Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Pajk Kosec

Optimizacija voznikove poti glede na način vožnje

diplomsko delo
na univerzitetnem študiju

Mentor: dr. Dejan Lavbič

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01952 / 2013
Datum: 3.9.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK PAJK KOSEC**

Naslov: **OPTIMIZACIJA VOZNIKOVE POTI GLEDE NA NAČIN VOŽNJE**
OPTIMIZING THE DRIVER ROUTE BASED ON DRIVING STYLE

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Obstoječi sistemi za navigacijo, ki pomagajo uporabnikom pri učinkovitem planiranju poti, so pri svojem delovanju večinoma omejeni na glavni vir podatkov - lokacijo uporabnika in kartografijo izbranega ponudnika. Uporaba drugih podatkov, ki so nam tudi na voljo pa je pogosto le izziv. V okviru diplomske naloge je potrebno pripraviti prototip sistema, ki pri optimizaciji voznikove poti upošteva dodatne vire podatkov (senzorski podatki avtomobila, vremenska napoved, stanje prometa na cesti) in omogoča planiranje varčne poti. Sistem naj omogoča odkrivanje pravil o vožnji uporabnika, na podlagi katerih zgradi profil voznika in mu tako svetuje učinkovitejšo ter varčno vožnjo glede na trenutne razmere. Sistem naj uporabniku ne daje navodil kako naj vozi, ampak se naj njegova pot optimizira glede na način vožnje. Prototip naj bo pripravljen v obliki mobilne aplikacije, ki zbira določene senzorske podatke in je nameščena na uporabnikovi mobilni napravi ter zalednega strežniškega dela, ki skrbi za hrambo podatkov, načrtovanje poti in zbiranje podatkov iz drugih virov.

Mentor:
doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Pajk Kosec, z vpisno številko **63060256**, sem avtor diplomskega dela z naslovom:

Optimizacija voznikove poti glede na način vožnje.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. januarja 2011 Podpis avtorja:

Iskreno se zahvaljujem mentorju doc. dr. Dejanu Lavbiču za vso strokovno pomoč in nasvete pri izdelavi tega diplomskega dela.
Posebna zahvala gre tudi dr. Nini Ledinek za lektorski pregled dela.
Za podporo in potrpežljivost pri študiju se zahvaljujem tudi svojim staršem ter punci Petri.

Kazalo

Seznam uporabljenih kratic

OBD (*ang.: On-Board Diagnostics*) - zmožnost vozila, da samo opravi diagnostiko in poroča o napakah

GPS (*ang.: Global Positioning System*) - sistem za določanje lokacije

EPA (*ang.: Environmental Protection Agency*) - agencija za varstvo okolja Združenih držav Amerike

CARB (*ang.: California Air Resources Board*) - agencija za nadzor kvalitete zraka zvezne države Kalifornija

EOBD (*ang.: European On Board Diagnostics*) - evropska različica sistema OBD

ECU (*ang.: Engine Control Unit*) - v vozilo vgrajena enota za nadzor delovanja motorja

MIL (*ang.: Malfunction Indicator Lamp*) - svetlobni indikator napake na motorju

DLC (*ang.: malfunction indicator lamp*) - priključek v vozilu, ki omogoča priklop na ECU

SAE (*ang.: Society of Automotive Engineers*) - združenje avtomobilističnih inženirjev

CAN (*ang.: Controller Area Network*) - podatkovno vodilo, ki se pogosto uporablja v avtomobilih

PWM (*ang.: Pulse-Width Modulation*) - tehnika za pretvarjanje elektromagnetnega signala

VPW (*ang.: Variable Pulse Width*) - tehnika za pretvarjanje elektromagnetnega signala, podobna PWM

ISO (*ang.: International Organization for Standardization*) - mednarodna organizacija za standardizacijo, po kateri se imenuje skupina standardov

GM (*ang.: General Motors*) - ameriški avtomobilski koncern

KWP (*ang.: KeyWord Protocol*) - komunikacijskih protokol v sistemih OBD

PID (*ang.: Parameter Identifier*) - identifikacijska oznaka ukaza v sistemih OBD

PBF (*ang.: Protocolbuffer Binary Format*) - podatkovna struktura, v kateri so na voljo zemljevidi OpenStreetMap

XML (*ang.: Extensible Markup Language*) - razširljivi označevalni jezik

OSM (*ang.: OpenStreetMap*) - digitalni zemljevid OpenStreetMap

REST (*ang.: REpresentational State Transfer*) - oblika arhitekture za razvoj spletnih storitev

CORBA (*ang.: Common Object Request Broker Architecture*) - standard za komunikacijo na osnovi storitev

RPC (*ang.: Remote Procedure Call*) - klic procedure na oddaljenem računalniku

SOAP (*ang.: Simple Object Access Protocol*) - protokol za prenos podatkov v spletnih storitvah

JSON (*ang.: JavaScript Object Notation*) - odprt standard za prenos podatkov v tekstovni obliki

KAZALO

BSON (*ang.: Binary JSON*) - binarno kodirana oblika dokumentov v obliki JSON

SQL (*ang.: Structured Query Language*) - strukturirani poizvedovalni jezik

NoSQL (*ang.: Not only SQL*) - skupno ime za nerelacijske podatkovne baze

UUID (*ang.: Universally Unique Identifier*) - univerzalni enolični identifikator

URI (*ang.: Uniform Resource Identifier*) - enolični identifikator vira

CBA-RG (*ang.: Classification Based on Associations – Rule Generator*) - klasifikacija na podlagi asociacij - generator pravil

CBA-CB (*ang.: Classification Based on Associations – Classifier Builder*) - klasifikacija na podlagi asociacij - generator klasifikatorja

CAR (*ang.: Class Association Rule*) - razredno asociativno pravilo

Kazalo slik

Kazalo izvornih kod

Povzetek

Pametna uporaba energije je eno izmed pomembnejših področij. To velja tudi za osebna vozila, ne glede na način pogona. Od vseh različic osebnih vozil pa so najbolj problematična tista s pogonom na fosilna goriva. Zaloge le-teh so zelo omejene, poleg tega pa pri njihovem izgorevanju nastajajo škodljive emisije v obliki plinov in prašnih delcev. Namen diplomskega dela je izdelava sistema za optimizacijo uporabnikove poti glede na njegov način vožnje. Sistem je razdeljen na dva dela, in sicer na mobilno aplikacijo za Android ter strežniški del, ki teče kot spletna storitev na aplikacijskem strežniku Tomcat. Oba sta v diplomskem delu tudi predstavljena. Predstavljen je standard OBD II, preko katerega mobilna naprava pridobiva podatke o porabi goriva, ter izračun porabe iz podatkov, ki jih testno vozilo omogoča. Predstavljeni so tudi trije algoritmi, ki jih sistem uporablja. To so algoritem za prilagajanje lokacije digitalnemu zemljevidu, prilagojeni algoritem A* za iskanje najboljše poti ter algoritem za klasifikacijo na podlagi asociativnih pravil. Mobilna naprava se na strežnik povezuje preko spletne storitve REST, vsebina prenesenih podatkov pa se nanaša na delno agregirane podatke o vožnji v smeri strežnika ter na izračunano najboljšo pot v smeri mobilne naprave. V zaključnem delu so opisani možnosti razširitev in rezultati testiranj.

Ključne besede:

Razredna asociativna pravila, REST, Android, Java, klasifikacija, OBD

Abstract

Smart usage of energy is one of the most important topics. It also includes personal vehicles, regardless of means of propulsion. Most problematic are those that run on fossil fuels. Reserves of fossil fuels are limited, also during combustion of those fuels, harmful emissions are produced in a form of gases and dust particles. The purpose of this diploma thesis is to develop a system for optimizing users path based on his driving habits. System is divided into two parts. First is a mobile application for Android device and second is a server application, which runs as a web service on application server Tomcat. Both parts are presented in this thesis. Presented in this thesis is OBD II standard, which is used to gather fuel consumption data available in a test vehicle, and fuel consumption calculation based on that data. Also presented are three algorithms, which are used by the system. These are map matching algorithm, modified A* algorithm for best path search and classification based on associations algorithm. Mobile device is connected to server via REST web service. Transferred data include partly aggregated data about driving in direction to server and calculated best path in direction of mobile device. In the final section testing and future development is described.

Key words:

Class association rules , REST, Android, Java, classification, OBD

Poglavje 1

Uvod

Število vozil v cestnem prometu se vztrajno povečuje, s tem pa tudi poraba goriva. Posledično to pomeni več emisij, ki nastajajo pri izgorevanju fosilnih goriv, in slabšo kakovost zraka. Zanemarljiva ni niti cena goriva, ki v zadnjem desetletju kaže naraščajoč trend, glede na omejene zaloge fosilnih goriv pa ne moremo pričakovati spremembe. Žal so vozila, ki energijo črpajo izključno iz obnovljivih virov, zelo redka, imajo pa tudi druge težave, zaradi katerih med vozniki niso zelo priljubljena. Reševanje problema porabe goriva je zelo pomembna tema, kar se kaže v mnogih rešitvah, ki so vgrajene v vozila ali pa delujejo kot samostojne aplikacije na mobilnih napravah ter vozniku pomagajo pri optimizaciji vožnje. Poraba goriva pa je tudi neposredno odvisna od izbire poti, po kateri uporabnik potuje do izbranega cilja. Ta odločitev je pomembna tudi s stališča časa potovanja, saj lahko predolga vožnja slabo vpliva na voznikovo počutje. Tehnologija, ki omogoča zbiranje podatkov, na podlagi katerih je mogoče izbiro poti optimizirati, je danes splošno razširjena. Standard OBD II [13] je prisoten že od leta 1996 in je danes najbolj primeren način, preko katerega lahko pridobimo podatke o vozilu, vključno s porabo goriva. GPS je del praktično vsakega pametnega mobilnega telefona in omogoča pridobivanje podatkov o premikanju vozila in vseh podatkih, ki jih iz tega lahko izračunamo. Obstaja tudi veliko število spletnih storitev, preko katerih lahko dobimo podatke o razmerah na neki

lokaciji, vključno s podatki o cestnih razmerah in vremenu. V tem diplomskem delu smo razvlili sistem, ki optimizira izbiro poti. Temelji na mobilni platformi Android in javanski spletni storitvi. Pri tem smo uporabili tehnologije, ki so opisane v poglavju 2. Pogledali si bomo mobilno platformo Android, aplikacijski strežnik Tomcat, digitalni zemljevid OpenStreetMap in pa uporabljene algoritme. Pod to spadajo algoritem za prilagajanje lokacije zemljevidom, algoritem za klasifiikacijo na podlagi asociativnih pravil in pa preiskovalni algoritem A*. V tretjem poglavju je predstavljena implementacija sistema. Zaključili bomo s predstavitvijo rezultatov testiranj natačnosti sistema in predlogi za nadaljnji razvoj.

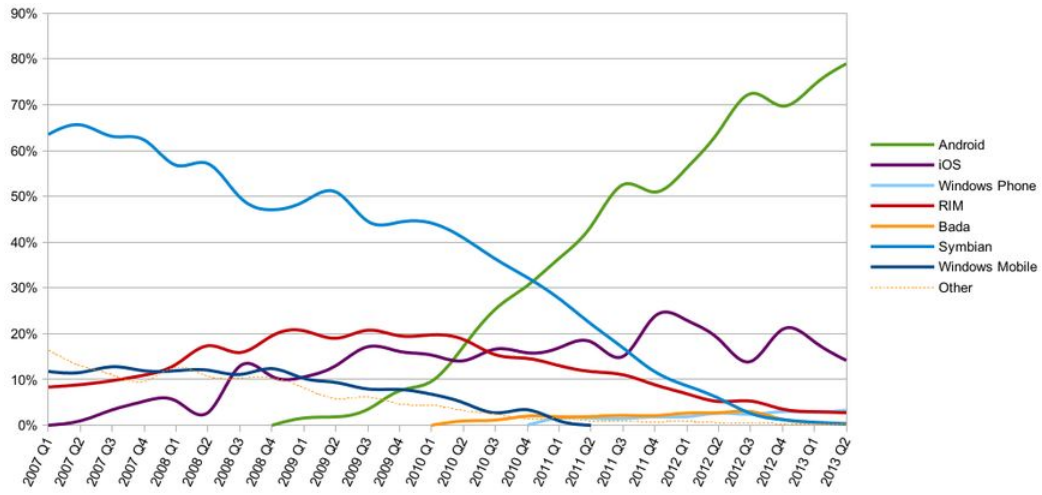
Poglavje 2

Uporabljene tehnologije

2.1 Mobilna platforma Android

Android je odprtokodna mobilna platforma podjetja Google. Namenjena je predvsem mobilnim napravam, kot so pametni telefoni ter tablični računalniki. Razvilo jo je istomensko podjetje, ki ga je leta 2005 prevzel Google, dve leti kasneje pa je bila platforma uradno predstavljena [1]. Leta 2008 je bil predstavljen prvi mobilni telefon, na katerem je tekel operacijski sistem Android 1.0 [2]. Mobilna platforma vključuje operacijski sistem, uporabniške vmesnike, posredniški sistem in osnovne aplikacije, trenutno pa je na trgu različica 4.3 s kodnim imenom Jelly Bean. Danes je Android najbolj priljubljena mobilna platforma, saj je v drugem četrtletju leta 2013 njen tržni delež znašal 79,3 % [3] kar kaže naraščajoč trend v primerjavi s prejšnjimi obdobji.

Android je na voljo pod licenco Apache Licence 2, ki omogoča svobodno uporabo in razvoj platforme. To je zanimivo predvsem za proizvajalce mobilnih naprav, saj lahko Android prilagodijo svojim potrebam, hkrati pa se je ustvaril velik krog domačih razvijalcev, ki izdajajo tako imenovane »custom« verzije sistema [4]. Zelo pomemben del Androida je možnost nalaganja aplikacij, ki omogočajo zelo osebno uporabniško izkušnjo, hkrati pa platforma omogoča preprost razvoj novih aplikacij. Razvijalci lahko aplikacije razvijajo v prilagojeni različici programskega jezika Java, možen pa je



Slika 2.1: Tržni deleži mobilnih platform.

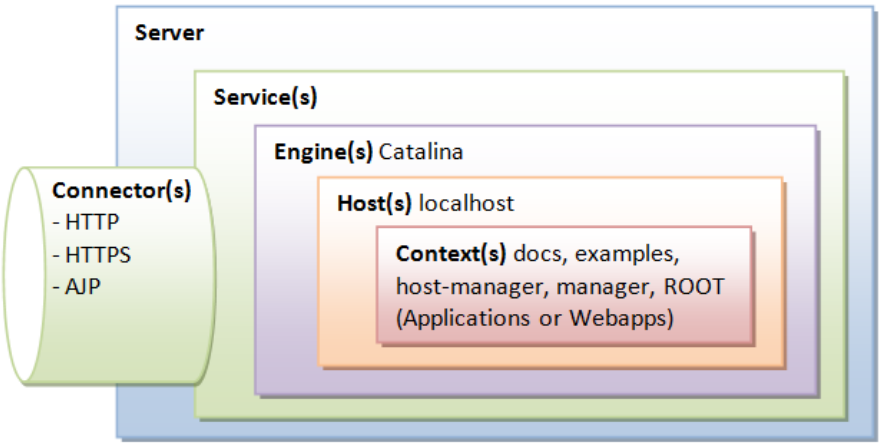
tudi razvoj aplikacij v jezku C/C++ [5].

2.2 Apache Tomcat

Apache Tomcat je odprtokodni aplikacijski strežnik, razvit s strani združenja Apache software foundation. Namenjen je poganjanju spletnih storitev ter spletnih strani, napisanih v programskem jeziku Java [11].

