas, koska näillä on eri tensor-määrä ensimmäisellä tasolla mallia. Näiden tensoreiden määrä määräytyy mallin aistien mukaan. Auton malli tensoreiden määrä on neljä, koska sen täytyy arvioida päästöt sen ajaman

tien tyyppi, tien nopeusrajoite, tien pituuden ja auton kantaman lastin painon perusteella.



Kuva 7. Teoreettinen auton koneoppimismalli.

Laivan malli olisi taas kaksi pienempi, koska tähän liittyen tiedämme vain laivan lastin ja arvion siitä, kuinka kauan matka kestää ja myös pituuden, mutta se on vähemmän arvokas meille.

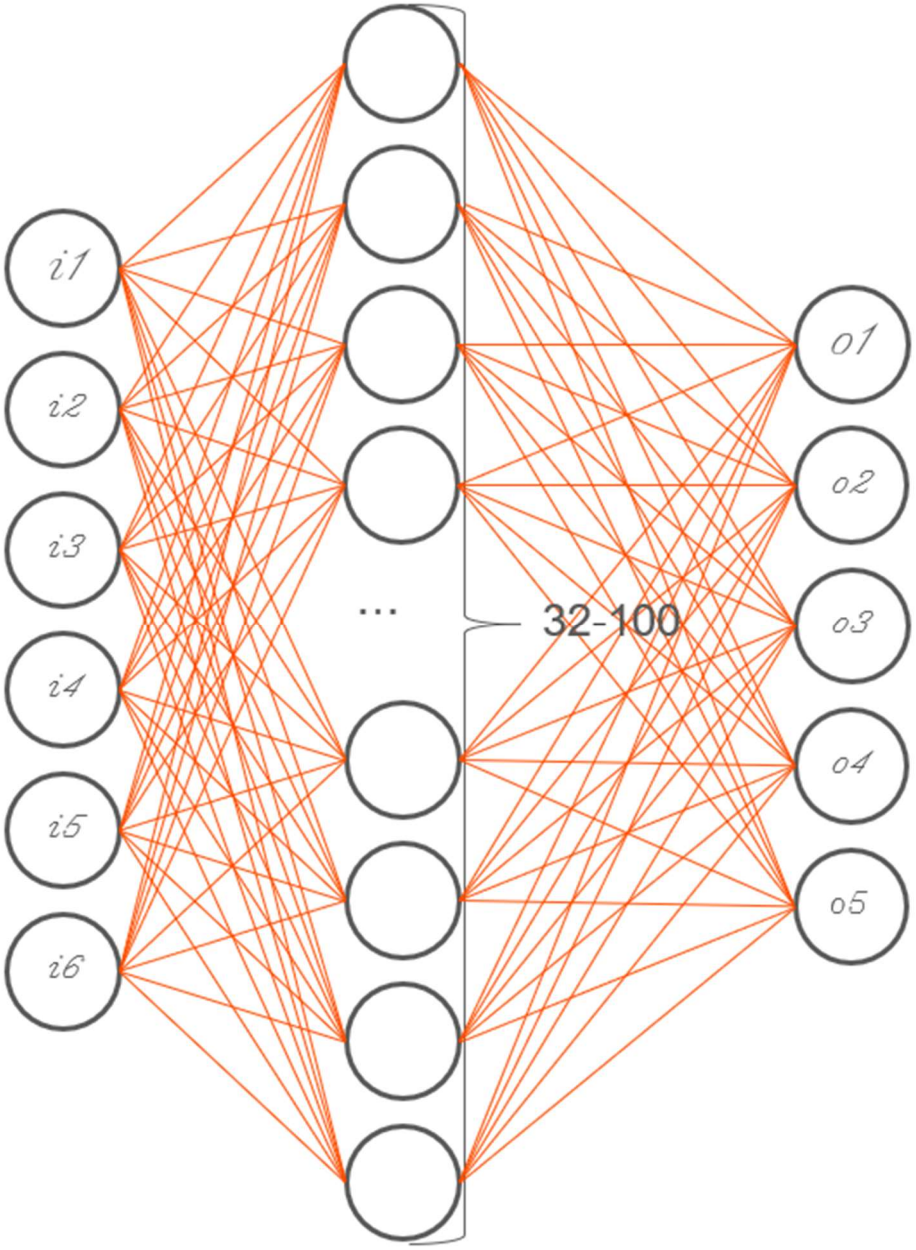


Kuva 8. Laivan teoreettinen koneoppimismalli

Mallien yhteenveto olisi ensimmäinen taso, joka koostuu x määrästä dense-neuroneista tai kutsumanimellä perceptroneista, joiden aktivointi jätetään täysin lineaariseksi, sillä nämä ovat regressiomalleja. Tämä tarkoittaa, että näiden ulostuloon ei lisätä lainkaan epälineaarisuutta, jota tuovat yleisimmät aktivointi-funktiot kuten ”relu” ja ”sigmoid”. Näitä yleensä käytetään klassifikaatiooperaati-oissa. Sellaisia malleja kutsutaan Deep Learning -malleiksi. Mallin toinen taso on hidden layer, jonka yhteydessä kaikki komputointi tapahtuu. Tämä koostuu samoista dense-neuroneista vaan näiden määrä on lisätty, joka antaa mallin luoda oikein ison differentiaalikaavan ja täten saavuttaa pienin virhemarginaali simuloidessaan auto mallia tai laivaa.

5.4.2 Digitaalinen kaksonen

Koneoppimisen avulla voidaan rakentaa autoista digitaalinen kaksonen, jonka avulla simuloidaan reitti alusta loppuun. Tällöin voidaan saada mallia tarkempia arvoja reiteistä luomalla algoritmi, joka ajaa auton halutun reitin halki ja emuloi päästöjä. Digitaalista kaksosta ajettaisiin siten, että algoritmi aloittaa tien alusta ja kytkee kaasun päälle. Ajan myötä algoritmi nostaa vaiheen tasoa, digitaalinen kaksonen nostaa nopeutta. Kun joka sekunnin välein nopeus summataan ja saadaan summan tulokseksi tien pituus. Algoritmi täten jarruttaa ja hidastaa au-toa tietyn tahtiseksi, jotta voidaan kuvitella sen tekevän kääntyminen oikeassa maailmassa. Kun digitaalinen kaksonen on saapunut päämäärään, voidaan ot-taa auton tuottamien päästöjen summa ja sanoa tätä päästöiksi koko matkalta. Tämä on hieman raskaampi operaatio laskennan tähden.



