Praćenje objekta mobilnim robotom primjenom računalnog vida

Miklaužić, Filip

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:655922

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-05**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Filip Miklaužić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Marko Švaco, mag. ing.

Student:

Filip Miklaužić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru Doc. dr. sc. Marku Švaci, Branimiru Ćaranu te Mateju Božiću na danim savjetima, komentarima i usmjerenjima tijekom izrade ovog rada.

Zahvaljujem i svojoj obitelji, djevojci te prijateljima na bezuvjetnoj podršci tijekom studiranja i pisanja ovog rada.

Filip Miklaužić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22-**	

ZAVRŠNI ZADATAK

Student: **Filip Miklaužić** JMBAG: **0035219360**

Naslov rada na hrvatskom jeziku: **Praćenje objekta mobilnim robotom primjenom računalnog vida**

Naslov rada na engleskom jeziku: **Object tracking with a mobile robot using computer vision**

Opis zadatka:

Snalaženje mobilnih robota u prostoru može biti zasnovano na različitim sustavima percepcije kao što su LIDAR-i, 2D i 3D vizijski sustavi i sl. S ciljem prepoznavanja i praćenja određenog objekta mobilni roboti najčešće koriste vizijske sustave s obzirom na njihovu široku dostupnost i relativno nisku cijenu. Vizijski sustavi omogućavaju akvizicije slike radnog okruženja dok se specifični algoritmi strojnog vida koriste za obradu i interpretaciju 2D ili 3D slike.

S ciljem upoznavanja sa sustavima percepcije mobilnih robota, u ovom radu potrebno je koristiti dostupan vizijski sustav na postojećem mobilnom robotu u Laboratoriju za računalnu inteligenciju i napraviti sljedeće:

- Istražiti postojeće algoritme za prepoznavanje i praćenje objekata u OpenCV-u (eng. „Open Source Computer Vision Library“) i ROS-u (eng. „Robot Operating System“),
- Istražiti algoritme za prepoznavanje i praćenje markera (npr. Aruco markeri),
- Odabrati najmanje jednu vrstu algoritma za prepoznavanje i praćenje objekata te ga implementirati na upravljačko računalo mobilnog robota,
- Implementirati razvijene algoritme na postojećem mobilnom robotu u Laboratoriju za računalnu inteligenciju kako bi robot održavao traženi prostorni položaj (poziciju i orijentaciju) u odnosu na objekt kojeg prati.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 30. 11. 2021.

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Zadatak zadao: Doc. dr. sc. Marko Švaco

Predsjednik Povjerenstva: Prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA IV
POPIS OZNAKA VI
SAŽETAK VII
SUMMARY VIII
1. UVOD 1
1.1. Računalni vid 2
1.1.1. Zadatci računalnog vida u robotici 2
1.1.1.1. Prepoznavanje 2
1.1.1.2. Praćenje 3
1.1.2. Primjena računalnog vida u robotici 4
1.2. Mobilni roboti 4
1.2.1. Osnovni podsustavi mobilnog robota 5
1.2.2. Mobilni robot s diferencijalnim pogonom 6
1.2.2.1. Kinematski model robota s diferencijalnim pogonom 7
1.3. Robotski operativni sustav (*ROS*) 10
1.3.1. Koncepti ROS-a 11
1.3.1.1. ROS filesystem razina 11
1.3.1.2. ROS computation razina 12
2. ALGORITMI ZA PREPOZNAVANJE I PRAĆENJE 14
2.1. Biblioteka računalnog vida - *OpenCV* 14
2.1.1. Uvod u OpenCV 14
2.2. Algoritmi za prepoznavanje (*engl. feature detection*) u *OpenCV-u* 14
2.2.1. Općenito o prepoznavanju 14
2.2.2. Harris Corner Detection algoritam 15
2.2.3. Shi-Tomas Corner Detection algoritam 17
2.2.4. SIFT algoritam 18
2.2.5. SURF algoritam 20
2.2.6. Detektor mrlja 21
2.2.7. HoG algoritam 22
2.2.8. Feature Matching algoritam 23
2.2.9. Usporedba algoritama detektiranja 23
2.3. Algoritmi za praćenje (*engl. object tracking)* u *OpenCV-u* 24
2.3.1. Općenito o praćenju 24
2.3.2. BOOSTING algoritam za praćenje 25
2.3.3. MIL algoritam za praćenje 26
2.3.4. KCF algoritam za praćenje 27
2.3.5. CSRT algoritam za praćenje 28

2.3.6. MedianFlow algoritam za praćenje ... 29
2.3.7. TLD algoritam za praćenje ... 30
2.3.8. MOSSE algoritam za praćenje ... 31
2.3.9. GOTURN algoritam za praćenje ... 32
2.3.10. Usporedba algoritama praćenja ... 33
2.4. Algoritmi za prepoznavanje i praćenje u *ROS-u* ... 34
2.4.1. Paket find_object_2d ... 34
2.4.1.1. Detekcija položaja predmeta u 2D prostoru ... 35
2.4.1.2. Detekcija položaja predmeta u 3D prostoru ... 40

3. PRAĆENJE I PREPOZNAVANJE MARKERA ... 43
3.1. Općenito o markerima ... 43
3.2. *ArUco* markeri ... 44
3.2.1. Općenito o ArUco markerima ... 44
3.2.2. Generiranje željenog tipa ArUco markera ... 45
3.3. Prepoznavanje i praćenje *ArUco* markera ... 46
3.4. Primjena algoritama za praćenje i prepoznavanje *ArUco* markera korištenjem *OpenCV* biblioteke ... 47
3.5. Primjena algoritama za praćenje i prepoznavanje *ArUco* markera korištenjem *ROS*-a i *aruco_ros* čvora ... 49
3.5.1. Paket ddynamic_reconfigure ... 49
3.5.2. Paket realsense-ros ... 49
3.5.3. Paket aruco_ros ... 52
3.5.4. Alat za 3D vizualizaciju RViz ... 55

4. IMPLEMENTACIJA ALGORITMA ZA PREPOZNAVANJE I PRAĆENJE NA UPRAVLJAČKO RAČUNALO MOBILNOG ROBOTA ... 56
4.1. Istraživački mobilni robot izrađen u laboratoriju *CRTA* ... 56
4.1.1. Glavne komponente mobilnog robota ... 57
4.1.1.1. NVIDIA Jetson Xavier NX razvojni komplet ... 57
4.1.1.2. Mikrokontroler Teensy 4.0 ... 58
4.1.1.3. odrive V3.6 ... 59
4.1.1.4. Intel RealSense D435i kamera ... 61
4.1.1.5. APS 5065 Outrunner motor ... 61
4.1.1.6. CUI AMT10E2-V enkoder ... 62
4.2. Pokretanje potrebnih čvorova ... 63
4.2.1. Launch datoteka za pokretanje čvorova mobilnog robota ... 63
4.2.2. čvor adding_frame_robot_link ... 64
4.2.3. čvor RViz ... 64
4.2.4. čvor static_tf_broadcaster ... 65
4.2.5. čvor view_frames ... 66
4.2.6. čvor odometry mobilnog robota ... 67
4.3. Čvorovi za upravljanje referencama brzina ... 68
4.3.1. čvor speed_controller_positioning ... 68
4.3.2. Launch datoteka za jednostavnije pokretanje čvorova za pozicioniranje ... 70
4.3.3. Pregled čvorova u zadatku pozicioniranja ... 70
4.3.4. čvor speed_controller_following ... 71
4.3.5. Launch datoteka za jednostavnije pokretanje čvorova za praćenje ... 73
4.3.6. Pregled čvorova u zadatku slijeđenja ... 73
4.4. Vizualizacija referenci brzine teme *cmd_vel* ... 74

4.5. Spajanje na *ROS* sustav montiran na mobilnom robotu 75
4.6. Testiranje razvijenih algoritama u stvarnom okruženju 76
5. ZAKLJUČAK 79
LITERATURA 81
PRILOZI 83

POPIS SLIKA

Slika 1. Algoritam prepoznavanja znaka [1] 3
Slika 2. Algoritam praćenja automobila [2] 3
Slika 3. Mobilni robot *TurtleBot 4* [3] 4
Slika 4. Prikaz mobilnog robota s diferencijalnim pogonom u X-Y ravnini [5] 7
Slika 5. Prikaz *ROS filesystem* razine [4] 11
Slika 6. Prikaz *ROS computation* razina [4] 12
Slika 7. Primjer otkrivenih kutova *Harris Corner Detection* algoritmom [10] 15
Slika 8. Primjer otkrivenih kutova *Harris Corner Detection* algoritmom 16
Slika 9. Primjer otkrivenih kutova Shi-Tomas algoritmom 17
Slika 10. Prikaz skalirane slike [13] 18
Slika 11. Primjer digitalnog zapisa značajki primjenom *SIFT* algoritma 19
Slika 12. Primjer podudaranja značajki sa slika primjenom *SURF* algoritma[11] 20
Slika 13. Primjer određivanja područja jednakih značajki sa slika primjenom *Blob Detection* algoritma [11] 21
Slika 14. Primjer određivanja područja značajki sa slike primjenom *HoG* algoritma 22
Slika 15. Pronalazak značajki primjenom *Feature Matching* algoritma 23
Slika 16. Praćenje rednih stezaljki *PLC-a* pomoću *BOOSTING* algoritma praćenja 25
Slika 17. Praćenje rednih stezaljki *PLC-a* pomoću *MIL* algoritma praćenja 26
Slika 18. Praćenje rednih stezaljki *PLC-a* pomoću *KCF* algoritma praćenja 27
Slika 19. Praćenje rednih stezaljki *PLC-a* pomoću *CSRT* algoritma praćenja 28
Slika 20. Praćenje rednih stezaljki *PLC-a* pomoću *MedianFlow* algoritma praćenja 29
Slika 21. Praćenje rednih stezaljki *PLC-a* pomoću *TLD* algoritma praćenja 30
Slika 22. Praćenje rednih stezaljki *PLC-a* pomoću *MOSSE* algoritma praćenja 31
Slika 23. Praćenje osobe pomoću *GOTURN* algoritma praćenja [18] 32
Slika 24. Prikaz objekata nad kojima je testiran *find_object_2d* čvor 36
Slika 25. Postupak dodavanja novog objekta detekcije korištenjem čvora *find_object_2d* 37
Slika 26. Prikaz objekata detektirani pomoću čvora *find_object_2d* 38
Slika 27. Prikaz koordinata detektiranih rubova skripte *print_objects_detected* 39
Slika 28. Prikaz parametara skripte *find_object_3d.launch* 42
Slika 29. Primjeri raznih veličina *ArUco* markera [19] 44
Slika 30. Prikaz online generatora *ArUco* markera [28] 45
Slika 31. Prikaz detektiranih *ArUco* markera [26] 48
Slika 32. Prikaz *RViz* sučelja prilikom detekcije *ArUco* markera 55
Slika 33. Mobilni robot iz CRTA-e 56
Slika 34. Prikaz *CAD* modela korištenog mobilnog robota 57
Slika 35. *NVIDIA Jetson Xavier NX razvojni komplet*[6] 58
Slika 36. *Teensy* 4.0 mikrokontroler [7] 59
Slika 37. Prikaz korištenog *Odrive 3.6* sustava [31] 60
Slika 38. *Realsense D435i* kamera [8] 61
Slika 39. *APS* 5065 *BLDC* motor[29] 62
Slika 40. prikaz korištenog enkodera CUI AMT10E2-V [30] 62
Slika 41. Prikaz predloška *RViz* sučelja sustava mobilnog robota 64
Slika 42. Prikaz *RViz* sučelja nakon pokretanja čvora *static_transformation.py* 65
Slika 43. Prikaz koordinatnih sustava dobivenih čvorom *view_frames* 66
Slika 44. Prikaz teme odometrije mobilnog robota naziva *odom_encoder* 67
Slika 45. Dijagram toka algoritma pozicioniranja 69
Slika 46. Prikaz pretplatnika/pošiljatelja u zadacima pozicioniranja 70

Slika 47. Dijagram toka algoritma slijeđenja markera 72
Slika 48. Prikaz pretplatnika/pošiljatelja u zadacima slijeđenja 73
Slika 49. Prikaz reference brzine objavljene na temu */cmd_vel* pomoću *rqt_plot* čvora 74
Slika 50. Prikaz mrežnih parametara korištenog mobilnog robota 75
Slika 51. Provođenje algoritama pozicioniranja na mobilnom robotu 76
Slika 52. Provođenje algoritama praćenja na mobilnom robotu 77
Slika 53. Prikaz brzine osvježavanja pozicije detektiranog *ArUco* markera 78

POPIS OZNAKA

Oznaka	Jedinica	Opis
x	m	Pozicija mobilnog robota u smjeru osi X
y	m	Pozicija mobilnog robota u smjeru osi Y
θ	rad	Orijentacija mobilnog robota u X-Y ravnini
v	m/s	Linearna brzina mobilnog robota
v_l	m/s	Linearna brzina lijevog kotača
v_r	m/s	Linearna brzina desnog kotača
ω	rad/s	Kutna brzina mobilnog robota u X-Y ravnini
ω_l	rad/s	Kutna brzina lijevog kotača
ω_r	rad/s	Kutna brzina desnog kotača

SAŽETAK

Kretanje mobilnog robota kroz prostor vrlo je složen zadatak. Snalaženje i planiranje trajektorije u istomu može biti zasnivano na različitim sustavima percepcije kao što je *LIDAR* te *2D* i *3D* vizijski sustavi. S ciljem prepoznavanja i praćenja određenog objekta u mobilnoj robotici zbog dostupnosti i relativno niske cijene najčešće se koriste vizijski sustavi. Vizijski sustav omogućava stvaranje percepcije radnog okruženja dok se za obradu i interpretaciju iste koriste algoritmi strojnog vida i umjetne inteligencije.

