RIS - Poročilo

Jan Gulič, Luka Podgoršek, Rok Poje, Sašo Cvitkovič

17. junij 2016

1 Uvod

Pri predmetu Razvoj inteligentnih sistemov (RIS) smo ekipo z imenom **team theta** sestavljali:

- Cvitkovič Sašo
- Gulič Jan
- Podgoršek Luka
- Poje Rok

2 Opis problema

Končna naloga je bila implementacija robotskega taksija. Poligon je predstavljal mesto, v katerem se je robot vozil in izpoljeval naloge. Mesto je bilo sestavljeno iz štirih ulic, štirih hiš in prometnih znakov, katere je robot moral upoštevati med samo vožnjo. Na začetku smo robota postavili v mesto in mu sporočili njegovo lokacijo. Nato je uporabnik preko telefona (uporabljena je bila Android aplikacija) sporočil robotu svoje ime in na kateri ulici se nahaja. Robot se je odpeljal na željeno ulico in poiskal uporabnika, ki ga je poklical. Nato je s potnikom opravil kratek dialog, preko katerega je izvedel željeno destinacijo. Uporabnika je odpeljal na destinacijo, kjer je moral poiskati stavbo. Stavbe so bile predstavljene kot barvni cilindri. Po uspešni dostavi se je robot odpeljal do naslednjega uporabnika. Robot je nalogo opravil, ko je uspešno odpeljal tri osebe na željeno destinacijo.

Slike mesta, obrazev in zgradb najdete v prilogi A.

3 Realizacija zahtev

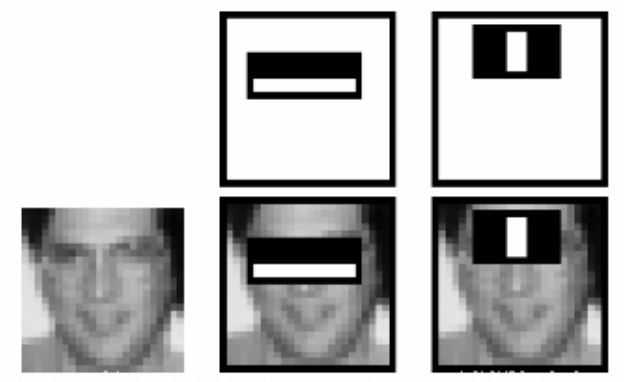
Zahteve, ki so obarvane z zeleno so bile realizirane. Rumeno obarvane zahteve niso bilo v celoti implementirane.

- Learn the appearance of nine faces
- Learn the appearance of five traffic signs
- Learn the appearance of four buildings
- Build the map of the competition area
- Manually mark the streets in the map
- Start at any position
- Travel around the city
- Detect and recognize the traffic signs
- Follow the traffic rules
- Detect and recognize the faces
- Detect and recognize the buildings
- Wait and understand the dispatcher's commands
- Ask the individuals where to take them
- Take the person to the correct building
- Decide whether to take two persons together

4 Metodologija

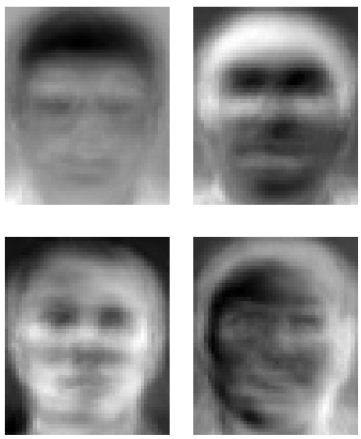
4.1 Detekcija in razpoznava obrazev

Za detekcijo obraza je bil uporabljen **Haarov klasifikator** (ang. haarcascade classifier). Ta klasifikator sta razvila Paul Viola in Michael Jones. Deluje tako, da poišče določene lastnosti na obrazu. Lastnost je predstavljkena kot število, ki se izračuna tako, da odštejemo vsoto belih pikslov od vsote črnih piksov. Algoritem poišče ogromno lastnosti v obrazu, vendar je večina neuporabnih. Zato nad vsemi lastnosti uporabimo algoritem **adaboosting**, ki poišče najbolše lastnosti obraza. Preden lahko uporabimo klasifikator, ga je potrebno natrenirati. Zato potrebujemo ogromno pozitivnih slik (slik obraza) in negativnih slik. Treniranje klasifikatorja poteka tako, da si shrani lastnosti obrazev in lastnosti slik, ki ne predstavljajo obraza.