Kuva 9. Digitaalisen kaksosen teoreettinen koneoppimismalli.

5.4.3 Datan kerääminen

Mallien kouluttaminen vaatii dataa ja tätä voi kerätä yksitellen, kun jollain autolla suoritetaan jotain esimerkkireittiä. Autolla ajetaan mahdollisimman lähellä nopeusrajoitetta ja silloin myös kerätään talteen sen teettämät päästöt. Ylös kirjataan, miten pitkä matka ajettu, mikä oli tiellä sallittu ajonopeus ja mitä tyyppiä tie oli. Näillä tiedoilla koitetaan simuloida auton olosuhteita ja saada koko päästöt

yhdelle tielle. Nämä sitten nimitetään yhdeksi datasetin lokeroksi ja sama toistetaan kaikille esimerkkireitin teille.

Digitaalisen kaksosen kouluttaminen on reaaliaikaista, datan kerääminen on samanlaista. Otetaan esimerkkiauto, jolla ajetaan joku reitti. Autosta kerätään talteen dataa joka sekunti ja ylös kirjataan seuraavat: onko kaasu pohjassa, painetaanko jarrua, mikä on auton tämänhetkinen nopeus, entä moottorin kierrosluku, paljonko tankissa on bensaa, mikä vaihe on päällä. Nämä ovat digitaalisen kaksosen sisääntulot. Ulostuloista kerätään seuraavat: mikä kierrosluku, mikä nopeus, onko paljon bensaa, paljonko syntyy päästöjä ja miten lämmin moottori on. Tähän voi tietenkin lisätä ominaisuuksia, jotka voisivat tarkentaa päästöjen lukua. Mallille koulutetaan tämän ulostulon i+1, jotta se voi ennustaa nykyisillä tiedoilla outputin seuraavan tiedon ja täten takaisinkytkennällä simuloida liikkuvaa autoa.

Datan kerättyä se täytyy jäsentää ja muuntaa muotoon, koneoppimismalli voi ymmärtää, joka on matriisi. Täytyy luoda x matriisi ja y matriisi. X-matriisi sisältää input datan ja y-matriisi sisältää saman input-datan, mutta se on offsetattu i+1. Täten koneoppiminen tapahtuu ja malli oppii tekemään ennustuksia, että miten tilanne muuttuu, jos kaasu on pois päältä tai vaihtaa vaihetta.

