Poročilo o projektni nalogi pri predmetu RINS

Jure Pahor, Domen Ogorevc, Grega Potočnik
Ekipa: Omicron

10. junij 2024

Kazalo

Poglavje 1

Uvod

Namen poročila je predstaviti metode in delo, ki smo ga izvedli v sklopu projekta pri predmetu Razvoj inteligentnih sistemov. Pri tej nalogi smo izdelali celostno rešitev za avtonomno navigacijo, prepoznavo obrazov, odkrivanje anomalij, govorno sintezo ter preprosto prepoznavo govora. V naslednjih poglavjih bodo vsi koraki, ki smo jih izvedli, podrobno opisani in razloženi. Vključno s samo predstavo problema in sistemom na katerem smo razvijali.

Poglavje 2

Predstavitev projekta

Naša naloga je bila programiranje robota Turtlebot 4, ki se je izvajal v simulaciji Gazebo. Uporabili smo orodje ROS 2 za zanesljivo komunikacijo med robotskimi komponentami. V naslednjih razdelkih so na kratko predstavljeni glavni sestavni deli projekta.

2.1 ROS2

[1]ROS 2 (Robot Operating System 2) je odprtokodni okvir za razvoj robotske programske opreme, ki izboljšuje podporo za realno-časovne sisteme, varnost in večnitnost. Podpira različne operacijske sisteme in komunikacijske protokole, kar omogoča večjo prilagodljivost pri razvoju robotskih aplikacij. Za naš projekt smo uporabljali različico Humble Hawksbill.

2.2 Turtlebot 4

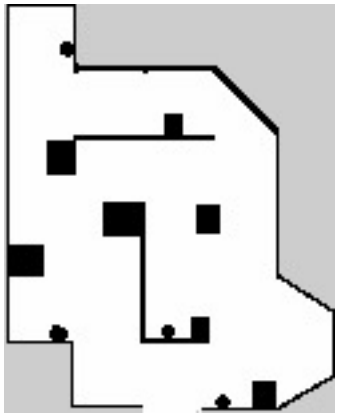
[2]TurtleBot 4 je odprtokodni mobilni robot za izobraževanje in raziskave. Vsebuje osnovo iRobot Create 3, Raspberry Pi 4 z nameščenim ROS 2 in senzorje, kot so OAK-D stereo kamera in 2D LiDAR. Naš robot je v simulaciji opremljen tudi z robotsko roko s kamero, ki se lahko pregiba v treh sklepih v eni smeri.

2.3 Gazebo

[3]Gazebo je napreden simulator za robotiko, ki omogoča natančno simulacijo robotov v kompleksnih okoljih. Uporabili smo ga zaradi časovne stiske, saj je programiranje na pravem robotu zamudno. Gazebo omogoča hitro testiranje in poganjanje robota prek simulacije. Vendar pa ima tudi pomanjkljivosti, saj ni vedno zanesljivo okolje.

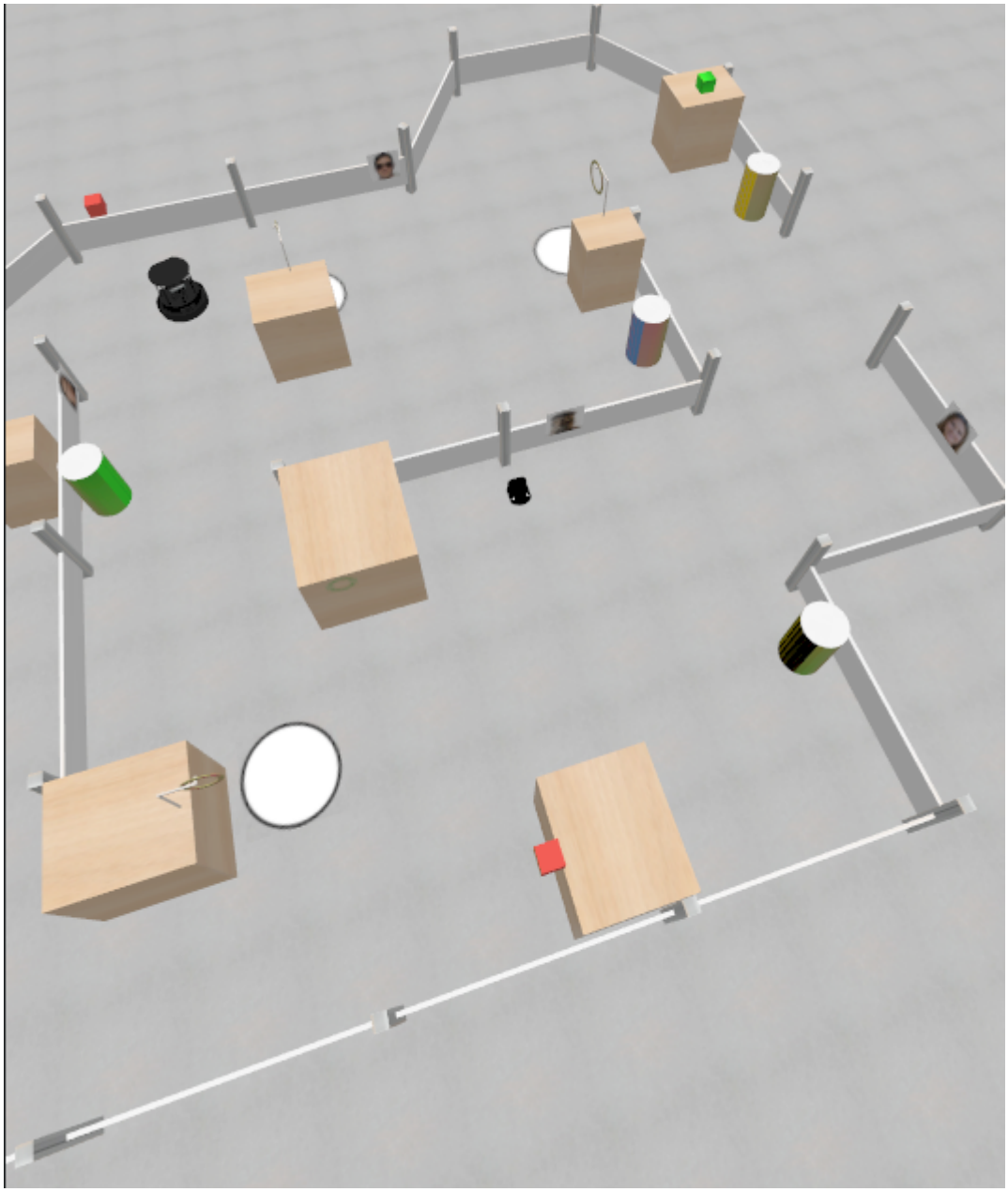
2.4 Svet simulacije

Simulacijski svet je sestavljen iz manjše ravnine, omejene s stenami in ovirami. Robot se lahko premika le po belih površinah (slika 2.1).



Slika 2.1: Svet robota

Simulacija je v treh dimenzijah, kjer so po svetu raztresene slike obrazov, cilindri in obroči, parkirna mesta in barvne kocke, ki naj bi zmedle robota (slika 2.2).



Slika 2.2: Gazebo prikaz simulacije

2.5 Naloga

Navodila naloge so bila sledeča: Robot se mora avtonomno premikati čez simulacijo in iskati obraze, slike, cilindre, obroče ter parkirna mesta. Ko najde obraz, mora sintetizirati govor in vprašati osebo o lokaciji pravilne slike Mona Lise. Glede na odgovor robot stori naslednje:

- Če je odgovor *ne*, nadaljuje iskanje.
- Če je odgovor *da*, vpraša za namig v obliki barve obroča, pod katerim mora parkirati.

Robot mora najti ta obroč, se postaviti v njegovo bližino in tam iskati parkirno mesto. Ko parkira, mora najti najbližji cilinder z QR kodo, ki predstavlja pravo sliko Mona Lise. Če se zmoti oziroma ga namig zavede in parkira pod napačen obroč, nadaljuje iskanje.

Ko najde pravo Mona Liso, se začne premikati po svetu in iskati slike Mona Lis na stenah. Robot mora ugotoviti, če ima slika kakšno napako, in napako prikazati. Ključna je robustnost in zanesljivost zaznavanja napak ter določanja ciljev. Pomembna je tudi optimalna pot robota, ki jo mora sam izračunati brez vnaprej določenih koordinat.

