Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

RINS končno poročilo

Rok Rajher, Jan Kuhta, Jaka Kužner

Skupina CHI
χ

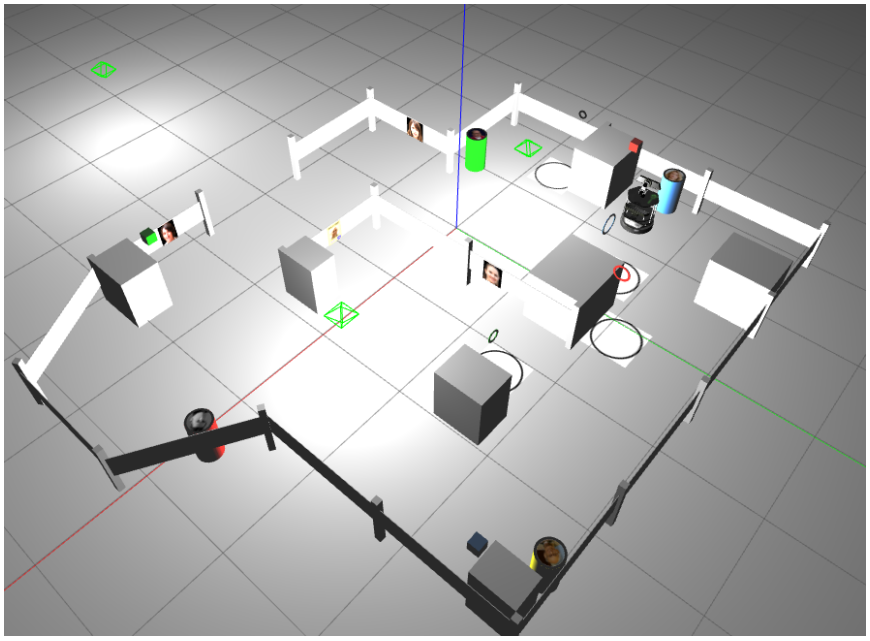
Ljubljana, 2023

1 Uvod

Tekom drugega semestra v študijskem letu 2022/23 smo pri predmetu Razvoj inteligentnih sistemov razvijali programsko opremo robota, da bi se ta znašel v simuliranem prostoru, ki imitira resnično življenjsko situacijo.

V tem poročilu bomo povzeli tako teorijo kot tudi implementacijo metod in tehnik, ki jih je robot uporabljal za identifikacijo in pomnjenje različnih objektov, kot so cilindri, obroči, plakati in človeški obrazi, parkiranje ipd. Robot se je moral gibati po poligonu, ki ponazarja majhno mestece. V okolju je zaznaval obraze, k njim pristopil in jih povprašal o ubežniku. Na stenah je zaznaval tudi plakate, na katerih so bili objavljeni ubežniki, njihova nagrada in simbol zapora, iz katerega so pobegnili. Robot je moral s pomočjo pridobljenih informacij ubežnika najti skritega na enem od cilindrov in ga pripeljati v zapor, iz katerega je pobegnil.

Sistem smo implementirali v okolju ROS in Gazebo, del naloge pa tudi na resničnem poligonu postavljenem na fakulteti.



Slika 1: Poligon task 3, 2023

2 Metode

V tem poglavju bomo predstavili pomembne metode, ki smo jih uporabili pri implementaciji.

2.1 Navigacija

Za raziskovanje prostora smo izbrali štirinajst najprimernejših točk, ki pokrijejo celoten poligon. Na vsaki točki je robot pregledal prostor okoli sebe in ustrezno odreagiral na zaznave ter nadaljeval do naslednjega zastavljenega cilja.

2.2 Zaznavanje obrazov in lokalizacija

Detekcija obrazov je potekala na podlagi RGBD slike. Uporabljamo vnaprej naučen model globoke nevronske mreže za klasificiranje obrazov. Model vzame vhodno sliko, izvede razširjanje naprej, tako da lastnosti slike primerja z naučenimi in vrne zaznane obraze. RGBD sliko torej najprej predpripravimo in ustvarimo "blob", ki ga nato pošljemo skozi model, da dobimo vse zaznane potencialne obraze. Če je zaupanje dovolj veliko, pomeni, da je na sliki zaznan obraz.

Nato uporabimo lokalizacijo, da obrazu določimo položaj v poligonu. Uporabili smo natančno kalibrirano kamero na RGBD sensor Kinect modulu. Položaj izračunamo s pomočjo globinske slike, na kateri najprej izračunamo kje se nahajo točka obraznega centra. Nato izračunamo kot do obraza po formuli:

$$\text{angle\_to\_target} = \arctan 2(face_x, k_f)$$

kjer je k_f enak goriščni razdalji v pikslih, $face_x$ p x-koordinata središča zaznanega obraza na sliki. Koordinate v robotovem prostoru pridobimo s formulama:

$$x = \text{dist} \cdot \cos(\text{angle\_to\_target})$$

$$y = \text{dist} \cdot \sin(\text{angle\_to\_target})$$

kjer je *dist* komponenta iz globinske slike. Točko smo nato transformirali v koordinatni sistem mape.

Za natančno določanje položaja smo uporabili gručenje, kjer smo za vsak obraz izračunali centroid položajev petih detekcij, kar je podrobno opisano v poglavju 3.3.4.