Slika 1: Iskanje lastnosti na obrazu

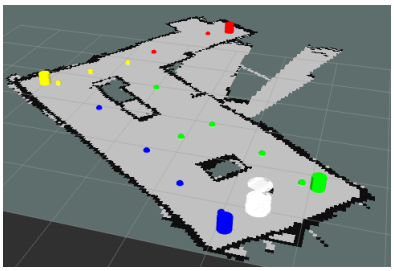
Da je sistem lahko prepoznal obraz, smo potrebovali učno množico obrazev. Za vsak obraz, ki ga je bilo potrebno prepoznati, smo shranili **50 različnih slik**. Te slike smo nato uporabili za izdelavo podatkovne baze **lastnih obrazev** (ang. eigenfaces).



Slika 2: Primer eigenface

4.2 Navigacija

Želeli smo, da robot čim hitreje pride iz točke A v točko B. Zato smo za podatkovno strukturo izbrali graf, saj poznamo algoritme za iskanje najkrajših poti v grafu. Uporabili smo Dijkstrin algoritem, s katerim poiščemo najboljšo pot do željene destinacije.



Slika 3: Mapa

Barvne pike na sliki predstavljajo vozlišča grafa. Barva vozlišča določa ulico na kateri se nahaja.

4.2.1 Upoštevanje prometnih znakov

Prometne znake robot upošteva s pomočjo cen povezav v grafu. Ko robot zazna prometni znak, popravi povezavo, ki vodi do tega oz. mimo tega znaka. Npr. robot vidi znak za prepovedan promet v eno smer. Ceno povezave poveča tako, da te povezave ne bo izbral za pot, saj je ta pot neoptimalna. V primeru, da so vse ostale poti blokirane robot, ne bo upošteval prometnega znaka in prišel bo do svoje destinacije po edini možni poti.

Detekcija znaka za zvočni signal je izjema in ne popravi cene povezave. Pred tem znakom robot zatrobi.

4.3 Dialog človek računalnik

Robot je mogel poslušati in se pogovarjati s človekom. Za to smo uporabili mobilno aplikacijo Voice Commander, katera je bila odgovorna za prepoznavanje govora, node SpeechProxy za povezavo med telefonom in robotom, ter node SoundPlay za sintezo govora. V dialog smo implementirali še Levenshteinovo razdaljo za boljšo natančnost dialoga.

Levenshteinovo razdalja med dvema nizoma predstavlja minimalno število operacij, ki jih potrebujemo za preoblikovanje enaga niza v drugega.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Slika 4: Formula Levenshteinove razdalje

5 Implementacija in integracija

5.1 Glavni pojmi sistema ROS

- **ROS** (Robot Operation System) je odprtokodni operacijski sistem za robotiko. Vsebuje mnogo knjižnic in orodij za razvoj robotskih sistemov.
- **Node** je samostojen proces, ki teče vzporedno z drugimi procesi.
- **Medprocesna komunikacija** je ena izmed glavnih komponent ROS-a. Komunikacija poteka preko **tem** na katere se vežejo poslušalci in pošiljatelji.
- **Rviz** je orodje s katerim lahko vizualiziramo delovanje robota, mapo po kateri se premika, detektirane objekte, obraze, itd.