6 Ehdotukset

Tässä osassa esitän omat näkemykseni projektin haitoista, missä voisi olla parantamisen aihetta vapaamuotoisesti.

6.1 Kartta

Tämä pelasi jotain roolia sovelluksessa, mutta kartan ei välttämättä tarvitse toimia muulla tavalla kuin datan visualisointina. Jos jotenkin olisi saatavilla DLS tai postin kuljetusreittejä ja varastoja, niin karttaan voisi lisätä layereitä, mistä nä-

kee eri kuljettajien suosimat reitit. Tällöin käyttäjä voisi vaikuttaa reitin generoinnissa asettamalla tietyn lähettäjän parametriksi. Tämä ei silloin rikkoisi koko reitin struktuuria, kun mennään vaihtamaan päätesijaintia.

6.2 Valikko

Valikko toimii ihan hyvin. Ainoa haaste on, kun vaihtaa määränpäitä, niin ajoneuvojen oletetaan vaihtuvan myös. Esim. Helsingistä Tallinnaan menee laiva, mutta jos vaihtaa määränpään Helsingistä Vantaan lentokenttään, niin laiva ei kulje tätä reittiä, joten meillä on haaste tähän nähden. En itse keksinyt ratkaisua. Melkein ainut ratkaisu tähän olisi ottaa käyttäjä pois valitsemasta näitä parametrejä. Kun ne tekevät muutoksen reittiin suunnittelun jälkeen, se täytyy melkein aloittaa alusta.

6.3 Lisäys

Reittiä suunnitellessa käyttäjälle voisi olla hyödyllistä nähdä, jonkinlaisia eri kuljettajien ja eri yritysten käyttämiä reittejä ja näiden avulla luoda se reitti. Osoitteen etsiminen kartalta voisi olla vain piste mitä sitten vertaillaan kuinka sen voi lisätä siihen reittiin eri kuljettajien määrittämien reittien avulla. Ei voi periaatteessa valita yhtään muuta ajoneuvoa tai sille rahdille lastia.

7 Yhteenveto

Projektissa saatiin kaikki asiakkaan laatimien mockup-kuvien haluamat piirteet toimimaan. Löydettiin monta eri keinoa, kuinka tätä ongelmaa voi ratkoa ja miten asiakas voi jatkaa tämän projektin edistämistä. Tässä työssä huomattiin, että mockup-kuvien haluama ulkonäkö ja toimintaperiaate eivät olleet tähän tarkoitukseen ja käyttäjäystävällisyyteen sopivia.

Pääasiassa projekti toimi tutkimuksena, missä katsottiin, työkaluilla voisi mahdollistaa tämänlaisen käyttöliittymän rakentamisen. Myös tässä projektissa syntyneen koodin pätkiä voi käyttää hyödyksi katsomalla, miten minä olin päättänyt ratkaista joitain käyttöliittymän ongelmia.

Lähteet

1 Vedia CVM, https://www.vedia.fi/fi/cvm-clean-vehicles-mobile-2/. Vedian oma päästösovellus. Luettu 29.6.2022.

2 Lipasto vtt, http://lipasto.vtt.fi/yksikkopaastot/. Lipasto poistaa tiedot käytöstä. Luettu 29.6.2022.

3 Lipasto vtt, http://lipasto.vtt.fi/yksikkopaastot/guide_tie.htm. Lipaston laskentakaava. Luettu 29.6.2022.

4 Openrouteservice, https://openrouteservice.org/. Dokumentaatio. Luettu 29.6.2022.

5 Graphhopper, https://www.graphhopper.com/. Dokumentaatio. Luettu 29.6.2022.

6 OpenStreetMap, https://wiki.openstreetmap.org/wiki/Main_Page. Overpass tietokanta ohjeita. Luettu 29.6.2022.

7 Leaflet, https://leafletjs.com/. Dokumentaatio. Luettu 29.6.2022.

8 Geojson, https://geojson.org/. Sisältöä miten overpassistä haetaan tietoa. Luettu 29.6.2022.

9 As The Crow Flies, https://gist.github.com/yevrah/0059ce2186dfd9d32d34 Menetelmä miten etäisyys lasketaan A ja B välillä. Luettu 29.6.2022.

10 Transfer Learning, https://machinelearningmastery.com/transfer-learning-for-deep-learning/. ML-koulutusstrategia. Luettu 5.9.2022.