2.3 Zaznavanje obročev

Potenicalne obroče smo zaznavali z prileganjem elips po sledeči enačbi:

$$\left(\frac{x - c_x}{a}\right)^2 + \left(\frac{y - c_y}{b}\right)^2 = 1$$

Če sta bili na sliki zaznani vsaj dve elipsi smo preverili, ali sta njena centra dovolj blizu. Ker so na poligonu bile tudi slike krogov, smo morali še preveriti, ali je v sredini obroča dejansko luknja. Le če je bil objekt votel, smo ga dodali med obroče. Barvo obročev smo zaznali iz barvne slike, tako da smo iz slike izrezali obroč in poiskali prevladujoči odtenek. Prilagajanje elips in ocenjevanje votlosti smo izvedli z globinsko sliko.

2.4 Segmentacija in zaznavanje cilindrov

Cilindre smo zaznavali na podlagi objekta Point Cloud. To je množica podatkov v 3D prostoru, ki predstavlja obliko in barvo površin, kot jih vidi robot. Vsaka točka je določena s tremi koordinatami in barvno komponento. Pridobljeni nabor točk smo najprej zmanjšali s pomočjo VoxelGrid filtriranja, ki prostor razdeli na enako velika območja v 3D prostoru, ki jim pravimo voksli. Velikost vokslov nam določi koliko detajlov želimo ohraniti, saj ohrani samo eno točko na voksel, ostale pa zavrže. Na ta način močno zmanjšamo nabor podatkov za nadaljnjo obdelavo, hkrati pa ohranimo glavne značilnosti oblike prostora. S pomočjo passthrough filtriranja smo nabor še bolj omejili. Izločili smo točke, ki so predaleč in točke, ki so prenizko/previsoko.

Za potrebe ocenjavanja normal smo podatke organizirali v k-dimenzijsko drevo, ki omogoča učinkovito iskanje najbližjih sosedov. Za vsako točko določimo sosede in prilagodimo lokalno ploskev, za katero nato izračunamo normalni vektor. Z izračunanimi normalami površin bomo definirali ploskve v prostoru po formuli:

$$n_x = a/(\sqrt{(a^2 + b^2 + c^2)})$$

$$n_y = b/(\sqrt{(a^2 + b^2 + c^2)})$$

$$n_z = c/(\sqrt{(a^2 + b^2 + c^2)})$$

in konstanto:

$$p = d/(\sqrt{(a^2 + b^2 + c^2)})$$

Ploskev definiramo v normalni obliki (ang. Hessian Normal Form):

$$\hat{n} \cdot x = -p$$

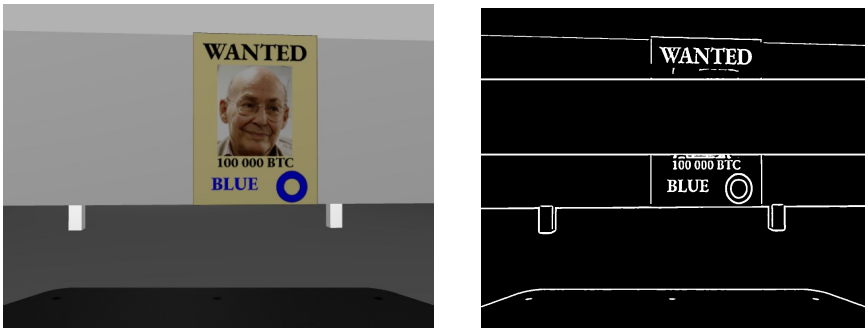
Za segmentacijo smo uporabili iterativni RANSAC, ki iz množice točk naključno izbira vsaj po tri nekolinearne točke in z njihovimi definira ploskev. Na podlagi razdalje med točko x_0 in ploskvo ($D = \hat{n} \cdot x_0 + p$) razdeli točke na inlierje in outlierje konvergira proti največji ploskvi. Tako iz nabora odstranimo tudi točke na steni.

Z ostalimi točkami in normalami ponovimo RANSAC, tokrat prilegamo model cilindra, definiranega s koordiatami centralne točke, koordinatami osi cilindra in njegovim polmerom. Tako iz točk izluščimo cilinder. Preverimo tudi, ali je njegov polmer v pričakovanih vrednostih in se odločimo ali se na v vidnem prostoru nahaja cilinder. Če se, njegov centroid transformiramo relativno na mapo, kjer ga označimo. Točke razdelimo v gruče in na sosednjih točkah na podlagi evklidske razdalje povprečimo centroid cilindra. Iz barvne komponente točk v segmentiranem točkovnem oblaku ocenimo tudi barvo cilindra.

2.5 Branje posterjev

Lokacijo posterjev smo zaznavali enako kot zaznavanje obrazov. K vsakemu obrazu je robot pristopil, nato pa preveril če gre za navaden obraz ali za poster. To smo izvedli s pomočjo OCR (Optical Character Recognition). Iz prebranega besedila smo tako ugotovili, ali gre za poster ali ne, prav tako pa smo iz besedila razbrali barvo obroča, ter denarno nagrado.

Pred izvedbo branja besedila iz slike smo na sliki izvedli predprocesiranje, kjer smo uporabili Gaussian Blur za zmanjšanje šuma, Adaptive Thresholding za povečanje kontrasta ter morfološko operacijo opening za izboljšanje povezljivosti znakov, kar je vidno na sliki 2.