10 slika celotnega sistema

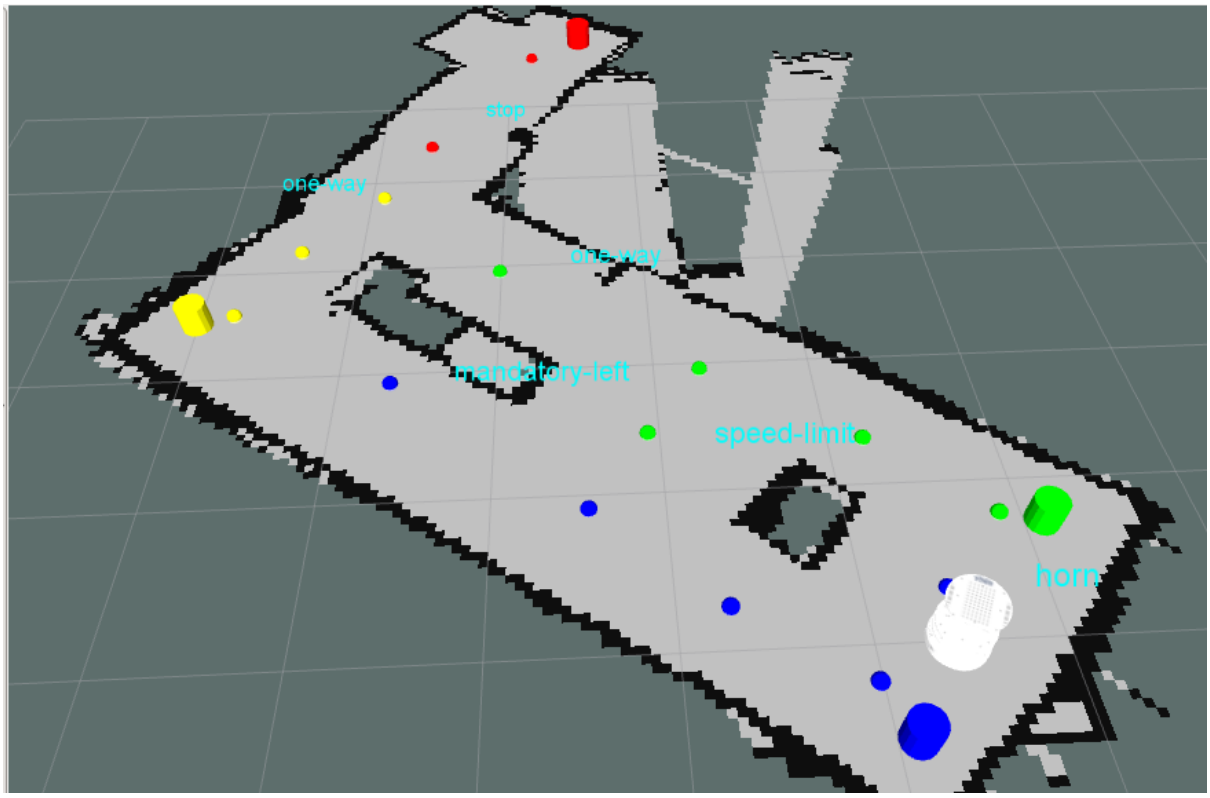
5.2 Integracija celotnega sistema

Naš sistem je sestavljen iz glavnega procesa in pomožnih procesov (glej sliko zgoraj). V glavnem procesu smo implementirali:

- Navigacijo
- Premik do koordinat
- Poslušalce
- Pošiljatelje
- Vizualizacijo delovanja

Implementacije celotnega sistema smo se lotili postopoma. Najprej smo naredili mapo celotnega mesta z uporabo Rviz-a in Kinecta ter jo shranili na disk. Sprva smo v kodo zapisali koordinate oseb, zgradb, in prometnih znakov, saj smo potrebovali testne podatke. Nato smo implementirali navigacijo in določili vozlišča grafa, ter označili posamezne ulice. Iz prejšnje naloge smo vzeli kodo za transformacijo koordinat med koordinatnimi sistemi in za premik do koordinat, ter jo dodali v nov sistem. Implementirali smo pošiljanje ti. markerjev orodju Rviz, ki je te markerje vizualiziralo. Markerji predstavljajo lokacije oseb, zgradb, prometnih znakov in vozlišč grafa. Vidna je tudi lokacija robota.

Tako smo si pripravili osnovo samega sistema, katero je bilo možno nadgrajevati. Nato smo implementirali ostale procese, ki so skrbeli za detekcijo razpoznavo. Implementacija teh procesov je podrobno opisana v spodnjih poglavjih.



Slika 5: Mapa z znaki in zgradbami

5.3 Detekcija in razpoznava obrazev

Za detekcijo obrazev smo uporabili Vicos-ovdetektor obrazev. Detektor je uporabljen kot **ROS node**. Registriran je na temo **detections**, kamor pošlje sporočilo v primeru detekcije obraza. Detektor vrača tudi neveljavne detekcije, zato smo v poslušalcu, ki teče v glavnem procesu implementirali filter neveljavnih detekcij.