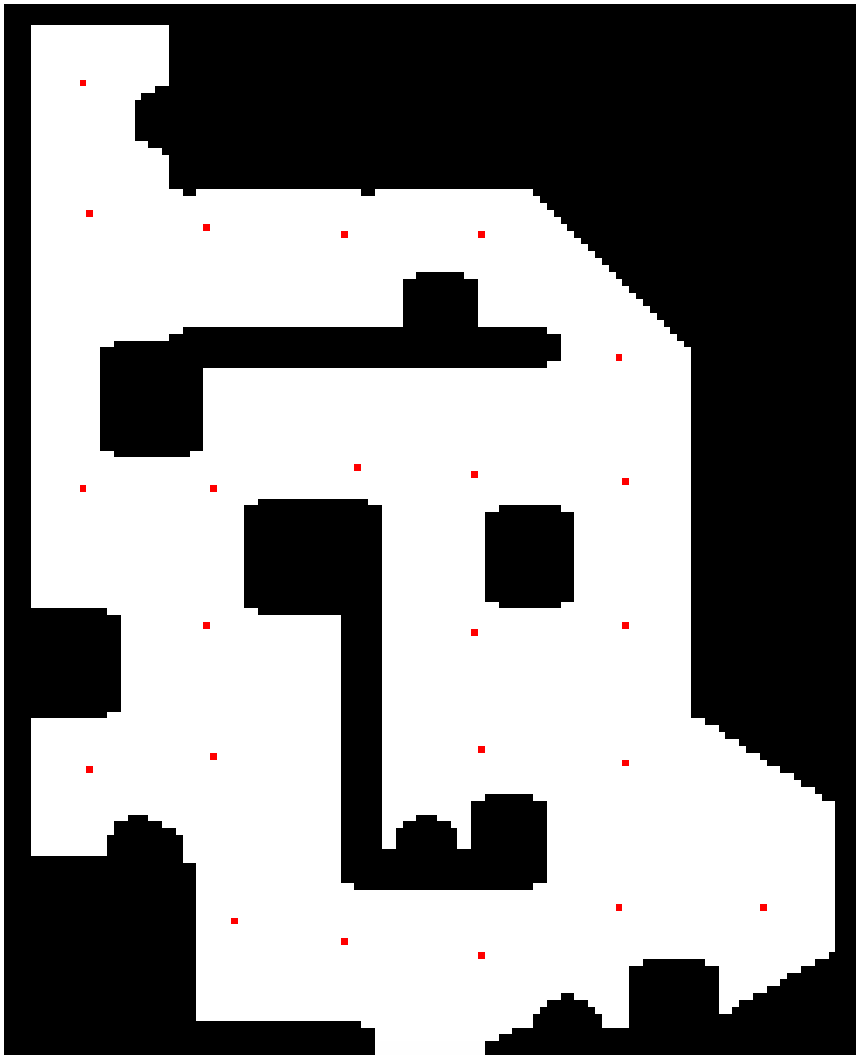
Poglavje 3

Metode

To poglavje je namenjeno podrobnemu pregledu metod, ki smo jih implementirali kot naš odgovor za rešitev te naloge. Poskusili bomo opisati razloge zakaj smo uporabili določene pristope in kaj so bile težave. Prav tako bodo omenjeni spodleteli poskusi in pa kaj bi lahko bili morebitni alternativni pristopi. Naslovi podpoglavij predstavljajo ključne točke naloge, v njih pa bodo navedene rešitve.

3.1 Avtonomna navigacija

Avtonomna navigacija uporablja PGM sliko mape, da določi točke kamor naj bi se robot peljal. Začne s tem, da pretvori vse sive odtenke v črno barvo, ostalo pa v belo. Nato na sliki izvede operacijo *dilate* iz knjižnice OpenCV in izračuna oddaljenost vseh belih slikovnih točk od črnih z funkcijo *cv2.distanceTransform*. Taka slika se razdeli na kose velikosti 20x20 slikovnih točk, s tem, da če velikost slike ni deljiva z 20, bodo robni kosi nekoliko večji. Za vsak kos, ki ima vsaj polovico slikovnih točk belih se določi točka, kamor naj bi robot šel, kot povprečje koordinat belih slikovnih točk na tem kosu. Če ta koordinata pristane na črni točki oziroma je od najbližje črne točke oddaljena za manj kot 5 točk, se jo ignorira. Preostale koordinate določajo cilje, ki bodo podane robotu. Pretvorijo se iz koordinat slikovnih točk v koordinatni sistem mape. Na koncu se izračuna še pot z požrešnim algoritmom. To pomeni, da bo robot izbral točko, ki je najbližja (po zračni razdalji) točki, kjer se trenutno nahaja. Če se bo robot kadarkoli (na primer med obiskovanjem slike) dovolj približal (manj kot 0.25m) kateri izmed točk, bo ta označena, kot da je bila že obiskana.



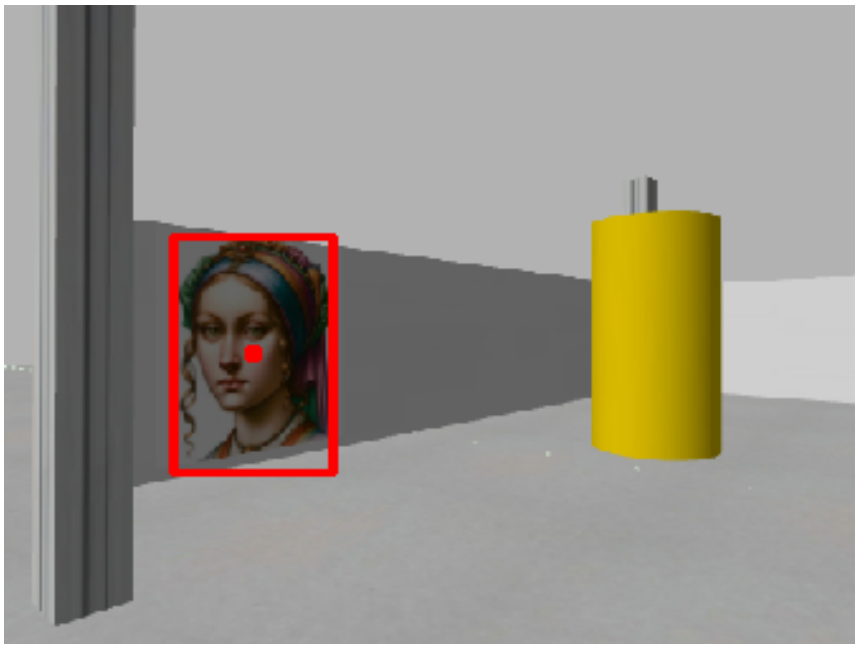
Slika 3.1: Točke dobljene iz avtonomne navigacije

3.2 Zaznavanje obrazov

Obraze robot zaznava na podlagi že naučenega modela YOLOv8. YOLOv8 (You Only Look Once, Version 8) je napreden model globokega učenja za detekcijo objektov v slikah in videih. Temelji na enostopenjskem pristopu, kjer celotna slika preide skozi nevronsko mrežo in izhodi so predikcije koordinat okvirjev in razredi objektov. Model je optimiziran za hitrost in natančnost ter uporablja napredne tehnike, kot so prilagodljive velikosti sidrnih okvirjev in izboljšane konvolucijske plasti.

Za detekcijo obrazov se YOLOv8 najprej nauči značilnosti obrazov iz označenih podatkovnih nizov. Nam tega ni bilo potrebno početi, saj smo prejeli že v naprej natreniran model, ki pa zaznava mnogo več kot samo obraze. Ostale detekcije smo seveda zanemarili. Ko mu podamo novo sliko ali video, model hitro prepozna in označi obraze z okvirji, ki vključujejo njihove koordinate in verjetnosti zaznavanja.

Za ta model smo se odločili, ker je eden izmed bolj znanih modelov na tem področju. Vemo, da deluje zanesljivo in hitro, hkrati pa je njegova implementacija zelo enostavna. Alternative kot so na primer *Faster R-CNN*, *RetinaNet* ali pa celo *Viola-Jones* smo sicer imeli v mislih, ampak ker je YOLOv8 deloval tako dobro jih nismo niti testirali.



Slika 3.2: Zaznan obraz

3.3 Sinteza in prepoznava govora

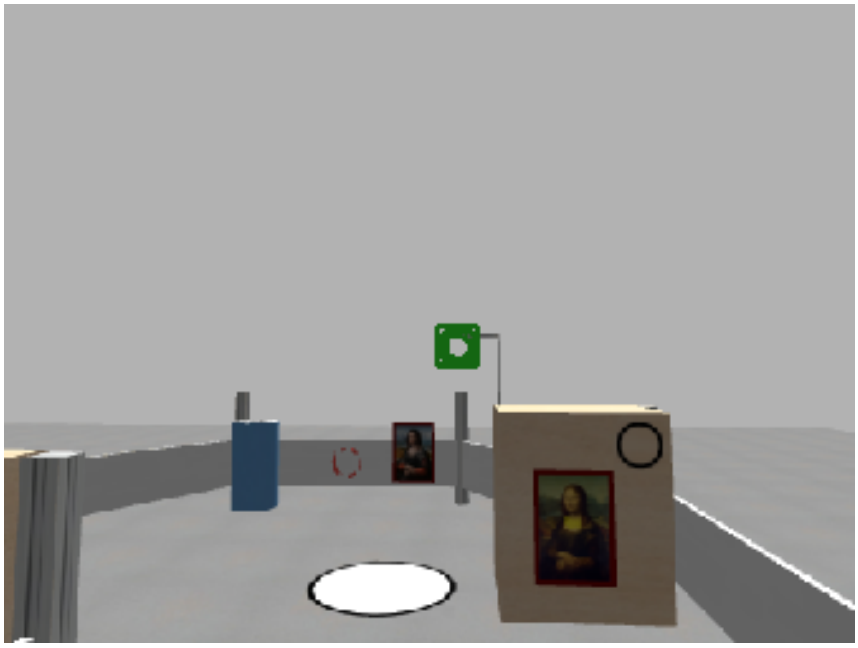
Prepoznava govora vključuje poslušanje zvoka, shranjevanje ter pretvorbo v tekst z uporabo knjižnice, ki je preprosta za integracijo in učinkovita. Nato sledi ločevanje med pozitivnim in negativnim odgovorom, kjer se lahko uporabi ročna izbira besed ali sentimentalna analiza za širši spekter vnosov. Za prepoznavo akcij, barv in oblik se uporablja model za obdelavo naravnega jezika, kar omogoča določitev cilja za robota. Za sintezo govora se uporablja model, ki tekstovni vnos pretvori v zvočni zapis.