(a) Zaznan poster (b) Predprocesirana slika

Slika 2: Predprocesiranje posterjev

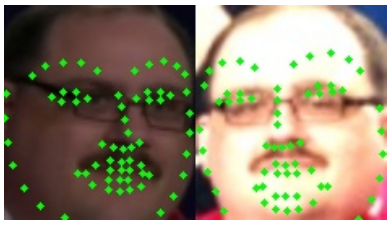
2.6 Primerjava obrazov

Ponovno identifikacijo obraza smo uporabili pri identifikaciji kriminalca. Obraz na vsakem pristopljenem cilindru smo primerjali z obrazom na glavnem posterju in preverili na katerem gre za isto osebo. Pri tem smo uporabili metodo računanja podobnosti dveh slik.

Za obe sliki smo najprej poiskali obrazne značilke (ang. face landmarks) z uporabo naučenega objekta `shape_predictor_68_face_landmarks.dat`, ki na sliki poišče 68 obraznih značilk, kot so npr. kotički oči, obrvi, nos, usta, oblika obraza ipd. (slika 3).

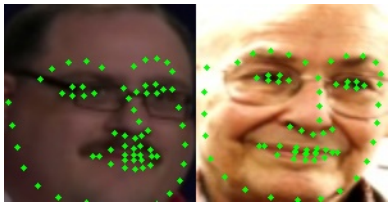
Za vsako sliko izračunamo deskriptor s vnaprej natreniranega modela globoke nevronske mreže `dlib_face_recognition_resnet_model_v1.dat`. Nato med deskriptorjema izračunamo evklidsko razdaljo, ki jo nato odštejemo od 1, da dobimo zaupanje. Če je zaupanje dovolj veliko, sta obraza na slikah ista.

(a) Enaki obrazi



(b) `Faces are the same!`
`Confidence: 0.492679979019`

(c) Različni obrazi



(d) `Faces are different!`
`Confidence: 0.29128492619`

Slika 3: Obrazne značilke za primerjavo obrazov

2.7 Pristop

Pristop (Approach), smo implementirali na podlagi Rviz costmapa. Costmap smo najprej pretvorili v binarno sliko in na njej izvedli erozijo (slika 4). Tako smo dobili sliko dostopnih točk na mapi.

Slika 4: Popravljen costmap

Vsakič, ko je potrebno pristopiti k obrazu, posterju cilindru ali ringu, se z uporabo preprocesirane slike costmapa po kriteriju evklidske razdalje izračuna najbližja dostopna točka(torej točka, ki je na mapi bela), na katero se robot prestavi.

$$(r, c) = \{(r', c') \mid r' \neq 0, c' \neq 0\}$$

$$\text{min\_idx} = \text{argmin}\left((r-y)^2 + (c-x)^2\right)$$

$$\text{closest\_c, closest\_r } = c[\text{min\_idx}], r[\text{min\_idx}]$$

Robot se pomakne proti izračunani točki. Ko prispe na cilj, se obrne tudi v smer obraza in spregovori.

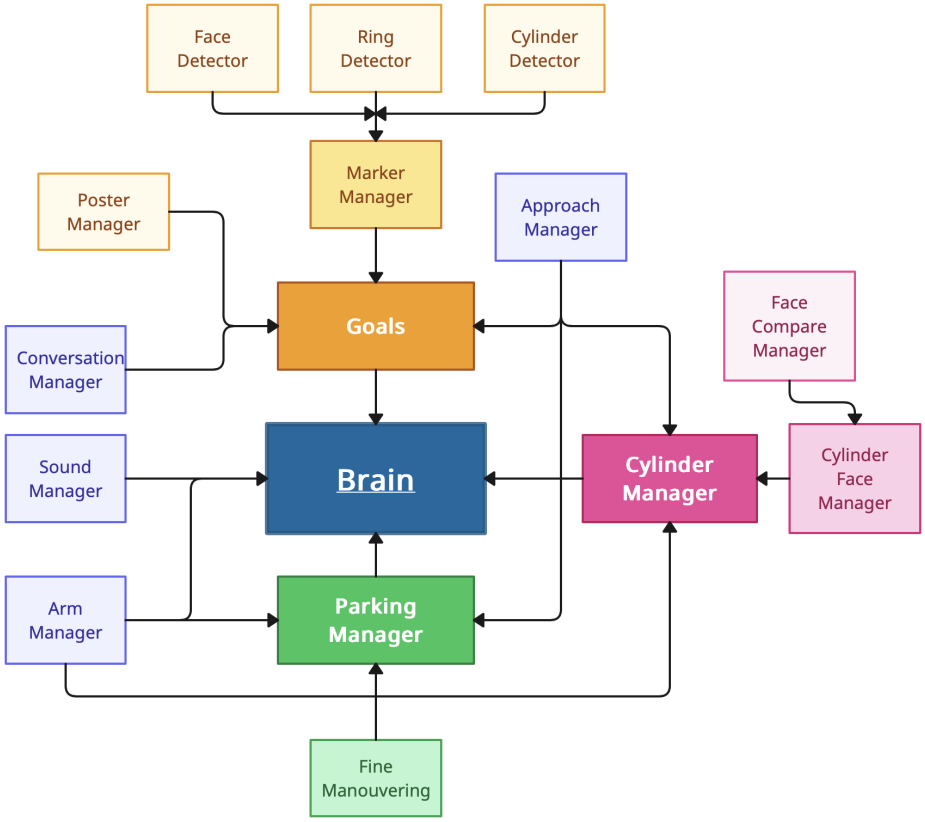
2.8 Iskanje parkirnega mesta in parkiranje

Tudi tukaj na sliki prilegamo dve elipsi s podobnim središčem, tokrat na tleh. Robota nato natančno poravnamo s povprečenim središčem teh dveh elips tako, da se to nahaja v majhnem vertikalnem pasu točk na sliki. Robot se nato premika počasi naprej in poravnava proti središču, dokler ne vidi več elips. Ko elips ni več na sliki, se samo še nekoliko približa centru elips in se tam ustavi.