Glavni zadatak ovog završnog rada je upoznati se s radom robotskog operativnog sustava (*engl. Robot Operating System - ROS*) te istražiti, proučiti i implementirati algoritme prepoznavanja i praćenja na postojećeg mobilnog robota u Laboratoriju za računalnu inteligenciju. Također je potrebno osmisliti, razviti te implementirati *planer* pomoću kojega će mobilni robot održavati traženi prostorni položaj (poziciju i orijentaciju) u odnosu na objekt kojeg prati.

Ključne riječi: mobilni robot, robotski operativni sustav (*ROS*), sustav percepcije, vizijski sustav, prepoznavanje, praćenje, *planer*

SUMMARY

Moving a mobile robot through space is a very complex task. Finding the right way and planning a trajectory can be based on different perception systems such as *LIDAR*, *2D* and *3D* vision systems. Whit a goal of recognizing and tracking a certian object, due to availability and relatively low cost, vision systems are most commonly used. The vision systems enables the perception of the working environment, while machine vision and artificial intelligence algorithms are used for its processing and interpretation.

The main task of this final paper is to become familiar with the *ROS* system and to research, study and implement recognition and tracking algorithms on an existing mobile robot in the Computer Intelligence Laboratory. It is also necessary to design, develop and implement a planner by means of which the mobile robot will be able to maintain the reauired spatial position and orientation in relation to the object it follows.

Key words: mobile robot, robot operating system, perception system, vision system, recognition, tracking, planner

1. UVOD

Mobilna i industrijska robotika su, u posljednjih desetak godina, doživjeli drastičan rast. Neki od razloga tome su veliki napredak u području elektronike koji se reflektirao na pojeftinjenje komponenata, potreba za automatizacijom raznih monotonih poslova ili procesa opasnih po zdravlje i život, sve to s ciljem što većeg profita i očuvanja zdravlja zaposlenika.

U ovom radu napravljen je pregled validacija postojećih algoritama za prepoznavanje i praćenje objekata. Za upravljanje ponašanjem robota korišten je robotski operativni sustav (*engl. Robot Operating System-ROS*) te *open source* (*hrv. otvorenog izvora*) biblioteka podataka specijalizirana za zadaće računalnog vida (*OpenCV*) koji su objašnjeni u nastavku rada. Nakon validacije odabran je najprikladniji algoritam koji je potom implementiran i validiran u realnim uvjetima na postojećem mobilnom robotu u Laboratoriju za računalnu inteligenciju.

Primjena mobilnih robota seže od malih kućanskih robota kao što su roboti za usisavanje i košnju trave pa sve do robota Rovera-namijenjenih za istraživanje Marsa. Za postizanje autonomije i samostalnosti robota u prostoru nužno je istomu omogućiti percepciju prostora u kojemu se nalazi. To se postiže primjenom raznih sustava percepcije kao što su senzori udaljenosti, 2D i 3D vizijski sustavi i slično. Potom je potrebno informacije iz vanjskog svijeta prikupljene pomoću spomenutih sustava prilagoditi *hardware-u* s ograničenim brojem izvršenih operacija u jedinici vremena s ciljem dobivanja željene pouzdanosti izvora i osiguravanja *real-time* (*hrv. u stvarnom vremenu*) informacije koja se dalje prosljeđuje i obrađuje u sustavu. Sljedeći element sustava prema određenim pravilima iste obrađuje te tako dobivene izlazne rezultate šalje dalje izvršnim članovima. U ovom završnom radu informacije iz vanjskog svijeta prikupljane su pomoću *Realsense D435i* kamere, spojene na *NVIDIA Jetson Xavier* razvojni sustav na kojemu se vrti *ROS* čija je glavna zadaća rad s informacijama, provođenje algoritama za snalaženja robota u prostoru te na posljetku slanje referenci na mikro-upravljač koji se zbog mogućnosti postizanja velikih brzina koristi u zadacima upravljanja i komunikacije s aktuatorima i senzorima. Korištena oprema je opisana u poglavlju *4. Implementacija algoritama za prepoznavanje i praćenje na upravljačko računalo mobilnog robota*.

1.1. Računalni vid

Računalni vid područje je umjetne inteligencije koje rješava probleme analize i prepoznavanja dvodimenzionalnih i/ili trodimenzionalnih predmeta. Koristi se u robotici u zadacima autonomnog kretanja i snalaženja u prostoru, po svojoj prirodi vrlo kompleksnih zadataka koji bi bez primjene računalnog vida bili neizvedivi. Pošto je svijet koji nas okružuje izrazito nepravilan i nepredvidiv ne može se na jednostavan način načiniti robustan algoritam koji će pravilno izvršavati određenu zadaću u prostoru te u svakoj situaciji ispravno odgovoriti na vanjski podražaj. Tu do izražaja dolazi proces razvoja raznih algoritama računalnog vida koji robotu stvaraju percepciju okoline prema kojima se donose odluke.

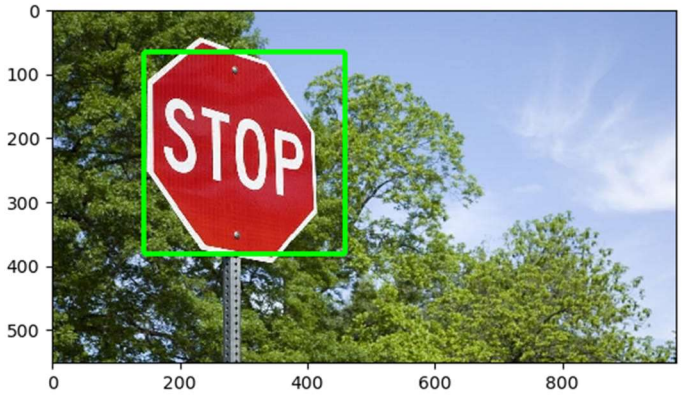
1.1.1. Zadatci računalnog vida u robotici

Zadatci računalnog vida mogu se, prema primjeni, podijeliti na dva osnovna dijela: prepoznavanje i praćenje.

1.1.1.1. Prepoznavanje

Jedan od najopćenitijih problema koju rješava računalni vid je problem prepoznavanja. Glavna mu je namjena obrada slike te utvrđivanje postoji li na istoj određena značajka, aktivnost ili objekt koji nam je od interesa u danom trenutku. Taj zadatak djeluje poprilično trivijalno svakoj osobi bez poteškoća ili oštećenja vidnog ili živčanog sustava, ali čak i uz pomoć računalnog vida, ni do dana pisanja ovog rada, nije pronađeno univerzalno rješenje koje bi iste bilo sposobno prepoznati bez greške u raznim nepoznatim situacijama i novim uvjetima rada, kao što su promjena osvjetljenja, pozadine, međusobnog položaja i orijentacije kamere u odnosu na predmet te ostali parametri koji izravno ili neizravno utječu na kvalitetu i sadržaj dobivene slike. Navedeni problemi mogu djelom biti riješeni primjenom neuronskih mreža te neprestanog učenja, ali i dalje se ne mogu isključiti nepovoljne reakcije sustava prilikom promjena nekih od prije spomenutih uvjeta rada. Također, algoritmi prepoznavanja mogu se koristiti u zadacima prepoznavanje lica, bilo korištenjem kamera te raznih pred definiranih značajki i unesenih modela prepoznaju lice, bilo pomoću metoda stvaranje 3D slike objekta pomoću senzora dubine i kamera. Tako se stvara oblak točaka koji nosi informaciju o skeniranom objektu koji se potom

uspoređuje s referentnim modelom. Slika 1. prikazuje algoritam koji na slici detektira prometni znak.



Slika 1. Algoritam prepoznavanja znaka [1]

1.1.1.2. Praćenje

Nakon algoritama koji provode detekciju željenih značajki, na red dolaze algoritmi koji su namijenjeni za zadatke praćenja u slijedu povezanih slika (videozapisu). Njihova zadaća je reducirati područje na slici koja se obrađuje te pratiti manji broj jedinstvenih značajka željenog predmeta. Zbog manjeg područja na slici koje se kontinuirano obrađuje te manjeg broja značajki za usporedbu, algoritmi praćenja su višestruko brži od algoritama prepoznavanja pa se mogu koristiti za praćenje bržih kretnji dok im se kao nedostatak može navesti daleko manja robusnost prema promjenama uvjeta. Slika 2. prikazuje algoritam koji na zadanom videozapisu prati automobil.

Slika 2. Algoritam praćenja automobila [2]

1.1.2. Primjena računalnog vida u robotici

Računalni vid se zbog velike dostupnosti i relativno niske cijene opreme može koristiti u raznim zadatcima u robotici kao što su primjerice zadaci selekcije predmeta na pokretnoj traci, analiza velikog broja predmeta po određenim kontrolnim točkama i kriterijima u području kontrole kvalitete, zadaci lokalizacije predmeta s ciljem određivanja njegova prostornog položaja te razni zadaci koji kompleksnim sustavima kao što su mobilni roboti omogućuju djelomičnu ili potpunu autonomiju.

1.2. Mobilni roboti

Koncept mobilnih robota, za razliku od industrijskih robota, veže na sebe pojam mobilnosti koja se ostvaruje pomoću pogonskih sustava (motora i aktuatora), sustava za dobavu informacija iz okoline (senzori, kamere) te sustava za upravljanje (računala, mikrokontroleri i mikroprocesori). Na slici 3. prikazan je mobilni robot *TurtleBot 4* istoimene tvrtke. Robot je specifičan jer za relativno pristupačnu cijenu donosi brojne funkcionalnosti daleko skupljih i složenijih industrijskih platformi te može služiti kao platforma za istraživanje i učenje. Za analizu okoline robot koristi stereo kameru uz 2D LIDAR, enkodere, infracrvene te razne ostale senzore koje mu omogućavaju kvalitetnu percepciju prostora.

Slika 3. Mobilni robot *TurtleBot 4* [3]

Pogon mobilnog robota može biti ostvaren i na druge načine, ovisno o topologiji i vrsti terena, pa tako kretanje mobilnog robota može biti ostvareno pomoću gusjenica, kotača, propelera. Osim podjele prema vrsti pogona, mobilne roboti se dijele i prema načinu pogona pa postoje autonomni i teleoperacijski mobilni roboti. Autonomni mobilni roboti se opremaju raznim senzorima, kao što su kamere, žiroskopi, *LIDAR* senzori, daljinomjeri uz pomoć kojih robot stvara percepciju okoline što mu omogućuje samostalno snalaženje, kretanje i autonomiju. Teleoperacijski roboti, uz senzore, sadrže komunikacijske module pomoću kojih se ostvaruje komunikacija između robota i upravljačkog uređaja koji šalje naredbe kretanja. Takvim robotom može upravljati čovjek ili centralno računalo koje na sebi provodi algoritme čiji su izlazni podaci reference po kojima se teleoperacijski robot kreće u prostoru.

1.2.1. Osnovni podsustavi mobilnog robota

Mobilni robot je kompleksan sklop interaktivnih i međusobno usklađenih sustava koji, samo u slučaju ispravne međusobne suradnje i komunikacije, istome omogućavaju ispravno kretanje. Baza cijelog sustava mobilnog robota naziva se mehanička konstrukcija ili postolje na kojeg se smještaju ostali potrebni sustavi te pomoću kojega se vitalni dijelovi robota štite od vanjskih utjecaja.