3.4 Prepoznava obročev in parkirnih mest

Obroče smo prepoznavali s pomočjo kontur, na katere smo prilagodili elipse, eno za zunanji in eno za notranji rob. Zaradi popačenosti nekaterih zaznanih zaznanih obročev smo sliko očistili s pomočjo morfoloških operacij na binarni sliki (zapiranje in odpiranje). S pomočjo tega smo dobili bolj čiste obrise elips. Izkazalo se je, da je najbolj učinkovito, če za pridobivanje binarne slike uporabimo Otsujevo metodo za iskanje praga, za jedro morfoloških operacij pa uporabimo elipse velikosti 3×3 in 5×5. To smo implementirali tako, da se najprej izvedi iskanje na morfološko obdelani binarni sliki za 3×3 jedro in če ta ne najde ničesar se prične prilaganje elips na konture slike obdelane s 5×5 jedrom. Da pa dejansko gre za obroč in ne le sliko kroga na steni pa preverjamo z opazovanjem globine v središču, če je ta neskončno (0 v kodi zaradi implementacije globinske

slike), se obroč obravnava kot pravi, drugače pa ga zavržemo. Parkirna mesta smo prepoznavali le z iskanjem kontur in prileganjem elips.

Slika 3.3: Morfološko obdelana binarna slika



Slika 3.4: Zaznan zelen obroč

3.5 Parkiranje robota

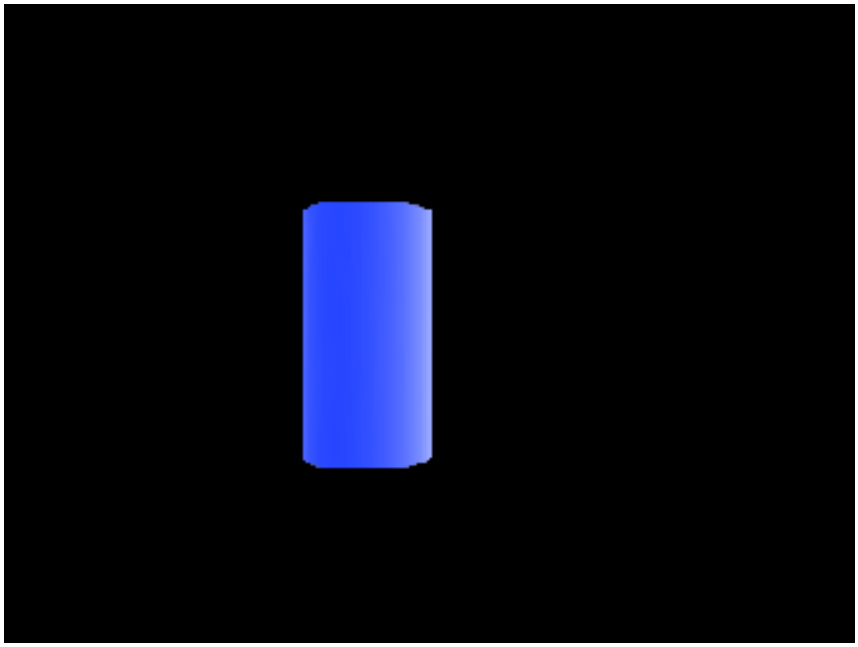
Na začetku smo se parkiranja lotili na preprost način: Robot je imel kamero na robotski roki usmerjeno v tla in ko je zaznal krog, je primerjal koordinate središča tega kroga (v slikovnih točkah) z koordinatami robota (v slikovnih točkah je to $X = \frac{W}{2}$ in $Y = H$. Nato se je robot zavrtel tako, da sta bili ti dve koordinati čim bolj poravnani in se premaknil naprej, dokler nista bili točki dovolj blizu (po potrebi se je rotacija vmes popravila).

To je dalo dobre rezultate na začetku, vendar pa so se pri nekaterih parkiriščih pojavile težave, na primer: robot je zaznal krog le enkrat, zato se je večno vrtel okrog svoje osi da bi popravil kot, kar mu ni uspelo. Zato smo se odločili za alternativno strategijo. Ko robot prvič zazna parkirišče, si zapomni središčno točko in jo pretvori v koordinatni sistem mape, nato pa popravlja svojo pozicijo in rotacijo tako, da se tej točki čim bolj približa. Pri tem uporablja funkcionalnost, ki robotu omogoča da vidi svojo pozicijo in rotacijo na mapi.

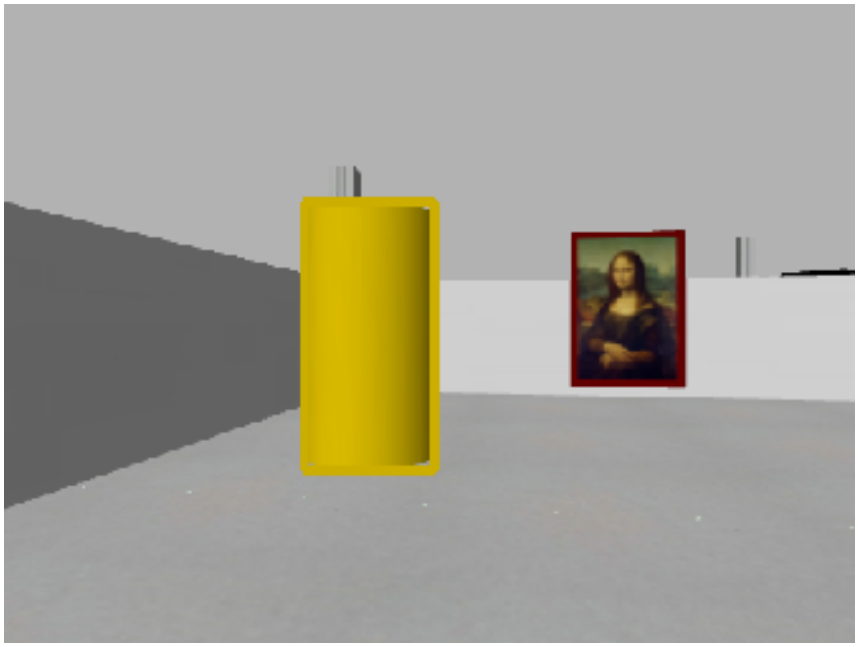
3.6 Prepoznava cilindrov

Cilindre smo sprva poskusili prepoznavati z obetavno a nekoliko težjo metodo. Lidar, ki je nameščen na robotu, nam je namreč nenehno vračal podatke o globini. S temi podatki, je bilo mogoče rekonstruirati 3 dimenzionalno podobo simulacije. Naša ideja je bila, da oblak tridimenzionalnih točk algoritem splošči v zgolj dve dimenziji v ravnino. Nato pa poskusi na te točke, ki so v ravnini narisati krog. Ker je cilinder okrogel, bi morale točke na ravnini že takoj predstavljati nek krožni odsek katerega notranji kot je manjši od 180°. Na takšen odsek pa je projeciranje kroga preprosto. Izkazalo se je, da je ta naloga dokaj zahtevna, saj oblak točk vključuje dosti šuma, hkrati pa je procesno zahtevna obdelava zaradi same količine točk. Algoritem bi moral biti odporen na šum in hkrati dovolj strog, da ne zaznava pravokotnih ali celo ravnih površin.

Prepričani smo bili, da bi ob dodatnem razmisleku ta algoritem lahko deloval zelo dobro, a vendar smo se zaradi časovnih omejitev odločili razviti alternativen algoritem, ki je posebej narejen za našo nalogo. Naloge smo se lotili s pomočjo barvne segmentacije. Sliko smo pretvorili v HSV barvni prostor in v temu prostoru poiskali relevantne maske za barve, ki smo jih poiskali. Te maske (vsako posebej) nato obdelamo s pomočjo morfoloških operacij, da se znebimo majhnih delov, za jedro uporabljamo kvadrat velikosti 9×9 s tremi iteracijami odpiranja. V maski tako dobimo le večje dele, kjer je bila barva zaznana. Da se znebimo potencialno napačnih detekcij pa uporabimo algoritem za sikanje povezanih struktur in vzamemo le največjo zaznano, pod pogojem, da je ta na obeh polovicah slike. Dobljene maske nato združimo v eno samo, kar omogoča detekcijo več kot enega cilindra hkrati. Združeno masko združimo z inverzno sliko, ki omogoča, da ne izgubimo črne barve. Na to sliko nato s pomočjo kontur prilegamo pravokotnike. Sprejmemo jih kot cilindre le če je če je njihova ploščina dovolj velika in če razmerje med višino in širino ustreza razmerju za cilindre.