3 Implementacija in integracija

Projekt smo zasnovali po modulih, ki smo jih ločili glede na funkcionalnost. Komponente med seboj komunicirajo bodisi preko topicov, kot tudi ROS vozlišča, ali pa z medsebojnimi klici med python moduli. Skupaj so komponente povezane v celoten sistem oz. omrežje, ki deluje koherentno in samostojno. 5.

Središče sistema predstavlja modul `brain.py`, ki skrbi za glavno logiko sistema. Neposredno je povezan s tremi glavnimi podkomponentami in sicer `goals.py` za raziskovanje prostora in zaznavanje objektov, `cylinder_manager.py` za iskanje kriminalca ter `parking_manager.py` za izvedbo parkiranja.



Slika 5: Struktura sistema

Modul Goals robota vodi po poligonu, ter preko `marger_manager.py` pridobiva informacije o zaznanih objektih, ki po topicih pridejo iz vozlišč za detekcije in sicer `face_localizer_dnn` za detekcijo obrazov, `detect_rings.py` za detekcijo obročev ter `cylinder_segmentation.cpp` za zaznavanje cilin-

drov. `marker_manager.py` te informacije prejme, ustvari objekte tipa `Face, Ring, Cylinder` ter objekte pošlje naprej v `goals.py`. Modul `approach_manager.py` izvede pristop do obrazov, kjer `poster_manager.py` preveri ali gre za poster. Modul `conversation_manager.py` pa izvede dialog z osebo.

Modul Cylinder Manager skrbi za iskanje kriminalca. S pomočjo `approach_manager.py` izvede pristop k cilindru, kjer `cylinder_face_manager.py` nastavi roko, da je usmerjena proti cilindru, poišče obraz ter preko `face_compare_manager.py` izvede primerjavo obraza, da preveri, če je dovolj podoben obrazu na posterju.

Parking Manager na koncu poskrbi, da robot pristopi do ciljnega obroča in izvede končno parkiranje s pomočjo modula `fine_manouvering_manager.py`.

Za pretvorbo besedila v zvok skrbi pomožni modul `sound_manager.py`, premike robotske roke, na kateri je kamera, pa izvede `arm_manager.py`.

3.1 Zagon programov

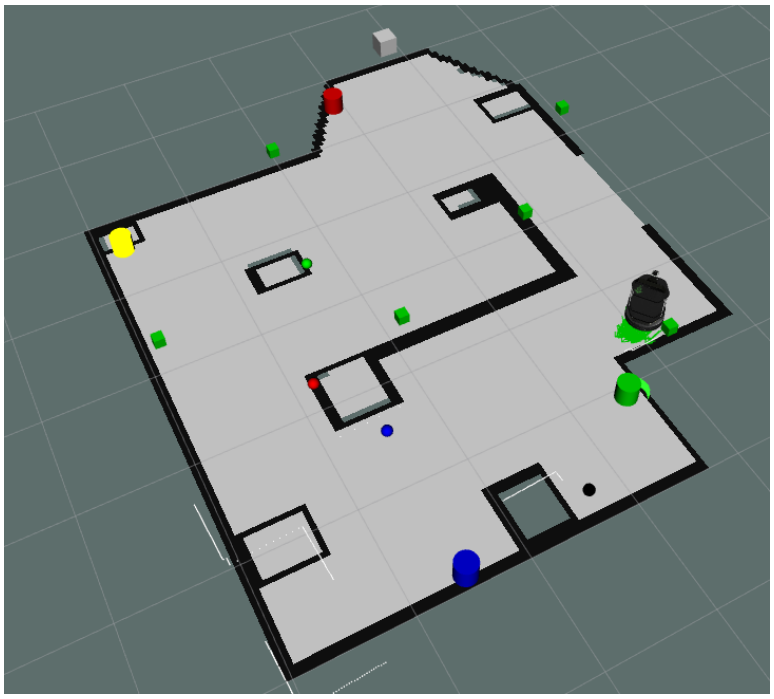
Za lažji zagon celotnega programa in modulov posebej, smo uporabljali datoteke za zagon (ang. launch files), ki omogočajo konfiguracijo in organizacijo več vozlišč hkrati. Olajšali smo si sam zagon programov ter spreminjanje posamičnih parametrov (npr. menjava poligona).

Za pogon celotnega sistema je treba zagnati zagonsko datoteko `rins_world.launch`, ter vozlišča `goals.py`, `face_localizer_dnn`, `detect_rings.py` ter `cylinder_segmentation`.

3.2 Vizualizacija

Za vizualizacijo zaznanih objektov smo uporabljali markerje, ki smo jih sproti objavlja na definiranih topicih (kot so `/face_markers`, `/ring_markers`, `/parking_markers` in `/detected_cylinders`). Markerje smo ločili glede na barvo in velikost posameznega objekta.