Prvi izdvojeni sustav koji je potreban za rad robota nosi naziv upravljački sustav. Njegove glavne zadaće su uspostavljanje i održavanje neprekidne komunikacije te upravljanje s radom ostalih sustava. Glavni element tog sustava je mikrokontroler, upravljački član koji je zadužen za prikupljanje podataka iz vanjskog svijeta, procesuiranje te upravljanje aktuatorima na temelju određenih referenci. Upravljački sustav može također biti podijeljen na više dijelova od kojih svaki izvršava određenu zadaću. Primjer takve odvojenosti upravljačkog sustava je upravo korišteni mobilni robot koji je detaljnije opisan u nastavku rada, čiji se upravljački sustav sastoji od 2 glavna dijela. Prvi dio sustava je sustav upravljanja visoke razine (*engl. high level control)* koji je zadužen za planiranja trajektorije, analiziranje vrijednosti koje nosi signal prikupljen putem senzora te procesuiranje ostalih vrsta podataka koje mu šalje upravljački sustav niske razine (*engl. low level control*). Sustav upravljanja visoke razine zatim šalje izračunate vrijednosti sustavu niske razine čija je zadaća upravljanje aktuatorima i očitavanje senzora prema zakonitostima prethodno implementiranim u programskom kodu upravljača niske razine.

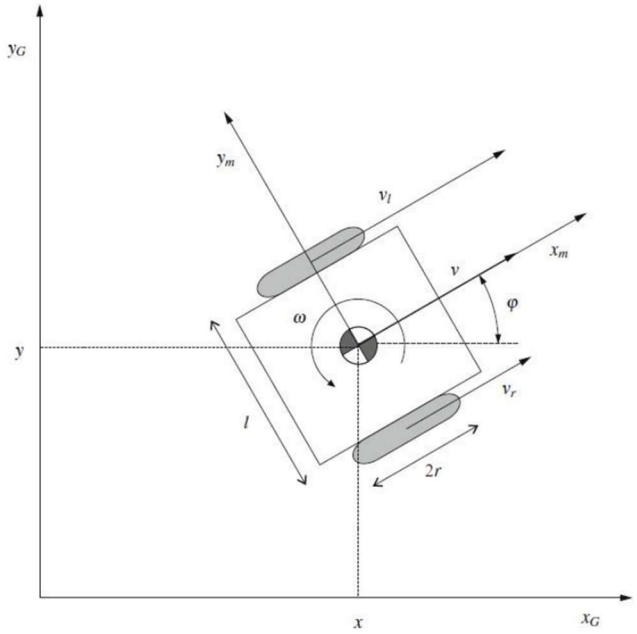
Sljedeći po redu sustav potreban za pravilan rad robota je sustav senzora. Kao što je već spomenuto, uloga sustava senzora je prikupljanje informacija iz okoline robota i pretvaranje iste u referentni naponski ili strujni signal. Tako dobiveni signal upravljačkom sustavu visoke razine, dobiven preko upravljačkog sustava niske razine i senzora služi kao parametar u zadacima prikazivanja okoline. U moru izbora senzora odabiru se oni pomoću kojih se na što jednostavniji način dobivaju što kvalitetniji podaci iz okoline, što bi primjerice mogli biti senzori udaljenosti i temperature, senzori za mjerenje brzine, akceleracije, žiroskopi, razni enkoderi i slično.

Treći osnovni sustav korišten pri izradi robota naziva se izvršni ili aktuatorski sustav. Sustav je zadužen za sva gibanja koja se izvode u sustavu mobilnog robota, bilo za procese kretanja kroz prostor bilo za manipulaciju predmetima ukoliko je u zadatku robota naznačena interakcija robota s predmetima iz okoline. Izvršnim sustavom također upravlja upravljački sustav niske razine nakon prethodno dobivene reference upravljačkog sustava dobivene prema matematičkom modelu sustava primjenom algoritma planiranja trajektorije. [5]

1.2.2. Mobilni robot s diferencijalnim pogonom

U ovom radu je kao temeljna robotska platforma korištena upravo ona koja se pokreće pogonom zasnivanom na diferencijalnom principu. Upravo platforme s takvim pogonom koriste se u zadacima gdje je iznimno bitna okretnost te mogućnost postizanja međusobno nezavisnih prostornih položaja i orijentacija u prostoru. Mobilni robot korišten u zadatku opisan je u odlomku *4. Implementacija algoritama za prepoznavanje i praćenje na upravljačko računalo mobilnog robota*.

Kako bi se upravljanje mobilnog robota moglo uspješno provoditi potrebno je odrediti međusobni odnos između translacijske brzine mobilnog robota te brzine vrtnje svakog od kotača. Takva međusobna veza tih dvaju parametara može se osigurati primjenom kinematskog modela diferencijalnog robota. Diferencijalni robot sastoji se od dva kotača pričvršćena na aktuatore pomoću kojih se ostvaruje gibanje te dva slobodno rotirajuća kotača koji služe za uspostavljanje ravnoteže. Slika 4. prikazuje shematski model diferencijalnog robota smještenog u X-Y ravnini.



Slika 4. Prikaz mobilnog robota s diferencijalnim pogonom u X-Y ravnini [5]

Kako bi se postavljanje i korištenje kinematskog modela ispravno sprovelo, potrebno je navesti njegova ograničenja i pojednostavljenja [5]:

- Gibanje mobilnog robota je planarno, tj. u horizontalnoj ravnini
- Kontakt između podloge i kotača je idealan, tj. nema proklizavanja kotača
- Mobilni robot se opisuje kao kruto tijelo na kotačima

1.2.2.1. Kinematski model robota s diferencijalnim pogonom

Položaj mobilnog robota može se definirati s tri prostorne koordinate, od kojih prve dvije definiraju poziciju u ravnini robota (X-Y ravnina s slike 4.), dok treća predstavlja orijentaciju mobilnog robota oko vertikalne Z osi. Na slici 4. je također vidljiv nepomični globalni koordinatni sustav označen s x_Gi $y_G$te lokalni pomični koordinatni sustav označen s x_m i y_m, gdje x_m predstavlja smjer kretanja mobilnog robota. Sljedeći izraz (**1**) predstavlja zapis vektora stanja mobilnog robota:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \qquad (\mathbf{1})$$

gdje su *x* i *y* relativne koordinate koordinatnog sustava mobilnog robota u odnosu na globalno definirani koordinatni sustav.

Kao što je spomenuto u uvodu, mobilni robot se pogoni s dva aktuatora na kojima su pričvršćeni kotači. U idealnom slučaju će, prilikom vrtnje oba kotača jednakom brzinom u istom smjeru, robot ostvariti pravocrtno gibanje, dok će u slučajevima jednakih iznosa brzina, ali suprotnih smjerova, robot vršiti rotaciju oko svoje osi. Ukoliko brzine kotača nisu međusobno jednake robot će se gibati po kružnici čiji se centar može nalaziti u bilo kojoj točki na pravcu koji spaja središta pogonskih kotača.

Izraz (**2**) prikazuje utjecaj brzina lijevog i desnog kotača na linearnu brzinu mobilnog robota u smjeru lokalne koordinatne osi x_m

$$v = \frac{v_l + v_r}{2}, \tag{2}$$

gdje je v_l translacijska brzina ostvarena na lijevom kotaču, a v_r translacijska brzina ostvarena na desnom kotaču.

Korištenjem izraza (**3**) određuje se kutna brzina mobilnog robota oko osi z:

$$\omega = \frac{\omega_l + \omega_r}{2}, \tag{3}$$

u kojem ω_l predstavlja kutnu brzinu lijevog, a $\omega_r$kutnu brzinu desnog kotača. Estimacija kutnih brzina kotača vrši se iz pozicije dobivene iz enkodera pričvršćenih na osovine motora.

Međusobni odnosi između linearnih i obodnih brzina lijevog i desnog kotača određeni su izrazima (**4**):

$$\begin{aligned} v_l &= \omega_l r, \\ v_l &= \omega_l r. \end{aligned} \tag{4}$$

Pošto izrazi (**2**) i (**3**) daju relativnu brzinu mobilnog robota u odnosu na lokalni koordinatni sustav definiran osima x_m i y_m, potrebno je provesti transformaciju koordinata u globalni koordinatni sustav kako bi se mogla odrediti brzina robota u odnosu na mirujući koordinatni sustav. Matrica rotacije *T* dana je izrazom (**5**):

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Upotrebom prethodno spomenute matrice translacije *T* dobiva se transformacija iz lokalnog u globalni koordinatni sustav mobilnog robota, koja je za translacijsku brzinu *v* zapisana izrazom:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \end{bmatrix}. \tag{6}$$

Transformaciju rotacijske brzine nije nužno provoditi jer su rotacijska brzina u lokalnom i globalnom koordinatnom sustavu jednake. Uzimajući u obzir tu pretpostavku kao i izraz (**6**), dobiva se izraz translacijske i rotacijske brzine u globalnom koordinatnom sustavu, zapisane izrazom (**7**):

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}. \tag{7}$$

Analiza sustava te jednadžbe kinematskog modela diferencijalnih robota [5].

Iz kinematskog modela mobilnog robota (**7**) vidljivo je kako se radi o pod-upravljanom sustavu. Mobilnim robotom se upravlja pomoću dvije upravljačke varijable, tj. kutnih brzina ω_l i ω_r, kojima se direktno utječe na tri varijable stanja, redom linearne brzine duž osi x i y te rotacijsku brzinu. Taj podatak dodatno indicira na složenost algoritama upravljanja brzina te samim time i pozicije mobilnog robota.

1.3. Robotski operativni sustav (*ROS*)

ROS se bez premca može nazvati jednom od najvažnijih prekretnica u svijetu robotike. Zbog sve veće upotrebe robota, kako u industrijskom tako i u kućanstvima, dolazi do potrebe za izgradnjom sustav koji bi standardizirao i olakšao mukotrpan proces izgradnje robotskog sustava. Počeci *ROS*-a sežu u 2007. godinu kada je student prestižnog Kalifornijskog sveučilišta *Stanford*, Morgan Quigley, pokrenuo projekt naziva *Switchyard* iz kojeg se razvio današnji *ROS*. Glavni cilj mu je bio načiniti platformu koja bi se mogla koristiti kao univerzalna polazna točka prilikom izrade robotskog sustava svake primjene. *ROS* na jednom mjestu objedinjuje veliku zbirku dostupnih alata, biblioteka i zakonitosti koji omogućavaju to pojednostavljenje i omogućavaju stvaranje vrlo kompleksnih robotskih sustava bez potrebe za pojedinačnim razvijanjem svakog aspekta sustava. Primjenom *ROS-a* omogućava se korištenje gotovih prethodno razvijenih i usavršenih rješenja te prilagođavanje svojim željama i potrebama zadatka. Velika prednost *ROS-a* kao operativnog sustava robota je ta što se svaki aspekt sustava može razvijati zasebno te se na posljetku sve može poprilično jednostavno uklopiti u upravljački sustav robota. Takav koncept omogućuje veliku razinu podjele zadatka među grupama različitih specijalnosti s ciljem što kvalitetnijeg konačnog rješenja problema i veće produktivnosti.

Prvo službeno izdanje *ROS-a* nosi naziv *ROS 0.4 Mango Tango* koje je tijekom narednih godina dobilo velik broj nasljednika s značajnim poboljšanjima i unaprjeđenjima. Politika cijele kompanije zadužene za održavanje i razvoj *ROS-a* je stvaranje nove verzije sustava svake godine koja svijetlo dana ugleda objavljivanjem službene verzije krajem svibnja. Svake parne godine izlazi *LTS* (*engl. long time support, hrv. dugotrajna podrška*) verzija koja osigurava pogodnosti dodatnih nadogradnji i programske podrške narednih 5 godina dok svake neparne godine izlazi verzija s podrškom u trajanju od dvije godine. Također, razvojni programeri prosječnom korisniku preporučaju korištenje *LTS* verzije sustava radi stabilnosti u radu i što jednostavnijeg korištenja i rješavanja problema dok se napredniji korisnici upuštaju u rad s novim verzijama sustava zbog mogućnosti koje su dostupne svakom novom inačicom. Posljednja *LTS* verzija *ROS-a* nosi naziv *ROS Humble Hawksbill* koja je postala službeno dostupna u svibnju 2022. godine. Uz održavanje postojećih te stvaranja novih inačica *ROS-a*, u trenutku pisanja ovog rada aktualan je i *ROS2* koji je još u razvoju, a trebao bi ispraviti većinu

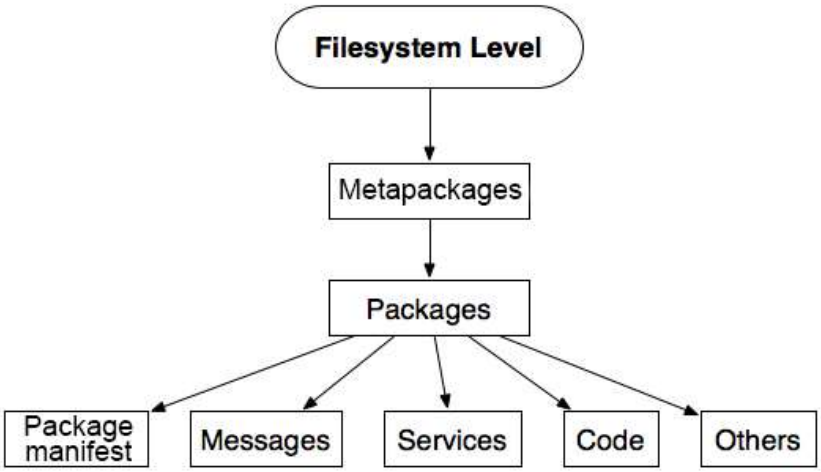
nedostataka operativnog sustava *ROS* te otvoriti nove mogućnosti u zadacima programiranja robota.