2.2.1 Arhitektura

Strežnik Tomcat je zgrajen na podlagi hierarhične in modularne arhitekture, kot je razvidno s slike 2.2. Najvišji nivo je instanca strežnika (server) in predstavlja celoten pogon za spletne storitve z imenom Catalina. Vsebuje eno ali več storitev (service). Vsaka storitev vsebuje enega ali več prijključkov (connector), ki skrbijo za dejansko upravljanje komunikacije z odjemalcem, kamor spadajo zahteve ter odgovori nanje. Vsi priključki si delijo isto instanco pogona znotraj storitve. Pogon skrbi za obdelavo vseh zahtev, ki jih prejmejo priključki. Nivo nižje se nahaja gostitelj (host), ki definira navidezne gostitelje znotraj vsake instance pogona in je lahko starš eni ali več spletnim



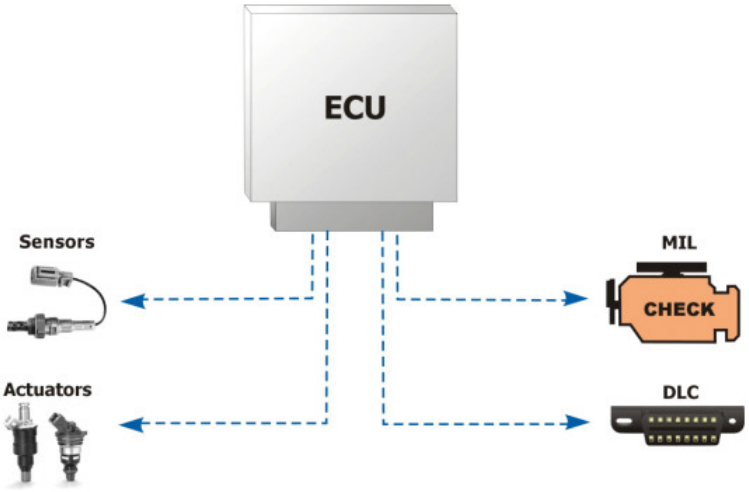
Slika 2.2: Arhitektura aplikacijskega strežnika Tomcat.

aplikacijam, ki so predstavljene kot kontekst (context). Znotraj konteksta se nahaja vsaka individualna spletna aplikacija. Število kontekstov znotraj gostitelja ni omejeno [12].

2.3 OBD II

OBD ali On-Board Diagnostics je izraz za sposobnost vozila, da samo zazna in opozarja uporabnika o stanju motorja in ostalih sistemov. Pod tem imenom poznamo vmesnik za priklop vozila na zunanjo napravo, kot je na primer računalnik [13].

OBD je nastal kot rezultat regulative zvezne države Kalifornija, ki je od proizvajalcev motornih vozil zahtevala, da v svoja vozila vgradijo standarden sistem za nadzor nad izpusti emisij. Regulativa se je nanašala na vsa vozila, ki so bila proizvedena od leta 1991 naprej. Podobne rešitve so sicer obstajale že od sedemdesetih let naprej, vendar so bile plod razvoja vsakega proizvajalca posebej in tako medsebojno nekompatibilne. Zato je leta 1988 SAE predlagal standarden sistem s standardnim priključkom in naborom ukazov, ki je znan pod imenom OBD I. Kasneje sta večino rešitev prevzela EPA in CARB, ki sta odgovorna za kvaliteto zraka v Kaliforniji. OBD I je



Slika 2.3: Struktura sistema OBD.

bil omejen na pridobivanje podatkov, ki so bili neposredno povezani z izpusti emisij motornih vozil. Zaradi potrebe po razširjeni funkcionalnosti je bila leta 1996 razvita izboljšana različica OBD I, znana pod imenom OBD II, z zelo razširjenim naborom ukazov [13].

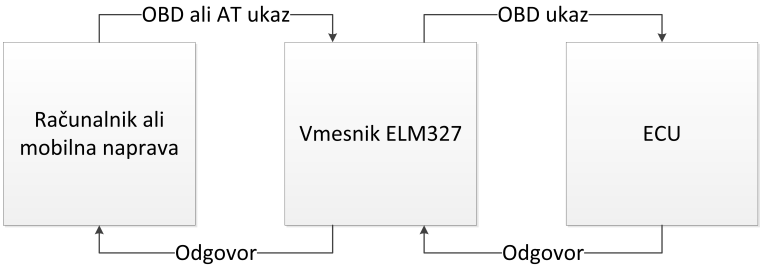
Danes sta v uporabi OBD II ter njegova evropska različica EOBD, ki omogočata pridobivanje podatkov iz senzorjev, ki nadzorujejo delovanje vozila. Osnovni sistem OBD prikazuje slika 2.3. Sestavlja ga krmilna enota ECU, na katero so priključeni različni senzorji in krmilni sistemi, preko katerih ECU zagotavlja željeno raven delovanja vozila. ECU voznika o napakah v delovanju obvešča preko opozorilne lučke MIL, ki se navadno nahaja na armaturni plošči vozila. Do enote ECU lahko dostopamo preko priključka DLC [14].

Standard OBD II podpira uporabo petih komunikacijskih protokolov [15]:

1. **SAE J1850 PWM** - Uporablja ga predvsem Ford, 41,6 kB/s.
2. **SAE J1850 VPW** - Uporablja ga predvsem GM, 10,4 kB/s.
3. **ISO 9141-2** - Starejši protokol, uporabljen predvsem v evropskih vozilih.

4. **ISO 14230-4** - Znan tudi pod imenom KWP2000, zelo razširjen protokol, obstajata dve različici:
   (a) **ISO 14230-4 KWP** - 5 baud init, 10,4 Kbaud.
   (b) **ISO 14230-4 KWP** - fast init, 10,4 Kbaud.
5. **ISO15765-4** - Znan tudi pod imenom CAN-BUS, najnovejši protokol, obvezen od leta 2008 za vsa vozila, prodana v Združenih državah Amerike. Obstajajo štiri različice:
   (a) **ISO 15765-4 CAN** - 11 bit ID, 500 Kbaud.
   (b) **ISO 15765-4 CAN** - 29 bit ID, 500 Kbaud.
   (c) **ISO 15765-4 CAN** - 11 bit ID, 250 Kbaud.
   (d) **ISO 15765-4 CAN** - 29 bit ID, 250 Kbaud.

Komunikacija z ECU po standardu OBD II poteka po načelu zahtev in odgovorov, kar pomeni, da vsaki poslani zahtevi sledi odgovor, po prejetem odgovoru pa lahko pošljemo novo zahtevo. Podatke o stanju željenega sistema



Slika 2.4: Komunikacija z ECU prek vmesnika ELM327.

dobimo tako, da krmilni enoti pošljemo ustrezno zahtevo. Te zahteve se imenujejo PID. Dolžina posameznega PID-a je en bajt, vsaka zahteva pa mora pred PID-om vsebovati tudi način (mode), v katerem pošiljamo zahtevo. OBD II predvideva deset načinov:

- **01** - prikaz trenutnih podatkov

- **02** - prikaz podatkov v času pojava napake v vozilu
- **03** - prikaz diagnostničnih napak v vozilu
- **04** - izbris podatkov v času pojava napake v vozilu
- **05** - rezultati testiranj, spremljanje senzorja za kisik
- **06** - rezultati testiranj, spremljanje ostalih komponent in sistemov
- **07** - prikaz diagnostičnih oznak napak v vozilu, ugotovljenih med trenutno in predhodno vožnjo
- **08** - nadzor delovanja komponent ali sistemov v vozilu
- **09** - prikaz informacij o vozilu
- **0A** - prikaz diagnostičnih oznak napak v vozilu, vključno z izbrisanimi

Podpora vsem načinom za proizvajalce vozil ni obvezujoča, hkrati pa lahko dodajajo svoje načine, ki jih uporabljajo za lastne potrebe) [16].

Primer ukaza je 010C, s katerim zahtevamo trenutno število vrtljajev motorja. Prva dva znaka (01) označujeta način, v katerem pošiljamo zahtevo, druga dva (0C) pa predstavljata dejanski PID. Odgovor na tako zahtevo je navadno oblike 41 0C XX XX. Prva znaka (41) pomenita, da gre za odgovor na zahtevo, druga dva (0C) označujeta, na katero zahtevo se odgovor nanaša, zadnji štirje znaki pa vsebujejo zahtevano vrednost. Dolžina te vrednosti je odvisna od zahteve, navadno pa je v načinu 01 dolga en, dva ali štiri bajte. Nekatere vrednosti je potrebno po prejemu še dodatno obdelati, kar velja tudi za podatek o številu vrtljajev motorja. V tem primeru lahko rečemo, da je prvi bajt zahtevane vrednosti označen z A, drugi pa z B. Pravo število vrtljajev motorja dobimo po formuli 2.1:

$$\frac{(256A) + B}{4}. \tag{2.1}$$

V naši aplikaciji uporabljeni PID-i se nahajajo v tabeli 2.1.

Način	PID	Opis	Enota	Formula
01	0B	senzor razdelilnika absolutnega tlaka	kPa	A
01	0C	število obratov motorja	vrtljaji/minuto	$\frac{(256A)+B}{4}$
01	0D	hitrost vozila	km/h	A
01	0F	temperatura vsesanega zraka	°C	$A-40$

Tabela 2.1: Uporabljeni PID-i.

Slika 2.5: Vmesnik ELM327 z Bluetooth povezavo.

Za uspešno povezavo s sistemom OBD je potreben vmesnik OBD II. V našem primeru smo uporabili ELM327 v1.3a, ki omogoča brezžično povezavo Bluetooth. Gre za brezžično verzijo osnovnega vmesnika ELM327, ki omogoča povezavo preko RS232 in ga izdeluje podjetje Elm Electronics iz Kanade [17]. Vmesnik prikazuje slika 2.5. Poleg standardnih zahtev OBD II podpira tudi vmesniku lastne ukaze. Ti se začnejo z znakoma AT in so namenjeni nastavitvam ter inicializaciji vmesnika. Da je bil ukaz izveden pravilno, vmesnik potrdi z odgovorom »OK«. Zaporedje ukazov, ki smo jih uporabili za inicializacijo vmesnika, se nahaja v tabeli 2.2.

PID	Opis
ATD	nastavitev vmesnika na privzete vrednosti
ATZ	ponastavitev vmesnika
ATE0	nastavitev vračanja vrednosti brez ponovitve ukaza
ATSP4	nastavi protkol na ISO 14230-4 KWP - 5 baud init, 10,4 Kbaud
0100	inicializacija povezave

Tabela 2.2: Inicializacija povezave.

Za lažji razvoj aplikacije smo uporabili tudi odprtokodni OBD II simulator OBDSim [18]. Gre za preprost simulator odgovorov na standardne zahteve OBD II, ki mu jih pošljemo prek željenih komunikacijskih vrat, zaženemo pa ga v ukazni vrstici. Program omogoča tudi beleženje komunikacije ter dva načina generiranja vrednosti. To sta naključni način ter postopno večanje generiranih vrednosti po modulu 256 za vsak bajt. V našem primeru smo uporabili povezavo Bluetooth, saj tako ni bilo potrebno dodatno spreminjati razvitega dela aplikacije na mobilni napravi, ter postopno večanje vrednosti.

```
C:\OBD\OBD Simulator>obdsim --help
Usage: obdsim [params]
   [-g|--generator=<name of generator>
       [-s|--seed=<generator-seed>]
       [-d|--customdelay=<ecu delay(ms)>]
     ]
   [-q|--logfile=<logfilename to write to>]
   [-V|--elm-version=<pretend to be this on ATZ>]
   [-D|--elm-device=<pretend to be this on AT@1>]
```

```
   [-L|--list-protocols]
   [-p|--protocol=<OBDII protocol>]
   [-w|--com-port=<windows COM port>]
   [-e|--genhelp=<name of generator>]
   [-l|--list-generators]
   [-n|--benchmark=<seconds>]
   [-v|--version] [-h|--help]
The generators built into this sim:
 "Random"
 "Cycle" (default)
 "Logger"
 "Error"

C:\OBD\OBD Simulator>obdsim -w COM4
SimPort name: COM4
Successfully initialised obdsim, entering main loop
10.010573 seconds. 0 samples, 0 queries. 0.00 s/s, 0.00 q/s
```

Izvorna koda 2.1: OBDSim.

2.4 OpenStreetMap

OpenStreetMap (v nadaljevanju OSM) so brezplačni zemljevidi celotnega sveta, ki so v obliki PBF in XML na voljo na uradni spletni strani openstreetmap.org. Hkrati gre za tako imenovani »comunity based« projekt, podprt s strani foundacije OpenStreetMap, ki omogoča vsakemu uporabniku sodelovanje pri nadaljnjem razvoju zemljevidov. Za lažje urejanje zemljevidov OSM je na voljo nekaj namenskih programov, kot sta JOSM ter Merkaartor. Poleg tega je na voljo tudi veliko druge odprtokodne programske opreme, katere jedro so ti zemljevidi. Vse spremembe zemljevidov in njihove nadgradnje so vsak teden objavljene na uradni spletni strani [19]. Struktura OSM temelji na kombinaciji štirih primitivnih podatkovnih tipov [19]. Ti so:

1. **Node** - Gre za vozlišče, ki definira eno prostorsko točko na osnovi

zemljepisne širine ter dolžine. Ta točka lahko predstavlja samostojno značilnost, kot je na primer lokacija bencinske črpalke.

2. **Way** - Pot je urejen seznam in lahko vsebuje med 2 in 2000 vozlišč. Pot lahko predstavlja linearne značilnosti, kot so na primer reke ali ceste. Poleg tega lahko predstavlja področja znotraj poligona, ki ga opisuje množica točk. V tem primeru morata biti prvo in zadnje vozlišče v seznamu ista. Primer takega področja so na primer zgradbe ali gozdovi. V primeru, da je za predstavitev neke značilnosti potrebnih več kot 2000 točk, je uporabljen bolj kompleksen podatkovni tip ≫Multipolygon relation≪.

3. **Relation** - Relacija je splošnonamenska struktura, ki opisuje povezavo med dvema ali več objekti. Primer take povezave je na primer kolesarska pot, ki je sestavljena iz več osnovnih objektov tipa pot. Prav tako se relacija uporablja za opis omejitev na cestah in ostalih poteh.

4. **Tag** - Vsak osnovni podatkovni tip lahko vsebuje oznake. Te so namenjene opisu elementa, ki so mu dodane. Struktura oznake je sestavljena iz dveh tekstovnih polj, ključa ter vrednosti, ki lahko vsebujeta poljubne vrednosti, vendar obstajajo smernice za vnos teh podatkov. Na uradni spletni strani je na voljo slovar vseh vrednosti, ki naj bi se uporabljale za opis značilnosti elementov.

```
<?xml version="1.0" encoding="UTF-8" ?>
<osm version="0.6" upload="true" generator="JOSM">
  <bounds minlat="46.1517728"
          minlon="14.4918036"
          maxlat="46.1538241"
          maxlon="14.4968247"
          origin="CGImap 0.3.0 (27047 thorn-03.openstreetmap
          .org)" />
  <node id="677031638"
```

```
        timestamp="2011-07-15T15:59:51Z"
        uid="151665" user="BlueSpark2001"
        visible="true" version="3"
        changeset="8732510"
        lat="46.1484342"
        lon="14.4833785" />
  <node id="677031721"
        timestamp="2011-07-15T15:59:53Z"
        uid="151665"
        user="BlueSpark2001"
        visible="true" version="2"
        changeset="8732510"
        lat="46.1584583"
        lon="14.5041275" />
  ...
  <way id="178305433"
       timestamp="2013-08-01T23:38:20Z"
       uid="398086" user="lesko987"
       visible="true" version="6"
       changeset="17186295">
    <nd ref="1364213274" />
    <nd ref="677031721" />
    <nd ref="677032791" />
    <nd ref="1364213266" />
    ...
    <nd ref="1362804906" />
    <nd ref="1362813094" />
    <tag k="highway" v="unclassified" />
    <tag k="source" v="Yahoo imagery" />
    <tag k="surface" v="asphalt" />
  </way>
</osm>
```

Izvorna koda 2.2: Primer dokumenta XML, ki predstavlja zemljevid.

2.5 Representational state transfer

REpresentational State Transfer ali krajše REST je oblika arhitekture za razvoj spletnih storitev ter aplikacij, opisuje pa jo šest omejitev [20]:

1. **Povezava odjemalec-strežnik** - Enoten vmesnik ločuje odjemalce od strežnikov. To pomeni, da imata odjemalec in strežnik vsak svojo točno določeno vlogo. Odjemalec tako na primer ni odgovoren za hranjenje podatkov, ki je notranja lastnost strežnika. To povečuje prenosljivost in strojno neodvisnost odjemalca. Na drugi strani strežnik ni odgovoren za obliko uporabniškega vmesnika ali za hranjenje stanja odjemalca, kar poenostavi strukturo ter povečuje skalabilnost strežnika. Razvoj strežnika in odjemalca lahko tako poteka neodvisno, dokler njun vmesnik ostane enak.

2. **Stanje povezave** - Stanje povezave med odjemalcem in strežnikom se ne hrani na strežniku med posameznimi zahtevami. Vsaka zahteva odjemalca vsebuje vse potrebne informacije, ki jih strežnik potrebuje za njeno obdelavo. Če je to potrebno, stanje povezave hrani odjemalec.

3. **Pomnjenje zahtev** - Odjemalec ima sposobnost pomnjenja odgovorov na zahteve, v katerih pa mora biti ta možnost eksplicitno definirana. Če to ne drži, lahko pride do napačne uporabe podatkov v teh odgovorih. Dobro definirano pomnjenje lahko zelo zmanjša količino prometa med odjemalcem in strežnikom.