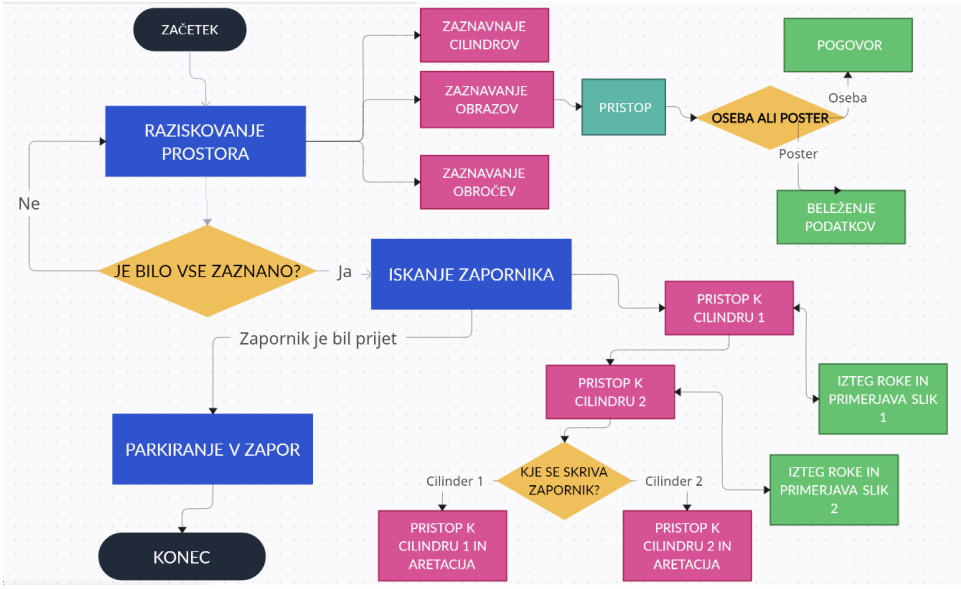
Vizualizacija poligona, kjer markerji predstavljajo položaje in barve posameznih objektov je vidna v sistemu Rviz, kar je prikazano na sliki 6.



Slika 6: Prikazani markerji v sistemu Rviz

3.3 Glavna logika

V tem poglavju bomo podrobneje opisali tok delovanja, ki je prikazan na sliki 7 in implementacijo metod zaznavanja objektov.



Slika 7: Tok delovanja brain.py

3.3.1 Raziskovanje prostora

Prostor preiskujemo po že opisani metodi 2.1. Med raziskovanjem robot zaznava obraze, cilindre obraze in obroče. Če zazna obraz, prekine raziskovanje prostora in pristopi k obrazu in preveri, ali gre za poster ali osebo. Če gre za osebo, se z njo pogovori, če pa zazna poster, potem določi barvo zapora in nagrado.

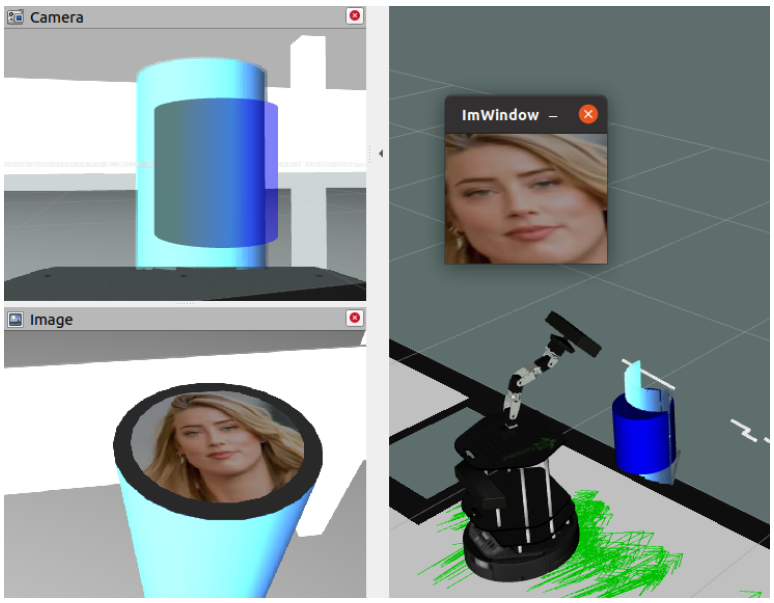
Raziskovanje prostora se zaključi, ko robot zazna podano število cilindrov, obročev, posterjev in obrazov na poligonu.

3.3.2 Iskanje zapornika

Zapornik se skriva na enem od dveh cilindrov. Robot gre do vseh cilindrov, ki so mu jih predlagali obrazi, ki jih je povprašal. Išče ubežnika, za katerega je razpisana največja nagrada. Obraz, skrit na cilindru primerja s sliko na posterju. Odpelje tistega, ki ima največjo podobnost z zapornikom.

Osrednja logika iskanja zapornika je implementirana v skripti `cylinder_manager.py`. Robot najprej izvede pristop do cilindra s klicom modula `approach_manager.py`.

Nato pokliče `cylinder_face_manager`, ki iztegne roko, poišče obraz na cilindru ter z kamero na robotski roki zajame RGB sliko. Nato dobljeno sliko preko `face_compare_manager.py` primerja s sliko na glavnem posterju in izračuna podobnost. Na koncu robot pristopi do cilindra, na katerem se skriva zaportnik z največjim ujemanjem in ga aretira. Postopek je prikazan na sliki 8.