1.3.1. Koncepti ROS-a

Za uspješno i efikasno korištenje svih značajki razvojne okoline *ROS-a* zaslužna je pomno isplanirana i uređena arhitektura koja je podijeljena *filesystem*, *computation* te *community* razinu [4].

1.3.1.1. ROS filesystem razina

Pošto se jedan projekt sastoji od velikog broja procesa koji su zaslužni za uspješno izvršavanje željene operacije stvoren je koncept *filesystem-a* (*engl. sustava datoteka).* Cilj ove razine je objediniti procese izgradnje svih sustava korištenih u projektu izrade programa robota. Takav koncept također pripomaže u pregledu strukture projekta te omogućuje jednostavne izmjene određenih dijelova zadatka koji se izvršava te relativno jednostavne preinake projekta. Slika 5 prikazuje ROS *filesystem* razinu:



Slika 5. Prikaz *ROS filesystem* razine [4]

Metapackages (*hrv. meta paketi*) je skupni naziv paketa koji sadrže samo "podatke o podacima". Mogu se također interpretirati kao skupovi paketa koji služe istoj svrsi. Iste ne sadrže nikakve programske kodove, datoteke ni ostale datoteke koje se inače nalaze u standardnim paketima.

Packages (*hrv. paketi*) najvažnija su jedinica u organizacijskoj strukturi *ROS-a* jer paketi mogu sadržavati *Nodes-e* (*hrv. čvorove*), *ROS-dependent library* (hrv. biblioteka ovisnosti), datoteke i biblioteke koje služe za prilagođavanje i pokretanje čvorova kao i ostale datoteke koje zbog funkcionalnosti i preglednosti moraju biti organizirane zajedno. Osnovna stvar stvar koja se može izraditi i objaviti je upravo paket.

Package manifest (*hrv. manifestacija paketa*) pružaju *metadata* (*hrv. meta podatke*) o paketu kao što su ime, verzija, opis, licenca, popis autora i održavatelja te ostale informacije koje bi mogle biti bitne osobama koje će koristiti preuzeti paket. Očituju se u vidu datoteke *package.xml*.

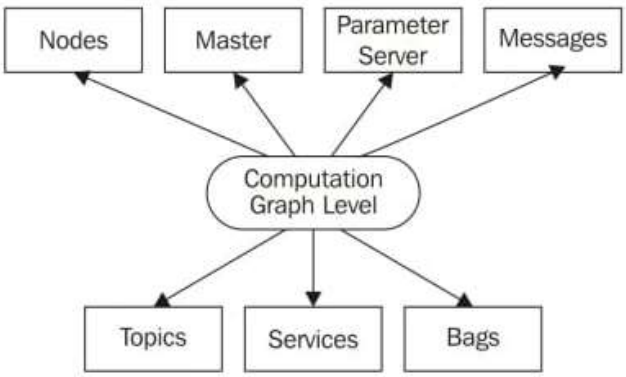
Messages (hrv. poruke) su vrste datoteka koji sudjeluju u komunikaciji između više istovremenih procesa unutar *ROS* okruženja. Sve poruke poslane između procesa spremaju se u mapu *msg* unutar korištenog paketa te sadržavaju nastavak *.msg*.

Code (hrv. program) naziv je za listing naredbi koji je zadužen za izvršavanje određenih zadaća.

Repository (hrv. repozitorij) koristi se za redovno praćenje i osvježavanje najnovijih verzija dostupnih unutar *ROS-a.* Najpoznatiji sustav koji se koristi za gore navedene zadaće nosi naziv *Git* te ga je također moguće koristiti za distribuciju paketa u *ROS* okruženje.

1.3.1.2. ROS computation razina

Glavna zadaća ove razine *ROS* sustava je upravljanje s procesima sustava, tj. čvorovima. Čvorovi su međusobno u interakciji te tvore mrežu koja je zadužena za funkcionalnost cijelog sustava. Slika 6. prikazuje strukturu *Computation (hrv. računske)* razine.



Slika 6. Prikaz *ROS computation* razina [4]

Glavni član ROS sustava koji je prijeko potreban za rad sustava naziva se *ROS Master.* Glavna zadaća mu je praćenje rada svih čvorova i servisa u tom sustavu te primanje i slanje poruka radi ostvarivanja komunikacije među čvorovima. Na jednom računalu u isto vrijeme može biti aktivan samo jedan *ROS Master* koji svoju komunikaciju s čvorovima i ostalim elementima ostvaruje putem *TCP/IP* protokola. [4.1]

Nodes (*hrv. čvorovi*) su članovi *ROS* sustava koji izvršavaju računanje. Sastoje se od skupa procesa koji izvode radnje potrebne za izvršavanje zadataka. Čvorovi mogu biti pisani u programskim jezicima *C++* i *Python* uz pomoć *ROS client library-ja* kao što su *roscpp* i *rospy*. Kao što je već spomenuto, čvorovi sa sadržanim procesima su osnovni dio ROS sustava robota te se sustav gradi prilikom dodavanja novih čvorova, međusobnim povezivanjem te međusobnim uspostavljanjem komunikacije. Također jedan od benefita izvršavanja koda preko čvorova umjesto izvršavanje cijelog sustava u jednom kodu je taj da uslijed greške u jednom čvoru, ostatak sustava može se normalno izvršavati ukoliko taj čvor nije krucijalan za neku dalju operaciju.

Parameters Server (hrv. server parametara) omogućava spremanje podataka na centralnu lokaciju te također omogućava pristup, čitanje i izmjene čvorovima koji posjeduju ključ servera.

Topics (hrv. teme) rade po modelu *publisher/subscriber (hrv. izdavač /pretplatnik)*. Čvor može biti u ulozi *publishera* i objavljivati određene podatke na temu ili može biti u ulozi *subscribera* te se može pretplatiti na temu i čitati podatke iz nje.

Serivces (*hrv. servisi*) drugi su model prijenosa podataka među čvorovima u *ROS-u*, koji, za razliku od modela *publisher/subscriber* i jednosmjerne komunikacije *many-to-many (hrv. mnogi prema mnogima*) rade na principu komunikacije pitanje-odgovor. Servisi se nalaze u *srv* datoteci radnog prostora korištenog čvora.

2. ALGORITMI ZA PREPOZNAVANJE I PRAĆENJE

2.1. Biblioteka računalnog vida - *OpenCV*

OpenCV je *open source* platforma koja se koristi u zadacima gdje postoji potreba za implementacijom računalnog vida i računalne inteligencije. Izrađena je kao biblioteka s uređenom infrastrukturom s ciljem ubrzanja i olakšavanja upotrebe vizije u komercijalnim i hobističkim projektima.

2.1.1. Uvod u OpenCV

Tijekom pisanja ovog rada *OpenCV* u svojoj bazi posjeduje više od 2500 razvijenih i upotrebljivih algoritama. Neki od algoritama koji se nalaze u biblioteci su primjerice algoritmi za prepoznavanje lica, detekciju objekata, praćenje ljudskog tijela, praćenje gesti, stvaranje i rad s 3D oblacima točaka (*engl*. *point cloud*), uređivanje slika, prepoznavanja scena te razni drugi algoritmi slične namjene. Biblioteka sadržava *C++*, *Python*, *Java* i *MATLAB* sučelja te ju je moguće koristiti na *Window*s, *Linux*, *Mac OS*, *Android* operacijskim sustavima. Jedni od glavnih skupina algoritama koje vežemo uz računalni vid su algoritmi prepoznavanja i praćenja, koji su detaljnije klasificirani i opisani u nastavku. Prije početka korištenja biblioteke potrebno je istu i preuzeti. Zbog potrebe raznih naprednijih algoritama odabire se *OpenCV* inačice - *contrib* jer ona sadrži proširenu listu algoritama detekcije i praćenja te sadrži sve algoritme koji su spomenuti u ovom radu.

2.2. Algoritmi za prepoznavanje (*engl. feature detection*) u *OpenCV-u*

2.2.1. Općenito o prepoznavanju

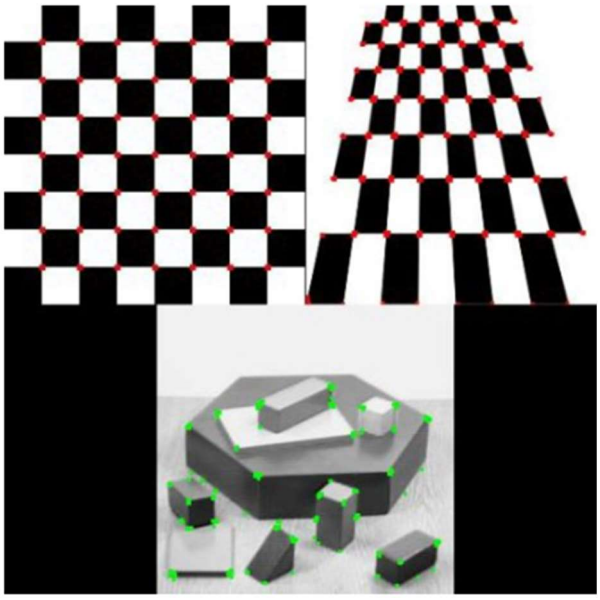
Primjena računala za rješavanje specifičnih zadataka povlači za sobom potrebu za izradom ili korištenjem razvijenih algoritama specijaliziranim i razvijenim za rješavanje određenih problema. Primjer toga je korištenje vizijskih sustava za upravljanje mobilnog robota iz ovog rada. Zadatak pravilne vizualizacije okoline nije nimalo jednostavan te ga je potrebno podijeliti na segmente kako bi se isti olakšao i što kvalitetnije sproveo.

Svijet oko nas, karakterom izuzetno specifičan i jedinstven, predstavlja velik problem u procesu "stvaranja" zakonitosti kojima se opisuju stvarni procesi. Primjerice, ne možemo očekivati da će računalo, ukoliko uslikamo nekoliko fotografija psa biti sposobno, na svakoj slici koju mu

zadano prepoznati da se radi upravo o psu. Stoga je, za opći slučaj, svaku sliku potrebno promatrati "piksel po piksel". Traže se rubovi, područja specifičnih karakteristika i ostale značajke metodama prepoznavanja značajki. Nakon pronalaska tih značajki, proučava se područje oko istih, koje se opisuje te se preko tih opisa područja karakteristika mogu pronaći i na drugim slikama te tako donijeti zaključci o detektiranom predmetu. Produkt takve kategorizacije naziva se opis značajki (*engl.feature description*), koji je temeljni element u radu s slikama prilikom korištenja vizijskih sustava. U nastavku je obrađeno 7 razrađenih algoritama pronalaska karakteristika i navesti područja primjene te prednosti i nedostatci svake od tih metoda.

2.2.2. Harris Corner Detection algoritam

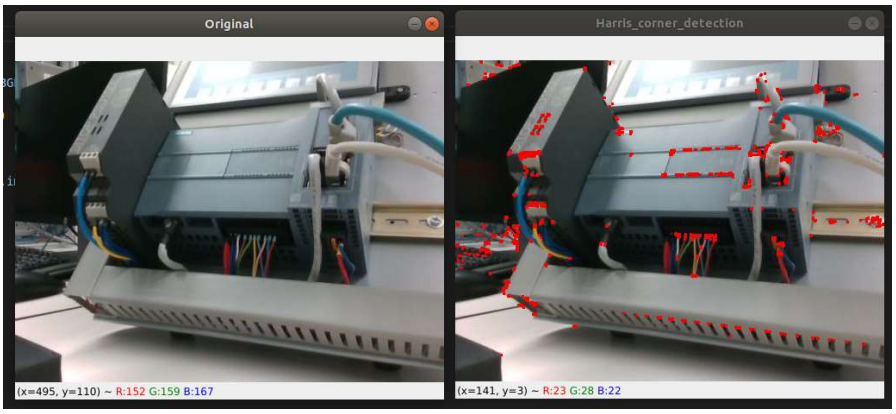
Prvi na listi algoritama korištenih za prepoznavanje koji su isprobani je algoritam *Harris Corner detection algorithm* (*hrv. Harris algoritam za prepoznavanje kutova*). Kao što i sam naziv govori, algoritam služi za otkrivanje najočitijih značajki na fotografiji, kutova. Glavna ideja algoritma je traženje razlike u težini piksela u svim smjerovima na slici. Za razliku od prethodno razvijenih algoritma detekcije kutova, *Harris Corner Detection* algoritam uzima u obzir razliku rezultata kuta u odnosu na smjer ruba, dok su prethodnici proučavali sliku pod fiksnim kutom od 45°. Taj pristup pokazao se bolji zbog kvalitetnijih rezultata.



