Razvoj inteligentnih sistemov: Končno poročilo skupine ε

Aleš Kert, Mark Žakelj, Matevž Eržen

Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

6. 6. 2020

1 Uvod

Cilj projekta je bil razvoj mobilnega robota, ki temelji na odprtokodni platformi TurtleBot. Robot bo zmožen opravljati različne naloge iz predhodno znanega scenarija.

Zaradi pandemije in posledične omejitve gibanja, robotove funkcionalnosti niso bile razvite na strojni opremi, v laboratoriju fakultete, temveč v simuliranem okolju Gazebo in vizualizatorju Rviz.

Končna naloga "*Matchmaker*" je definirana kot sledeči scenarij;

Robot se giblje po navideznem mestu, ki vsebuje šest slik obrazov - 5 ženskih in 1 moškega, štiri "*vodnjake želja*", ki so ponazorjeni z valji različnih barv in štiri obroče enakih barv kot valji.

Ultimativni cilj je *Gargamelu* poiskati partnerico, ki ustreza njegovim željam. Za dosego tega cilja moramo najprej poiskati (sliko) *Gargamela* in ga povprašati o njegovih preferencah - dolžini in barvnemu tonu las. Temu sledi iskanje primernega dekleta - v mestu je potrebno poiskati dekle, ki ustreza njegovim željam.

Če najdeno dekle *Gargamela* zavrne iščemo naprej, v nasprotnem primeru pa se vrnemo do *Gargamela* in ga povprašamo kakšna se mu zdi izbranka. Če privoli, ljubezen zapečatimo s pomočjo vodnjaka želja in prstana.

Dekle smo že ob prejšnji interakciji povprašali po najljubši barvi. Sedaj poiščemo vodnjak želja v enaki barvi. Z robotsko roko vanj navidezno vržemo kovanec. Nato poiščemo še prstan (obroč) enake barve, ga z robotsko roko poberemo in prinesemo nazaj do dekleta. Na tej točki se dekle lahko odloči če bo sprejela *Gargamelovo* ženitveno ponudbo ali ga zavrnila. Če privoli, je naša naloga uspešno končana, če ga zavrne, pa se celoten postopek iskanja dekleta ponovi.

2 Uporabljene metode

2.1 Zaznava & prepoznava obrazov

2.1.1 Detekcija obrazov

Za prepoznavo obrazov smo uporabili globoko nevronsko mrežo (*dnn*), implementirano znotraj *OpenCV*. Učnega modela nismo trenirali sami, temveč smo uporabili že prej narejeni model `res10_300x300_ssd_iter_140000.caffemodel`, ki je dosegljiv na *GitHub* repozitoriju *OpenCV*.

Pomemben del uporabe *dnn* je predprocesiranje vhodnih podatkov. Vsem vhodnim slikam smo spremenili velikost na 300x300 točk. Parameter zaupanja za prepoznavo obraza pa po testiranju nastavili na 70%, saj je to predstavljalo optimalno razmerje med minimalnim številom lažnih in največjim številom pravilnih zaznav.

2.1.2 Prepoznavanje obrazov

Obraze, ki smo jih zaznali v prejšnji točki, nato potrebujemo še prepoznati. To smo naredili z uporabo modela SVM ter vložitvijo obraza v že naučeno globoko konvolucijsko nevronsko mrežo *FaceNet*, specifično `facenet_keras`. *FaceNet* vzame sliko obraza ter vrne vektor števil, ki opisuje lastnosti obraza. SVM nato ta vektor klasificira kot eno izmed oseb, glede na vrednosti v vektorju. Uporabili smo linearni model SVM, ki v učni fazi išče linearne enačbe, ki razdelijo prostor obrazov v klasifikacijska območja. S pomočjo slednjih nato modela poišče v katero območje spada ravnokar zaznan obraz, ter ga klasificira glede na enačbe modela.