4. **Večnivojski sistem** - Odjemalec ne ve, ali je povezan neposredno na strežnik ali povezava poteka preko posrednikov. Razlika med obema možnostma s strani odjemalca ne sme biti vidna. Posredniki omogočajo skalabilnost sistema, saj lahko razporejajo breme na več strežnikov.

5. **Koda na zahtevo** - Strežnik lahko začasno razširi ali prilagodi funkcionalnost odjemalca, tako da mu kot odgovor na zahtevo pošlje izvršljivo kodo. To je edina omejitev arhitekture REST, katere implementacija ni obvezna.

6. **Enoten vmesnik** - Vmesnik med odjemalcem in strežnikom olajša neodvisen razvoj delov sistema.

Največkrat se pri arhitekturi REST uporablja protkol HTTP, ki že ima vgrajene funkcije za izvajanje klicev storitev. Z izkoriščanjem te lastnosti ne potrebujemo dodatnih, bolj kompleksnih in za vsako aplikacijo prilagojenih mehanizmov, kar je navadno lastnost bolj kompleksnih rešitev, kot so CORBA, RPC ali SOAP. Aplikacije, ki delujejo skladno s smernicami arhitekture REST navadno uporabljajo naslednje klice HTTP [21]:

1. **POST** - Uporablja se za dodajanje vnosa v spletni vir.

2. **PUT** - Uporablja se za osveževanje vnosa v spletnem viru.

3. **GET** - Uporablja se za branje oziroma pridobivanje podatkov iz spletnega vira.

4. **DELETE** - Uporablja se za brisanje podatkov iz spletnega vira.

Lahko pa se uporabljajo tudi druge funkcije, ki so del protokola HTTP. Kombinaciji teh klicev rečemo tudi vmesnik, ki je eden izmed osnovnih omejitev arhitekture REST. GET se pojmuje kot varna funkcija, kar pomeni, da pri njeni uporabi ni nevarnosti, da bi prišlo do sprememb na podatkih strežnika, medtem ko za ostale funkcije to ne velja. Pomembno je omeniti, da REST ni standard, zato uporaba določenih funkcij ali protokola ni nikjer določena. Vsak razvijalec lahko skladno s smernicami arhitekture implementira svojo rešitev.

Tip in oblika podatkov, ki jih odjemalec lahko zahteva, nista predpisana, pomembno je le da jih zna interpretirati. Navadno so podatki v obliki dokumenta HTML, XML ali JSON, lahko pa gre tudi za slike, tekst ali kakšno drugo vsebino.

2.6 MongoDB

2.6.1 Splošno o dokumentno usmerjenih podatkovnih bazah

Dokumentno usmerjena podatkovna baza je NoSQL podatkovna baza, sestavljena iz zbirke samostojnih dokumentov. To pomeni, da so vsi podatki o dokumentu shranjeni v dokumentu samem in ne v povezani tabeli, kot bi bili v relacijski podatkovni bazi. V dokumentni bazi ne govorimo o tabelah, vrsticah, stolpcih ali relacijah, kar pomeni, da je vloga sheme nad podatki veliko manjša kot pri relacijskih sistemih. Oblika podatkov se vzpostavi, ko jih shranjujemo v podatkovno bazo, hkrati pa ni zavezujoča, saj lahko v primeru, da neki dokument potrebuje novo polje, to polje preprosto dodamo brez sprememb na ostalih dokumentih ali podatkovni bazi sami. To hkrati pomeni, da če za neko polje nimamo podatkov, v to polje ni potrebno shranjevati praznih vrednosti, ampak jih preprosto izpustimo. Dokumentno usmerjene podatkovne baze navadno namesto primarnih ključev, ki se zaporedno večajo z dodajanjem zapisov, uporabljajo unikatne univerzalne identifikatorje (UUID), pri čemer je s takim identifikatorjem označen vsak dokument v zbirki. Dokumentno usmerjene baze imajo zelo omejeno in zgolj posredno podporo stikom. To je posledica dejstva, da te baze ne poznajo primarnih in tujih ključev [22].

```
{ "_id" : ObjectId("521f8b5691c8afdaff7504fe"),
  "userName" : "rok",
  "latitude" : 46.10363601912316,
  "longitude" : 14.47601930576183,
  "heading" : 100,
  "consumption" : 3.74874666929247,
  "speed" : 58.32614811312768,
  "roadType" : "secondary",
  "speedLimit" : 90,
  "weather" : "Scattered Clouds",
```

```
  "windSpeed" : 0,
  "windDirection" : 237,
  "temperature" : 16.8,
  "obstacleType" : "No obstacle",
  "noPass" : 0,
  "timestamp" : "19:56"
}
```

Izvorna koda 2.3: Primer dokumenta v podatkovni bazi MongoDB.

2.6.2 MongoDB

MongoDB [25] je predstavnik dokumentno usmerjenih podatkovnih baz. Podatki v bazi so shranjeni v obliki dokumentov v formatu, ki je soroden formatu JSON in se imenuje BSON. MongoDB je odprtokodna ter brezplačna programska oprema, ki je na voljo pod licenco GNU Affero Heneral Public License in Apache License. Razvilo jo je podjetje 10gen leta 2007 kot del produkta tipa platforma kot storitev, dve leti kasneje pa je MongoDB postal samostojen produkt podjetja z enakim imenom. Danes je na voljo različica 2.4.6 in je najbolj razširjena NoSQL podatkovna baza na svetu [23] [24]. Podatki v MongoDB so shranjeni kot dokumenti. Vsak dokument je lahko velik največ 16 megabajtov in vsebuje eno ali več polj. Vsako polje vsebuje ključ in vrednost, ki jo ta ključ predstavlja, lahko pa namesto polja dokument vsebuje tudi poddokumente. Dokument je enolično predstavljen z univerzalnim identifikatorjem (UUID), ključ tega identifikatorja pa je »_id« in je rezerviran za ta namen. Sorodni dokumenti so shranjeni v zbirkah (collection), ki sestavljajo podatkovno bazo. Ti koncepti so podobni tistim v relacijskih podatkovnih bazah [26].

Glavne lastnosti MongoDB so [26]:

1. Iskanje po podatkovni bazi je podobno iskanjem po relacijskih podatkovnih bazah z razliko, da ne uporabljamo sintakse SQL, ampak sin-

Koncepti SQL	Koncepti MongoDB
podatkovna baza	podatkovna baza
tabela	zbirka
vrstica	dokument
stolpec	polje
indeks	indeks

Tabela 2.3: Preslikava konceptov iz SQL v MongoDB.

takso, lastno MongoDB. Iskanja lahko vključujejo tudi uporabniško definirane funkcije JavaScript.

2. Indeksiranje podatkov konceptualno poteka podobno kot pri relacijskih bazah.

3. MongoDB podpira način replikacije podatkov gospodar - suženj. Gospodar lahko izvaja branja in pisanja, medtem ko suženj kopira podatke od gospodarja in lahko izvaja le branja podatkov.

4. Razporejanje bremena poteka na način horizontalnega skaliranja (horizontal scaling) ali shardinga. Vsaka zbirka dokumentov se tako lahko nahaja na več strežnikih, razporejanje pa je avtomatično. V primeru potrebe po več virih lahko strežnike dodajamo delujoči podatkovni bazi.

5. MongoDB se lahko uporablja tudi kot datotečni sistem.

2.6.3 Morphia

Morphia je programska knjižnica, ki omogoča pretvorbo javanskih objektov v zapise v podatkovni bazi MongoDB in obratno. Omogoča preprostejše programiranje aplikacij, ki za hranjenje podatkov uporabljajo MongoDB [27].

2.7 SQLite

SQLite je odprtokodna vgrajena relacijska podatkovna baza. Od ostalih relacijskih podatkovnih baz se razlikuje predvsem po tem, da ne deluje kot ločen strežniški proces. SQLite podatke zapisuje neposredno v klasične datoteke, ki se nahajajo na disku. V takšni datoteki je shranjena celotna podatkovna baza. Zaradi svoje kompaktnosti je zelo primerna za uporabo na mobilnih napravah, kot so na primer telefoni in multimedijski predvajalniki. Vgrajena je tudi kot del platforme Android. Za poizvedbe po podatkovni bazi se uporablja jezik SQL [28].

2.8 Programski aplikacijski vmesniki

2.8.1 Weather Underground API

Weather Underground je spletna storitev, ki uporabnikom omogoča pridobivanje podatkov o trenutnem vremenu na izbrani lokaciji ter vremensko napoved za prihodnje dni. Za uporabo programskega aplikacijskega vmesnika potrebujemo ključ, ki ga dobimo ob registraciji na uradni spletni strani wunderground.com. Storitev je sicer plačljiva, na voljo pa je licenca za razvijalce in ima omejitve glede števila zahtev. Omejitev znaša 500 zahtev na dan ter 10 na minuto, kar je bilo za naše potrebe dovolj. Za nas so bili zanimivi podatki o trenutnem vremenu, ki vsebujejo podatek o tipu vremena, temperaturi ter hitrosti in smeri vetra. Do spletne storitve dostopamo z uporabo zahtev HTTP na URI, ki je oblike

```
http://api.wunderground.com/api/Your_Key/conditions
/q/46.186961,14.491315.xml,
```

odgovor pa je lahko oblike JSON ali XML [29].

```
<response>
```

```
        <version>0.1</version>
        <termsofService>
                http://www.wunderground.com/weather/api/d/
                terms.html
        </termsofService>
        <features>
                <feature>conditions</feature>
        </features>
        <current_observation>
                <display_location>
                        <full>Bukovica pri Vodicah, Slovenia
                        </full>
                        <city>Bukovica pri Vodicah</city>
                        <state_name>Slovenia</state_name>
                        <country>LJ</country>
                        <country_iso3166>SI</country_iso3166>
                        <zip>00000</zip>
                        <latitude>46.186961</latitude>
                        <longitude>14.491315</longitude>
                </display_location>
                <observation_location>
                        <full>Komenda - Moste, Komenda
                         - Moste, CE</full>
                        <city>Komenda - Moste, Komenda
                         - Moste</city>
                        <country>SLOVENIA</country>
                        <country_iso3166>SI</country_iso3166>
                        <latitude>46.195549</latitude>
                        <longitude>14.541242</longitude>
                </observation_location>
                <observation_time>Last Updated on October 23,
                 11:10 AM CEST</observation_time>
                <weather>Mostly Cloudy</weather>
                <temp_c>17.3</temp_c>
                <wind_degrees>270</wind_degrees>
                <wind_kph>0.0</wind_kph>
        </current_observation>
</response>
```

Izvorna koda 2.4: Primer odgovora na zahtevo storitve Weather Underground.

2.8.2 Prometno-infromacijski center API

Slovenski prometno-informacijski center ima na voljo programski aplikacijski vmesnik na naslovu http://kazipot1.promet.si/kazipot/services/dataexport/ in omogoča pridobivanje podatkov o razmerah na cestah. Podatki, ki jih lahko dobimo, so lokacija izrednega dogodka, tip dogodka in podatek, ali je cesta zaprta. Primer naslova URI, na katerega pošljemo zahtevo HTTP:

```
http://kazipot1.promet.si/kazipot/services/dataexport/
exportDogodki.ashx?format=XML&version=1.0.0&reportType=FULL&
language=SI&sortOrder=VELJAVNOSTODDESC&icons=NONE.
```

Odgovor je lahko v obliki XML ali JSON, na voljo pa je tudi kot Excelova datoteka [30].

```
<feed xmlns:georss="http://www.georss.org/georss"
xmlns="http://www.w3.org/2005/Atom">
<title>
Traffic conditions, 10/23/2013 11:35:33 AM local time
</title>
<subtitle>Traffic events</subtitle>
<updated>2013-10-23T09:23:26.2Z</updated>
<link href="http://www.promet.si"/>
<author>
<name>DARS d.d.</name>
<email>info@promet.si</email>
</author>
<entry>
```

```
<title>A1-E57 Maribor - Ljubljana: Roadworks</title>
<id>148733</id>
<updated>2013-10-23T09:23:26.2Z</updated>
<summary>
On the A1-E57 motorway between Celje and Trojane in the
direction of Ljubljana until 23/10/2013 the fast
lane will be closed due to roadworks.
</summary>
<category term="Roadworks"/>
<georss:point>46.27179053948,15.1296378322297</georss:point>
</entry>

...

</feed>
```

Izvorna koda 2.5: Primer odgovora na zahtevo storitve promet.si.

2.8.3 Google Geocoding API

Geokodiranje je proces pretvarjanja uličnih naslovov v geografske koordinate in obratno. Google to ponuja kot spletno storitev prek zahtev HTTP. Primer naslova URI, na katerega pošljemo zahtevo:

`http://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false.`

Odgovor je lahko v formatu XML ali JSON, omejitev pa je 2500 zahtev na dan. Če želimo večje število zahtev, storitev postane plačljiva [31].

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "1",
               "short_name" : "1",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Dunajska cesta",
               "short_name" : "Dunajska cesta",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Ljubljana",
               "short_name" : "Ljubljana",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Slovenia",
               "short_name" : "SI",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "1000",
               "short_name" : "1000",
               "types" : [ "postal_code" ]
            },
            {
               "long_name" : "Ljubljana",
               "short_name" : "Ljubljana",
               "types" : [ "postal_town" ]
            }
         ],
         "formatted_address" : "Dunajska cesta 1,
         1000 Ljubljana, Slovenia",
```