Slika 7. Primjer otkrivenih kutova *Harris Corner Detection* algoritmom [10]

Na slici 7. prikazano je otkrivanje dijelova slike koji su prema matematičkom opisu definirani kao kutovi. Kao što je već spomenuto, kut se može definirati kao spoj dvaju rubova, tj. mjesto nagle promjene svjetline. Važni su iz razloga što, prilikom translacije, rotacije i promjene osvjetljenja, relativni položaj kutova nekog objekta ostaje nepromijenjen stoga se ti algoritmi mogu koristiti s ciljem minimizacije količine potrebnih obrađenih podataka u zadacima praćenja pokreta, vizije te ostalih područja koje vežemo za računalni vid.

Postupak provođenja algoritma može se podijeliti na 5 glavnih koraka. Prvo se provodi pretvaranje slike u sivi format (*engl. grayscale*) s ciljem smanjenja šuma između piksela izazvanih prisutnosti različitih boja te što kvalitetnijeg i bržeg procesuiranja slike. Drugi korak je proučavanje slike i otkrivanje područja naglih promjena intenziteta svjetline. Time se određuje pomični prozor kojim prolazimo kroz cijelu sliku i otkrivamo područja s sličnim karakteristikama. Računanjem poklapanja svakog prozora slike s referentnim dobivamo vrijednost poklapanja svakog od kontrolnih prozora koja se sprema te se po završetku dobiva lista rubova (detaljna matematička podloga *Harris Corner Detection* algoritma s korištenim matematičkim izrazima [10]).



Slika 8. Primjer otkrivenih kutova *Harris Corner Detection* algoritmom

Slika 8. prikazuje detektirane rubove primjenom *Harris Corner Detection* algoritma. Vidljivo je da je, uz prikaz dominantnih, prikazan i značajan broj manje dominantnih kutova, tj. šumova (primjerice, lijeva strana slike, zbog veće promjene svjetline u odnosu na ostatak slike, promjene intenziteta osvjetljenja tipkovnice detektirani su kao kutovi). Također se može primijetiti pogrešna detekcija kutova kod *ethernet* kabela čiji odsjaj također uzrokuje promjenu osvjetljenja na slici te algoritam tu značajku deklarira kao kut. Detekcija kutova sa slike 8.

provedena je na *gray-scale* slici, ali rezultati su, radi preglednosti, prikazani na slici u *RGB* formatu (osnovni kod korištenja algoritma preuzet s [11], dodatno prilagođen potrebi zadatka).

Algoritam je svoju primjenu pronašao u svim područjima obrade slika, detekcijama pokreta, praćenju objekata, modeliranjima i rekonstrukciji 3D scena te srodnim područjima u kojima se ističe njegova prednost brzine, pouzdanosti i rasterećenja procesorskog sustava. Nedostaci metode su nemogućnost odbacivanja manje dominantnih detekcija te prikaz samo značajnijih rubova.

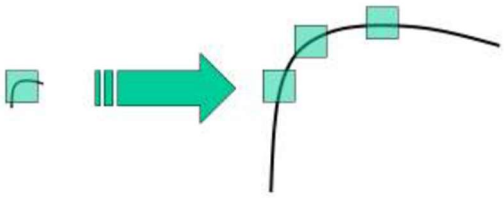
2.2.3. Shi-Tomas Corner Detection algoritam

Prema modelu *Harris Corner Detection* algoritma, uz manje modifikacije, načinjen je *Shi-Tomas Corner Detection* algoritam. Glavna razlika je korištenje funkcije za ocjenjivanje rezultata (matematički zapis funkcije ocjenjivanja [12]). Kao izlaz iz algoritma detekcije dobiva se lista kutova uz dodani prikaz rezultata intenziteta svakog detektiranog kuta. Zbog postojanja liste s vrijednostima intenziteta, algoritam nudi mogućnost prikaza N najznačajnijih kutova koji su, prema zadanoj metodi ocjenjivanja ostvarili najviše vrijednosti, zbog čega nam omogućava filtriranje te odbacivanje kutova koji su ostvarili manje rezultate, tj. manje dominantne kutove te pogrešne detekcije. Slika 9. prikazuje 20 najdominantnijih kutova, tj. područja koja su ostvarila najveće ocjene (kod korištenog algoritma preuzet s [11], dodatno prilagođen potrebama zadatka).

Slika 9. Primjer otkrivenih kutova Shi-Tomas algoritmom

2.2.4. SIFT algoritam

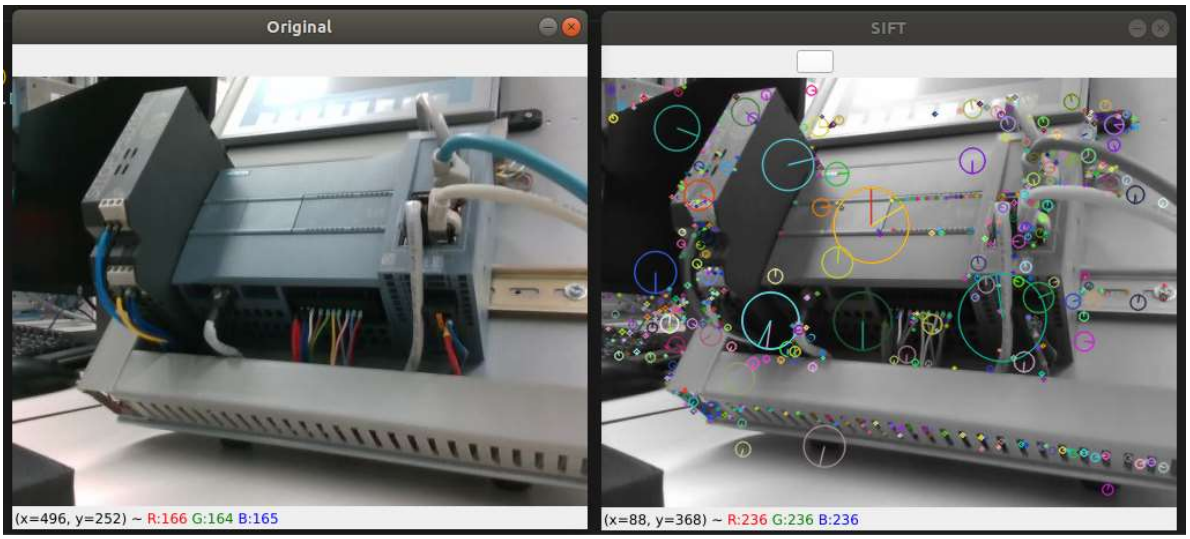
Pri korištenju dosad spomenutih algoritama javlja se problem očitavanja određenih značajki koji je nastao zbog primjene veličine ili nepogodne orijentacije iste. Primjer nemogućnosti očitavanja značajki zbog premale veličine slike prikazan je na slici 10 gdje je vidljivo kako algoritam detekcije rubova za istu liniju, ali različite veličine iste daje drugačije očitanje. Pošto većina algoritama detekcije radi na principu testnih matrica potrebno je voditi računa o redu veličine očekivanih značajki koje se žele otkriti. Primjenom jednake testne matrice na dvije identične, ali skalirane slike dobit ćemo različita očitanja značajki.



Slika 10. Prikaz skalirane slike [13]

Taj problem dobivanja različitih rješenja za isti problem doprinio je razvitku *SIFT* (*engl. scale invariant feature transform, hrv. transformacija invarijantne značajke mjerila*) algoritma. Glavna ideja koja stoji iza tog algoritma je omogućiti prepoznavanje određenih značajki sa slike bez da se mora paziti o transformacijama i skaliranjima koje se izvršavaju nad testnom slikom. Za bilo koji objekt sa slike mogu se izdvojiti zanimljive točke čiji opis želimo dobiti. Takav opis dobiven *SIFT* algoritmom može se zatim koristiti za identifikaciju tog istog objekta na slici gdje se nalazi mnoštvo drugih značajki. Problem koji *SIFT* algoritam pokušava riješiti je pronalazak istih značajki na više različitih fotografija iste scene te stvaranje digitalnog opisa te značajke. To je proces relativno jednostavan za čovjeka, ali računalo može relativno brzo izgubiti vezu između dvije fotografije ukoliko malo promijenimo neke parametre kao što je orijentacija, dimenzija, kontrast i slično. *SIFT* algoritam pokušava, proučavanjem rubova kao predstavnika skipine "snažnijih" gradijenata neosjetljivih na promjene gore navedenih parametara, stvoriti što robusniji opis značajke. Nakon tako provedene obrade dobiva se opis slike dobiven iz otkrivenih rubova, koji može služiti za prepoznavanje istog tog objekta u drugoj sceni (detaljnija matematička podloga *SIFT* algoritma [12]).

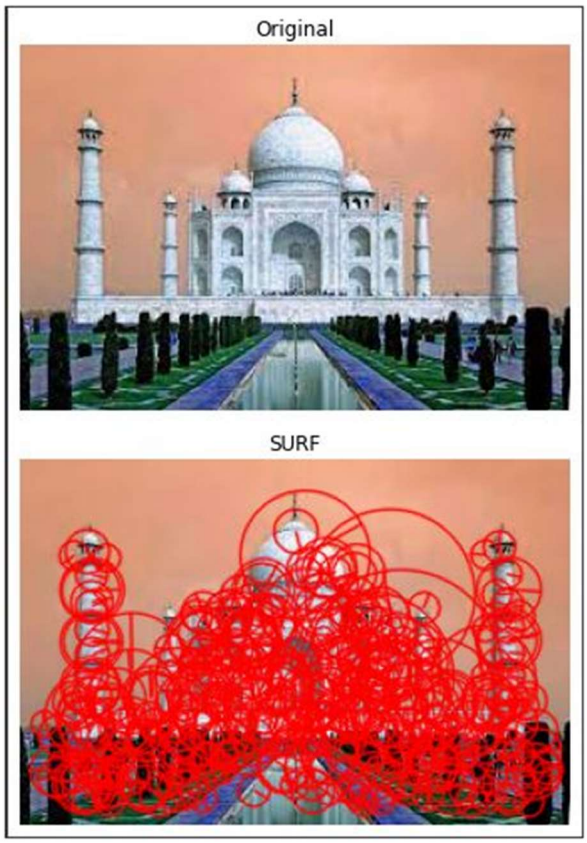
Slika 11. prikazuje digitalni zapis značajki fotografije industrijskog *PLC-a* uz pomoć krugova s upisanim kutovima primjenom *SIFT* algoritma (osnovni kod korištenog algoritma preuzet s [11], dodatno prilagođen potrebama zadatka).



Slika 11. Primjer digitalnog zapisa značajki primjenom *SIFT* algoritma

2.2.5. SURF algoritam

SURF (*engl. Speeded-up Robust Features, hrv. ubrzane robusne značajke*) algoritam nastao je kao unaprjeđenje *SIFT* algoritma koji nudi daleko veće brzine provođenja. Promjene u metodama računanja ključnih točaka i korištenje adaptivnih veličina deskriptora, *SURF* algoritam nakon provedenih usporedbi s prije spomenutim *SIFT* algoritmom ostvaruje do 3 puta bržu detekciju značajki. Također valja naglasiti da je preporuka koristiti *SURF* algoritam ukoliko će prilikom procesa dolaziti do zamućivanja te rotacije slika, dok ga prilikom promjene točke gledišta i rukovanja nije preporučljivo koristiti radi veće mogućnosti pogreške (opis algoritma te matematička podloga [14]).



Slika 12. Primjer podudaranja značajki sa slika primjenom *SURF* algoritma[11]

Slika 12 prikazuje digitalni zapis 400 najdominantnijih otkrivenih značajki koji se može upotrijebiti ukoliko taj objekt želimo pronaći na nekoj većoj slici ili za praćenje tog objekta kroz više slijednih scena.