2.1.3 Prepoznavanje obraznih lastnosti

Prepoznave dolžine in barve las smo se lotili na podoben način, le da smo za prepoznavo in treniranje uporabili povečane obraze, saj naš algoritem za zaznavo obrazov detektira le obraz brez las. Posledično potrebujemo povečati območje zaznanega obraza ("*bounding-box*"), da znotraj obrezane slike detektiranega obraza dobimo tudi lase.

2.2 Zaznava objektov

2.2.1 Zaznava obročev

Za detekcijo obročev smo uporabili *Houghovo transformacijo* za zaznavo krogov, ki je implementirana v knjižnici *OpenCV*. *Houghova transformacija* deluje na podlagi iskanja robov v dani sliki. Le te išče z uporabo konvolucije nad sliko, predstavljeno kot matrika, ter S pomočjo dobljene matrike robov nato išče določene oblike v sliki.

V našem primeru iščemo kroge. To naredimo s sestavo akumulacijskega prostora s tremi dimenzijami, ki predstavljajo: prvo in drugo koordinato središča kroga, ter polmer kroga. V ta prostor nato za vsako robno točko označimo vse možne krožnice, ki potekajo skozi to točko. Ko to naredimo za vse točke, v akumulacijskemu prostoru poiščemo vse lokalne maksimume. Slednje vrnemo kot rezultat operacije oz. kot zaznane kroge.

Rezultat transformacije smo nato še dodatno obdelali, da smo prišli do bolj robustne rešitve. Opis obdelave je vključen v poglavju implementacije.

2.2.2 Zaznava cilindrov

Za zaznavo cilindrov smo nad oblakom točk uporabili iterativno metodo *RANSAC*, ki je implementirana v knjižnici *PCL*. *RANSAC* deluje na principu naključnega vzorčenja, ki v množici danih točk poišče iskano obliko. Algoritem iterativno izbira naključne točke iz množice toliko časa, dokler izbrane točke ne opišejo dovolj velike množice ne-izstopajočih točk ali pa ne dosežemo maksimalnega števila iteracij. Rezultat tega procesa je v našem primeru cilinder, ki mu uspe pokriti največ ne-izstopajočih točk.

2.3 Prepoznavanje barv

Za identifikacijo prevladujočih barv smo uporabili metodo K-means. Barvne točke slike v 3 dimenzionalnem prostoru gručimo v 4 skupine, kjer le te centriramo na podlagi R, G in B komponent barvnega zapisa. Za implementacijo smo uporabili `Python scikit-learn`, specifično model `sklearn.cluster.MiniBatchKMeans`, saj smo med testiranjem ugotovili da osnovni algoritem K-means, implementiran v `sklearn.cluster.KMeans`, deluje prepočasi.

Algoritem *Mini Batch K-means* je hitrejši, saj k rešitvi konvergira hitreje - za ceno malo slabše natančnosti. Algoritem v osnovi deluje enako kot običajen K-means, s to razliko, da namesto na vseh vhodnih podatkih, dejansko operira le na delu teh. V vsaki iteraciji iz vhodnih podatkov vzorči vrednosti in s tem zmanjša časovno zahtevnost procesiranja.

Koordinate (parametri R, G, B) končnih centroidov, dobljenih po izvedbi algoritma, predstavljajo prevladujoče barve. Le te nato klasificiramo v iskane razrede (rdeča, modra, zelena, rumena...) s pomočjo HSV zapisa barv. Barvni ton (komponenta H) se da na osnovi splošno znanih pragov preprosto razdeliti na več sekcij, ki ustrezajo našim barvam.

2.4 Prepoznavanje govora

Za prepoznavanje govora smo uporabili *Google Speech Recognition*, ki je nevronska mreža, namenjena prepoznavanju govora. Mreži kot vhod pošljemo posneti govor ter kot odgovor dobimo besedilo, prepoznano iz posnetka. To besedilo pretvorimo v fonetični zapis, o katerem povemo več pri implementaciji.