```
         "geometry" : {
            "bounds" : {
               "northeast" : {
                  "lat" : 46.0592626,
                  "lng" : 14.5066324
               },
               "southwest" : {
                  "lat" : 46.0592587,
                  "lng" : 14.5066149
               }
            },
            "location" : {
               "lat" : 46.0592626,
               "lng" : 14.5066149
            },
            "location_type" : "RANGE_INTERPOLATED",
            "viewport" : {
               "northeast" : {
                  "lat" : 46.06060963029149,
                  "lng" : 14.5079726302915
               },
               "southwest" : {
                  "lat" : 46.0579116697085,
                  "lng" : 14.5052746697085
               }
            }
         },
         "partial_match" : true,
         "types" : [ "street_address" ]
      }
   ],
   "status" : "OK"
}
```


Izvorna koda 2.6: Primer odgovora na zahtevo storitve Google Geocoding.

2.9 Osmdroid

Osmdroid je brezplačna programska knjižnica, ki služi kot zamenjava za razred MapView platforme Android. Z njo lahko namesto privzetih Google Maps uporabljamo OpenStreetMap pri razvoju aplikacij. Osmdroid vključuje podporo za izrisovanje zemljevidov ter lastnosti, ki jih želimo programsko dodati tem zemljevidom. V našem primeru smo knjižnico uporabili za izrisovanje poti od naše lokacije do izbranega naslova [32].

2.10 Algoritem za topološko prilagajanje lokacije zemljevidom

Sistem GPS ima svoje omejitve in ena od njih je napaka pri določanju lokacije, ki je navadno reda nekaj metrov. Ta napaka se veča s slabšanjem kakovosti signala, čemur botrujejo različne ovire, kot so na primer visoke stavbe, mostovi, drevje, ipd. V avtomobilski navigaciji to pomeni, da lokacija, ki jo pridobimo iz sprejemnika GPS ne sovpada z lokacijo ceste, po kateri potujemo. Zato potrebujemo algoritem, ki to napako odpravi. Obstaja več različnih algoritmov, ki ta kriterij izpolnjujejo, mi pa smo se odločili za uporabo topološkega algoritma [33].

Algoritem deluje na podlagi kriterijev podobnosti med pridobljenimi lokacijami ter zemljevidi. Ti kriteriji so podobnost v smeri premika ter povezave, razdalja pridobljene lokacije do povezave ter relativna lokacija glede na obravnavane povezave in določajo sistem uteži, ki je uporabljen za prilagajanje lokacij.

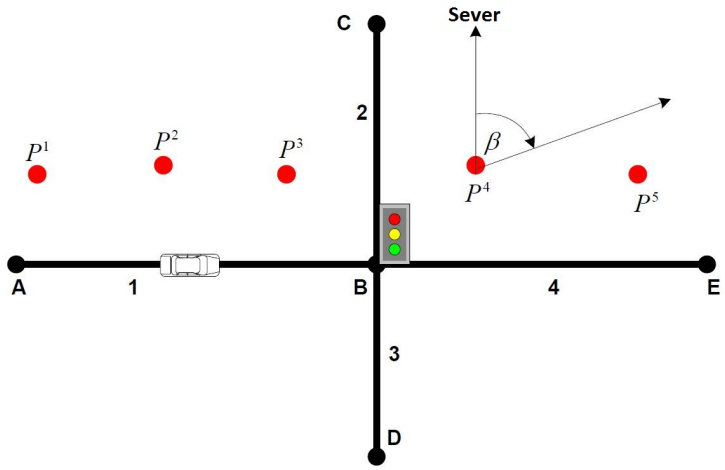
Algoritem kot vhodne podatke uporablja navigacijske podatke, ki vključujejo zemljepisno širino ter dolžino, hitrost ter smer, in podatke iz digitalnih zemljevidov.

2.10.1 Iskanje začetne lokacije

Prvi korak v delovanju algoritma je začetno določanje lokacije. To je zelo pomembno, saj lahko napačna izbira vodi v napačno prilagajanje kasnejših lokacij. Začetno lokacijo določimo tako, da najprej poiščemo pridobljeni lokaciji najbližje vozlišče na zemljevidu. Vse povezave, katerih začetno ali končno vozlišče je najdeno vozlišče, so kandidati za izbiro prave povezave. Nato z izračunom uteži določimo pravo povezavo in točno lokacijo na tej povezavi. Ta postopek je enak tudi za vse naslednje pridobljene lokacije in je opisan v nadaljevanju.

2.10.2 Izračun uteži za smer premika

Predpostavimo, da je algoritem pravilno prilagodil lokacije P_1, P_2 ter P_3 na povezavi AB, kot je razvidno iz slike 2.6. Naslednja lokacija, ki jo moramo prilagoditi, je P_4. Kandidati za povezavo, na kateri se nahaja prava lokacija,



Slika 2.6: Izračun uteži za smer premika.

so BC, BE ter BD, zaradi možnosti polkrožnega obračanja v križišču B pa tudi AB. Za izračun uteži uporabimo smer premika iz točke P_3 v točko P_4 ter podatke o smeri povezav. Kot β predstavlja kot med smerjo premika in

smerjo, ki kaže na sever. Tako dobimo smer premika v stopinjah, v enaki obliki pa so shranjeni tudi podatki o povezavah znotraj digitalnih zemljevidov. Smeri povezav označimo z β', razliko med njima pa $\Delta\beta$. Formula za izračun uteži za smer premika WS_H je tako:

$$WS_H = A_H \; cos(\Delta\beta'); \; A_H > 0, \tag{2.2}$$

kjer je

$$\begin{aligned}
&\Delta\beta = \beta - \beta', \\
&\Delta\beta' = \Delta\beta; \; -180^\circ \leq \Delta\beta \leq 180^\circ, \\
&\Delta\beta' = 360^\circ - \Delta\beta; \; \Delta\beta > 180^\circ, \\
&\Delta\beta' = 360^\circ + \Delta\beta; \; \Delta\beta < -180^\circ.
\end{aligned}$$

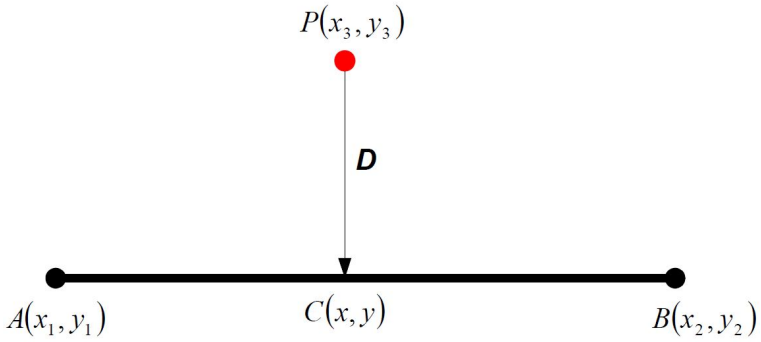
A_H je parameter za izračun uteži za smer premika, njegova vrednost pa je pridobljena po enačbah 2.7 in 2.8. Manjša kot je $\Delta\beta$, večja je verjetnost, da je obravnavana povezava prava in obratno.

2.10.3 Izračun uteži za razdaljo pridobljene lokacije od povezave

Izračun uteži za razdaljo pridobljene lokacije od povezave temelji na izračunu višine na daljico, kar je najkrajša razdalja med točko in daljico. Višino na povezavo izračunamo na podlagi treh točk. To sta točki $A(x_1, y_1)$ in $B(x_2, y_2)$, ki definirata povezavo, ter točka $P(x_3, y_3)$, ki predstavlja pridobljeno lokacijo. Točka, v kateri se višina ter povezava sekata, je točka $C(x, y)$. Na tem mestu ta točka še ni pomembna, kasneje pa bo uporabljena za preslikavo pridobljene lokacije na ustrezen odsek ceste. Višino na povezavo D izračunamo po enačbi:

$$D = \frac{x_3(y_q - y_2) - y_3(x_q - x_2) + (x_1y_2 - x_2y_1)}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}. \tag{2.3}$$

Manjša kot je višina na povezavo, večja je verjetnost, da je obravnavana povezava prava in obratno. Formula za izračun uteži za razdaljo pridobljene



Slika 2.7: Izračun uteži za razdaljo pridobljene lokacije od povezave.

lokacije od povezave WS_{PD} je tako:

$$WS_{PD} = A_{PD}\,\omega \quad ;\ A_{PD} > 0, \tag{2.4}$$

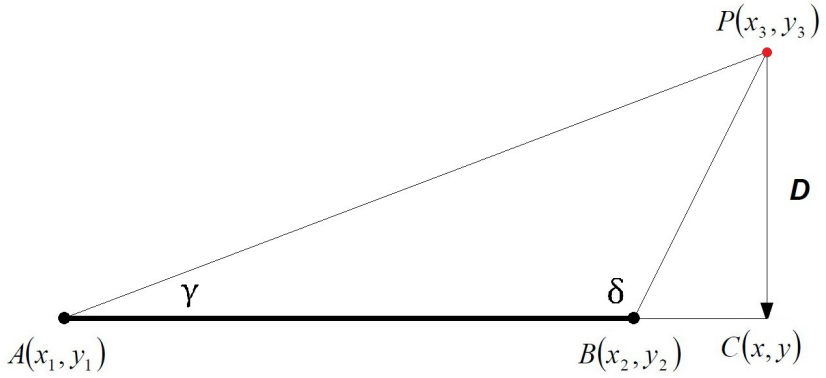
kjer je ω faktor v območju od 0 do 1, odvisen pa je od izračunane razdalje D. ω določimo po naslednjem kriteriju:

$$\omega = 1;\ D < 5\ m,$$
$$\omega = 1.0 - 0.01D;\ 5\ m \leq D \leq 100\ m,$$
$$\omega = -1;\ D > 100\ m.$$

A_{PD} je parameter za izračun uteži za razdaljo lokacije od povezave, njegova vrednost pa je prav tako pridobljena po enačbah 2.7 in 2.8.

Prikazan način izračuna uteži za razdaljo lokacije od povezave se je med testiranji izkazal kot problematičen, saj upošteva le višino na povezavo, ne glede na to, ali je točka res nad povezavo ali pa je zunaj tega območja. To pomeni, da se lahko pridobljena lokacija nahaja relativno daleč od obravnavane povezave, izračunana razdalja med njima pa bo še vedno lahko zelo majhna, za kar lahko trdimo, da ni res.

Zato smo dodali dodatno preverjanje tega pogoja, in sicer tako, da smo pred izračunom razdalje preverili, ali sta kot, ki ga definirajo točke $P(x_3, y_3)$, $A(x_1, y_1)$ in $B(x_2, y_2)$ ter je označen z γ, in kot, ki ga definirajo točke $P(x_3, y_3)$, $B(x_2, y_2)$ in $A(x_1, y_1)$ ter je označen z δ, večja ali enaka 90 stopinj. V primeru,

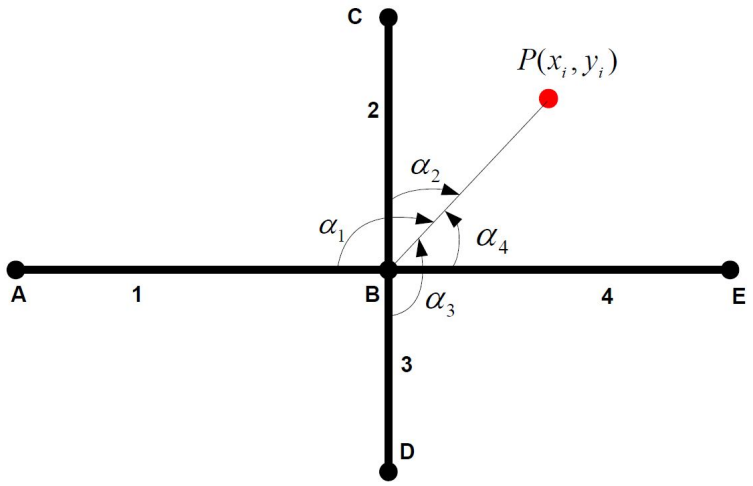


Slika 2.8: Preverjanje kota za izračun uteži.

da je γ večji ali enak 90 stopinj, definiramo razdaljo D kot razdaljo med A in P, ter v primeru, da je δ večji ali enak 90 stopinj, definiramo razdaljo D kot razdaljo med B in P. Poleg tega smo utež WS_{PD} v takem primeru zmanjšali za 10 %, kar je pri testiranju dalo dobre rezultate.

2.10.4 Izračun uteži za relativno lokacijo glede na povezavo

Pri izračunu te uteži gre za primerjavo kotov med daljico BP ter vsemi povezavami, ki jih obravnavamo kot kandidate. α_1, α_2, α_3 in α_4 so najmanjši koti med daljico BP ter obravnavanimi povezavami AB, BC, BD ter BE, kot je to razvidno iz slike 2.9. Manjši kot je kot alfa za določeno povezavo, večja je verjetnost, da gre za pravo povezavo in obratno. To sicer ne drži povsem v primeru, kjer je gostota povezav zelo velika, in lahko pride do napake, še posebej, če ima pridobljena lokacija veliko napako zaradi slabega signala GPS. Formula za izračun uteži za relativno lokacijo glede na obravnavano



Slika 2.9: Izračun uteži za relativno lokacije glede na povezavo.

povezavo WS_{RP} je:

$$WS_{RP} = A_{RP}\ cos(\alpha_i) \quad ;\ A_{RP} > 0, \tag{2.5}$$

kjer je A_{RP} parameter za izračun uteži za relativno lokacijo glede na povezavo in ga izračunamo po enačbah 2.7 in 2.8. Manjši kot je kot α, večja je verjetnost, da gre za pravo povezavo in obratno.

2.10.5 Izračun končne uteži za povezavo

Končna utež TWS je vsota vseh treh uteži. Dobimo jo po formuli:

$$TWS = A_H\ cos(\Delta\beta') + A_{PD}\ \omega + A_{RP}\ cos(\alpha). \tag{2.6}$$

Vrednosti cos($\Delta\beta$), ω in cos(α) se gibljejo med –1 in 1 v odvisnosti od izmerjenih vrednosti $\Delta\beta$, ω in α. Zato lahko vrednost končne uteži nadzorujemo z uporabo parametrov A_H, A_{PD} in A_{RP}. Določata jih enačbi:

$$A_H = a\ A_{PD} \tag{2.7}$$

in

$$A_{RP} = b\ A_{PD}, \tag{2.8}$$

kjer sta a in b nenegativna faktorja, uporabljena za izračun relacije med faktorji uteži. Vrednost faktorjev a in b je odvisna od natančnosti uporabljenih senzorjev ter digitalnih zemljevidov. V literaturi sta predlagani vrednosti $a = 3$, $b = 2$ in $A_{PD} = 10$, vendar se je po nekaj empiričnih testih pokazalo, da je 4 boljša vrednost za a. To posledično pomeni, da ima smer premika večjo težo. Po določitvi a in b bo vsaka pozitivna vrednost A_{PD} vračala smiselne rezultate.

2.10.6 Ocena lokacije vozila na povezavi

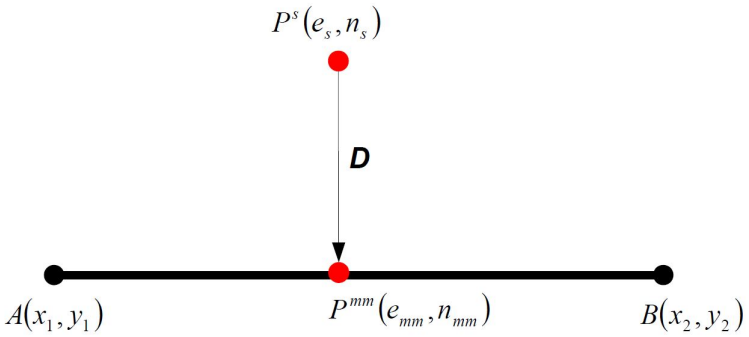
Ob predpostavki, da smo z opisanim postopkom določili pravilno povezavo, na kateri se vozilo nahaja, sledi določanje lokacije na sami povezavi. To storimo z uporabo ortogonalne projekcije pridobljene lokacije $P(x_3, y_3)$ na povezavo, ki je določena z $A(x_1, y_1)$ in $B(x_2, y_2)$. V literaturi [33] so za izračun uporabljene kartezične koordinate, ki določajo odmik od ničelnega poldnevnika in ekvatorja, vendar se zaradi relativno majhnih razdalj ukrivljenost zemlje praktično ne pozna. Zato smo se, v želji po manjši kompleksnosti algoritma ter v izogib pretvarjanju koordinatnega sistema iz kartezičnega v zemljepisno širino in dolžino, odločili, da za koordinate uporabimo kar slednje. Praktična testiranja so potrdila, da je izračun lokacije enako učinkovit. Ortogonalno projekcijo pridobljene lokacije na povezavo izračunamo po formulah:

$$e_{mm} = \frac{(x_2 - x_1)[e_s(x_2 - x_1) + n_s(y_2 - y_1)] + (y_2 - y_1)(x_1y_2 - x_2y_1)}{(x_2 - x_1)^2\ (y_2 - y_1)^2} \quad (2.9)$$

in

$$n_{mm} = \frac{(y_2 - y_1)[e_s(x_2 - x_1) + n_s(y_2 - y_1)] + (x_2 - x_1)(x_1y_2 - x_2y_1)}{(x_2 - x_1)^2\ (y_2 - y_1)^2}. \quad (2.10)$$

Rezultat ortogonalne projekcije je ocena lokacije, ki je predstavljena s točko $P_{mm}(x_4, y_4)$.



Slika 2.10: Ocena lokacije vozila na povezavi.

2.10.7 Preverjanje, ali je vozilo še na trenutni povezavi

Ko smo določili lokacijo vozila na povezavi za pridobljeno lokacijo, začnemo obravnavati naslednjo pridobljeno lokacijo. To lahko storimo tako, da za novo lokacijo ponovno izračunamo vse uteži ter lokacijo na povezavi, lahko pa pred tem preverimo, ali se nova lokacija še vedno nahaja na isti povezavi, kar prihrani veliko računanja. Ali je vozilo še vedno na isti povezavi, preverimo s tremi pogoji:

1. Razlika v smeri med dvema zaporednima daljicama, ki ju določata para točk $P_1(x_i, y_i)$ ter $P_2(x_{i+1}, y_{i+1})$ in $P_2(x_{i+1}, y_{i+1})$ ter $P_3(x_{i+2}, y_{i+2})$, je večja od 45 stopinj.

2. Razlika v smeri med dvema pridobljenima lokacijama je večja od 45 stopinj.

3. Kot alfa, ki je uporabljen za določanje relativne lokacije je večji od 90 stopinj.

Če je katerikoli izmed teh pogojev izpolnjen, lahko rečemo, da vozilo ni več na isti povezavi ter da moramo za novo lokacijo ponovno izračunati vse uteži. V nasprotnem primeru je vozilo še vedno na isti povezavi.

2.10.8 Koraki v delovanju algoritma

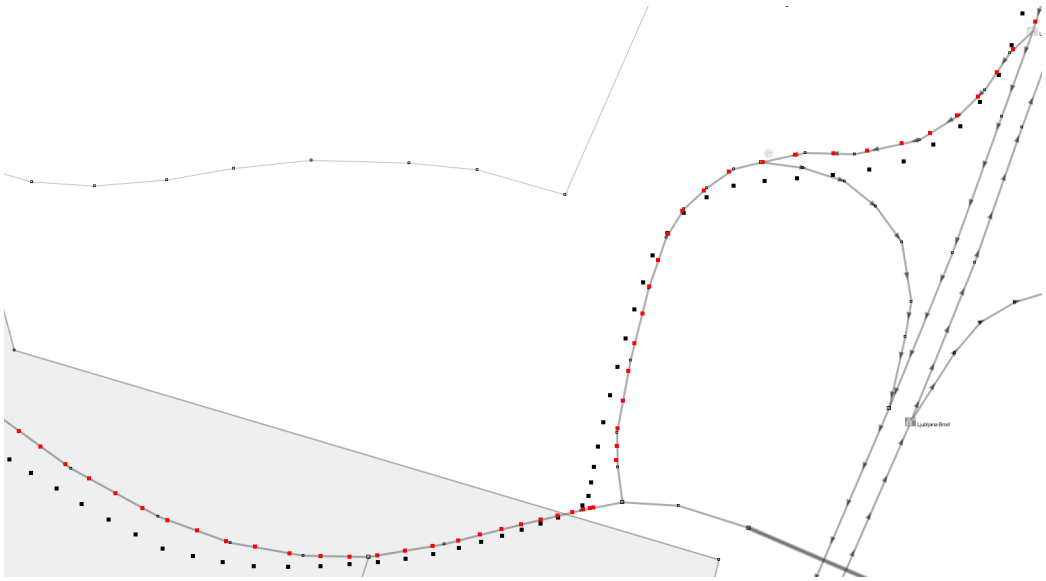
1. Poiščemo najbližje vozlišče lokaciji, ki smo jo pridobili iz GPS za določanje začetne lokacije.

2. Določimo povezave, ki imajo začetno ali končno vozlišče v vozlišču, ki je bilo določeno v prvem koraku.

3. Na podlagi uteži, ki jih izračunamo po formuli 2.6, izberemo pravo povezavo.

4. Ko imamo pravo povezavo, določimo lokacijo na tej povezavi z uporabo formul 2.9 in 2.10.

5. Premaknemo se na novo pridobljeno lokacijo in preverimo, ali je vozilo še vedno na isti povezavi. Če je, izračunamo novo lokacijo na tej povezavi in ponovimo peti korak, sicer pa se vrnemo na prvi korak algoritma.



Slika 2.11: Prilagajanje lokacije.

2.11 A* algoritem

A* je preiskovalni algoritem, ki temelji na best-first algoritmu. Uporablja se v zelo veliko različnih situacijah, od iskanja najboljših poti v grafih do iskanja poti v prostoru stanj. Oboje je zelo pogosto pri razvoju računalniških iger ter navigacijskih sistemov. A* je informirani algoritem, kar pomeni, da poti ne išče slepo v vse smeri, ampak uporablja hevristiko, ki vodi preiskovanje. Hevristika je ocena dolžine poti do cilja od trenutno obravnavanega vozlišča. Posledica tega je, da algoritem preišče manjše območje in tako hitreje najde najboljšo pot do cilja [34]. V naši implementaciji smo za hevristično oceno h uporabili evklidsko razdaljo, ki smo ji dodali utež $1/d$. V zanki smo testirali hitrost iskanja poti v odvisnosti od d z uporabo nabora celih števil od 1 do 30. Najboljše rezultate smo dobili pri vrednosti $d = 11$.