2.2.6. Detektor mrlja

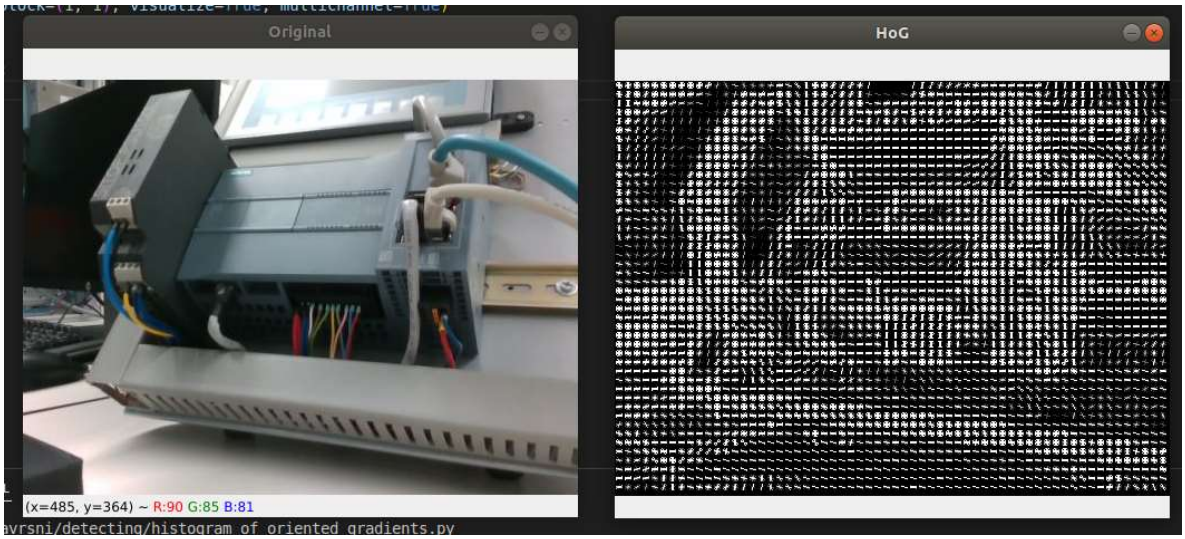
Prilikom primjene računalnog vida postoji potreba za radom s područjima koja se izdvajaju iz cjeline, a sadrže jednake ili približno jednake značajke, kao što su boja, svjetlina, tekstura i slično. Pojam mrlja (*engl. blob*) referira se na grupu koncentriranih piksela ili područja koji dijele određena zajednička svojstva. Detektori mrlja imaju prednost pred korištenjem detektora rubova ili detektora kutova jer nude potpuni zapis regije koju opisuju. Tako dobivene informacije o objektu mogu dalje se koriste za detekciju i praćenje objekata orijentirano na teksture i vizualnu interpretaciju predmeta (boja i oblik). Iz tako dobivenih podataka se, primjenom histograma, također mogu izlučiti rubovi i vrhovi predmeta, što omogućuje istu primjenu kao dosad spomenuti algoritmi detekcije značajki.

Slika 13. prikazuje jedinstveni zapis područja jednakih značajki. Vidljivo je da su područja fasade, područja kula te sva ostala područja slične teksture sjedinjeni i prikazani kao jedan *Blob*.

Slika 13. Primjer određivanja područja jednakih značajki sa slika primjenom *Blob Detection* algoritma [11]

2.2.7. HoG algoritam

Histogram orijentiranih gradijenata (*engl. histogram of oriented gradients)* algoritam je koji se koristi u računalnom vidu s ciljem detekcije objekata. Algoritam se temelji na praćenju ponavljanja orijentacije gradijenata u lokaliziranim dijelovima slike. Izračunava se na gustoj mreži jednoliko raspoređenih ćelija (uzoraka) te koristi preklapajuću mrežu (lokalnu normalizaciju kontrasta) s ciljem poboljšanja točnosti. Teorija detekcije objekta za *HoG* algoritam nalaže da se neki objekt unutar slike može opisati distribucijom usmjerenja rubova ili gradijenta intenziteta. Slika se raščlanjuje na male regije (ćelije) te se potom za sve piksele unutar jedne regije sastavlja histogram smjerova gradijenta (matematička osnova i detaljno objašnjenje *HoG* algoritma [16]).

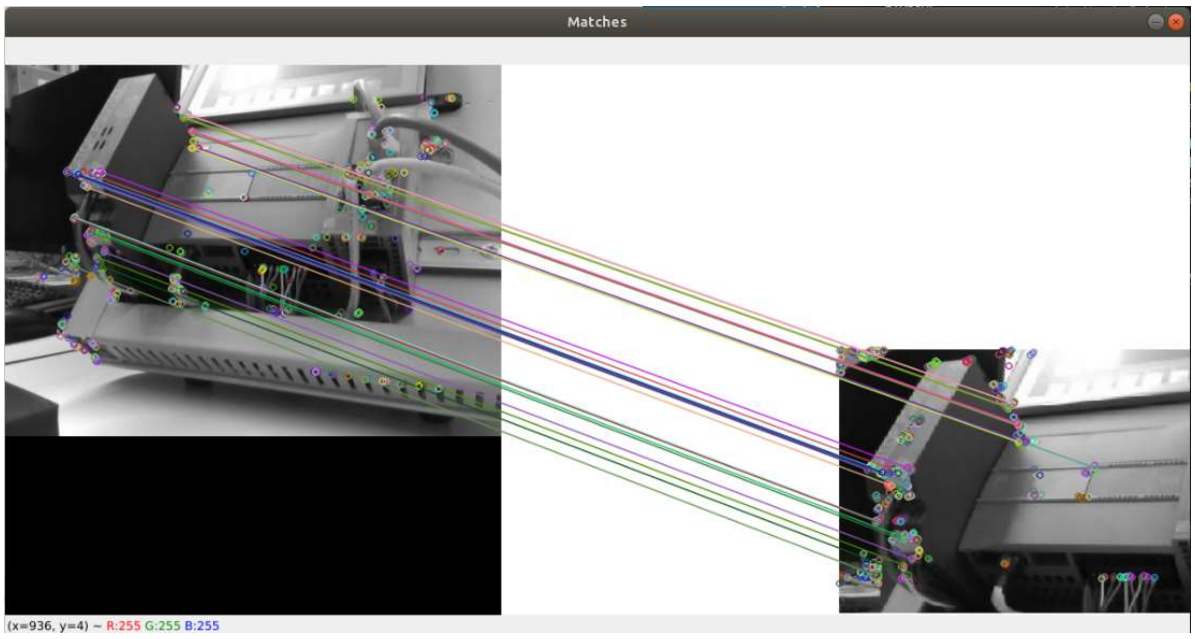


Slika 14. Primjer određivanja područja značajki sa slike primjenom *HoG* algoritma

Slika 14. prikazuje određivanje područja značajki sa slike primjenom *HoG* algoritma. Ovisno o parametrima ćelija (dimenzijama uzorka kojim se provodi analiza slike) dobivamo različite finoće i preglednosti dobivenih histograma (osnovni kod preuzet s [11], dodatno dorađen potrebama zadatka)

2.2.8. Feature Matching algoritam

Kao što mu i naziv govori, cilj algoritma je prepoznati i povezati određene značajke različite orijentacije, perspektive, osvjetljenja, veličine i/ili boje s učene slike na slici koju detektiramo. Računalo, prilikom procesa obrade učene slike traži određene ključne značajke (*engl. key points*), koje također traži i na slici koja se detektira te ih pokušava u što većoj mjeri povezati te tako dobiti veću razinu poklapanja. Slika 15. prikazuje pronalazak značajki na učenoj slici te povezivanje željenih značajki s zadanom slikom (kod primjene *Feature Matching* algoritma [11]).



Slika 15. Pronalazak značajki primjenom *Feature Matching* algoritma

2.2.9. Usporedba algoritama detektiranja

Među algoritmima detektiranja značajki, kao što su kutovi, u upotrebi s *Harris corner detection* algoritmom prednjači *Shi-Tomas* algoritam zbog mogućnosti selekcije određenog broja najdominantnijih kutova otkrivenih na zadanoj sceni. Prilikom velike vjerojatnosti značajnih skaliranja elemenata na slici scene, odabire se *SIFT* algoritam koji značajke opisuje preko rubova koji predstavljaju "snažnije" gradijente uz značajno povećanje robusnosti na račun povećavanja opterećenost procesora. Ukoliko je potrebno pratiti brže scene, na račun smanjenja robusnosti, odabire se *SURF* algoritam koji implementira adaptivne veličine deskriptora te tako osigurava i do 3 puta brže izvršavanje algoritma detekcije u usporedbi s *SIFT* algoritmom.

Ukoliko se na slikama na kojima se provode algoritmi detekcije nalaze elementi koji slična područja koja se mogu izdvojiti iz cjeline, radi povećanja brzine izvršavanja algoritma u takvim situacijama, koristi se algoritam detekcije mrlja (*engl. blob detection*) koji grupira elemente jednakih značajki te tako značajno smanjuje opterećenost procesorskog vremena sustava na kojemu je algoritam primijenjen. Ukoliko brzina algoritma nije od prevelikog značenja, već je cilj zadatka dobiti informaciju o mjeri poklapanja predefinirane scene s trenutnom slikom, odabire se algoritam povezivanja značajki koji provodi analizu predodređene i trenutne slike te povezuje značajke koje se u najvećoj mjeri poklapaju.

2.3. Algoritmi za praćenje (*engl. object tracking)* u *OpenCV-u*

2.3.1. Općenito o praćenju

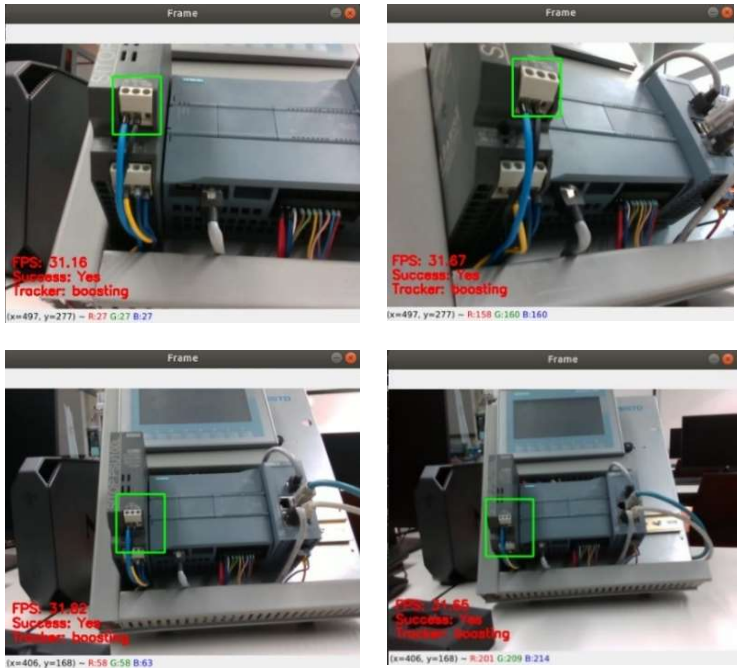
Većina gotovih algoritama za praćenje, koji su razvijeni do vremena pisanja ovog rada, pretpostavljaju postojanje unaprijed obučene neuronske mreže ili nekog drugog klasifikatora obučenog na velikom broju unesenih fotografija objekata koji se prate. Taj princip temelji se na proučavanju značajki praćene scene, uspoređivanje s prethodno naučenim uzorcima te donošenje izvještaja o poklapanju. Ukoliko ne postoji gotovi klasifikator ili ne postoji mogućnost istrenirati vlastiti, potrebno je koristiti metode i algoritme praćenja razvijene i implementirane u biblioteku *OpenCV*, koje se, zbog svoje prirode mogućnosti momentalnog korištenja bez dodatnih učenja mreža, nazivaju "*online*" ili "*on the fly*" algoritmi praćenja.

Algoritmi praćenja su, radi svoje prirode izvršavanja i potrebnih izračuna, višestruko brži od algoritama detekcije. Prilikom kontinuiranog provođenja algoritma detekcije na videozapisu u svakom kadru potrebno je provesti cijeli kod usporedbe prethodno naučenih klasifikatora s trenutnom slikom. Uzimajući u obzir ograničeni broj operacija procesora u jedinici vremena te simultano izvršavanje više procesa na istom upravljačkom računalu (kao što je slučaj u mobilnoj robotici gdje jedno računalo provodi čitanje slike s kamere, algoritme detekcije, planiranje trajektorije, računanje referenci te slanje istih na *low level* kontrolere) može dovesti do značajnog kašnjenja i nemogućnosti donošenja odluka u željenom vremenskom intervalu. Uzimajući u obzir gore navedenu problematiku kontinuiranog detektiranja predmeta nastali su algoritmi praćenja kojima, jednom kad se odredi okvir praćenja, prema određenim značajkama nastoje pratiti zadano područje na slici. Svojom arhitekturom i principom rada nastoje manje opteretiti sustav te omogućiti sustavu dodatne mogućnosti kao što je izračun položaja, brzina, smjera kretanja ili drugih sličnih značajki koje nisu bile dostupne korištenjem algoritama

detekcije zbog daleko kompleksnijih izračuna. Druga važna prednost korištenja algoritama praćenja nad korištenjem algoritama detekcije je stabilnost. Prilikom isprobavanja algoritama detekcije (primjerice *feature matching algoritma*) vidljivo je da je svako iskrivljenje ili djelomično preklapanje testne slike s drugom slikom značilo pogrešnu detekciju. Pri takvim situacijama algoritmi praćenja rade puno bolji posao jer su zbog svojeg načina rada i kontinuiranog praćenja jednog okvira scene robusniji te samim time smanjuju mogućnost gubljenja predmeta koji se prati. Programski kod algoritama praćenja korišten za testiranje preuzet je s izvora [17], uz naknadnu doradu i izmjene željenog tragača (*engl. tracker*).