Filter najprej omeji detekcije po višini. Imeli smo problem, saj je naš robot zaznal obraze ljudi, ki so stali ob poligonu. Ti obrazi so v danem scenariju neveljavni, povzročili pa so to, da je robot zaznal obraz na neveljavni lokaciji. V prejšnji nalogi se je robot moral zapeljati do zaznanega obraza, zato so neveljavni obrazi predstavljali problem, saj se robot ni znal zapeljati do njih in se je posledično ustavil preden je končal svojo nalogo. To smo rešili tako, da smo mu omejili vidno polje. Vse detekcije, ki so bile višje od stene poligona smo zavrgli.

Kljub omejitvi "vidnega polja", je detektor vračal neveljavne detekcije. Npr. ročaje omare je detektor zaznal kot obraz. Zato smo v filter dodali, da more detektor zaznati obraz vsaj 5-krat v pol meterskem radiju okoli točke, kjer je bil zaznan prvi obraz. Ko detektor zazna obraz vsaj 5-krat, filter pošlje sporočilo **procesu za razpoznavo obrazev**.

Tako kot detektor obrazev, je tudi razpoznavalec obrazev vračal napačne rešitve. To smo rešili, z dodatnim filtrom v razpoznavalcu obrazev. Razpoznavalec najprej izračuna podobnost med obrazem, ki ga vidi preko kamere in med naučenimi obrazi. V primeru, da je podobnost večja ali enaka **95%**, poskusi razpoznati obraz. Zaradi robustnosti in izničitve napačnih razpo-

znav, mora razpoznavalec razpoznati obraz vsaj 10-krat. Nato pogleda kateri obraz izmed vseh obrazev je zaznal največkrat in tega označi kot veljavnega na dani lokaciji. Lokacijo veljavnega obraza pošlje orodju Rviz, ki izriše ime zaznane osebe na mapi. Lokacijo obraza prejme tudi glavni proces, ki si shrani lokacijo zaznanega obraza in jo nato uporabi pri planiranju.

Uporabljeni razpoznavalec obrazev je na voljo, kot ROS orodje. Izvorno kodo smo predelali za naše potrebe. Implementirali smo zgoraj opisani filter, popravili, da sliko prejme preko kamere in spremenili temo na kateri je poslušal. Dodali smo tudi pošiljatelja, ki je pošiljal razpoznane obraze orodju Rviz za vizualizacijo. Preden je pošiljatel poslal koordinate obraza, smo jih transformirali v koordinatni sistem mape.

5.4 Detekcija in razpoznava cilindrov

Detekcija in razpoznava cilindrov ni bila dokončno implementirana. V glavnem procesu smo ročno vnesli koordinate zgradb in jih shranili v seznam. Zaradi tega je robot lahko dostavil osebo do željene zgradbe, kljub temu, da je ni "videl". Imeli smo pripravljenega poslušalca, ki je poslušal na temi za cilindre, vendar smo to kodo zakomentirali. V primeru, da bi uspešno zaključili implementacijo detekcije cilindrov, bi poslali informacijo z koordinatami zgradbe glavnemu procesu. Glavni proces, bi nato dodal zgradbo v seznam zgradb. Seznam zgradb bi bil ob zagonu glavnega procesa prazen.

Za detekcijo cilindrov smo uporabili detektor iz knjižnice PCL. Detektor nam je vračal preveč neveljavnih detekcij, česar nismo znali odpraviti s spremembo parametrov samega orodja. V primeru, da bi detekcijo izboljšali, bi dodali še barvno preverjanje. Barvno preverjanje bi implementirali z uporabo histogramov. Sliko cilindra bi razbili na več manjših histogramov velikosti N * N in jih združili v en sam histogram. Ta histogram, bi nato uporabili za barvno primerjavo. Poleg tega bi dodali pošiljatelja, ki bi imel enako vlogo kot pošiljatel pri razpoznavi obrazev.