Slika 3.5: Maska rumenega cilindra na inverzni sliki



Slika 3.6: Zaznan rumen cilinder

3.7 Branje QR kode

Implementacija branja QR kode je presenetljivo preprosta, saj nam to omogoča že knjižnica *OpenCV*. Ta rešitev se izkaže za dovolj robustno brez kakršnih koli nadgraditev ob predpostavki, da imamo pravilno nastavljeno kamero za branje.

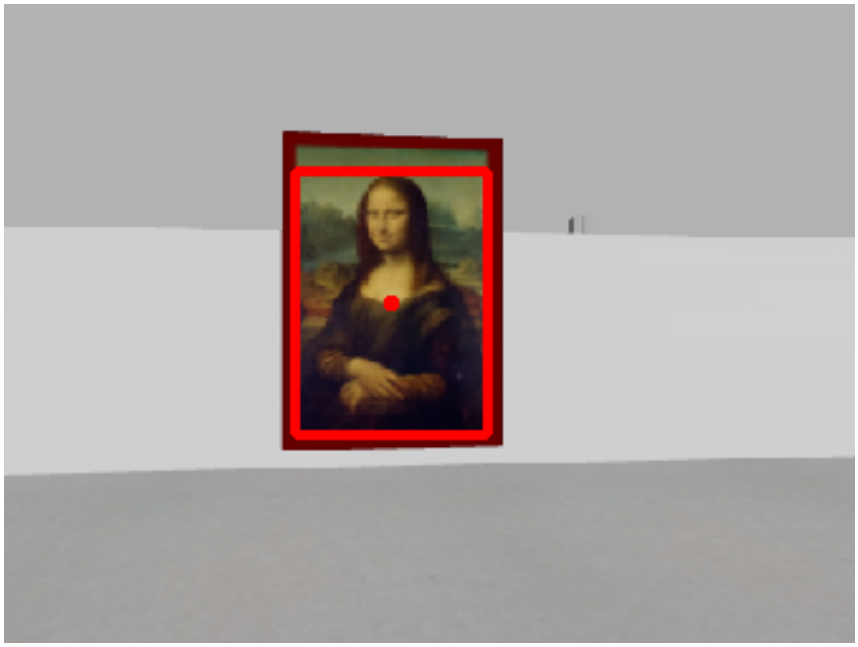
3.8 Prepoznava slik

Metoda za prepoznavo slik je zelo podobna metodi za prepoznavanje obrazov. Prav tako se namreč uporablja model YOLOv8, le da je tokrat ponovno naučen specifično na našem naboru podatkov.

Torej za osnovo smo vzeli originalen v naprej naučen model YOLOv8, nato pa smo zbrali nekaj čez 5000 instanc slik in obrazov. To smo naredili tako, da smo zagnali simulacijo in snemali izhod kamere, ko je bil robot okrog obrazov ali slik. Poskrbeli smo tudi, da smo beležili pozicije objektov v sliki. Nastala je podatkovna zbirka okrog 5000 slik in 5000 tekstovnih datotek, ki se paroma navezujejo na slike. Format teh tekstovnih anotacij je sledeč:
"identifikator razreda, center objekta po x, center objekta po y, širina, višina". Pri tem smo previdni, da so podatki normirani med 0 in 1.

Slika 3.7: Primer iz učne množice z nenormiranim zapisom: 0 205 114 51 87

Trenirali smo več iteracij modelov z različnimi ciljnimi razredi. Poskušali smo namreč doseči, da bi model ločil med vsako sliko in obrazom posebej - za identifikacijo različnih. A smo hitro ugotovili, da je najbolje, da imamo samo dva razreda, in sicer razred 0 - slika in pa razred 1 - obraz. Model ki smo ga naučili sicer ima več razredov - za vsako sliko in obraz svojega ampak se ti razredi programsko ločijo le na 2, torej slika in obraz.

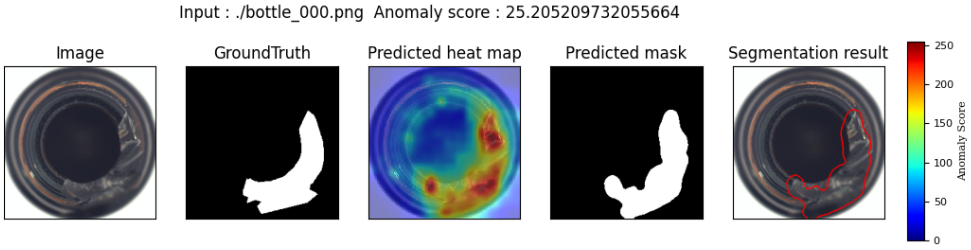


Slika 3.8: Zaznana slika

Izkazalo se je, da model zelo dobro loči med slikami in obrazi, vendar pa pogosto zazna tudi popolnoma ne korelirane objekte. Za to smo vpeljali dvojno detekcijo, ker mora sliko zaznati tako pred naučen model, kot tudi naš model. Rezultat je bil presenetljivo dober, ni bilo več lažnih detekcij, hkrati pa je robot zanesljivo ločil med sliko in med obrazom.

3.9 Prepoznava napak na sliki

Za prepoznavanje napak na sliki smo uporabili [5] PaDiM. PaDiM (Patch Distribution Modeling) je metoda za detekcijo in segmentacijo anomalij v slikah. Deluje tako, da najprej iz normalnih slik izračuna značilnosti z uporabo konvolucijskega nevronske mreže. Nato te značilnosti razdeli na manjše obliže in za vsak obliž izračuna povprečje in kovariančno matriko, kar omogoča modeliranje normalne distribucije značilnosti. Ko obdelujemo novo sliko, se izračunane značilnosti primerjajo s tem modelom z Mahalanobisovo razdaljo, ki meri odstopanja od normalne distribucije. Področja z visokimi rezultati so označena kot anomalije, kar omogoča natančno prepoznavanje nepravilnosti v slikah.



Potrebno je določiti mejo, kjer se bodo področja določila kot anomalija. Da bo to robustno delovalo na poljubni anomaliji, meja ne sme biti konstantna, temveč se mora prilagajati sliki.

Odločili smo se, da jo določimo z standardnim odklonom odstopanj. To omogoči, da slike, z zelo značilnimi in lokalnimi napakami zvišajo mejo in s tem ne prikazujejo dodatnega šuma. Slikam, ki pa imajo morebiti več napak in zaradi tega večji standardni odklon, pa jim omogoči da prikaže večjo površino slike, kjer se te napake nahajajo.

Za PaDiM smo se odločili, ker se nam je zdel preprost za implementacijo hkrati pa je ponujal vse potrebno za dokončanje naloge. Predvsem pa zato ker je že kmalu po raziskovanju pokazal obetavne rezultate. Obstajajo seveda alternativni pristopi, ki bi morda delovali še bolje, vendar pa nam jih v sklopu tega projekta žal ni uspelo testirat.

Kaj pa če je na vhodu pravilna slika? Algoritem se izvaja normalno do točke, kjer se preračuna tako imenovan *anomaly score* oziroma stopnja anomalije. Ta se izračuna preprosto tako, da se pogleda točka z največjim odstopanjem od originalne slike. Na podlagi te vrednosti se lahko sami odločimo ali je zares anomalija, saj bo zaradi prilagodljive meje, algoritem vedno poiskal največja odstopanja na sliki. Ta so pri originalni sliki navadno zgolj šum.

Pomanjkljivosti tega pristopa so, da morajo biti pogoji, v katerih se slike primerjajo, dokaj podobni z zelo malo ali brez šuma. Prav tako morajo biti slike stisnjene na enake velikosti in obrezane.

Edino drugo metodo, ki smo jo poskusili je bil namenski trening modelov za prepoznavo vseh napak posebej. Čeprav se je izkazalo da so takšni modeli zelo natančni in presenetljivo učinkoviti, so tudi zelo omejeni. Delujejo namreč zgolj na treniranih podatkih in zatajijo takoj, ko vidijo prej ne poznano sliko.