2.3.2. BOOSTING algoritam za praćenje

Ovaj algoritam temelji se na *AdaBoost* algoritmu (algoritam koji se koristi za strojno učenje) fokusiranom na provođenje u stvarnom vremenu. Upravo zbog svoje namjere, tj. korištenja u stvarnom vremenu, korisnik postavlja okvir u kojemu se nalazi objekt koji se želi pratiti. Označeni objekt smatra se kao uzorak kojega se prati dok se ostatak trenutne slike smatra kao pozadina te se ne proučava. Prilikom dobave nove slike tj. prilikom osvježavanja kadra, klasifikator *BOOSTING* algoritma ocjenjuje piksele unutar i u užoj okolici područja praćenja iz prethodnog koraka te određuje novu poziciju praćenog objekta u novom kadru [18].



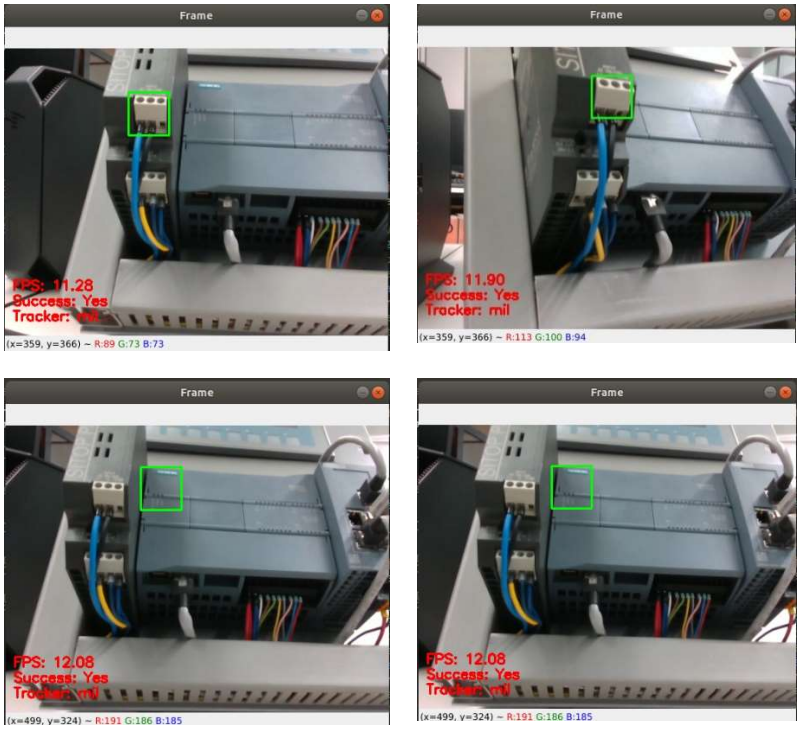
Slika 16. Praćenje rednih stezaljki *PLC-a* pomoću *BOOSTING* algoritma praćenja

Iz slike 16 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja, ali okvir praćenja nije smanjio svoju veličinu prilikom udaljavanja kamere od predmeta već je dio okoliša također postao objektom praćenja.

Proučavanjem rada algoritma i provedenog praćenja raznih objekata dolazi se do zaključka da algoritam relativno dobro prati objekte, ali je brzina niža u odnosu na ostale novije algoritme praćenja. Drugi nedostatak je nastajanje pogrešnih detekcija prilikom izlaska objekta praćenja iz kadra ili prekrivanja objekta drugim objektom. U tim slučajevima algoritam samo nastavlja pogrešno pratiti neku drugu scenu koja mu prema zapisu djeluje najsličnije prethodno praćenoj.

2.3.3. MIL algoritam za praćenje

MIL stoji kao skraćenica izraza za učenja na više instanci (*engl. Multiple Instance Learning*) koji ima jednak pristup praćenju predmeta kao i *BOOSTING* algoritam, ali umjesto da prilikom prelaska iz jedne scene u drugu nagađa mogući položaj praćenog objekta, on promatra okolinu oko objekta koju klasificira i stvara listu potencijalno pozitivnih objekata iz te neposredne blizine starog položaja [18.]



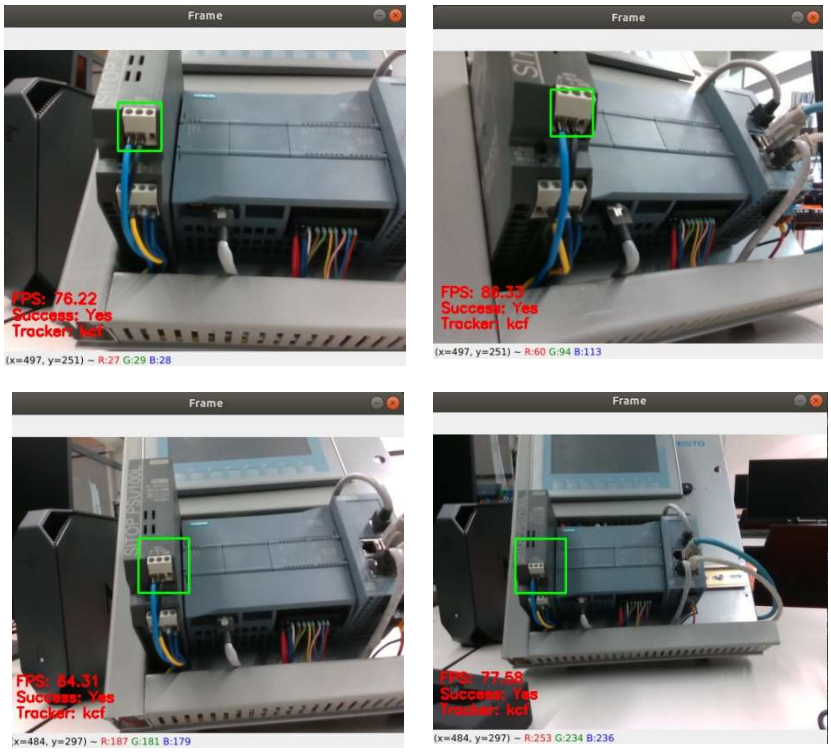
Slika 17. Praćenje rednih stezaljki *PLC-a* pomoću *MIL* algoritma praćenja

Iz slike 17 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja ukoliko se promjena scena događa postepeno i bez prevelikih trzaja što je potrebno izbjegavati zbog relativno sporijeg izvršavanja algoritma. Na donjem dijelu slike vidljivo je da je, zbog nagle promjene kadra i udaljavanja od predmeta praćenja, algoritam zbog svoje prirode i načina rada u kojemu proučava okolinu, predmet ispao iz praćenja te je algoritam nastavio pratiti područje okoliša redne stezaljke iz neposredne blizine predmeta.

Proučavanjem rada algoritma i provedenog praćenja raznih objekata dolazi se do zaključka da algoritam relativno dobro prati objekte ukoliko se scene mijenjaju relativno sporo. Ukoliko se odabrani predmet praćenja izgubi iz okvira, algoritam nije u mogućnosti javiti grešku jer će uvijek barem jedna scena biti pozitivno ocijenjena te će algoritam nastaviti pratiti dio okoline.

2.3.4. KCF algoritam za praćenje

KCF stoji kao kratica za kerneliziranu poveznicu filtera (*engl. Kernelized Correlation Filters*). Načinjen je kao kombinacija dva prethodno spomenuta algoritma, *BOOSTING*-a i *MIL*-a. Kod ovog algoritma se također kontinuirano prati okolina, ali se primjenom korelacijskog filtriranja traže područja najveće mjere preklapanja koje omogućuje da se prati položaj, a samim time i kretanje željenog objekta [18.].



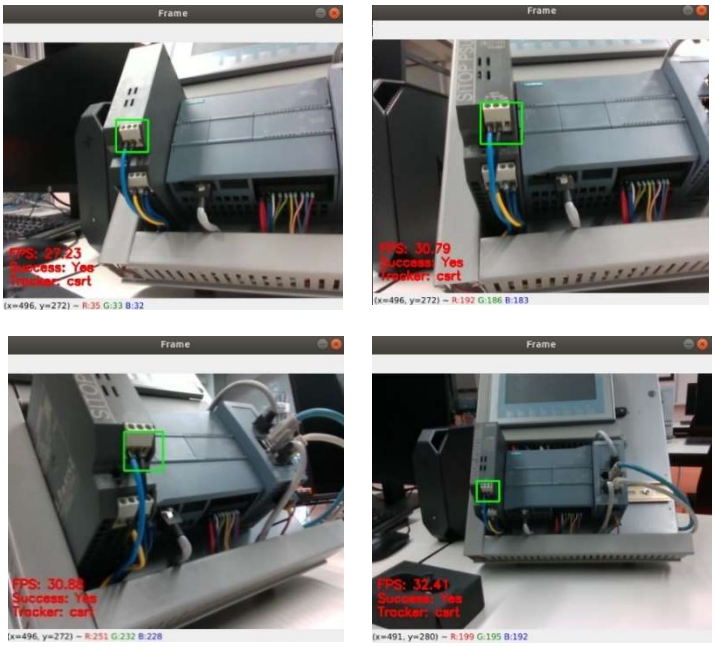
Slika 18. Praćenje rednih stezaljki *PLC-a* pomoću *KCF* algoritma praćenja

Iz slike 18 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja i u slučajevima kada se scena mijenja većom brzinom i uz umjerene trzaje zbog veće brzine računanja koju je moguće postići zbog dobre optimizacije algoritma i načina izvršavanja istog. Na donjem dijelu slike vidljivo je da je, zbog načina izvršavanja algoritma temeljenom na *MIL* algoritmu i postepenog udaljavanja kamere od predmeta, okvir praćenja zahvatio dijelove područja oko predmeta. Ukoliko se predmet prekrije drugim predmetom ili izađe iz okvira kamere, algoritam je sposoban zaključiti da je željeni objekt praćenja izgubljen.

Proučavanjem rada algoritma i provedenog praćenja raznih objekata dolazi se do zaključka da algoritam relativno dobro prati zadane objekte. Ukoliko dođe do gubljenja predmeta praćenja, algoritam će prekinuti isto, ali zbog svoje prirode praćenja kontinuiranim promatranjem okoline nije sposoban nastaviti praćenje pri povratku predmeta u kadar.

2.3.5. CSRT algoritam za praćenje

CSRT stoji kao kratica za diskriminativni korelacijski filtar s kanalnom i prostornom pouzdanošću (*engl. Discriminative Correlation Filter with Channel and Spatial Reliability*). Algoritam svoj rad temelji na učenju korelacijskih filtera u stvarnom vremenu primjenom histograma orijentiranih gradijenata (*engl. Histogram of oriented gradients*, *HOG*). Tako utrenirani filteri se zatim koriste za pretraživanje područja na novoj slici oko mjesta posljednje poznate pozicije na kojoj se nalazio objekt praćenja [18.].



Slika 19. Praćenje rednih stezaljki *PLC-a* pomoću *CSRT* algoritma praćenja

Iz slike 19 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja ukoliko se promjena scena događa relativno postepeno i bez prevelikih trzaja zbog relativno sporijeg izvršavanja algoritma. Na donjem dijelu slike vidljivo je da je, unatoč promjeni kuta i udaljenosti od predmeta praćenja, isti ostao unutar okvira. Prilikom udaljavanja okvir se smanjio na veličinu predmeta te dijelovi okoline nisu postali dio objekta praćenja kao što je bio slučaj u svakom od gore isprobanih algoritama.

Proučavanjem rada algoritma dolazi se do zaključka da je algoritam precizniji ali sporiji od isprobanog *KCF* algoritma uz zamijećenu robusnost prilikom naglih promjena kadrova bez obzira na manju brzinu izvođenja. Algoritam se dobro ponaša prilikom rotacije scene kao i značajnijeg približavanja ili udaljavanja od predmeta praćenja. Nedostatak algoritma je nemogućnost oporavka i nestabilan rad nakon preklapanja predmeta ili gubitka istog iz kadra.