```
function A*(start,cilj)
    closedset = prazna mnozica //mnozica vozlisc, ki so ze
    bila obdelana
    openset = {start} //mnozica vozlisc, ki se niso bila
    obdelana
    came_from := prazna tabela    // Mnozica neobdelanih
    vozlisc z njihovimi predhodniki
    g[start] = 0
    f[start] := g[start] + hevristicna_ocena(start, cilj)
    while openset ni prazen
        trenutni = vozlisce v openset z najmanjso vrednostjo f
        if trenutni = cilj
            return rekontruiraj_najdeno_pot(came_from, cilj)
        odstrani trenutno vozlisce iz openset
        dodaj trenutno vozlisce v closedset
        for vsako sosedno vozlisce(trenutni)
            g := g[trenutni] + napovedana_teza_poti(trenutni,
            sosed)
            f := g + hevristicna_ocena_do_cilja
            if sosodeno vozlisce v closedset in f >= f[sosed]
```

```
                    continue
            if sosoedno vozlisce ni v closedset ali f <
            f[sosed]
                came_from[sosed] := trenutni
                g[sosed] = g
                f[sosed] = f
                if sosedno vozlisce ni v openset
                    dodaj sosedno vozlisce v openset
    return napaka

function rekontruiraj_najdeno_pot(came_from, trenutni)
    if trenutno vozlisce v came_from
        p = rekontruiraj_najdeno_pot(came_from,
        came_from[trenutni])
        return (p + trenutni)
    else return trenutni
```

Izvorna koda 2.7: Psevdokoda algoritma A*.

Za uporabo v naši rešitvi smo v algoritem vpeljali sprotno določanje teže povezav g med vozlišči v grafu. To pomeni, da namesto da težo povezave preberemo iz grafa, jo določimo na podlagi trenutnih voznih razmer in zgodovinskih podatkov o voznikovi vožnji. Težo predstavlja absolutna poraba ali čas potovanja po povezavi, odvisno od izbranega kriterija. Do teh uteži pridemo po enačbah 2.11 in 2.12:

$$g = \frac{3.6\ l}{v + \Delta v} \tag{2.11}$$

in

$$g = \frac{l\ c}{100000}, \tag{2.12}$$

kjer je Δv odstopanje od omejitve hitrosti, v omejitev hitrosti, l dolžina odseka in c povprečna poraba v litrih na sto kilometrov, ki jih dobimo s klasifikacijo na podlagi asociativnih pravil. Ta postopek je opisan v nadaljevanju.

2.12 Algoritem za klasifikacijo na podlagi asociativnih pravil

Algoritem za klasifikacijo na podlagi asociativnih pravil (CBA) [35] je sestavljen iz dveh delov. To sta algoritem za iskanje razrednih asociativnih pravil ter algoritem za generiranje klasifikatorja. Rezultat algoritma je klasifikator, ki ga nato uporabimo za klasifikacijo testnih primerov.

2.12.1 Algoritem za iskanje razrednih asociativnih pravil

Asociativna pravila imajo obliko:

$$\check{C}e\ velja\ X,\ potem\ velja\ Y,$$

kjer sta X in Y podmnožici vseh atributov v podatkovni zbirki, iz katere želimo izluščiti pravila. Pri tem ni pomembno, katera podmnožica je na levi (pogoj) ali na desni (posledica) strani pravila, saj je cilj iskanja asociativnih pravil najti vse zanimive zakonitosti v podatkih, ki zadovoljujejo željen kriterij podpore ter zaupanja. Podpora pravilu je definirana kot:

$$\frac{X \cup Y}{D}, \tag{2.13}$$

kjer sta X pogoj in Y posledica pravila ter D število zapisov v podatkovni zbirki. Z drugimi besedami to pomeni, kolikokrat se podmnožica $X \cup Y$ pojavi v podatkovni zbirki. Zaupanje pa je definirano kot:

$$\frac{X \cup Y}{X}, \tag{2.14}$$

kjer sta X pogoj in Y posledica pravila. To lahko razumemo kot podatek, kolikokrat pravilo drži med vsemi pojavitvami podmnožice X. Asociativna pravila tako nimajo določenega razreda, na podlagi katerega bi iskali zakonitosti. V določenih primerih pa nas zanimajo pravila s točno določenim

ciljnim razredom. Za tako uporabo lahko asociativna pravila prilagodimo v razredna asociativna pravila (CAR). Ta pravila imajo obliko:

$$(pogoj, y),$$

kjer je pogoj podmnožica vseh atributov v podatkovni zbirki razen y ter ciljni razred y. Taka pravila išče tudi prvi del algoritma CBA. Pri iskanju teh pravil govorimo o podpori pogoju, ki je definirana kot:

$$\frac{X}{D}, \tag{2.15}$$

ter o podpori pravilu, ki je definirana kot:

$$\frac{(X, y)}{D}, \tag{2.16}$$

kjer je D število zapisov v podatkovni zbirki. Zaupanje v pravilo izračunamo po formuli

$$\frac{(X, y)}{\frac{X}{D}}. \tag{2.17}$$

Pravila, ki zadovoljujejo izbrani kriterij minimalne podpore, so pogosta pravila, medtem ko so ostala napogosta. Pravila, ki zadovoljujejo kriterij minimalnega zaupanja, so zanesljiva, medtem ko so ostala nezanesljiva.

Algoritem CBA-RG [35] temelji na algoritmu Apriori [36] in prav tako kot algoritem Apriori išče pravila na podlagi več prehodov čez podatkovno zbirko. S prvim prehodom čez podatke išče pogosta prvila ki imajo v pogoju en atribut, zato rečemo, da gre za pravila dolžine 1. Nato iz teh pravil z združevanjem pravil dolžine 1 gradi nova pravila dolžine 2. To deluje tako, da algoritem generira vse možne pare atributov v pogoju, ki imajo enak ciljni razred. Nadaljuje z novim prehodom čez podatke, pri čemer za vsako generirano pravilo dolžine 2 izračuna podporo ter zaupanje. Iz množice pravil odstrani nepogosta in nezanesljiva pravila. To se ponovi za pravila dolžine 3 in tako naprej. Vse možne pogoje ter zaporedje, v katerem so lahko generirani, prikazuje slika 2.12.

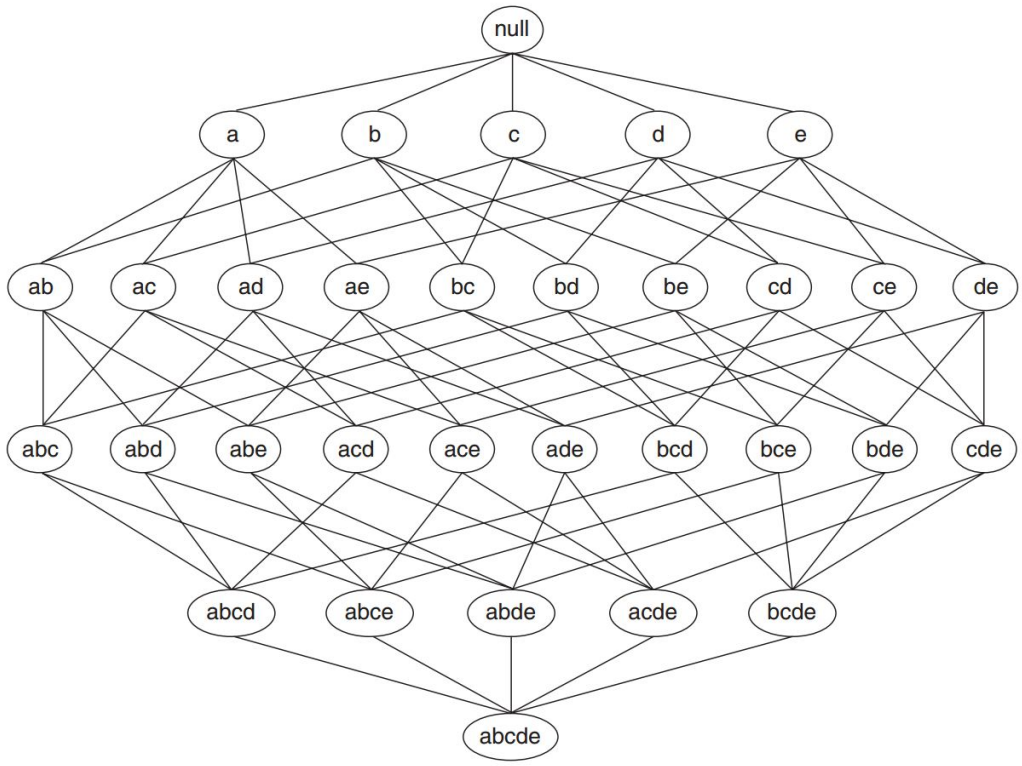
```
F(1) = mnozica pravil dolzine 1
CAR(1) = generiraj_pravila_dolzine(1) iz F(1) //pravila, ki
zadovoljujejo pogojema minimalne podpore in zaupanja
for(k = 2; F(k-1) != prazna mnozica; k++)
        C(k) = generiraj kandidate za pravila iz F(k-1)
        for(vsak zapis d v podatkovni bazi)
                C(d) = podmnozica kadidatov C(k)
                for(vsak kandidat c v C(d))
                        podpora_pravilu++;
                        if(cilnji razred d == ciljni razred c)
                                zaupanje_v_pravilo++;
        F(k) = mnozica kandidatov, katerih podpora_pravilu je
        vecja od zeljene
        CAR(k) = generiraj_pravila_dolzine(k) //pravila, ki
        zadovoljujejo pogojema minimalne podpore in zaupanja
return unija cseh mnozic CAR(k)
```

Izvorna koda 2.8: Psevdokoda algoritma CBA-RG.

2.12.2 Algoritem za generiranje klasifikatorja

Klasifikator je množica pravil, na podlagi katerih se odvija klasifikacija testnih primerov. Gre za podmnožico razrednih asociativnih pravil, ki proizvede najmanjšo napako na učnih podatkih. Za to bi morali preveriti vse možne podmnožice pravil, kar znese 2^m podmnožic, kjer je m število najdenih pravil. Ta številka je lahko zelo velika, kadar gre za veliko število pravil, in preverjanje vsake je časovno nesprejemljivo. Zato lahko uporabimo algoritem za generiranje klasifikatorja, ki je predstavljen v nadaljevanju in je časovno veliko bolj sprejemljiv. Za delovanje algoritma je pomembno, da najdena pravila razvrstimo. To pomeni, da če imamo dve pravili r_i in r_j, ima r_i prednost pred r_j, če je zadoščeno enemu izmed treh pogojev:

1. Zaupanje pravila r_i je večje od zaupanja r_j.



Slika 2.12: Graf možnih kombinacij pogojev.

2. Pravili sta enako zanesljivi, vendar je podpora pravilu r_i večja od podpore pravilu r_j.

3. Pravili sta enako zanesljivi in imata enako podporo, vendar je bil r_i generiran pred r_j.

Generirani klasifikator je tako oblike:

$$< r_1,\ r_2,\ r_3,\ r_4,\ ...,\ r_n,\ privzeti\ razred >,$$

kjer velja, da ima r_a prednost pred r_b, če velja $a > b$. Pri klasifikaciji novega primera bo tako prvo pravilo, ki pokriva primer, klasificiralo ta primer. Algoritem CBA-CB lahko razdelimo na tri korake.

1. Prvi korak je razvrščanje pravil glede na prednost.

2. V drugem koraku algoritem izvede prehod čez podatkovno zbirko za vsako pravilo r iz urejenega seznama pravil, pri tem pa poiščemo vse primere, ki jih r pokriva (zadostujejo pogojem pravila r). Če r pravilno klasificira vsaj enega izmed teh primerov, ga označi in tako postane kandidat za klasifikator in je dodan v seznam C. V tem primeru iz podatkovne zbirke odstranimo vse primere, ki jih r pokriva. Sledi še določanje privzetega razreda, ki postane najbolj pogosta vrednost ciljnega razreda v preostalih podatkih. Korak se konča s preverjanjem števila napak, ki jih naredi množica vseh označenih pravil C ter privzeti razred na učnih podatkih. Ta korak se ponavlja, dokler ne ostane nobenega pravila ali učnega primera več.

3. V tretjem koraku algoritem odstrani vsa označena pravila, ki ne izboljšajo natančnosti klasifikatorja. To so vsa pravila v seznamu C do pravila, od katerega naprej se je natančnost, ki je bila izračunana v drugem koraku, začela manjšati. Klasifikator je tako množica označenih pravil, ki so ostala na seznamu C, ter privzeti razred.

Algoritem tako zadostuje dvema pogojema:

1. Vsak učni primer je pokrit s pravilom z največjo prednostjo med vsemi pravili, ki ga lahko pokrijejo.

2. Vsako pravilo v klasifikatorju pravilno klasificira vsaj en učni primer.