Za fonetični zapis uporabimo fonetični algoritem *Metaphone*, ki preslika črkovni zapis angleškega jezika v fonetični zapis s pomočjo devetnajstih pravil za izgovorjavo angleškega jezika.

2.5 Navigacija

Avtonomno navigacijo čez poligon smo implementirali s segmentacijo podanega zemljevida poligona (ki ga predhodno "*napihnemo*" - da dobimo točke na zemljevidu, katere robot dejansko lahko doseže) na enako velike dele. Za le te izračunamo centroide, ki ustrezajo točkam, ki jih bo robot obiskal v fazi preiskovanja poligona.

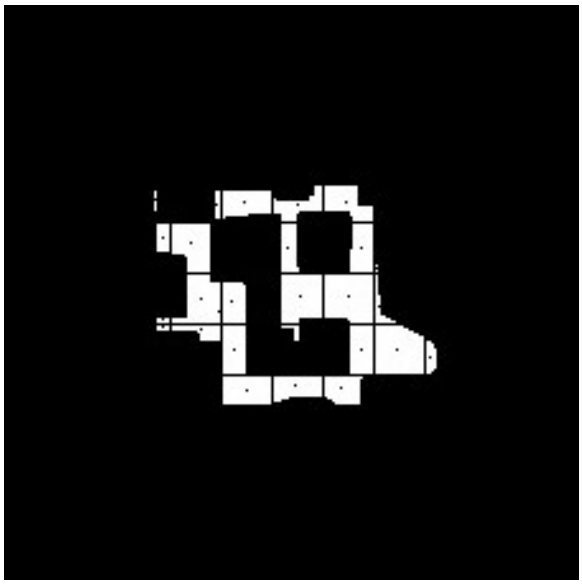


Figure 1: Pridobitev točk za preiskovanje prostora

3 Implementacija in integracija

3.1 Zaznava & prepoznava obrazov

3.1.1 Detekcija obrazov

Pri zaznavi obrazov s kamere pridobimo RGB sliko, nad katero poženemo algoritem, ki smo ga opisali v metodah. Ta nam vrne pozicijo zaznanega obraza na sliki, ter njegov mejni kvadrat. Kot zaznan obraz, ki ga uporabimo v naslednjih korakih, uporabimo ta izrez slike.

3.1.2 Treniranje prepoznavnega modela

Pri treniranju SVM modela za prepoznavo obrazov nam je bilo danih 21 različnih obrazov, katere smo posneli iz različnih zornih kotov, da smo naredili učno množico za naš model. Vsak obraz smo fotografirali iz okoli 8 zornih kotov. Na teh slikah smo nato izvedli zgoraj opisani algoritem za zaznavanje obrazov, da smo dobili izrez obrazov. Te slike smo nato vložili v vektorje z mrežo *FaceNet* in s to učno množico trenirali naš SVM model za klasifikacijo obrazov s prečnim preverjanjem.

Za prepoznavanje obraznih lastnosti smo na tej točki natrenirali 2 modela, enega za dolžino las in enega za barvo.

3.1.3 Prepoznavanje obrazov

Iz prejšnjega koraka dobimo obrezano sliko obraza, katero vložimo v vektor z mrežo *FaceNet* ter pridobljeni vektor klasificiramo z uporabo našega SVM modela. Za potrebe poznejše obravnave, vrnjeno identiteto shranimo predstavljeno kot indeks med 0 in 20.

3.1.4 Prepoznavanje obraznih lastnosti

Za prepoznavo obraznih lastnosti prav tako vzamemo izreze obrazov, pridobljene pri detekciji obrazov. Za razliko od prepoznave obrazov, v tem primeru slike "*povečamo*" za 25%, da so v sliki vključeni tudi lasje posameznika. Dobljeno povečano sliko vložimo, ter jo podamo v oba prepoznavna modela, ki nam vrneta lastnosti las osebe.