5.5 Dialog človek robot

Na začetku programa nas robot s pomočjo SoundPlay-a lepo pozdravi, nato pa čaka na ukaz. S pomočjo Android aplikacije Voice Commander, ki prepoznava govor ustvarimo ukaz in ga preko SpeechProxy-ja pošljemo robotu. Ta s pomočjo Levenshteinove razdalje preveri ali gre za ustrezen ukaz. Na podlagi danega ukaza in ≫situacije≪ v kateri se nahaja se nato odloči kaj bo naredil. Robot pozna več situacij. V vsaki je dialog malce drugačen. Situacija iskanja osebe v dani ulici: Robot posluša ukaze in poizkuša s pomočjo Levenshteinove razdalje iz njih dobiti ime ulice, ter ime osebe, katero more poiskati. Situacija določanja cilja: Ko robot pride do stranke jo vpraša kam želi iti, ta pa more nato povedati barvo cilindra, do katerega želi biti prenešena. Robot nato ponovi ukaz, ki ga je prejel od stranke, da ga lahko ta potrdi ali ovrže. Situacija potrditve cilja: Robot posluša, ali je ukaz, ki ga je prejel pravilen ali napačen. Če prejme odgovor ≫yes≪, se odpravi na cilj, v primeru pa da stranka ovrže uakz in odgovori z ≫no≪, pridemo spet v situacijo določanja cilja ter se dialog določanja cilja ponovi. Ko robot pripelje stranko do določenega cilja ga lepo pozdravi in se postavi nazaj v začetno situacijo.

5.6 Navigacija in planiranje

Za navigacijo v najprej znani mapi smo uporabili **graf** in algoritem **Dijkstra**. Vozlišča v grafu smo definirali v sami kodi. Vsako vozlišče vsebuje koordinate kjer se nahaja. Cene povezav med vozlišči so na začetku prazne in se posodabljajo med samim delovanjem robota. Npr. robot razpozna znak za prepovedan promet v eno smer in popravi povezavo na kateri se nahaja znak.

Dijkstra je algoritem za iskanje najkrajših poti v grafu. Preden se robot premakne si izdela načrt, po katerih vozliščih se bo peljal. Glavni proces požene dijkstrin algoritem in izračuna najkrajšo pot do željene destinacije. Robot prejme željeno destinacijo preko glasovnih ukazov v (5.5) dialogu. V primeru, da robot ne ve kje se nahaja oseba ali zgradba, to osebo oz. zgradbo poišče na podani ulici. Ulice smo tako kot vozlišča v grafu, definirali v sami kodi.

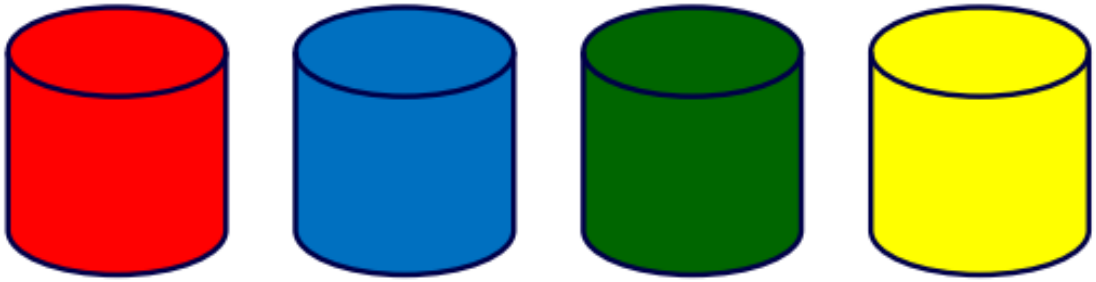
6 Delovanje celotnega sistema na tekmovanju

Naš sistem je na končnem tekmovanju zbral 24 točk. Pravilno je dostavil 2 osebi do dveh zgradb. Zataknil se je pri pobiranju tretje osebe s čimer je končal svoj poiskus.