```
R = uredi_pravila(R) //pravila uredimo glede na njihovo
podporo in zaupanje
for(vsako pravilo v R)
        temp = prazna mnozica
        for(vsak ucni primer d)
                if(r pokriva d)
                        shrani d v temp in oznaci r, ce
                        pravilno klasificira d
        if(r je oznacen)
```

```
                dodaj r na konec C //C je mnozica pravil, ki
                sestavljajo klasifikator
                izbrisi vse ucne primere, ki so v temp
                doloci privzeti razred za C
poisci prvo pravilo p v C z najmanjso napako in odstrani vsa
pravila, ki so v klasifikatorju za p
dodaj privzeti razred na konec klasifikatorja
```

Izvorna koda 2.9: Psevdokoda algoritma CBA-CB.

Poglavje 3

Obstoječe rešitve

3.1 Navigacija Google Maps

Google Maps je navigacija v obliki spletne storitve, namenjena pa je tako pešcem kot voznikom, saj nudi izračun poti za pešpoti, ceste ter pri uporabi javnega prevoza. Pri izračunu najboljše poti uporablja hitrostne omejitve ter podatke o prometu, če so ti na voljo. Ocena trajanja poti je tako izračunana na podlagi omenjenih podatkov in ne upošteva voznih navad uporabnika. Storitev ponuja tudi oceno cene poti glede na porabo goriva, vendar je ta ocenjena glede na vneseno velikost vozila in zato zelo nenatančna. V najnovejši verziji Google Maps je ocena porabe na voljo le še za ZDA [37].

3.2 Sistemi za optimizacijo varčne vožnje

Sistemi za optimizacijo varčne vožnje niso namenjeni iskanju poti, ampak uporabnika med vožnjo opozarjajo o popravkih, ki bi jih moral narediti, da bi porabil čim manj goriva. Tu govorimo predvsem o trenutni prestavi ter pritisku na stopalko za plin. Obstaja kar nekaj tovrstnih aplikacij za pametne telefone v kombinaciji z modulom OBD II. Ena izmed njih je EcoShifter OBD2 Car [39], ki uporabnika obvešča o najbolj primernem trenutku za menjavo prestave. Veliko avtomobilov pa ima tak sistem tudi že vgrajen [38].

Poglavje 4

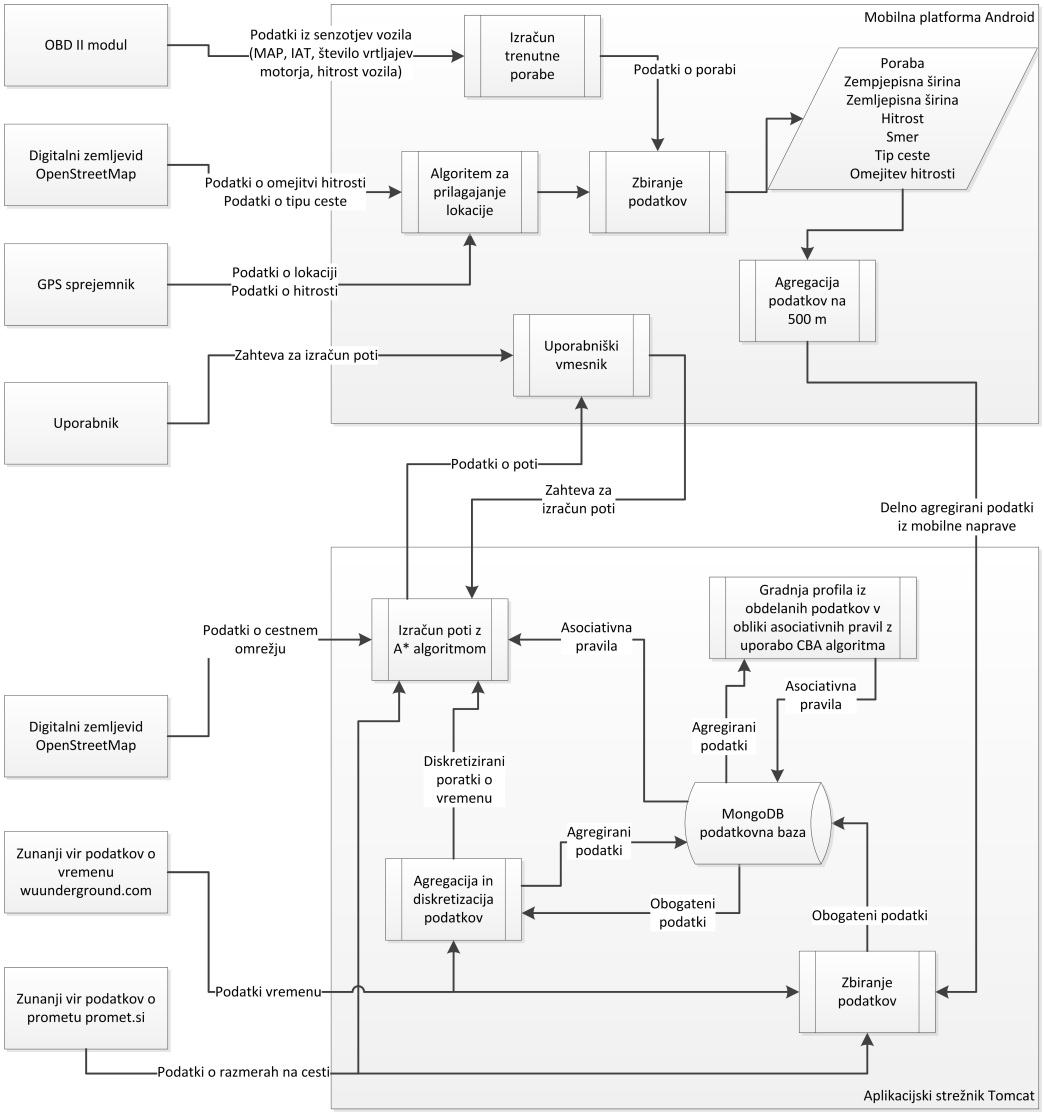
Predlog rešitve in implementacija

4.1 Ideja

Ideja našega sistema je združiti lastnosti obstoječih rešitev, kot sta Google Maps in sistem za optimizacijo varčne vožnje, ter jih nadgraditi. Vožnja je namreč rutinsko opravilo, če ne štejemo izrednih dogodkov. To pomeni, da lahko iz podatkov o vožnji izluščimo pravila. Ta pravila lahko nato uporabimo za bolj učinkovito svetovanje vozniku, kako naj vozi, da bo njegova vožnja čimbolj varčna. Ker pa ljudje včasih težko upoštevamo navodila in ker lahko tak pripomoček odvrača voznikovo pozornost od vožnje, smo uporabili drugačen pristop. Namesto da bi vozniku dajali navodila, kako naj vozi, smo mu pustili njegov način vožnje in optimizirali njegovo pot glede na ta način vožnje. Način vožnje smo modelirali v obliki pravil iz zbranih podatkov.

4.2 Načrtovanje sistema

Sistem, ki smo ga razvili, je sestavljen iz dveh glavnih komponent. To sta mobilna naprava Android ter aplikacijski strežnik Tomcat. Na njiju tečejo vsi procesi, ki so potrebni za obdelavo podatkov ter izračun poti.



Slika 4.1: Shema sistema.

Mobilna naprava črpa podatke iz sprejemnika GPS, digitalnih zemljevidov OpenStreetMap ter modula OBD II. Sprejemnik GPS je vir podatkov o lokaciji, ki so podani z zemljepisno širino in dolžino, ter podatkov o hitrosti. Digitalni zemljevid je vir podatkov o omejitvi hitrosti in tipu ceste, na katerem se trenutno nahaja vozilo. Zaradi napake, ki je del sistema GPS, se iz GPS pridobljena lokacija najprej prilagodi glede na zemljevid in tako se določi pravi odsek ceste, na katerem se vozilo nahaja. Ti podatki se nato prenesejo v proces zbiranja podatkov. Podatki iz modula OBD II so dodatno obdelani, saj testno vozilo, ki smo ga uporabili, ne podpira PID-a za trenutno porabo, prav tako pa tudi ne podpira PID-a za senzor masnega pretoka (MAF). To pomeni, da smo uvedli še proces izračuna trenutne porabe iz podatkov, ki so na voljo. Izračun je bolj natančno predstavljen v nadaljevanju. Podatek o trenutni porabi se nato prenese v proces zbiranja podatkov. Ti podatki se v naslednjem koraku delno agregirajo. Gre za izračun povprečij porabe ter hitrosti na vsakih 500 prevoženih metrov. Na tej točki se podatki pošljejo na strežnik.

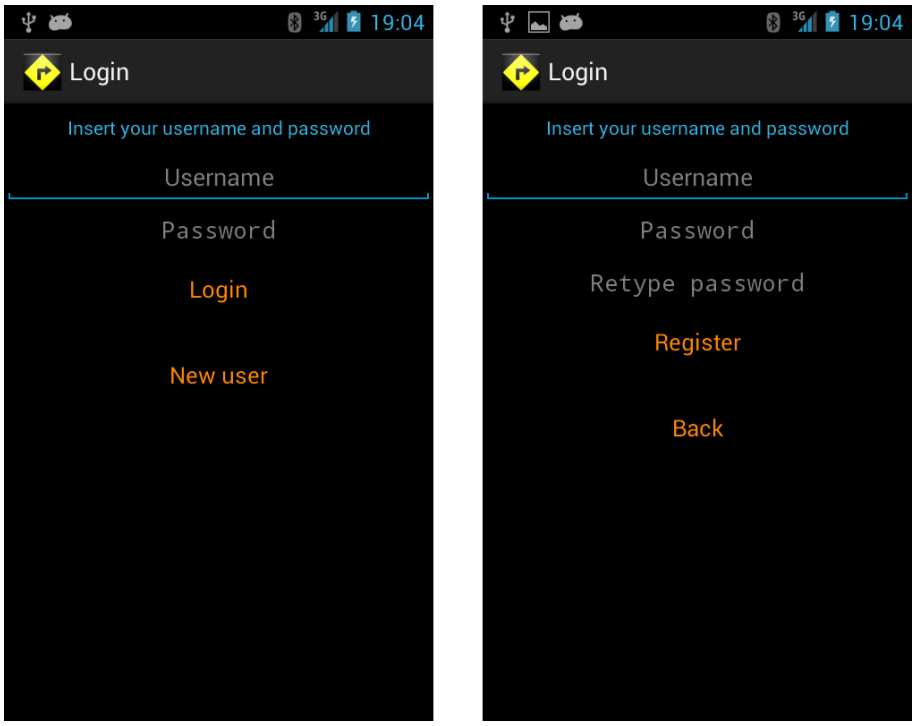
Na strežniku se podatki iz mobilne naprave obogatijo s podatki o vremenu na lokaciji, na kateri se vozilo trenutno nahaja, ter s podatki o prometnih razmerah na prevoženem odseku. Tako obogateni podatki se shranijo v podatkovno bazo, iz katere črpa podatke proces agregacije in diskretizacije podatkov. V tem procesu se zvezni podatki pretvorijo v diskretne, kar je pogoj za učinkovito delovanje algoritma za klasifikacijo na osnovi asociativnih pravil, ki je opisan v poglavju 2.12 in ki je naslednji korak v obdelavi podatkov. Rezultat tega algoritma so asociativna pravila, ki tvorijo klasifikator. Ta je osnova za klasifikacijo razmer in napoved porabe ali hitrosti, ki jih za izračun poti potrebuje nadgrajeni algoritem A*, ki je opisan v poglavju 2.11. Algoritem A* poleg asociativnih pravil potrebuje še podatke o prometnih razmerah, cestnem omrežju ter diskretizirane podatke o vremenu. Na podlagi teh podatkov izračuna najboljšo pot glede na izbran kriterij, ki je lahko odstopanje od omejitve hitrosti ali pa poraba goriva.

Za poganjanje naše aplikacije smo uporabili mobilno napravo Samsung

Google Nexus S, na katerem je tekel operacijski sistem Android 4.1.2 Jelly Bean. Mobilna naprava vsebuje sprejemnik GPS ter modul Bluetooth, ki sta pomembna za delovanje naše aplikacije. Preko povezave Bluetooth smo na mobilno napravo priključili modul OBD II ELM327 1.3a, ki je bil priključen na testno vozilo Volkswagen Polo 9n3 z 1.4-litrskim bencinskim motorjem. Na strani strežnika smo za poganjanje spletne storitve uporabili aplikacijski strežnik Apache Tomcat 7.0.40, v času pisanja pa je sicer postala na voljo različica 8.0.

4.3 Uporabniški vmesnik

4.3.1 Prijava v sistem

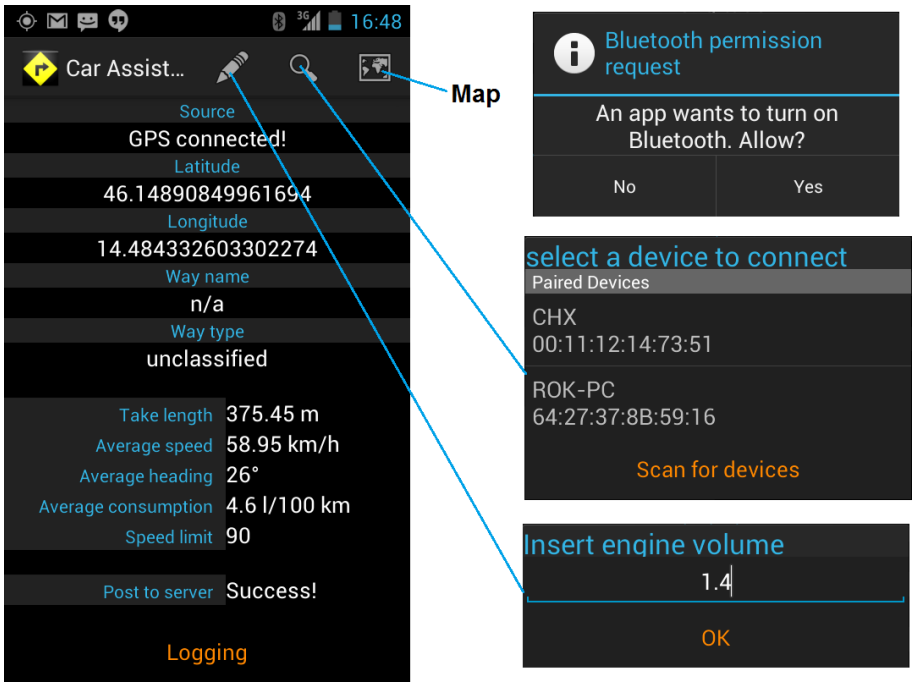


Slika 4.2: Prijava v sistem.

Uporabnik se v sistem prijavi z uporabniškim imenom in geslom ter pri-

tiskom na gumb »Login«. Če gre za novega uporabnika, lahko z gumbom »New user« preidemo v način registracije, kjer na podoben način v sistem vpišemo novega uporabnika.