3.2 Zaznava objektov

3.2.1 Zaznava obročev

Pri implementaciji zaznave obročev nad globinsko sliko, prijeto od kamere, poženemo *Houghovo transformacijo* za zaznavo krogov. Zaradi globinske slike, so narisani obroči na steni že takoj ignorirani. Transformacija nam vrne seznam potencialnih obročev, ki jih je potrebno dodatno obdelati, saj se v okolju lahko pojavijo tudi krogle, ki so zaznane kot krogi. Ker krogle nimajo luknje v sredini, še dodatno preverimo, da je znotraj vsakega kroga luknja.

Za ta del smo vzeli zaznano krožnico iz prejšnjega koraka, ter na njen pogledali kakšna je največja globinska vrednost na krožnici in pridobljeno vrednost primerjali z globinsko vrednostjo v središču. Če je globinska vrednost v središču občutno drugačna, to pomeni, da je na sredini našega zaznanega objekta luknja, in zabeležimo njegovo lokacijo kot nov obroč.

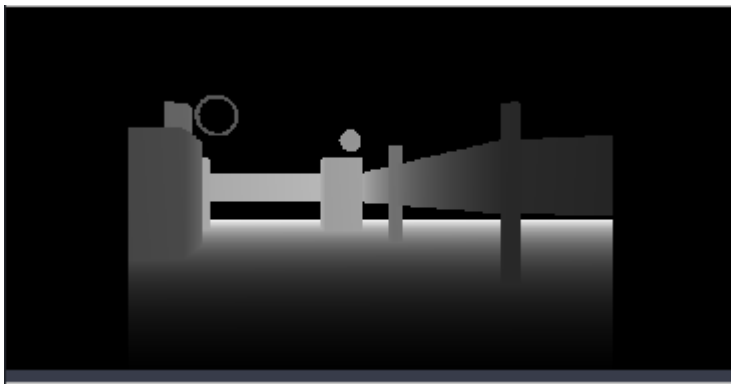


Figure 2: Razlika v globinski sliki kroga ter krogle

3.2.2 Zaznava cilindrov

Za zaznavo cilindrov najprej vzamemo oblak točk, ki go dobimo iz globinske kamere robota. Oblak predstavi okolico, ki jo vidi robot, ponazorjeno s točkami v prostoru. Dobljene točke vsebujejo še veliko drugih objektov poleg cilindrov, zato je treba pred nadaljnjo uporabo to množico očistiti. To naredimo z metodo *RANSAC*, s katero poiščemo vse večje ravnine v oblaku, ter jih odstranimo. S tem odstranimo približno do 70% točk, ki predstavljajo tla ter stene, kar naredi naše iskanje cilindrov bolj zanesljivo in manj časovno zahtevno. Na obdelani množici točk nato ponovno uporabimo metodo *RANSAC* za iskanje cilindrov, ki nam vrne pozicije cilindrov, ki jih robot v tistem trenutku lahko vidi.

3.3 Prepoznavanje barv

Pri prepoznavi barv nismo predpostavljali da dobimo za vhod idealno obrezano sliko barvnega objekta ali vzorec barvnih toč z njegove površine. Funkcijo smo naredili dovolj robustno, da lahko v našem okolju (ki ima določene predpostavke - tla, stene itd. so v sivinskih tonih) prepozna barvo objekta tudi če funkciji podamo sliko, ki vsebuje tudi velik del okolice, pomembno je le, da je naš opazovani objekt večji (zaseda večjo površino na sliki) od drugih (posebnih, barvnih) objektov, ki se lahko pojavljajo na sliki v ozadju.