Poglavje 4

Implementacija in integracija

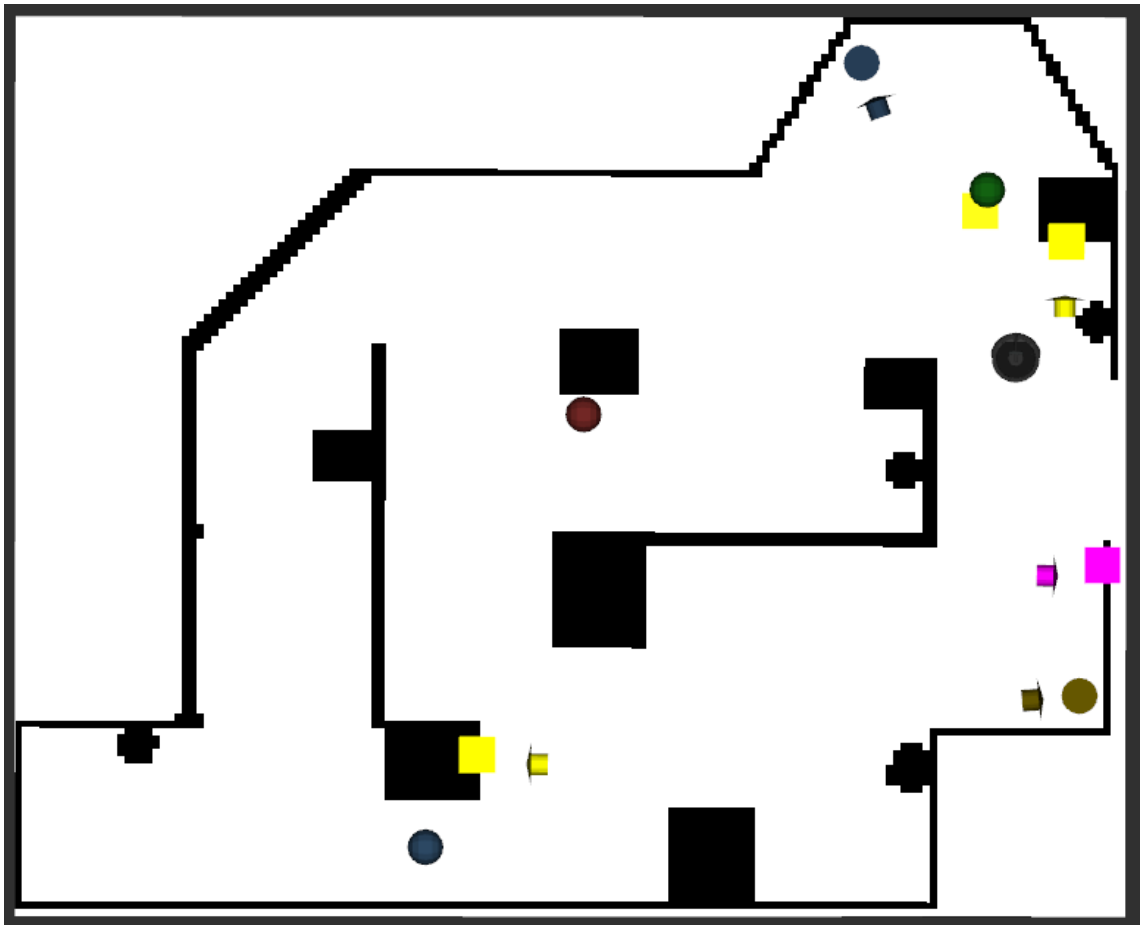
Najbolj kompleksen in časovno zahteven del naloge je seveda bilo samo programiranje rešitev tako, da so se uspešno povezovale. Ko smo neko metodo implementirali smo jo najprej testirali brez ostalih, šele nato smo se lotili povezovanja.

4.1 Implementacija prepoznave obrazov, slik in oblik ter zvokovna obdelava

4.1.1 Koordinate

Zaznavanje posameznih oblik in ljudi ter razlikovanje med slikami in obrazi je prikazano na RVIZ zemljevidu 4.1. Iskanje koordinat za posamezne oblike je sledeče:

- **Obroči 3.4:** iz obdajajoče škatle smo odstranili vse elemente z globino v neskončnosti. Ostale vrednosti smo sortirali po njihovi oddaljenosti od središča $(0, 0, 0)$ in vzeli sredinski element. Izkazalo se je, da je ta pristop dovolj robusten tudi za zaznavanje iz daljave. Označeni z kroglami v barvi obročev.
- **Parkirna mesta 3.4:** vzeli smo center kroga. Označeni z rumeni kvadrati (ki so v bližini obročev in nimajo pripadajočih puščic).
- **Cilindri 3.6:** koordinate pridobimo iz sredinske točke na cilindru, zaradi njihovega dovolj majhnega radija se to izkaže za dovolj robustno rešitev. Označeni z krogi v njihovi barvi.
- **Točke za branje QR kod 3.7:** pridobimo jih tako, da izračunamo razliko med točko cilindra ter pozicijo robota, to normiramo, zmnožimo z 0.4 in odštejemo od točke cilindra. Orientacijo dobimo iz orientacije robota. Označene z puščicami v barvi pripadajočega cilindra.
- **Obrazi 3.2 in slike 3.8:** za točke vzamemo centralne koordinate zaznanih slik oz. obrazov. Označene z kvadrati rumene (za slike) in vijolične (za osebe) barve.
- **Točke za pogovor in detekcijo anomalij:** Na sliki poleg središča $\mathbf{s}$ vzamemo še točki ki se nahajata na polovici od središča do zgornjega $\mathbf{t_1}$ in levega roba $\mathbf{t_2}$ (saj so robovi včasih izven slike in v neskončnosti). S temi točkami izračunamo normalo po formuli $\mathbf{n} = (\mathbf{t_1} - \mathbf{s}) \times (\mathbf{t_2} - \mathbf{s})$. Točko za pogovor oz. detekcijo pa dobimo po formuli $\mathbf{p} = \mathbf{s} - 0.5 \cdot \frac{\mathbf{n}}{\|\mathbf{n}\|}$. Orientacijo izračunamo iz vrednosti $\arctan(n_y, n_x)$. Označene z puščicami rumene (za anomalije) in vijolične (za pogovor) barve.



Slika 4.1: Prikaz nekaterih zaznanih oblik, obrazov in slik
Opomba: *Slika je narejena v alternativnem svetu, z ročnim premikanjem robota.*

4.1.2 Unikati

Vsaka od zaznanih oblik oziroma ljudi uporablja instanco naše knjižnice, ki omogoča hrambo unikatov na podlagi njihove lokacije. Na ta način se izognemo detekciji duplikatov. Poleg samih koordinat s pomočjo katerih računamo razdaljo in na podlagi katerih določimo ali gre za duplikat ali ne, hranimo tudi ime barve ter RGB zapis barve, če ne gre za obraz ali sliko.

4.1.3 Barva

Barve posameznih oblik smo pridobili na sledeč način:

- **Obroči 3.4:** naredili smo masko slike, kjer se naš obroč nahaja (vzeli smo vse med notranjo in zunanjo elipso, ostalo pa nastavili na 0). Iz tega kar nam je ostalo smo odstranili barvo ozadja in nato poiskali najbolj pogosto barvo.
- **Cilindri 3.6:** zaradi natančne detekcije pridobimo barvo kar iz centra cilindra, saj se izkaže

da je ta vsaj tako svetla kot najbolj pogosta in na ta način dobimo najbolj reprezentativno barvo iz cilindra.

Imena barv pridobivamo s pomočjo knjižnice *webcolors*, ki uporablja standardna imena iz *CSS2* ali *CSS3*. Ime poiščemo na podlagi tiste barve iz izbranega zapisa barv, ki ima najbližji RGB zapis (izračunano kot norma vektorja razlike). Sprva smo uporabljali *CSS3* barve, ampak smo za potrebe tretje naloge uporabili *CSS2* barve, saj je teh veliko manj. Ročno smo zapisali funkcijo, ki preslika te barve v nekaj v naprej določenih barv (zaradi lažjega usmerjanja robota k pravim obročem, sicer bi namesto rdeče kot *red* morali uporabljati *maroon*).

4.1.4 Zvočna obdelava in razumevanje govora

Ta del naloge je razdeljen v pretežno dva dela. Prvi je sinteza govora, ki je dokaj preprosto implementiran. V naprej so določeni stavki, ki jih robot lahko pove. Nato pa se z uporabo [6] coqui-ai TTS *(text to speach)*, zažene model, ki pove stavek. Mi smo uporabili model v angleškem jeziku *tacotron2-DDC*.

Drug del naloge je nekoliko bolj kompleksen. Robot mora poslušati zvok, ga shraniti kot tekstovni zapis in nato iz njega pridobiti podatke. Za poslušanje in beleženja zvoka v besedilo je uporabljena *speech_recognition* knjižnica.Za pridobivanje pomena pa je uporbljena NLTK knjižnica in SpaCy knjižnica skupaj z *en_core_web_sm* modelom za naravno obdelavo jezika.

Implementirano pa je tako, da program počaka, da se robot postavi nasproti obraza, dokler ne prejme sporočila na ros temo. Nato generira zvok iz vhodnih podatkov *"Hello! Do you know where the mona lisa painting is?"*. Posluša do 5 sekund. Če v tem času ne prepozna niti ene besede, je mogoče v terminal tudi ročno vnesti odgovor. Odgovor nato pošlje v NLTK-jev semantični analzirator, iz katerega pridobi numerično oceno semantike. Če je več kot 0,05 vrne True, drugače vrne False. Če je prejel True, generira naslednje besedilo v zvok: *Šo where is it?"*. Ponovno čaka na odgovor, le da tokrat tega ne pošlje v semantični analizator, temveč v pridobivanje akcije, barve in objekta iz odgovora. Prepozna glavne glagole (akcije), barve in predmete, ter razlikuje med pozitivnimi in negativnimi akcijami (npr. "do"ali "don't"). Rezultate shrani v slovar, ki vsebuje sezname akcij, barv in predmetov za oba tipa akcij. Izkaže se da potrebujemo za našo nalogo zgolj podatke o barvi saj vedno parkira pod obroč. Zato program vzame zgolj barvo in na podlagi nje določi nov cilj. Če v odgovoru ni barve, robot ne določi novega cilja in nadaljuje proti staremu.