4.3.2 Zbiranje podatkov

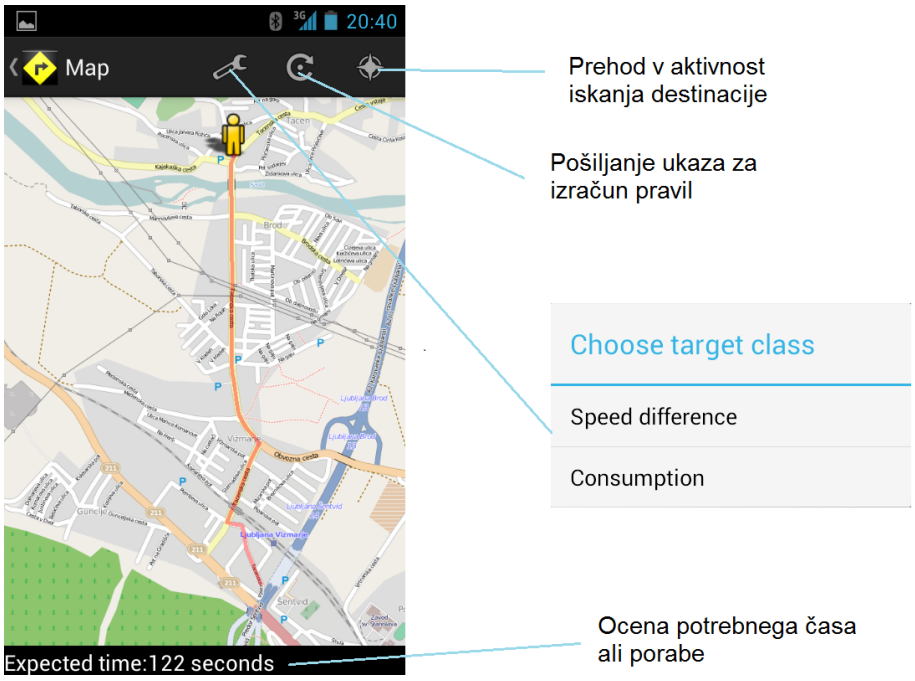


Slika 4.3: Zbiranje podatkov.

Po uspešni prijavi se uporabniku na zaslonu prikaže glavna aktivnost za zbiranje podatkov o vožnji. V primeru, da povezava Bluetooth ni vklopljena, se prikaže opozorilo, v katerem jo uporabnik lahko vklopi. Če tega ne želi, bodo zbrani podatki brez podatka o porabi. V zgornjem delu zaslona se nahaja preprost meni, ki vključuje 3 gumbe. Prvi omogoča uporabniku vnos prostornine motorja. Drugi gumb v meniju omogoča uporabniku povezovanje z Bluetooth modulom OBDII prek preprostega menija, kjer lahko uporabnik izbere že poznano napravo ali napravo poišče. Na dnu zaslona sta še indikator, ki sporoča, ali je bilo pošiljanje podatkov na strežnik uspešno ali ne, ter

gumb, s katerim lahko zbiranje podatkov izklopimo ali vklopimo.

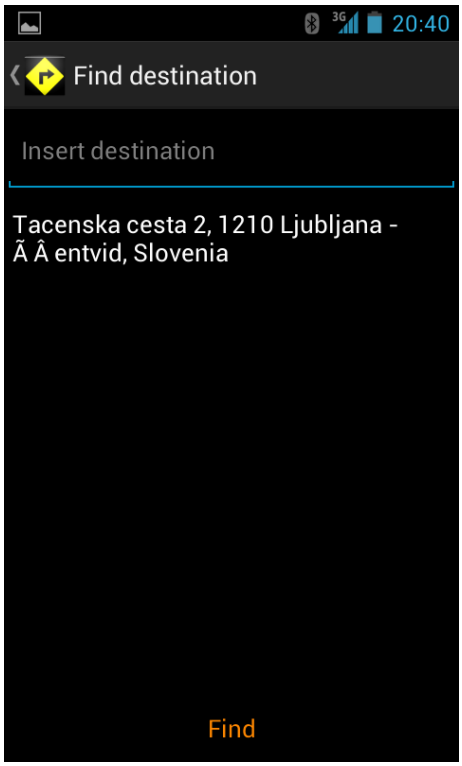
4.3.3 Zemljevid



Slika 4.4: Izris zemljevida in najboljše poti.

V tej aktivnosti lahko uporabnik pregleduje zemljevid, na katerem se mu tudi izriše pot do cilja. Tudi tu je na zgornjem delu zaslona meni. Prvi gumb uporabniku omogoča izbiro kriterija za izračun poti glede na pravila o njegovi vožnji, drugi gumb pošlje na strežnik zahtevo o ponovnem izračunu pravil, tretji gumb pa uporabniku prikaže novo zaslonsko masko, kjer lahko z vnosom naslova določi cilj poti. Na dnu zaslona se prikaže pričakovan čas poti oziroma absolutna poraba goriva na poti v litrih.

4.3.4 Iskanje cilja



Slika 4.5: Iskanje cilja željene poti.

Na vrhu zaslona se nahaja polje za vnos, kjer uporabnik vnese željen cilj potovanja. To je lahko naslov, podjetje, javna ustanova ali kaj podobnega. Za iskanje je uporabljena storitev Google Geocoding, zato so tudi rezultati podobni tistim na Google Maps. Po pritisku gumba »Find«, se na zaslonu v obliki seznama izpišejo rezultati iskanja. Zaradi omejitev storitve Geocoding (2500 iskanj na dan) in enotnega proxy strežnika mobilnega operaterja so iskalni ukazi poslani na strežnik, kjer tečejo ostale storitve, ki jih aplikacija uporablja, ter od tam preusmerjeni na Google.

4.4 Zajemanje podatkov na mobilni napravi

Podatke na mobilni napravi smo zbirali periodično na vsakih 500 prevoženih metrov iz treh različnih virov. To so sprejemnik GPS, modul OBD II ter digitalni zemljevid. Podatki, ki jih mobilna naprava črpa iz sprejemnika GPS so hitrost, smer in lokcija. Za prvi dve smo po koncu vsakega 500-metrskega odseka izračunali povprečja. Razdaljo 500 metrov smo izbrali na podlagi empiričnih testov, saj se je izkazalo, da krajši kot je odsek, večja so odstopanja, saj povprečje izračunamo iz manjšega števila podatkov. Daljša razdalja bi pomenila manjše število zbranih podatkov, kar pa tudi ni dobro.

Podatke smo zbirali le na odsekih, kjer je vozilo prevozilo celotnih 500 metrov pod enakimi pogoji. V primeru, da se je med vožnjo spremenil tip ceste ali omejitev hitrosti, smo vse vrednosti ponastavili in začeli nov odsek. Tak pristop se je izkazal kot dober tudi v primeru, ko je algoritem za prilagajanje lokacije to napačno prilagodil in določil lokacijo na cesti, kjer se vozilo ni nahajalo. Take napake so bile sicer redke in so navadno trajale do 150 metrov. Še vedno pa obstaja možnost, da je cesta, ki jo je algoritem napačno določil, enakega tipa in ima enako hitrostno omejitev, ampak v tem primeru s stališča podatkov ni pomembno, ali je cesta prava, saj so razmere enake.

Senzor GPS je tudi vir podatka o lokaciji v obliki zemljepisne širine in dolžine. Tega smo dodali k ostalim zbranim podatkom, saj ga je za nadaljnjo obdelavo potreboval strežnik. Drugi vir podatkov je OBD II. Iz tega vira mobilna naprava črpa podatek o porabi. Ker testno vozilo ni podpiralo PID-a za trenutno porabo goriva, prav tako pa tudi ni podpiralo PID-a za masni pretok zraka (MAF ali mass air flow), s katerim se sicer lahko izračuna trenutna poraba, je bil izračun porabe bolj kompleksen. Na voljo so bili PID-i za senzor razdelilnika absolutnega tlaka (MAP ali manifold absolute pressure), senzor števila obratov motorja, senzor hitrosti vozila ter senzor temperature vsesanega zraka (IAT ali intake air temperature). Na podlagi teh podatkov smo izračunali psevdomasni pretok zraka z uporabo enačbe 4.1:

$$maf = \frac{map}{iat} \frac{rpm}{60} \frac{ed}{2} \frac{mm}{R} ve. \tag{4.1}$$

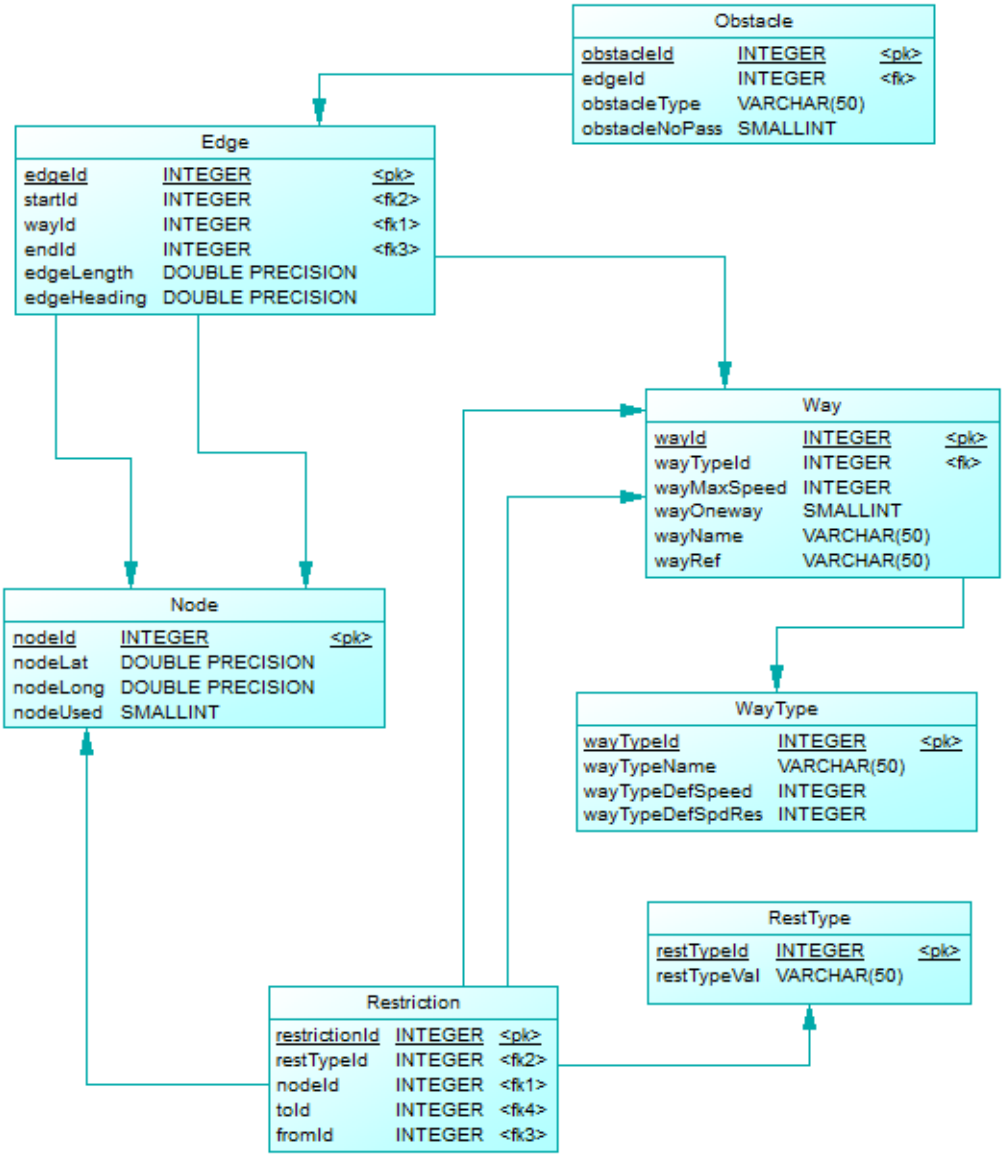
Na podlagi izračunane vrednosti masnega pretoka zraka smo po enačbi 4.2 izračunali trenutno porabo motorja v litrih na 100 kilometov:

$$c = \frac{3600\ maf}{14.7\ 730.0\ v}\ 100. \tag{4.2}$$

Pomen spremenljivk v enačbah 4.1 in 4.2:

- *map* - vrednost senzorja razdelilnika absolutnega tlaka
- *iat* - temperatura vsesanega zraka
- *rpm* - število obratov na minuto
- *ed* - prostornina motorja
- *v* - hitrost vozila
- *mm* - molekulska masa zraka (28.97 g/mol)
- *ve* - izkoristek motorja
- 14,7 - idealno razmerje med zrakom in gorivom
- $R = 8.314 J/^{\circ}K/mol$ - splošna plinska konstanta
- 730 - gostota goriva v g/l za 95-oktanski bencin

Izkoristek motorja se sicer spreminja med delovanjem motorja, vendar so spremembe dovolj majhne, da lahko privzamemo, da gre za konstantno vrednost. V našem primeru smo uporabili približek 0.75. Tretji vir podatkov pa je digitalni zemljevid. Iz tega smo pridobili podatke o omejitvi hitrosti ter tipu ceste, na kateri se vozilo trenutno nahaja. Digitalni zemljevid smo iz oblike XML pretvorili v graf in ga nato shranili v podatkovno bazo SQLite. Shema baze, ki smo jo uporabili za shranjevanje zemljevida, je vidna na sliki 4.6.



Slika 4.6: Shema podatkovne baze SQLite za hranjenje digitalnega zemljevida.

4.5 Prenos podatkov na strežnik

Na strežniku smo implementirali spletno storitev REST, ki sprejema zahteve POST. Preko te storitve smo na mobilni napravi implementirali pošiljanje podatkov na strežnik. Postopek ima tri korake:

1. Podatke, ki so na mobilni napravi shranjeni kot javanski objekt, smo pretvorili v obliko JSON. Pri tem smo uporabili odprtokodno knjižnico Gson [40], ki podpira preprosto pretvorbo med javanskimi objekti in zapisom JSON teh objektov.

2. Na strežnik smo poslali zahtevo POST s podatki v obliki JSON.