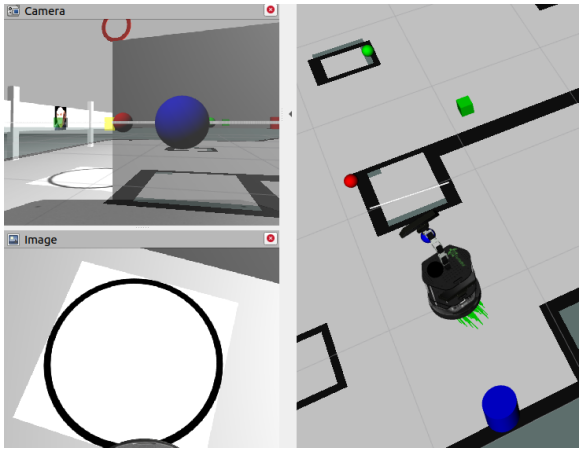


Slika 8: Iskanje zapornika

3.3.3 Parkiranje

Parkiranje je razdeljeno na dva dela, grobo parkiranje in fino parkiranje. Logiko parkiranja nadzoruje `parking_manager.py` Najprej robot grobo parkira, To pomeni, da pristopi k lokaciji obroča (`approach_manager.py`), ki predstavlja pravi zapor. Ko doseže cilj, iztegne roko in se pomakne nekoliko nazaj, in zavrti dokler parkirno mesto ni na sliki.

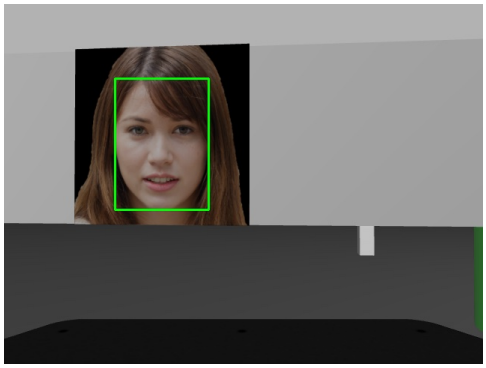
Sledi fino parkiranje (`parking_detector.py`), kjer robot s pomočjo kamere na robotski roki na tleh zaznava elipse in natančno parkira 2.8. Za to smo uporabili `Twist` sporočila, ki robota natančno poravnavajo s središčem. Ko je poravnan se lahko premika naprej in približuje parkirišču. Na koncu še pomaha z robotsko roko in se poslovi.



Slika 9: Robot se pri finem parkiranju natančno poravnava glede na lokacijo središč elips na sliki in se počasi približuje.

3.3.4 Zaznavanje obrazov

Za zaznavanje obrazov smo uporabili skripto `face_localizer_dnn`. V zanki zajemamo RGB sliko iz topica /camera/rgb/image_raw ter Depth sliko iz topica /camera/depth/image_raw. Sliki pretvorimo v format `cv2`. RGB sliko predpripravimo in pošljemo skozi model globoke nevronske mreže, ki vrne tabelo zaznanih obrazov.

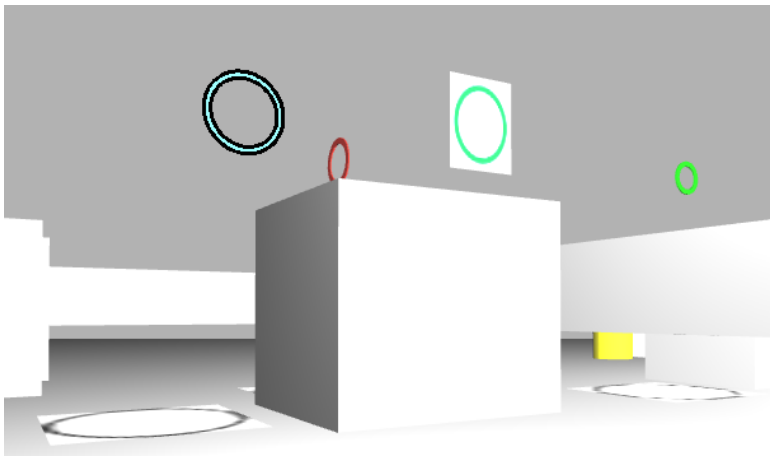


Slika 10: Zaznan obraz na RGB sliki

Če je zaupanje dovolj veliko, razdalja do obraza znana ter manjša od 1,5m, lokacijo obraza dodamo v najbljižjo gručo (`class Cluster`), oz. ustvarimo novo, če v bližnji okolici gruča še ne obstaja. V gruči sproti računamo centorid, dokler ni v gruči 5 slik. Nato gručo označimo kot polno, prenehamo dodajati nove lokacije v to gručo, ter postavimo marker.

3.3.5 Zaznavanje obročev

Za zaznavanje obrazov smo uporabili skripto `detect_rings`. Podobno kot za zaznavo obrazov, v zanki zajemamo RGB sliko iz topica /camera/rgb/image_raw ter globinsko sliko iz topica /camera/depth/image_raw, ter pretvorimo v format `cv2`. RGB sliko uporabimo za zaznavanje barv obročev, globnisko sliko pa za prilagajane elips in preverjanje votlosti obročev.



Slika 11: S črno smo označili elipsi, ki se prilegata zaznanemu obroču.