4.2 Implementacija navigiranja, premikanja in parkiranja

Robot uporablja avtonomno navigacijo za navigiranje po mapi. Ko se zažene *commander*, bo ta najprej zgradil točke, ki jih bo obiskal ter njihov vrstni red. Po teh točkah se bo premikal, dokler ne bo zaznal obraza, ki ga še ni obiskal. Če bo robot zaznal ne obiskan obraz medtem ko je zaseden (ker je že na poti do drugega obraza, je sredi *undockinga*...), si bo zapomnil njegovo lokacijo in ga šel obiskati kasneje. Ko robot pride do obraza bo dobil informacijo o tem, kje naj parkira. Če je bil kateri od teh obročev že zaznan, se bo robot takoj začel premikati do njih, sicer bo to storil, ko bodo zaznani. Ko bo zaznano parkirišče, ki je dovolj blizu iskanega obroča, bo robot roko premaknil tako, da bo kamera gledala pravokotno navzdol, navigacijskemu sistemu bo poslan nov cilj - lokacija središčne točke parkirišča, ter vozlišče zadolženo za parkiranje bo prejelo signal, da naj se aktivira.

Ko je robot v načinu parkiranja, bo z kamero na robotski roki (ki bo gledala v tla) iskal kroge s pomočjo *hough transform-a*. To se bo dogajalo po tem ko prednja kamera že zazna parkirišče kamor naj bi robot parkiral, spusti kamero in napoti robota v smer parkirišča s pomočjo navigacijskega sistema. Ko je torej krog najden, se koordinate sredine tega kroga pretvorijo iz robotovrga

koordinatnega sistema (pointcloud) v koordinatni sistem mape. Nato robot uporablja svojo pozicijo in pozicijo centra kroga, da izračuna kotno hitrost in linearno hitrost. Te posreduje cmd_vel kot sporočilo Twist. Če robot ni pravilno obrnjen, se najprej pošlje samo kotna hitrost, sicer pa (če je robot obrnjen pravilno z napako manj kot 5 stopinj) se pošilja linearna hitrost. Ko je robot enkrat dovolj blizu želene koordinate (v našem primeru 0.1m), se ustavi in parkiranje je končano.

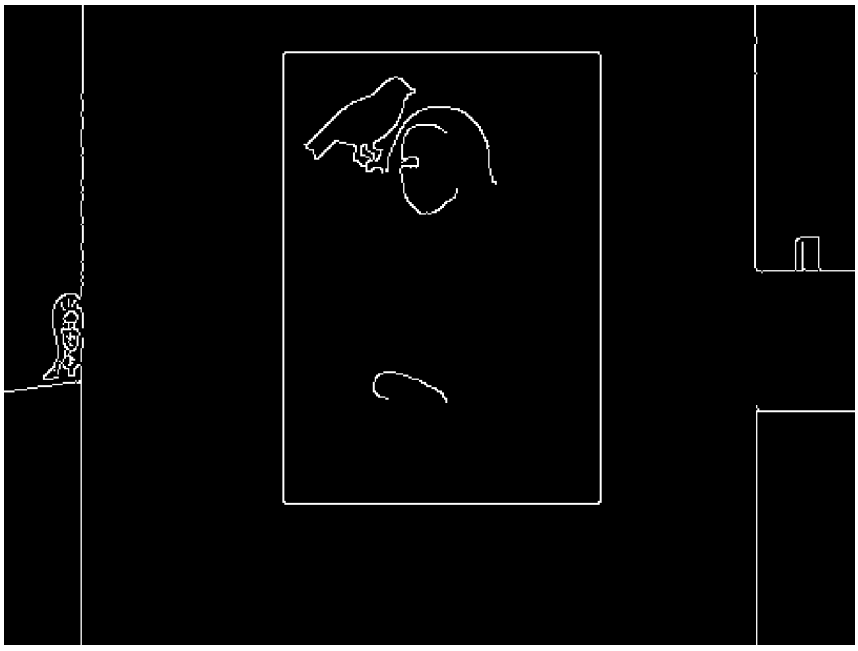
Po parkiranju se robot zavrti okoli svoje osi, ter se premakne do najbližjega cilindra. Pri tem kamero premakne v položaj primeren za branje QR kod iz cilindra. Ko prebere QR kodo, nadaljuje z premikanjem, razen, če ta QR koda vsebuje originalno sliko monalize. V tem primeru bo robot namesto iskanja obrazov in obročev iskal slike monaliz, šel do njih in preveril ali so originalne ali ne. Anomalije bo tudi prikazal. Ko najde originalno sliko monalize, se ustavi.

4.3 Implementacija detektorja anomalij

Da bi čim bolj zmanjšali variabilnost pri postavitvi robota, kadar zaznava anomalije, smo se odločili za pristop, kjer robot vedno pride na 0.5m od slike ter je obrnjen pravokotno na sliko. Zaradi značilnega okvirja slike, smo se odločili to uporabiti tudi pri detekciji anomalij, da smo še dodatno zmanjšali variabilnost slik, ki jih podamo našemu modelu.

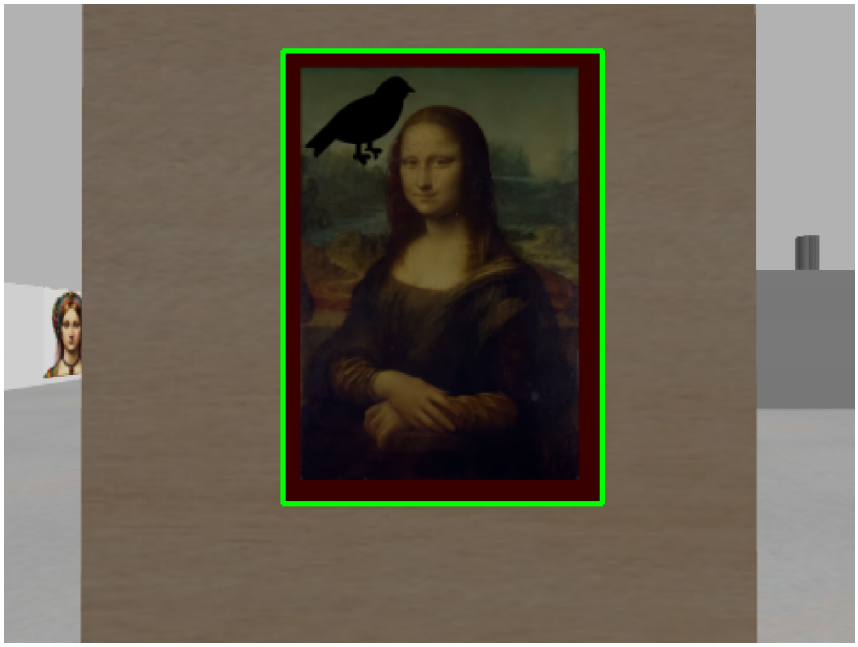
4.3.1 Prepoznavanje okvirja

Okvir prepoznamo s pomočjo uporabe Canny detektorja robov. Na sliko robov nato prilagodimo konture, na katere prilagodimo pravokotnike. Najdenih je sicer ogromno ampak zaradi naše pozicije v primeru, ko nas dejansko zanimajo, je dovolj, da vzamemo le centralnega, kjer hkrati zagotovimo pravilno velikost in razmerje.



Slika 4.2: Slika robov okvirja

Ko je robot postavljen na pozicijo za detekcijo anomalij, obrežemo sliko glede na zaznan pravokotnik. Te pravokotnike sicer zaznavamo stalno ampak ker je detekcija anomalij prožena le takrat, ko je robot v poziciji bo najden pravokotnik vedno okvir slike.



Slika 4.3: Zaznan okvir slike

4.3.2 Prepoznavanje anomalij

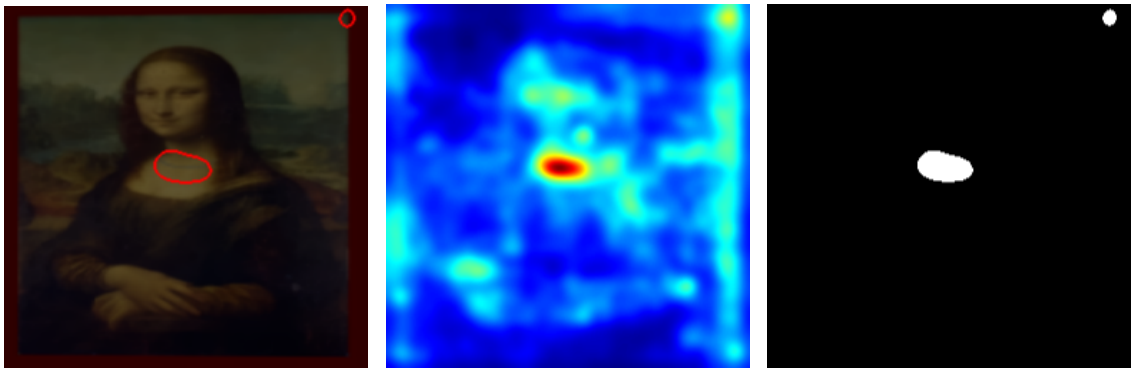
Glede na prej določen okvir dobimo obrezano sliko, na kateri poženemo naš model. Ta se požene tako, da posluša na dodeljeno temo (ros topic) in prejme prazen signal, ko je v poziciji. Sliki se nastavi velikost na 256 x 256 slikovnih točk, nato se normalizira s srednjimi vrednostmi [0.485, 0.456, 0.406] in standardnimi odkloni [0.229, 0.224, 0.225], pred tem pa deli vse vrednosti v sliki z 255. Zadnji del predobdelave je transponiranje in dodajanje prazne dimenzije na začetek. Iz slike z dimenzijami *višina, širina, kanal* naredi sliko z dimenzijami *empty, kanal, višina, širina*. Slika je sedaj pripravljena na ekstrakcijo značilk, za kar v splošnem poskrbi [4]ailia SDK.