Vhodno sliko preoblikujemo na velikost 128x128 točk in jo po potrebi pretvorimo v RGB zapis. Njene slikovne pike nato s pomočjo algoritma K-means gručimo v 4 skupine. S tem v središčih centroidov pridobimo štiri prevladujoče barve na sliki. Za štiri gruče smo se odločili ker na tak način v našem okolju zagotovimo, da bo vsaj ena od gruč centrirana na (prevladujoči) barvi opazovanega objekta. Okolica je na sliki črno-sivo-bele barve, kar se v praksi po poteku gručenja pokaže kot tri gruči; bela, siva in črna. Četrta gruča je centrirana na barvi našega objekta. Testiranje z večjim številom gruč v našem primeru ni prineslo večje natančnosti ali robustnosti, le daljši čas izvajanja.

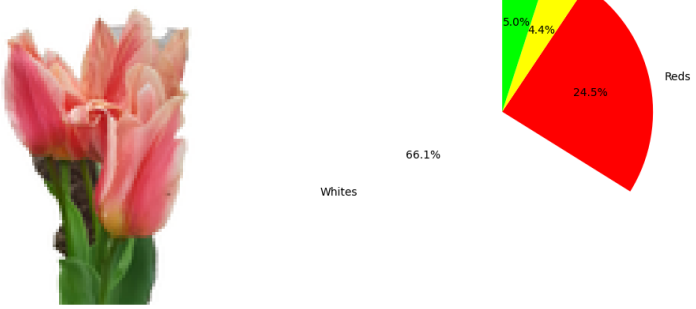


Figure 3: Klasifikacija prevladujočih barv

Po izvedbi gručenja, dobljene barve pretvorimo v HSV zapis in s splošnimi, preddoločenimi mejami (za nasičenost in svetlost) določimo za kakšno barvo gre. V našem primeru se osredotočamo na rdečo, rumeno, zeleno in modro barvo, ter barve okolja (črno, belo in sivo). Razultate, ki ustrezajo barvam okolja ignoriramo, prevladujočo barvo (iz seta rdeče, modre, rumene in zelene) pa vrnemo kot zaznano barvo.

3.4 Prepoznavanje govora

Prepoznavanje govora uporabljamo v fazi iskanja partnerke, kjer sprašujemo osebe po njihovih preferencah.

Naš algoritem za generacijo dialoga deluje tako, da najprej robot vpraša uporabnika po tistem podatku, ki ga trenutno potrebuje (najljubša barva, preference las) in nato počaka nekaj sekund, da dobi odgovor. le tega posnamemo in pošljemo v oblak, Googlovem algoritmu za prepoznavo, ki vrne govor v obliki besedila.

To besedilo nato razčlenimo na besede, ter primerjamo fonetični zapis vsake besede s fonetičnim zapisom vsake ključne besede(*dark, long, red, blue, yes,...*), da zaznamo morebitne napačne zaznave in če se ujemata zapisa, zabeležimo, da je bila izrečena ključna beseda. Ko pregledamo vse besede v besedilu, pogledamo še, če so bili izraženi vsi podatki, po katerih je vprašal robot, da preverimo če je potrebno uporabnika še enkrat vprašati po teh podatkih. Če so bili vsi podatki izrečeni, na koncu uporabnika vprašamo še za potrdilo, če je robot slišal prave podatke. Uporabnik odgovori z da ali ne.

```
Recording input
Finished recording
Converting audio transcripts into text ...
I like long dark hair
[]
['long']
['dark']
Color hairstyle: long dark
```

Figure 4: Primer procesiranja stavka

3.5 Navigacija

3.5.1 Spoznavanje okolja

Pri spoznavanju okolja, zaznavi objektov in obrazov najprej uporabimo metodo, ki smo jo opisali v prejšnjem poglavju, da dobimo točke ki jih bo robot obiskal za potrebe odkrivanju okolja. Iz izračunanih točk sestavimo seznam pozicij, ki jih bomo obiskali na sledeč način: najprej izbere točko, ki je trenutno najbližja robotu, ter nanjo postavimo 4 pozicije z različnimi orientacijami - kot smeri neba. Za naslednjo točko potem vedno izbere tisto točko, ki je najbližja prejšnji. Ko je točka obiskana, se izbriše iz seznama.