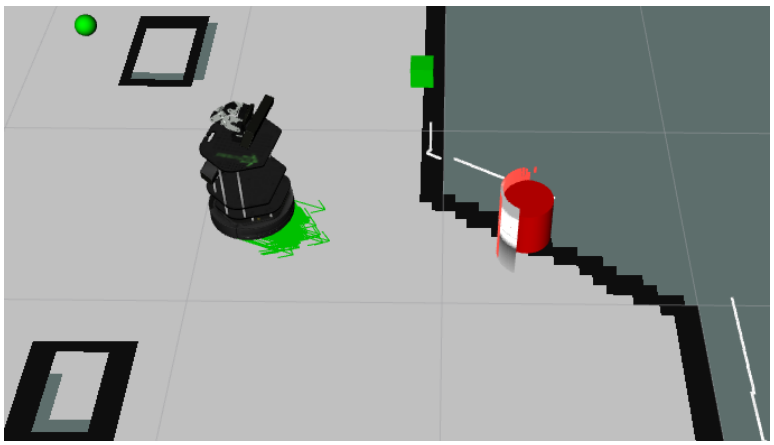
Če je potencialni obroč zaznan bližje kot 1.6m in je votel, potem se izvede gručenje, kjer se pogleda ali obroč že pripada kakšni gruči ali če gre za nov obroč. Obroč je dokončno zaznan, ko je v določeni gruči vsaj šest zaznanih obročev.

3.3.6 Segmentacija in zaznavanje cilindrov

Obdelava točkovnega oblaka in segmentacija cilindrov nekoliko težji zalogaj za procesorje. Zaradi optimizacijskih potreb, smo kodo razvijali v C++. Uporabili smo odprtokodni projekt Point Cloud Library (`PCL`), ki nudi vsa orodja za obdelavo 2D/3D slik in točkovnih oblakov. Na osnovi kode podane na vajah, smo razvili bolj robustno rešitev, ki predvsem bolje izključi nepotrebne točke. Obdelava je tako nekoliko bolj tekoča in se lažje izvaja v resničnem času.

Točkovni oblak, na katerem smo izvajali segmentacijo cilindrov smo osveževali z naročnikom na temo /camera/depth/points, ki ga objavlja RGBD sensor Kinect, nameščen na robota in z osmi robota poravnan tako, da zajema prostor pred robotom. Za potrebe določanja barve cilindra smo točke definirali kot `pcl::PointXYZRGB`, ki vsebuje tudi barvno komponento. Izračunane centroide samih cilindrov smo kot barvne markerje objavljali na

drugo temo, ter jih predstavili v RVizu. Za namene razvoja in testiranja smo objavljali tudi segmentirani oblak točk, da smo lahko v resničnem času opazovali rezultat segmentacije.



Slika 12: Vizualizacija barvnega točkovnega oblaka, segmentiranega kot cilinder in markerja, ki se je pripel po potrditvi cilindra.

3.3.7 Branje posterjev

Branje posterjev smo implementirali v skripti `poster_manager.py`. Uporabili smox metode OCR opisane v poglavju 2.5 ter omenjene metode za predprocesiranje slike. OCR smo izvedli s pomočjo pythonovske knjižnice `pyocr`. Za predprocesiranje slike smo uporabili knjižnico `cv2`. Celotni postopek smo na vsakem posterju ponovili do 3x, saj se je včasih kakšen podatek izgubil. Tako smo v vsaki iteraciji dopolnili še neznane podatke o posterju, torej barvo obroča in nagrado. Oboje smo prebrali iz besedila.

4 Rezultati

Z rezultatom na task 3 smo zelo zadovoljni, prav tako z evalvacijo v živo, nekoliko manj pa z rezultati na task 1 in task 2. Na prvih dveh evalvacijah smo imeli tudi nekaj smole, saj program ni bil dovolj robusten in robot, zato robot včasih ni mogel opraviti vseh nalog. Do tretje evalvacije nam je napake uspelo popraviti tako, da je robot praktično vsakič uspel izvesti celotno nalogo.

Po težavah na task 1 in task 2 smo se projekti lotili praktično znova in na novo spisali celotno logiko, ohranili smo le komponente za zaznavanje objektov. Projekt smo razdelili v module, ki se med seboj kličejo. Tako smo se znebili ene dolge skripte, ki smo jo imeli prej in ni bila več berljiva. Modularna zgradba je omogočala boljši pregled, lažje razhroščevanje in lažje nadgrajvanje. Razvijali smo vsak modul zase, ter sproti testirali. Vse module smo združili v glavno skripto, ki poveže program v logično celoto. Ponosni smo na končno strukturo našega ROS projekta, prav tako pa na našo celotno logiko, ki robota popelje čez celoten poligon od začetka, do končnega parkiranja z izvedbo vseh vmesnih nalog. Predvideli smo vse možne zaplete in robota na njih tudi pripravili.

Robot uspešno in zelo robustno zaznava obraze, obroče in cilindre, tako barvo kot lokacijo. Pri tem se praktično nikoli ne zmoti, tudi napačno zaznanih objektov (false detections) praktično nikoli ni. Vse objekte je najde v enem obhodu poligona. Zelo smo zadovoljni s hitrostjo zaznavanja objektov, predvsem pri cilindrih, saj gre za obdelavo zelo obsežnih PCL podatkov. Včasih (zelo redko) se zgodi, da robot zazna napačno barvo, kar je posledica pretirane/premajhne osvetlitve poligona. Na senčni strani obroča je zaznal črn obroč, kar pa smo rešili z dodajanjem luči v poligon. Zaradi preosvetljenosti je včasih moder cilinder zamenjal za belega, vendar redko.