Ko imamo značilke, preračunamo razdalje od originala in dodamo Gaussovo glajenje. Sledi normalizacija točk v zglajeni matriki razdalj. Anomalije so praktično že znane, vendar pa dodamo še izračun stopnje anomalije, ki je trivialen, če sledimo opisu v 3.9, ki pravi, da samo pogledamo največjo razdaljo v matriki.

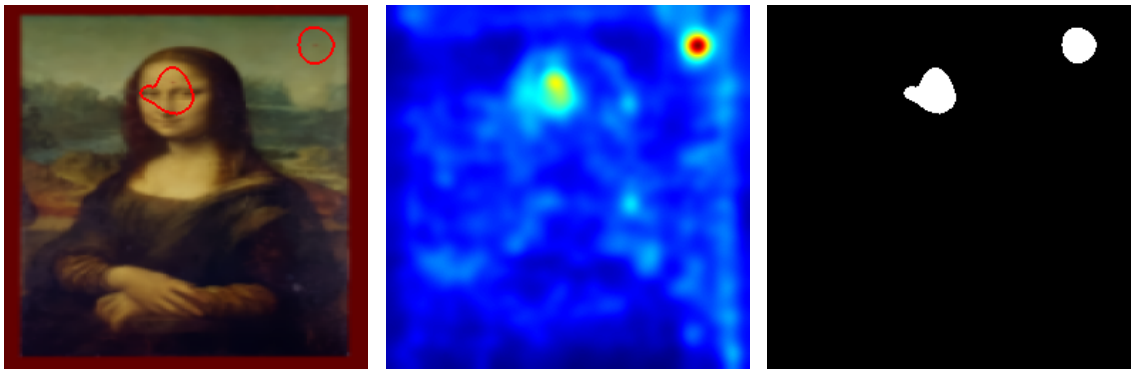
Za konec poskrbimo še za vizualizacijo. Iz matrike razdalj pridobimo standardno deviacijo in povprečje, s katerim določimo mejo po algoritmu 1. S to mejo pridobimo črno-belo masko slike, ki označuje področje napake. Knjižnica skimage ponuja funkcijo za obrobo maske, ki jo združimo z originalno sliko. Na koncu dodamo barvne popravke na matriko razdalj oziroma *heatmap*. Če je stopnja anomalije nad 17, se prikažejo okna, kot prikazujeta sliki 4.6 in 4.5, sicer se v terminal le izpiše, da anomalija ni bila zaznana. Na sliki 4.4 pa lahko vidimo primer ko algoritmu ne uspe izločiti šuma.

Algorithm 1 Izračun praga na podlagi srednje vrednosti in standardne deviacije maske

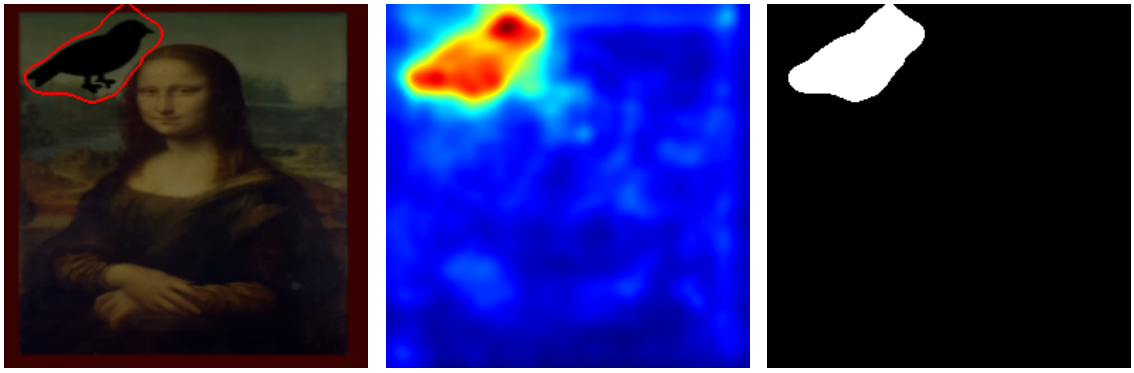
Vhod: maska (seznam vrednosti)
Izhod: prag (threshold)
Izračun srednje vrednosti maske:
$mean\_score \leftarrow$ srednja vrednost maske
Izračun standardne deviacije maske:
$std\_dev \leftarrow$ standardna deviacija maske
Privzeti prag:
$threshold \leftarrow mean\_score + 2 \times std\_dev$
Prilagoditev praga glede na standardno deviacijo:
if $std\_dev < 10$ **then**
    $threshold \leftarrow mean\_score + 2.5 \times std\_dev$
else if $std\_dev > 30$ **then**
    $threshold \leftarrow mean\_score + 1.5 \times std\_dev$
end if
Vrni prag:
return *threshold*



Slika 4.4: Rezultat detekcije napake s šumom v zgornjem desnem kotu



Slika 4.5: Primer uspešne detekcije anomalij



Slika 4.6: Primer uspešne detekcije anomalij

4.4 Gradnja povezane rešitve

4.4.1 Teme

Vozlišča, ki imajo specifično nalogo skupaj povezuje vozlišče RobotCommander. Vozlišča med sabo komunicirajo z uporabo "publisherjev"in šubscriberjev". Commander ima dve fazi operacije. Ali išče QR kode v iskanju slike Mona Lise, ali pa išče dejanske slike Mona Lise na mapi in preverja njihovo pristnost. Commander uporablja naslednje teme za komuniciranje z ostalimi vozlišči:

Sprejemniki:

- /greet_ppl_marker: ta tema vsebuje lokacije (tip Marker) vseh unikatnih zaznanih obrazov (brez slik). Ko commander dobi novo lokacijo reagira takole: Če je v drugi fazi, ga obrazi ne zanimajo več, sporočilo ignorira. Če je robot zaseden (sredi obiska drugega obraza, branja QR kode, ...), si lokacijo obraza shrani kot ne obiskano. Obiskal jo bo potem ko izpolni prejšnje naloge (razen če vmes že najde pravo Mona Liso). Sicer obišče obraz in z njim "komunicira". Rezultat komunikacije interpretira v en sam seznam, ki je seznam barv obročev, ki naj bi jih robot obiskal. Ta seznam se nato kot String ([green, red] -¿ "green;red") sporočilo pošlje na temo "/searched_ring", na katero posluša vozlišče zadolženo za zaznavo obročev. To vozlišče interno že shranjuje iskane barve od prejšnjih namigov, zato bodo na vozlišče vplivali samo namigi z novimi barvami.
- /greet_paint_marker: ta tema vsebuje lokacije vseh unikatnih zaznanih slik Mona Lise. V prvi fazi si commander te lokacije zgolj shranjuje, na začetku druge faze pa gre nato po shranjenem seznamu obiskovati slike, ki si jih je shranil (v takem vrstnem redu, da izmed ne obiskanih slik najprej obišče tisto, ki je najbližje robotovi trenutni poziciji). Ko pride do slike objavi prazno sporočilo na /detect_anomalies, ki signalizira vozlišču za anomalije da naj preveri avtentičnost Mona Lise pred katero robot stoji. ko bo vozlišče končalo s procesiranjem slike, bo objavilo sporočilo na /anomaly_detector/anomaly_detected, ki vsebuje zastavico (Bool) z informacijo o tem, ali je slika avtentična. Če je, se robot ustavi, če ni, nadaljuje pot.
- /cylinder_marker: ta tema vsebuje lokacije vseh unikatnih cilindrov, ki jih zazna vozlišče za cilindre. V vsakem primeru si commander te lokacije zapomni in jih uporabi kasneje

- /qr_scanner/found: vozlišče zadolženo za branje QR kod na to temo pošlje sporočilo, ki vsebuje zastavico, zastavica je postavljena na "True", če je bila QR koda z Mona Liso najdena, "False", če QR koda ni vsebovala Mona Lise. V primeru da je zastavica "True", bo commander prešel v drugo fazo. Sicer bo nadaljeval z iskanjem.
- /parker_node/parkplatz_found: ko commander sprejme prazno sporočilo na to temo, bo prekinil trenutni cilj, nadzor bo prevzelo vozlišče zadolženo za parkiranje z uporabo /cmd_vel. Sporočilo se pošlje po tem, ko je kamera že obrnjena navzdol in je bilo parkirišče najdeno.
- /green: prvotno je bila ta tema uporabljena za zelene obroče, saj smo iskali zgolj te, sedaj je uporabljena za vse iskane obroče, ime pa je ostalo iz zgodovinskih razlogov. To sporočilo vsebuje lokacijo enega od iskanih obročev (Marker). Ime Marker-ja je barva tega obroča. Če je robot zaseden si sporočilo shrani za kasneje. Če je robot v drugi fazi, sporočilo ignorira. Sicer commander poda nov cilj navigacijskemu sistemu in to je lokacija tega obroča.
- /parking_coordinates: Ta tema vsebuje lokacijo parkirišča, kjer naj bi robot parkiral. Parkirišče je že vnaprej preverjeno da je dovolj blizu iskanega obroča. Commander prekine trenutni cilj, spusti roko, aktivira vozlišče zadolženo za parkiranje in pošlje navigacijskemu sistemu cilj, ki je središče zaznanega kroga.
- /parker_node/parking_finished: Ta tema sprejme prazno sporočilo takrat, kadar vozlišče zadolženo za parkiranje konča s svojim opravilom. Robot se zavrti za 360 stopinj, pri tem si zapomni novo videne cilindre, če je kakšen, nato pa se premakne do najbližjega cilindra in aktivira vozlišče za branje QR kod.