```
{
  "username" : "rok",
  "userpass" : "rok",
  "latitude" : 46.186961,
  "longitude" : 14.491315,
  "heading" : 204,
  "consumption" : 5.6,
  "speed" : 56.87,
  "roadType" : "primary",
  "speedLimit" : 50,
  "wayId" : 34554365,
  "weatherDataCheck" : true
}
```

Izvorna koda 4.1: Primer dokumenta JSON, ki je uporabljen za prenos podatkov iz mobilne naprave na strežnik.

3. Na strežniku smo prejete podatke v obliki JSON z uporabo knjižnice Gson pretvorili nazaj v javanski objekt. Ta objekt smo nato neposredno uporabili za nadaljnjo obdelavo.

4.6 Obdelava podatkov na strežniku

Na strežniku smo podatkom, ki smo jih dobili iz mobilne naprave, dodali še podatke o vremenu iz spletne storitve Weather Underground ter podatke o razmerah na cestah s Prometno-informacijskega centra Slovenije. Vremenski podatki obsegajo tip vremena, temperaturo, smer vetra ter hitrost vetra, podatki o razmerah na cestah pa izredne dogodke na cesti. Podatke o vremenu in prometnih razmerah smo zbirali v petminutnih intervalih. Tako obogatene podatke smo shranili v podatkovno bazo MongoDB. Nadaljevali smo z obdelavo teh podatkov. Iz podatkov o smeri vožnje, smeri vetra in hitrosti vetra smo z uporabo enačbe 4.4 izračunali koeficient zračnega upora:

$$hitrost\ vetra * cos(smer\ vetra - smer\ potovanja). \tag{4.3}$$

Poleg tega smo združili hitrost vožnje in omejitev hitrosti ter s tem dobili odstopanje od omejitve hitrosti:

$$hitrost\ vožnje - omejitev\ hitrosti. \tag{4.4}$$

Vse zvezne podatke smo diskretizirali. To smo storili tako, da smo razpon zbranih vrednosti razdelili v deset košev z enakimi intervali. Za vrednost koša smo vzeli njegovo srednjo vrednost. S temi koraki smo zmanjšali število atributov in njihovih vrednosti, saj algoritem za generiranje asociativnih pravil deluje bolje z manjšim številom atributov in vrednosti. Tako obdelane podatke smo shranili v svojo zbirko dokumentov v podatkovni bazi. S tem nismo izgubili originalnih podatkov.

4.7 Generiranje pravil iz podatkov

Na obdelanih podatkih smo izvedli algoritem za iskanje razrednih asociativnih pravil, ki smo jih uporabili kot vhod v algoritem za generiranje klasifikatorja. Oba algoritma sta opisana v poglavju 2.12. Za stopnjo podpore pravilu smo uporabili vrednost 0,1, za stopnjo zaupanja pa 0,3. Nizke vrednosti teh parametrov smo uporabili zaradi majhnega števila in netočnosti

testnih podatkov, kar je opisano v poglavju 5.2. Višje postavljene vrednosti teh parametrov na testnih podatkih bi izločile večino vseh najdenih pravil. Implementirali smo tudi možnost izbire kriterija, glede na katerega se generira klasifikator. To sta odstopanje od omejitve hitrosti in poraba goriva. Vsak kriterij ima svoj nabor pravil, ki sestavljajo dva ločena klasifikatorja, ki se glede na željen kriterij tudi uporabi pri izračunu poti.

4.8 Izračun optimalne poti in prenos rezultata na mobilno napravo

Izračun optimalne poti smo implementirali z nadgrajenim algoritmom A*, ki je opisan v poglavju 2.11. Težo povezave smo pridobili s klasifikacijo trenutnih razmer, ki smo jih pridobili s pomočjo spletnih storitev Weather Underground ter Prometno-informacijskega centra Slovenije. Poleg tega smo za klasifikacijo uporabili podatke o povezavi, katere težo smo klasificirali. Te podatke smo pridobili iz digitalnega zemljevida OpenStreetMap. Pri tem smo uporabili ustrezen klasifikator glede na kriterij, ki določa, ali želimo najti najboljšo pot glede na porabo goriva ali glede na odstopanje od omejitve hitrosti. Rezultat algoritma A* je pot od začetne do končne točke v obliki seznama vozlišč na digitalnem zemljevidu.

Iskanje najboljše poti se začne, ko uporabnik z uporabo mobilne naprave na strežnik pošlje zahtevo, ki sproži izvajanje algoritma. Po koncu izvajanja algoritma smo seznam vozlišč pretvorili v obliko JSON in ga kot odgovor spletne storitve poslali nazaj na mobilno napravo, kjer se je pot izrisala na zaslonu. Bolj podroben opis izrisa poti se nahaja v poglavju 4.3.3.

Poglavje 5

Sklepne ugotovitve in zaključek

5.1 Rezultati testiranj

Algoritem A* za izračun najboljše poti uporablja težo povezave, ki jo dobi na podlagi klasifikacije trenutnih razmer z uporabo razrednih asociativnih pravil. To pomeni, da je natančnost napovedi absolutne porabe goriva ali časa potovanja na izbrani poti neposredno odvisna od natančnosti klasifikacije trenutnih razmer. Poleg tega test, pri katerem bi izračunano pot dejansko prevozili in primerjali porabo in čas z napovedjo, ni ponovljiv. Zato smo se odločili, da natančnost našega sistema testiramo na koraku klasifikacije razmer.

To smo storili tako, da smo na zbranih podatkih izvedli k-prečno preverjanje, pri čemer smo za k vzeli 5. Izvedli smo dve ločeni testiranji. Prvo smo izvedli le na podatkih, ki smo jih zbrali na avtocestah in hitrih cestah, saj so omejitve za ta dva tipa ceste pravilno vnesene v digitalni zemljevid. Drugo testiranje smo izvedli na vseh zbranih podatkih.

Za vsako testiranje smo izmed petih učnih in testnih množic poiskali stopnjo podpore in zaupanja, pri kateri smo dobili največjo natančnost. Preiskali smo vse vrednosti od 0 do 1 s korakom 0.01. Rezultati so vidni v tabelah 5.1 in 5.2. Na podlagi tega smo izračunali povprečne stopnje podpore in zaupanja za vsak primer, ki smo jih uporabili pri testiranju. Določitev

Kriterij	Test	Podpora	Zaupanje	Natančnost (%)
Odstopanje od omejitve	1	0.02	0.26	26
Odstopanje od omejitve	2	0.02	0.37	15
Odstopanje od omejitve	3	0.03	0.31	19
Odstopanje od omejitve	4	0.03	0.24	40
Odstopanje od omejitve	5	0.03	0.09	9
Poraba goriva	1	0.02	0.41	32
Poraba goriva	2	0.04	0.39	28
Poraba goriva	3	0.04	0.67	26
Poraba goriva	4	0.03	0.25	40
Poraba goriva	5	0.06	0.33	37

Tabela 5.1: Najboljše vrednosti zaupanja in podpore za podatke z avtocest in hitrih cest.

najboljše stopnje podpore in zaupanja je namreč nerešljiv problem, zato smo uporabili opisani pristop in si s tem poskusili izboljšati možnosti za uspeh. Kljub temu pa ni nujno, da so izbrane vrednosti najboljše. Rezultati posameznih testiranj so vidni v tabelah 5.3, 5.4, 5.5 in 5.6.

Rezultati za odstopanje od omejitve hitrosti so slabi, iz česar lahko sklepamo, da nismo zajeli pravih podatkov, na podlagi katerih bi lahko uspešno napovedali hitrost vožnje v trenutnih voznih razmerah. Kot lahko vidimo iz tabel 5.3 in 5.5, pa je natančnost napovedi podatkov na avtocestah in hitrih cestah za 8 odstotkov višja kot napoved za vse ceste, kar pomeni, da se natančnost skoraj podvoji. To pove, da je podatek o omejitvi hitrosti pomemben za izboljšanje natančnosti napovedi.

Ravno nasprotno pa je natančnost napovedi porabe goriva večja pri testih, ki so bili izvedeni na vseh zbranih podatkih, kot pri testih, ki se nanašajo le na avtoceste in hitre ceste. To je razvidno iz tabel 5.4 in 5.6. Poleg tega je rezultat na splošno boljši kot pri napovedi odstopanja od omejitve hitrosti. Na podlagi teh rezultatov lahko sklepamo, da število podatkov vpliva na

Kriterij	Test	Podpora	Zaupanje	Natančnost (%)
Odstopanje od omejitve	1	0.01	0.31	18
Odstopanje od omejitve	2	0.01	0.32	13
Odstopanje od omejitve	3	0.01	0.36	11
Odstopanje od omejitve	4	0.01	0.31	8
Odstopanje od omejitve	5	0.01	0.31	8
Poraba goriva	1	0.06	0.61	41
Poraba goriva	2	0.01	0.56	41
Poraba goriva	3	0.01	0.66	51
Poraba goriva	4	0.06	0.51	47
Poraba goriva	5	0.11	0.41	58

Tabela 5.2: Najboljše vrednosti zaupanja in podpore za vse zbrane podatke.

Test	Pravilno klasificirani primeri	Napačno klasificirani primeri	Natančnost (%)
1	9	44	17
2	6	47	11
3	4	49	08
4	21	32	40
5	4	50	7
Povprečje			17

Tabela 5.3: Rezultati testov natančnosti za kriterij odstopanja od omejitve hitrosti na podatkih, ki smo jih zbrali na avtocestah in hitrih cestah. Za stopnjo podpore smo vzeli 0,03 in za stopnjo zaupanja 0,25.

Test	Pravilno klasificirani primeri	Napačno klasificirani primeri	Natančnost (%)
1	12	41	23
2	12	41	23
3	14	39	26
4	18	35	34
5	20	34	37
Povprečje			29

Tabela 5.4: Rezultati testov natančnosti za kriterij porabe goriva na podatkih, ki smo jih zbrali na avtocestah in hitrih cestah. Za stopnjo podpore smo vzeli 0,04 in za stopnjo zaupanja 0,40.

Test	Pravilno klasificirani primeri	Napačno klasificirani primeri	Natančnost (%)
1	31	141	18
2	17	155	10
3	6	166	3
4	12	160	7
5	15	158	9
Povprečje			9

Tabela 5.5: Rezultati testov natančnosti za kriterij odstopanja od omejitve hitrosti na vseh zbranih podatkih. Za stopnjo podpore smo vzeli 0,01 in za stopnjo zaupanja 0,30.

Test	Pravilno klasificirani primeri	Napačno klasificirani primeri	Natančnost (%)
1	70	102	41
2	71	101	41
3	82	90	48
4	78	94	45
5	99	74	57
Povprečje			46

Tabela 5.6: Rezultati testov natančnosti za kriterij porabe goriva na vseh zbranih podatkih. Za stopnjo podpore smo vzeli 0,05 in za stopnjo zaupanja 0,55.

kvaliteto klasifikatorja. Poleg tega lahko rečemo, da so v primeru napovedi porabe goriva zbrani podatki bolj relevantni.

5.2 Težave pri razvoju

Pri razvoju smo imeli težave predvsem z nepopolnimi podatki. Najbolj so nam primanjkovali podatki o omejitvah hitrosti, ki niso na voljo za zelo velik delež cest v digitalnem zemljevidu OpenStreetMap. Posledica tega so napačni podatki o odstopanju od omejitve hitrosti, kar je vzrok za zelo slabo delovanje sistema. Težavo je predstavljal tudi podatek o porabi, saj smo morali za prostorninsko učinkovitost motorja uporabiti približek, čeprav se ta realno med vožnjo nekoliko spreminja. Zadnjo večjo težavo pa so predstavljale spletne storitve, iz katerih smo črpali podatke o vremenu ter prometnih razmerah, ter storitev Google Geocoding. Weather Underground ter Google Geocoding imata omejene brezplačne licence. Predvsem prvo, ki omogoča 500 zahtev na dan in do največ deset zahtev na minuto, smo nekajkrat presegli. Taka situacija je otežila testiranje, saj eden izmed ključnih virov podatkov ni bil na voljo. Problem podatkov o prometnih razmerah pa se je nanašal

predvsem na strukturo in nepopolnost podatkov Prometno-infromacijskega centra Slovenije, saj so podatki o čakalnih dobah in dolžinah zastojev podani opisno, največkrat pa jih sploh ni.

5.3 Možnosti za izboljšave in nadgradnje

Sistem, ki smo ga izdelali za potrebe diplomske naloge, je le prototip, ki prikazuje opisano idejo in realne možnosti njene uresničitve. Zato je prostora za izboljšave in nadgradnje veliko.

Prva in verjetno najpomembnejša je varnost, saj delamo z občutljivimi podatki o uporabnikovi lokaciji in vedenju. Osnovno avtentikacijo smo sicer implementirali za namen delovanja prototipa, vendar bi bilo potrebno dodati še varno povezavo, ki bi jo lahko implementirali z uporabo protkola HTTPS. Rezultate bi lahko izboljšali tudi tako, da bi dodali zaznavanje vozila pred seboj, saj sta hitrost vožnje in odstopanje od omejitve hitrosti drugačna, če nekdo vozi pred nami. Naslednja pomembna izboljšava bi bila uporaba ekspertnega znanja s področja prometa ter psihologije človeka. To znanje bi lahko vključili v samo načrtovanje in razvoj sistema in tako dosegli bolj pametno in učinkovito delo z zbranimi podatki. V implementaciji smo uporabnikove vozne navade predstavili z uporabo razrednih asociativnih pravil, a ne moremo trditi, da je to najboljša možnost. Področje podatkovnega rudarjenja ponuja tudi druge algoritme in pristope in kak drug pristop bi lahko dal boljše rezultate.

Zanimiva bi bila tudi obdelava vseh podatkov v podatkovni bazi, ne le za vsakega uporabnika posebej. Najdene splošne zakonitosti bi lahko na uporabnikovo željo uporabili pri izračunu najboljše poti za nove uporabnike, za katere še nimamo zbranih podatkov. Zadnja nadgradnja, ki jo predlagamo, pa so obogateni zemljevidi. To pomeni, da bi vsak uporabnik imel svojo verzijo digitalnega zemljevida, na katerem bi vsaka cesta vključevala zakonitosti vedenja voznikov in posameznega uporabnika. To bi bila osnova za nadaljnji razvoj sistema.

5.4 Zaključek

V diplomskem delu smo izdelali prototip sistema za optimizacijo poti glede na način uporabnikove vožnje. Protip smo izdelali na podlagi ideje, katere bistvena lastnost je, da smo se usmerili v prilagajanje poti načinu vožnje uporabnika, namesto da bi poskušali spreminjati njegove vozne navade. Te navade smo modelirali v obliki pravil iz zbranih podatkov.

Sistem temelji na arhitekturi odjemalec-strežnik. Na strani odjemalca se nahaja mobilna naprava Android, na kateri teče aplikacija. Ta je namenjena periodičnemu zbiranju podatkov o vožnji, kamor spadajo hitrost, smer, poraba goriva, tip ceste in omejitev hitrosti. Te podatke naprava pridobi iz sprejemnika GPS, digitalnega zemljevida ter vmesnika OBD II, ki je priključen v vozilo. Ti podatki se nato prenesejo na strežnik, kjer se jim dodajo podatki o vremenu in razmerah na cesti na lokaciji, na kateri se vozilo trenutno nahaja. Vsi podatki so shranjeni v dokumentni podatkovni bazi MongoDB. Tako pridobljene podatke smo diskretizirali in obdelali, s čimer smo zmanjšali število atributov, ki smo jih nato uporabili pri iskanju razrednih asociativnih pravil. Pravila so rezultat prvega dela algoritma za klasifikacijo na podlagi asociativnih pravil. V drugem delu algoritma smo generirali klasifikator iz najdenih pravil.

Klasifikator smo uporabili za klasifikacijo trenutnih razmer, za ciljna razreda pa smo uporabili odstopanje od omejitve hitrosti ter porabo goriva. Ciljni razred je odvisen od kriterija, ki ga izbere uporabnik.

Klasifikacijo trenutnih razmer smo uporabili pri iskanju najboljše poti z algoritmom A*, ki smo ga spremenili tako, da namesto fiksne teže poti uporablja težo, ki jo dobimo na osnovi omenjene klasifikacije. Izračunana pot je tudi rezultat, ki se po končanem iskanju s strežnika prenese na mobilno napravo, kjer se izriše na digitalnem zemljevidu.

Pokazali smo, da je tak sistem mogoč in da je z zbiranjem podatkov mogoče izdelati profil uporabnika. Hkrati smo ugotovili, da podatki, iz katerih smo zgradili profil, niso nujno najprimernejši in da je možnosti za izboljšave še veliko. Testi za napoved odstopanja od omejitve hitrosti namreč

niso bili uspešni, saj so pokazali zelo nizko natančnost. Boljše rezultate smo dobili pri napovedi porabe in ugotovili, da se natančnost izboljša s številom zajetih podatkov. Izvorna koda rešitve ja na voljo na spletnem repozitoriju Github na naslovu `https://github.com/RokPajkKosec`.

Pri razvoju smo uporabili znanje z več področij študija na Fakulteti za računalništvo in informatiko in ga uspešno združili v sistem, za katerega verjamemo, da je korak v pravo smer na področju razvoja in optimizacije prometa.

Literatura

[1] Industry Leaders Announce Open Platform for Mobile Devices. Dostopno na:
http://www.openhandsetalliance.com/press_110507.html

[2] Wikipedia: HTC Dream. Dostopno na:
http://en.wikipedia.org/wiki/HTC_Dream

[3] idc.com - Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC. Dostopno na:
http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[4] Uradna spletna stran Android Licence. Dostopno na:
http://source.android.com/source/licenses.html

[5] Uradna spletna stran Android Developer Tools. Dostopno na:
http://developer.android.com/tools/index.html

[6] FRI Android wiki. Dostopno na:
http://android.fri.uni-lj.si/index.php/Platforma

[7] IBM DeveloperWorks - What is Eclipse, and how do I use it? Dostopno na:
http://www.ibm.com/developerworks/opensource/library/os-eclipse/index.html

[8] Uradna spletna stran Android Development Tools - Eclipse vtičnik. Dostopno na:
http://developer.android.com/tools/sdk/eclipse-adt.html

[9] Android Software Development Kit. Dostopno na:
http://developer.android.com/sdk/index.html

[10] Dokumentacija verzij Android API. Dostopno na:
http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

[11] Uradna spletna stran Apache Tomcat. Dostopno na:
http://wiki.apache.org/tomcat/FrontPage

[12] Apache Tomcat 7: More about the Cat. Dostopno na:
http://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_More.html

[13] On-Board Diagnostic II (OBD II) Systems - Fact Sheet. Dostopno na:
http://www.arb.ca.gov/msprog/obdprog/obdfaq.htm

[14] OBD Solutions: What is OBD? Dostono na:
http://www.obdsol.com/articles/on-board-diagnostics/what-is-obd/

[15] Obdtester.com - protokoli OBD II. Dostopno na:
http://www.obdtester.com/obd2_protocols

[16] Uradna spletna stran OBD Console. Dostopno na:
http://obdcon.sourceforge.net/2010/06/obd-ii-pids/

[17] Uradna dokumentacija za modul OBD II ELM327. Dostopno na:
http://www.elmelectronics.com/DSheets/ELM327DSF.pdf

[18] Uradna spletna stran OBDSim. Dostopno na:
http://icculus.org/obdgpslogger/obdsim.html

[19] Uradna dokumentacija OpenStreetMap. Dostopno na:
http://wiki.openstreetmap.org/wiki

[20] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, doktorska disertacija, University of California, 2000

[21] M. Elkstein, Learn REST: A Tutorial, spletni dnevnik. Dostopno na: http://rest.elkstein.org/2008/02/what-is-rest.html

[22] IBM DeveloperWorks - Exploring CouchDB. Dostopno na: http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html

[23] Gigaom.com - 10gen embraces what it created, becomes MongoDB Inc. Dostopno na: http://gigaom.com/2013/08/27/10gen-embraces-what-it-created-becomes-mongodb-inc/

[24] Db-engines.com - DB-Engines Ranking. Dostopno na: http://db-engines.com/en/ranking

[25] Uradna spletna stran MongoDB. Dostopno na: http://www.mongodb.org/

[26] Uradna dokumentacija MongoDB. Dostopno na: http://docs.mongodb.org/manual/

[27] Uradna spletna stran Morphia. Dostopno na: https://github.com/mongodb/morphia/wiki/Motivation

[28] Uradna spletna stran SQLite. Dostopno na: http://www.sqlite.org/

[29] Uradna dokumentacija Weather Underground API. Dostopno na: http://www.wunderground.com/weather/api/d/docs

[30] Uradna dokumentacija API Prometno-informacijskega centra Slovenije. Dostopno na: http://kazipot1.promet.si/kazipot/services/dataexport/

[31] Uradna dokumentacija Google Geocoding API. Dostopno na: https://developers.google.com/maps/documentation/geocoding/

[32] Uradna spletna stran osmdroid. Dostopno na: https://code.google.com/p/osmdroid/

[33] M. A. Quddus, High Integrity Map Matching Algorithms for Advanced Transport Telematics Applications, doktorska disertacija, Imperial College London, 2006

[34] I. Bratko, Prolog Programming for Artificial Intelligence, 3rd edition, Addison-Wesley, 2001

[35] B. Liu, Web Data Mining - Exploring Hyperlinks, Contents, and Usage Data, 2nd edition, Springer, 2011

[36] P. Tan, M. Steinbach, V. Kumar - Introduction to Data Mining, 1st edition, Addison-Wesley, 2005

[37] Google Maps help - Estimated driving and fuel costs. Dostopno na: https://support.google.com/maps/answer/81106?hl=en

[38] M. Končan, Aplikacija za optimizacijo sistema varčne vožnje vozil na motorni pogon, diplomsko delo, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2013

[39] Uradna spletna stran EcoShifter OBD2 Car. Dostopno na: http://ecoshifter.com/

[40] Uradna spletna stran knjižnice Gson. Dostopno na: https://code.google.com/p/google-gson/