Med tem ko se robot premika med točkami, stalno pregleduje prostor za cilindre, obroče ter obraze. V primeru slednjih se jim, ko jih zazna, približa in izvede prepoznavo obraza ter njegove lastnosti. V primeru obroča ali cilindra pa izvede detekcijo barve.

3.5.2 Iskanje partnerke

Ko robot pregleda celoten prostor, se začne glavni del scenarija "*Matchmaker*", kjer iščemo nevesto za *Gargamela*. Najprej izmed prepoznanih obrazov poiščemo pozicijo, na kateri smo zaznali *Gargamela* ter ga vprašamo po njegovih preferencah. Ko *Gargamel* odgovori, robot poišče lokacijo ženske, ki ustreza njegovim preferencam, ter jo vpraša za njeno najljubšo barvo. Ko dobi odgovor, gre nazaj do *Gargamela*, da ga vpraša, če mu je dekle všeč. Če odgovor pritrdilno, gre robot do cilindra njene najljubše barve in vanj spusti kovanec. Nato nadaljuje do obroča ter ga (navidezno) "*pobere*". Na koncu vpraša izbranko, če bi se poročila z Gargamelom. Če je odgovor da, se igra konča, drugače poiščemo novo žensko.

3.6 Integracija celotnega sistema

Pri integraciji celotnega sistema v sklopu končne naloge nismo imeli večjih težav. Večji podvig nam je pred drugim zagovorom predstavljala integracija približevanja, saj je sama delovala kot bi morala, znotraj robota kot celote pa je povzročala težave z navigacijo in detekcijo predmetov. Težave so bile tudi posledica slabšega razumevanja jezika C++, v katerem so bile napisane komponente za detekcijo predmetov v 3D prostoru. Vsa ostala koda je bila napisana v Pythonu, ki nam je bolj poznan - posledično tudi lažje odkrivanje napak.

4 Rezultati

4.1 Zaznava & prepoznava obrazov

4.1.1 Detekcija obrazov

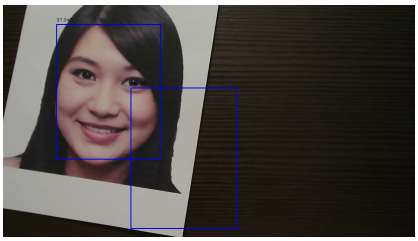
Pri detekciji obrazov smo dosegli najboljše rezultate na razdalji do 2.5m in do kota 45°. Tu, je bila detekcija skoraj 100% pravilna, z izjemo določenih obrazov temnopoltih ljudi, ki jih občasno nismo uspeli zaznati.

Pod večjimi koti, med 45°in 60°, je razdalja na kateri smo lahko z veliko gotovostjo zaznali obraze padla le še na okrog 1.5m, nad kotom 60°, pa je bila detekcija skoraj neuporabna, saj je bila minimalno boljša od naključne napovedi.

Hkrati smo opazili, da večja resolucija zajete slike omogoča boljšo detekcijo na daljših razdaljah, kar je logično, a hkrati lahko povzroča lažne zaznave na manjših razdaljah.

(a) Uspešna detekcija

(b) Neuspešna detekcija



(c) Lažna detekcija

Figure 5: Detekcija obraza

4.1.2 Prepoznavanje obrazov

Naš model za prepoznavanje obrazov ima v vseh primerih 100% natančnost pri klasifikaciji. Pri testiranju in implementaciji nismo imeli omembe vrednih težav.