Oddajniki:

- /detect_anomalies: Commander bo na to temo poslal prazno sporočilo z namenom da se aktivira vozlišče zadolženo za zaznavanje anomalij in avtentičnosti Mona Lise. To sporočilo se pošlje v drugi fazi, ko robot stoji pred sliko Mona Lise.
- /searched_ring: Tema se uporablja za posredovanje informacije o iskanih obročih vozlišču za zaznavanje obročev. Sporočilo se pošlje po dialogu z obrazom.
- /robot_commander/go_to_ring: Ko se bo commander odločil obiskati nek obroč bo poslal to sporočilo. Namen je, da bo vozlišče za zaznavanje parkirišč vedelo, kateri obroč robot trenutno obiskuje, da bo potem lahko preverilo ali zaznano parkirišče pripada temu obroču.
- /look_for_qr: Tema s praznim sporočilom, ki aktivira vozlišče za branje QR kod.
- /parker_node/enable_parking: Tema s praznim sporočilom, ki aktivira vozlišče za parkiranje.

4.4.2 Prioritete premikanja v prvi fazi

Ko robot v prvi fazi zaključi z dialogom, najde QR kodo, ki ne vsebuje slike Mona Lise, konča z "undocking-om, pokliče funkcijo continue_moving. Ta funkcija stori sledeče: Če obstaja kakšen zaznan obraz, ki še ni bil obiskan, ga obišče. Sicer, pogleda če obstaja kakšen zaznan obroč, ki še ni bil obiskan in ga obišče. Če ni znanih ne obiskanih obrazov ali obročev se robot premakne na naslednjo ne obiskano točko avtonomne navigacije.

Poglavje 5

Rezultati

Ob končani implementaciji kode, je bil robot sposoben opraviti zadano nalogo opisano v podpoglavju 2.5.

Ocenjevali smo sposobnost avtonomnega navigiranja. Se je robot kje zataknil? Je obiskal celotno polje? Je izbral učinkovito oziroma hitro pot?

Nadaljevali smo s sposobnostjo prepoznave obrazov, slik in oblik. Ali loči slike in obraze med sabo? Prepozna obraze robustno? Kaj pa ostale oblike? Lahko zanesljivo prepozna in zabeleži obroče, parkiran mesta in cilindre? Ko prepozna obraz, ali vpraša za namig? Razume odgovor?

Ne nazadnje nas pa je seveda zanimalo kje so napake v slikah? Jih robot zna prikazati? Sploh loči originalno sliko od ostalih?

Rezultati so bili zadovoljivi. Robot je uspešno navigiral skozi svet, kljub občasnim težavam v ozkih hodnikih, ki smo jih zmanjšali s prilagoditvijo nastavitev. Vedno je našel najkrajšo pot do cilja in se uspešno postavil na parkirno mesto ter pred slike, obraze in cilindre. Prepoznava obrazov in oblik je bila zelo natančna, brez napačnih pozitivov, in robot je primerno reagiral na preprosta navodila, kot na primer "Park under the red ring".

Posebej zadovoljni smo bili z detekcijo napak na slikah. Robot je zanesljivo izoliral in prikazal vse testirane napake, kar kaže na robustnost in učinkovitost algoritma, ki deluje tudi na nikoli prej videnih napakah.

Skupno so rezultati pokazali, da robot deluje zelo dobro pod normalnimi pogoji, brez motenj, kar potrjuje uspešnost naše rešitve.

Poglavje 6

Razdelitev dela

Pri projektu smo sodelovali trije člani: Jure Pahor, Domen Ogorevc in Grega Potočnik. Razdelitev dela v odstotkih prikazuje tabela 6.1 Na grobo bi porazdelitev dela lahko opisali nekako takole:

Domen se je primarno posvečal navigaciji in premikanju robota. V zasluge se mu šteje implementacija avtonomne navigacije, parkiranja in pravokotne postavitve robota pred poljuben objekt. Razvil je metode za izbiro in navigacijo do cilja. Njegov največji prispevek pa je verjetno integracija vseh komponent, kar sicer ni naredil sam, je pa imel vodilno vlogo.

Jure je delal predvsem v zaznavi objektov. Implementiral je metode za zaznavo cilindrov, obročev, okvirjev slik in parkirnih mest, ki so se izkazale za učinkovite. Poudariti je treba, da je razvil tudi razrede za hranjenje obročev in cilindrov, da si je robot zapomnil njihove pozicije in barve. Veliko je svetoval in komuniciral ter v splošnem vodil načrtovanje projekta.

Grega je največ prispeval z implementacijo Padima in modelom za ločevanje med slikami in obrazi. Poleg tega pa je razvijal nekaj ne implementiranih pomožnih rešitev za parkiranje in zaznavo cilindrov. Veliko dela sta Jure in Grega opravila skupaj. To zavzema sintezo in zaznavanje govora in zajemanje podatkov za treniranje modelov. Pomagal je tudi pri detekciji robov in pred obdelavo slike za bolj natančno zaznavanje obročev.

Član ekipe	**delež opravljenega dela**
Jure Pahor	34%
Domen Ogorevc	33%
Grega Potočnik	33%

Tabela 6.1: Razdelitev dela med člani ekipe

Poglavje 7

Zaključek

Izdelava te naloge je bila poučna in zahtevna izkušnja. Izdelali smo programski paket, ki robota vodi po svetu in mu omogoča zaznavanje oblik, obrazov in slik. Zagotovo pa je prostora tudi za nekatere izboljšave. Ena izmed bolj očitnih je širši spekter zaznanega govora. Čeprav je robot sposoben razumeti navodila, ki so definirana v nalogi, vseeno ne razume večjega dela angleškega jezika in bolj kompleksnih povedi. Izboljšala se bi lahko tudi detekcija objektov, ki bi povečala robustnost. Ob namerno slabših pogojih z več motilci, se robot namreč ne bi odrezal tako dobro kot se je za tole nalogo.

Na poti do uspeha nam je stalo kar nekaj problemov, kot so nestabilna programska oprema in pa ne zadostna strojna oprema. Mnogokrat se je namreč zgodilo, da simulacija ni delovala pravilno oziroma se sploh ni zagnala. Pogost problem je bil tudi ne optimalno navigiranje robota, ki se je ne malokdaj zataknil ob steni ali vogalu. Seveda pa nam je uspelo verjetnosti teh problemov kar se da zmanjšati.

Omejitev je bila tudi to, da smo za učinkovito delo morali uporabljati računalnike fakultete za računalništvo in informatiko. Poganjanje simulacije zahteva namreč dokaj sposobno grafično enoto, ki pa je žal nismo imeli vsi na voljo. Z uporabo tujih računalnikov pa seveda pridejo tudi omejitve dostopnosti, saj nimamo privilegijev super uporabnika. Poleg tega pa je število samih računalnikov prav tako omejeno, kar je pomenilo, da zaradi velikega števila ekip, niso mogli vsi delati. Bila pa je tudi prednost uporabe fakultetnih računalnikov, in sicer to, da je bilo okolje več ali manj že vzpostavljeno in ni bilo večjih problemov z združljivostjo.

Na koncu se je izkazalo, da so vsi ti problemi obvladljivi, saj smo še pravi čas nalogo uspešno dokončali.

Literatura

[1] Open Robotics, *ROS 2 Documentation: Humble Hawksbill*, Dostopno na: https://docs.ros.org/en/humble/index.html

[2] Generation robots, *TurtleBot 4 Overview*, Dostopno na: `https://www.generationrobots.com/media/turtlebot4/Turtlebot_4_OverviewBrochure.pdf`

[3] Open Robotics, *Gazebo Simulation*, Dostopno na: `https://gazebosim.org/about`

[4] ax Inc.axinc-ai *ailia models* Dostopno na: `https://github.com/axinc-ai/ailia-models`

[5] PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization Thomas Defard and Aleksandr Setkov and Angelique Loesch and Romaric Audigier 2020

[6] Coqui-ai *TTS* Dostopno na: `https://github.com/coqui-ai/TTS`