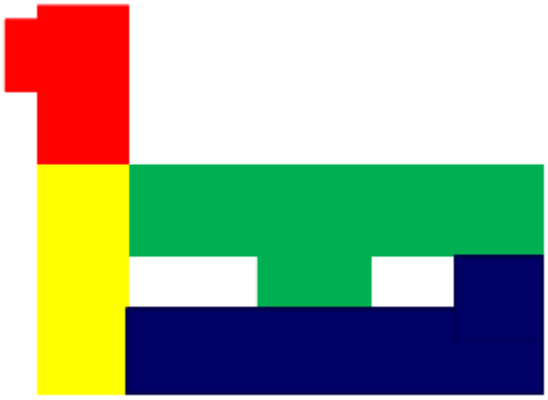
Priloge

A Slike

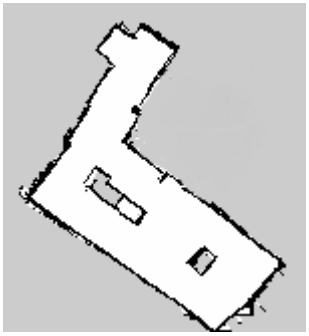
Slika 6: Veljavni prometni znaki



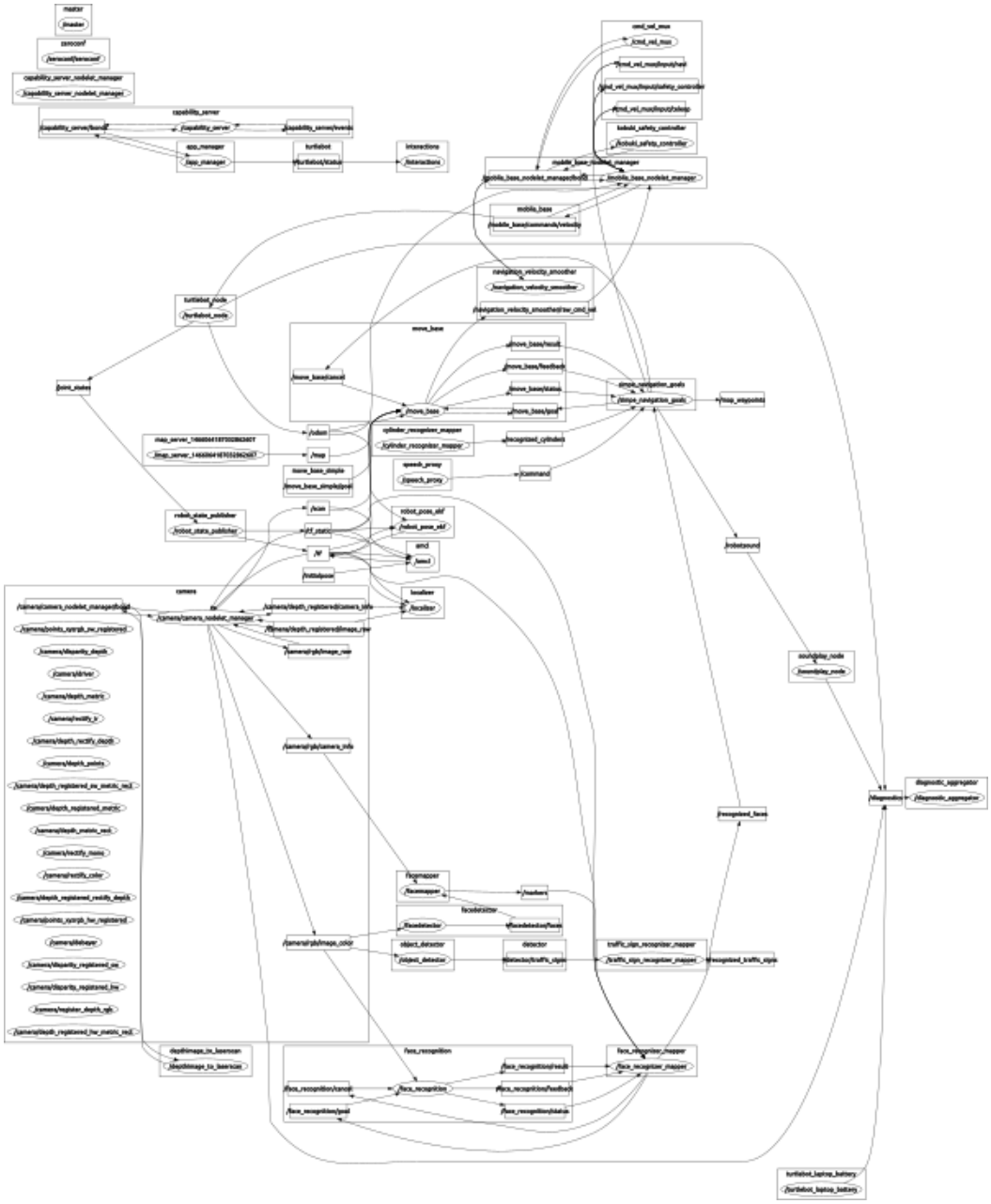
Slika 7: Izgled zgradb



Slika 8: Skica mesta



Slika 9: Mapa po kateri se je navigiral robot



Slika 10: Graf celotnega sistema