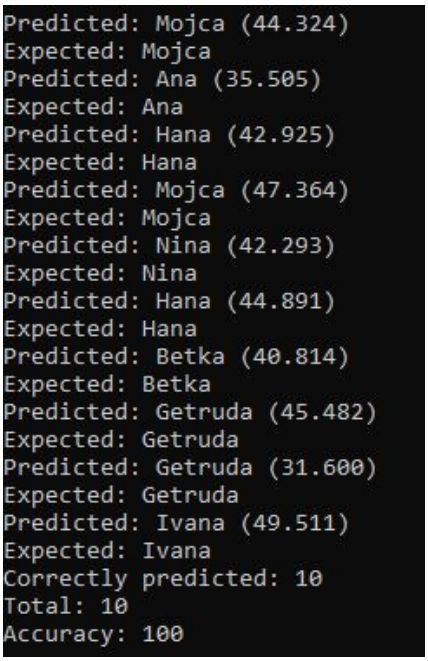


Figure 6: Klasifikacija 10 naključnih obrazov

4.1.3 Prepoznavanje obraznih lastnosti

Zgoraj opisana implementacija prepoznave lastnosti, nam je pri predpostavki zaprtega sveta nudila 90-95% natančnost pri zaznavi, pri odpravi le te pa je natančnost našega modela padla na občutno nižje 60-70%. Glede na potrebe našega projekta smo se zato odločili, da bomo obdržali predpostavko in v končnem robotu uporabili prvi model.

Spodaj sta matriki zamenjav za model, ki predpostavlja zaprti svet.

		Napovedana barva	
		Temni	Svetli
Dejanska barva	Temni	1.00	0.00
	Svetli	0.6	0.94

		Napovedana dolžina	
		Dolgi	Kratki
Dejanska dolžina	Dolgi	0.95	0.05
	Kratki	0.06	0.94

4.2 Zaznava objektov

Naš robot na poligonu uspešno zazna vse objekte in se uspešno izogne lažnim pozitivnim zaznavam, tako pri obročih, kot pri cilindrih. Vsako zaznavo objekta tudi shrani v seznam, na podlagi katerega izračuna center objekta z povprečjem 10% najbolj centralnih točk. Pozicioniranje objektov je torej tudi v primeru slabe zaznave robustno.

4.3 Prepoznavanje barv

Prepoznavanje, ob upoštevanju predpostavljenih omejitev okolja, je imelo 100% natančnost. Napake so se pojavljale le v primeru *ekstremnih* vhodov, pri katerih zaradi osvetlitve prihaja do distorzije barve (npr. bela kocka osvetljena z rdečo lučjo). Algoritem je na podlagi vrednosti slikovnih pik prepoznal *pravilno* barvo (saj so pike same po sebi resnično npr. rdeče barve), a hkrati na podlagi našega znanja vemo, da odgovor dejansko ni pravilen. Ta problem bi verjetno lahko rešili s kompleksno nevronsko mrežo, a v našem primeru ni potrebe po takšni natančnosti in kompleksnosti.

Hkrati je trenutni algoritem dovolj hiter in nezahteven, da v tem smislu ne vpliva na delovanje celotnega sistema.

4.4 Prepoznavanje govora

Pri dialogu smo se poskušali zaščititi pred pogostimi napakami, ki so posledica nejasnega govora - če uporabnik izreče napačne podatke ali da naš algoritem prepozna neko besedo kot njeno istozvočnico (npr. *red* in *rat*). Po rezultatih sodeče je bilo to dovolj za robustno delovanje algoritma. S tem smo se uspešno izognili napačnim podatkom.

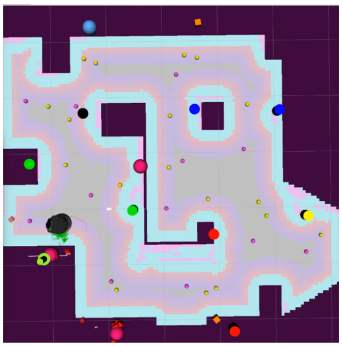
4.5 Navigacija

Algoritem za odkrivanje objektov na poligonu deluje zanesljivo ter v enem obhodu poligona odkrije vse iskane objekte. To načeloma naredi v zmernem času, vendar je v vizualizirani simulaciji zaradi preobremenjenosti računalnika čas občutno daljši. Rezultati se nam zdijo zadovoljivi in bi v realnem svetu težko dosegli drastično boljše rezultate.