Prav tako robot brez težav uspešno parkira v krog. Pri tem najprej pristopi do lokacije kroga, se pomakne nazaj, nato pa izvede natančno manevriranje in na koncu vedno parkira znotraj kroga.

Tudi ponovna identifikacija obrazov dela uspešno, in v 90% pravilno primerja obraza na slikah. Včasih se je zgodilo, da nobenega od obeh obrazov na cilindrih ne prepozna kot osebo na posterjih, takrat se odloči za obraz, kjer je podobnost večja. Zalo smo ponosni na implementacijo pristopa do objektov (approach). S tem smo imeli največ težav na prvih dveh evalvacijah, zato smo temu pred task 3 namenili največ časa in preizkusili različne pristope. Zgodilo se je, da je točka za pristop bila v prepovedanem območju, včasih je robot pristopil iz napačne smeri, težave nam je povzročala tudi orientacija. Na koncu smo vse težave odpravili in zimplementirali pristop s

pomočjo costmapa, za katerega lahko pritrdimo, da dela 100% brez izjeme, tako pri pristopu obrazov, kot tudi cilindrov in obročev.

Tudi prepoznavanje in branje posterjev dela uspešno. Robot vsak poster prepozna kot poster in ne le kot navaden obraz. Tudi z prepoznavo barve iz posterja v večini časa nima nobenih težav, v redkih primerih ni uspel zaznati barve na posterju ki je bil preveč osvetljen. V 90% tudi pravilno prebere denarno nagrado, težave je le imel pri števki 1, saj jo je včasih zaznal kot 7 in obratno.

Na končni evalvaciji smo bili zelo zadovolnji z uspešnostjo našega robota. Vse objekte je zaznal pravilno, tako barve kot tudi lokacije, braz napačnih detekcij. Vse pristope je izvedel uspešno, tudi dialog z osebami ni predstavljal nobenih težav. Uspešno je prebral tudi posterje in primerjal obraze, ter našel in pobral pravilnega kriminalca, ter ga uspešno odpeljal v pravi zapor, kjer je uspešno parkiral znotraj kroga. Vse detekcije je izvedel v prvem obhodu. Edina pomanjkljivost je bila, da iz enega posterja ni uspel izluščiti barve obroča, predvidevamo zaradi prevelike osvetlitve. Zelo smo zadovoljni kako smo izboljšali našega robota v primerjavi s task 1 in task 2.

5 Delitev dela

Ker smo kodo organizirali modularno, je lahko vsak delal na svojem modulu in ga ločeno testiral. Na fakulteti smo se dobili in drug drugemu kodo predstavili ter jo povezali. Po testiranju je vsak predlagal izboljšave v nedelujočem modulu, dokler nismo napake odkrili in odpravili.

5.1 Rok

Rok je bil odgovoren za implementacijo algoritmov za zaznavanje obročev in pristopa, ter oblikovanje celotne logike projekta. Skupaj z ekipo je določil zahteve projekta in razvil arhitekturo sistema.

5.2 Jan

Jan je prevzel vodilno vlogo pri zaznavanju obrazov in razpoznavanju posterjev. Njegovo delo je obsegalo raziskovanje in implementacijo algoritmov za zaznavanje in prepoznavanje obrazov in branje posterjev ter splošno organizacijo arhitekture projekta, ter za implementacijo premikov robotske roke.

5.3 Jaka

Jaka je bil zadolžen za implementacijo algoritmov za zaznavanje cilindrov, izvedbo parkiranja ter vzpostavitev komunikacije med robotom in osebami. Njegovo delo je vključevalo razvoj in implementacijo algoritmov za natančno zaznavanje cilindrov ter identifikacijo parkirnih mest. Poleg tega je bil odgovoren za izvedbo pogovorov in interakcijo z osebami.

5.4 Ocena delitve

Z razdelitvijo nalog na ta način smo omogočili, da je vsak član ekipe prevzel jasno določeno področje dela. Kljub temu smo tesno sodelovali kot ekipa, da smo zagotovili usklajenost projekta in enakovreden prispevek vsakega člana. Skupaj smo uspešno zaključili projekt in vsak član je **enakovredno prispeval k uspehu**.

6 Zaključek

Z rezultati smo zadovoljni. Kljub slabemu začetku s prvo in drugo nalogo smo uspeli kodo pravočasno izboljšati in jo narediti mnogo bolj robustno. Za enostavnejši razvoj smo praktično vso kodo za prvi dve nalogi prepisali v python (razen segmentacije cilindrov) in bolje organizirali logiko. Naš robot je brezhibno opravil končno predstavitev tretje naloge. Možna izboljšava našega projekta bi bila implementacija avtonomnega raziskovanja prostora.

Delo nam je utežila strojna oprema, saj noben izmed nas ni imel native operacijskega sistema Linux. Tako smo bili primorani uporabljati virtualke, večino časa pa smo preživeli v laboratoriju na šolskih računalnikih. Z boljšo strojno opremno bi prihranili veliko časa pri testiranju.

Zaključimo lahko, da je razvoj in implementacija robota v resničnem svetu še težja od tiste v simuliranem okolju, saj je nepredvidenih faktorjev še toliko več. Naučili smo se, da je za robustno delovanje potrebno ogromno testiranja, razhroščevanja, optimiziranja in izboljševanja in šele zdaj znamo ceniti uspešne implementacije podobnih sistemov.