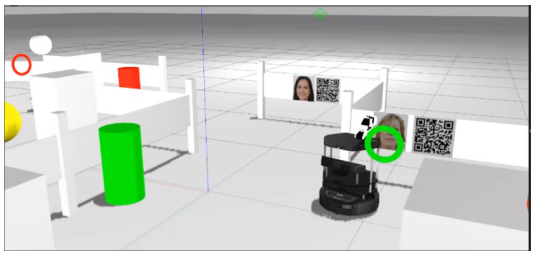
4.6 Celoten sistem

Celoten sistem je na koncu deloval zelo robustno in v realnem času opravi celoten proces v 4-6 minutah, vendar je na predstavitvi zaradi pomanjkanja dovolj močne strojne opreme in preobremenitve računalnika proces trajal občutno več časa.

Kljub temu, projekt deluje brez večjih težav.



(a) Pogled na izvajanje od zgoraj



(b) Pogled znotraj Gazebo

5 Razdelitev dela

5.1 Aleš Kert

Je kot edini član ekipe z boljšim znanjem programskega jezika C++ delal na razvoju detekcije objektov. Poleg tega je sodeloval pri razvoju zaznave in prepoznave obrazov in razvil osnovno verzijo modula za razpoznavanje govora. Hkrati je pomagal tudi pri delu razvoja navigacije in pisanju poročil.

Opravil je približno 30% vsega dela.

5.2 Mark Žakelj

Je razvil večji del logike za navigacijo in povezal posamezne komponente v celoto. Hkrati je razvil tudi komponento, ki upravlja z robotsko roko. Poleg naštetega je sodeloval tudi pri razvoju detekcije in prepoznave obrazov.

Opravil je približno 40% vsega dela.

5.3 Matevž Eržen

Je delal na razvoju zaznavanja in prepoznavanja obrazov, implementiral celotno komponento za prepoznavo barv in dodelal algoritem za prepoznavo govora. Hkrati je pomagal tudi pri manjših delih razvoja nekaterih ostalih komponent in pisanju poročil.

Opravil je približno 30% vsega dela.

6 Zaključek

Kljub omejitvam in delu le v simuliranem okolju, smo se lahko dodobra spoznali z razvojem *kompleksnejših* sistemov, kjer mora več komponent sodelovati med seboj. To je bil hkrati tudi največji izziv razvoja celotnega robota. Čeprav smo imeli dobršnjo mero težav pri vzpostavitvi razvojnega okolja in tudi kasneje pri delu, smo s predmetom generalno zadovoljni in veseli, da smo se lahko spoznali s tehnologijo na "*hands-on*" način.

Med samim delom smo imeli največ težav s samo procesorsko zahtevnostjo poganjanja vseh komponent, ki so potrebne za delo v virtualnem okolju. To se je videlo tudi pri končni predstavitvi, ko je robot za izvedbo celotnega scenarija potreboval skoraj 15 min. Vse to je bila primarno posledica počasne vizualizacije, saj naj bi robot za izvedbo enakega postopka v realnosti (sodeč po faktorju hitrosti uporabljanja znotraj *Gazebo*) potreboval le med 6 in 7 minut.

Na strani programske opreme nam je največ preglavic povzročalo samo okolje ROS in komponenti za virtualizacijo; *Gazebo* in *Rviz*. Ker gre za odprtokodne rešitve, je nesmiselno pričakovati uporabo brez najmanjših težav, a v tem primeru so bile le te vse prej kot majhne. Glavna težava je bila kompatibilnost z različnimi verzijami in distribucijami *Linuxa* in tudi verzijami odvisnih komponent. Vse to nas je pripeljalo do tega, da smo si poleg naših operacijskih sistemov morali naložiti še (zdaj že relativno zastaran) operacijski sistem *Ubuntu 16.04*.