2.3.6. MedianFlow algoritam za praćenje

MedianFlow algoritam se temelji na *Lucas-Canade* algoritmu praćenja, a radi na principu traženja istih točaka na trenutnoj i sljedećoj slici te njihovo praćenje u svim smjerovima u stvarnom vremenu uz davanje procjene pogreški tih trajektorija što mu omogućava relativno dobru i brzu djelomičnu procjenu pozicije objekta praćenja u svakoj novoj slici [18.].

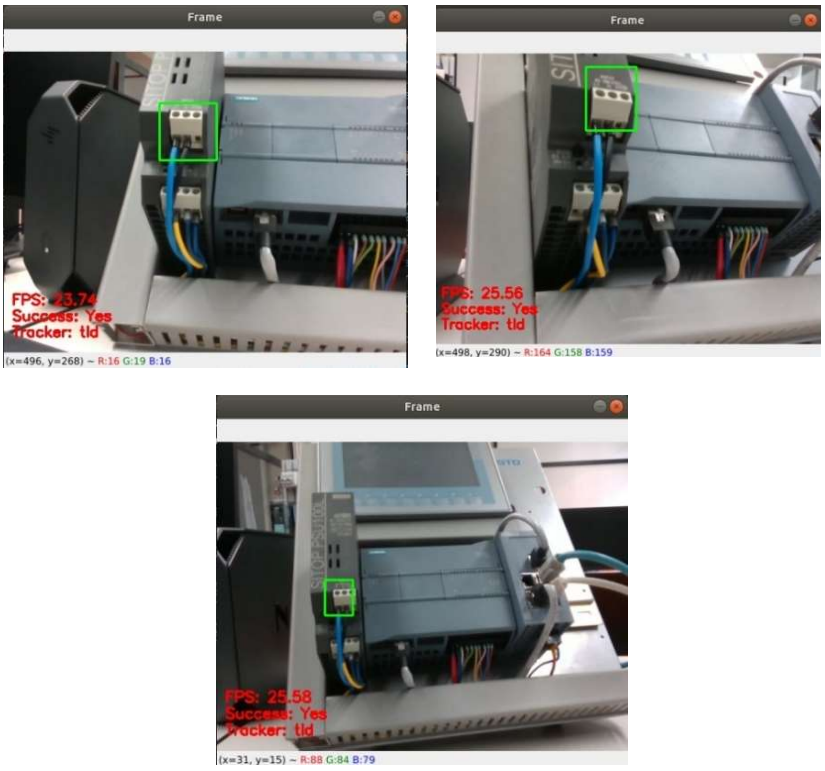
Slika 20. Praćenje rednih stezaljki *PLC-a* pomoću *MedianFlow* algoritma praćenja

Iz slike 20 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja i u slučajevima kada se scena mijenja većom brzinom i uz značajnije trzaje zbog veće brzine računanja koja je ostvarena zbog dobre optimizacije algoritma i načina izvršavanja istog.

Proučavanjem rada algoritma i provedenog praćenja raznih objekata donosi se zaključak da algoritam relativno dobro prati zadane objekte ukoliko se ti objekti kroz provjeru ne preklapaju s drugim objektima, izađu iz kadra te ukoliko brzina gibanja objekata nije prevelika. Ukoliko dođe do gubljenja predmeta praćenja, algoritam će prekinuti isto, ali neće biti sposoban nastaviti praćenje prilikom ponovnog otkrivanja predmeta ili ponovnim ulaskom u kadar.

2.3.7. TLD algoritam za praćenje

TLD stoji kao kratica za praćenje-učenje-detekciju (*engl. Tracking Learning Detection*). Kao što mu i ime govori, algoritam izrađen za zadatak praćenja rastavlja se u 3 zasebna procesa s ciljem dobivanja što boljih rezultata. Algoritam tragača je temeljen na *MedianFlow* algoritmu koji je zadužen za praćenje objekta dok je zadaća detektora analizirati scenu oko regije praćenja i po potrebi prilagoditi okvir praćenja, tj. ispraviti tragač. Treći dio algoritma usmjeren je na učenje značajki objekata koji se prate uz procjenu relativne pogreške nastale u procesu te sprječavanje nastanak novih [18.].



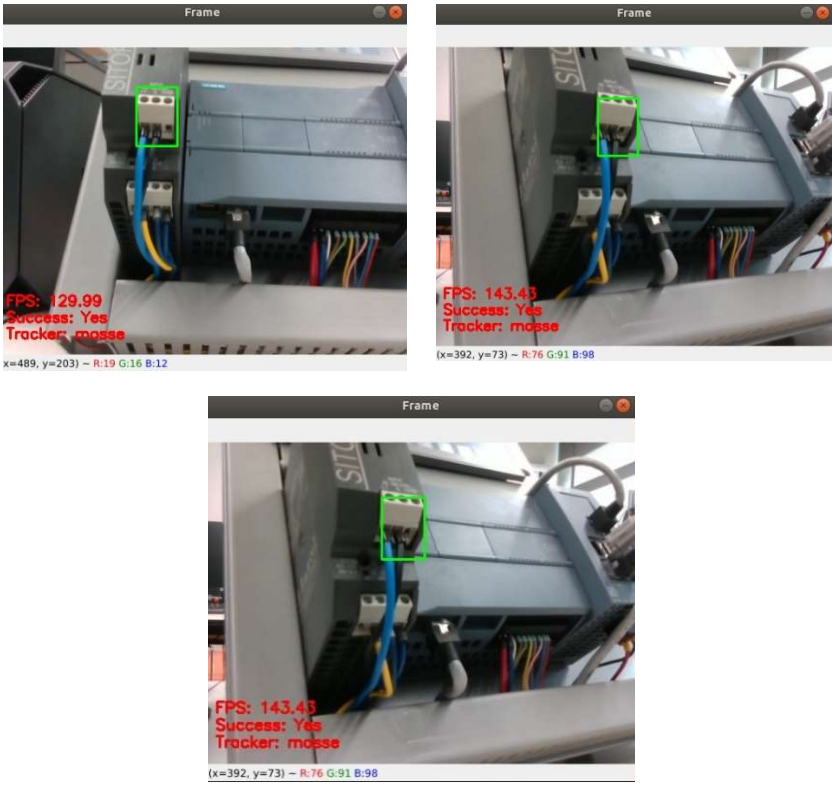
Slika 21. Praćenje rednih stezaljki *PLC-a* pomoću *TLD* algoritma praćenja

Iz slike 21 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja ukoliko se promjena scena događa relativno postepeno i bez prevelikih trzaja zbog relativno sporijeg izvršavanja algoritma.

Ukoliko se objekt praćenja prekrije te se naknadno opet otkrije, algoritam će, zbog svoje prirode izvršavanja najvjerojatnije biti sposoban nastaviti praćenje. Problem algoritma je nastavak praćenja sličnih objekata koji se nalaze u blizini zbog sličnih karakteristika koje algoritam uzima za reference. Pri testiranju rada algoritma isti je nekoliko puta prilikom bržih promjena položaja kamere u odnosu na objekt kao objekt praćenja označio donju rednu stezaljku sa slike koja prvobitno nije bila objekt praćenja, ali izgledom nalikuje na praćeni objekt.

2.3.8. MOSSE algoritam za praćenje

MOSSE stoji kao kratica za minimalnu izlaznu sumu kvadrata greške (*engl. Minimum Output Sum of Squared Error*). Svoj rad temelji na računanju adaptivnih korelacija u *Fourierovom* prostoru. Algoritam filtera minimizira zbroj kvadrata pogreške između stvarnog i predviđenog korelacijskog izlaza [18.].

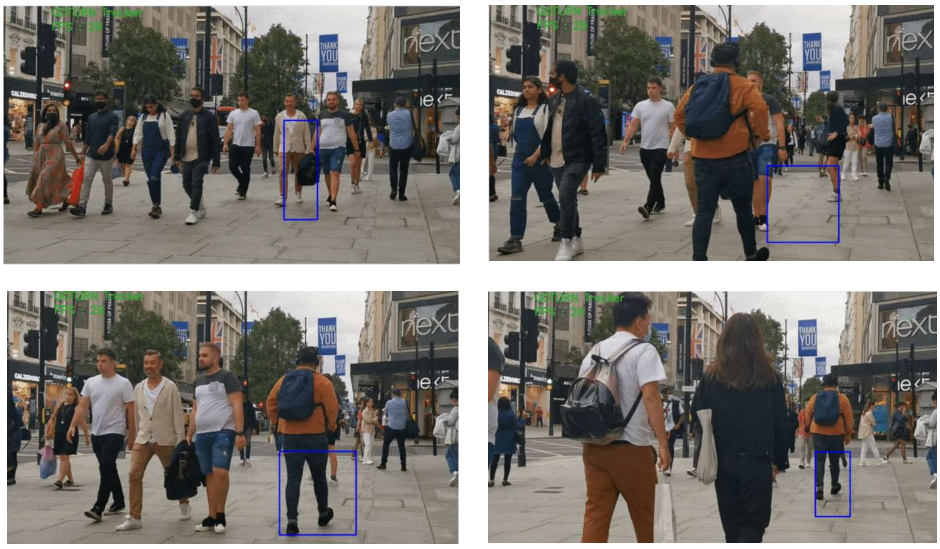


Slika 22. Praćenje rednih stezaljki *PLC-a* pomoću *MOSSE* algoritma praćenja

Iz slike 22 vidljivo je da objekt praćenja (redna stezaljka) relativno dobro ostaje u okviru praćenja i u slučajevima kada se scena mijenja većom brzinom i uz umjerenije trzaje zbog veće brzine računanja koja je ostvarena zbog dobre optimizacije algoritma i načina izvršavanja istog. Algoritam relativno dobro reagira ukoliko se predmet izgubi iz kadra te se relativno brzo pojavi, no ukoliko objekta nema jedno vrijeme u kadru, algoritam će najvjerojatnije započeti pratiti krivi objekt.

2.3.9. GOTURN algoritam za praćenje

GOTURN stoji kao kratica za generičko praćenje objekta pomoću regresijske mreže (*engl. Generic Object Tracking Using Regression Network*). Za razliku od ostalih, ovaj algoritam je "*offline*" tragač jer u svojoj osnovi sadrži duboku konvolucijsku neuronsku mrežu. Algoritam uvijek radi s dvije slike scene, trenutnom i prethodnom. Dok je na prethodnoj slici poznat, na trenutnoj slici položaj praćenog objekta mora se predvidjeti. Obje slike definiraju se kao ulazni parametri koji prolaze kroz konvolucijsku neuronsku mrežu čiji je izlaz skup od 4 točke koje kao koordinate definiraju rubove okvira praćenog objekta. Budući da je algoritam temeljen na korištenju neuronskih mreža, prije upotrebe algoritma potrebno je preuzeti i specificirati model s njegovim težinama koje služe kao individualni parametri za uspješno definiranje neuronske mreže za praćenje specifičnog objekta [18.].



Slika 23. Praćenje osobe pomoću *GOTURN* algoritma praćenja [18]

Iz slike 23. vidljivo je da je, u tijeku praćenja tijela osobe, u jednom trenu došlo do preklapanja praćenog objekta s drugim objektom, prilikom čega je algoritam postepeno nastavio pratiti taj

drugi objekt. Preciznost korištenog algoritma ovisi o parametrima neuronske mreže i kvaliteti ulaznih podataka na kojima je izvršeno treniranje iste te se ista može značajno popraviti daljnjim učenjem mreže na više primjera.

2.3.10. Usporedba algoritama praćenja

Dok se algoritmi detekcije mogu koristiti, kako u analizi videozapisa tako i u radu sa pojedinačnim slikama, algoritmi praćenja svoju namjenu mogu ostvariti isključivo kada se ulazi u algoritam sastoje od niza slijednih slika-videozapis. Ukoliko objekt praćenja mijenja svoj položaj bez drastičnih trzaja te izbivanja izvan vidnog polja kamere ili prekrivanja objekta praćenja drugim objektima, može se koristiti *Boosting* algoritam, koji je prilikom testiranja provedenog s korištenom opremom ostvario prosječno 30 ciklusa izračuna nove pozicije predmeta u sekundi. Ukoliko je za zadatak praćenja potrebna veća robusnost, a objekt se giba relativno sporo, kao algoritam praćenja može se odabrati *MIL* tragač koji se temelji na algoritmima kontinuiranog učenja na više instanci koji je, na korištenoj opremi ostvario prosječno 11 ciklusa izračuna nove pozicije objekta koji se prati. Kao kombinacija dva prethodno spomenuta algoritma nastao je *KCF* algoritam koji, na račun smanjenja robusnosti i mogućnosti pogrešnih praćenja postiže prosječno 80 ciklusa izračuna nove pozicije praćenog objekta. Ukoliko je potrebno precizno praćenje objekta prilikom približavanja i udaljavanja predmeta od kamere, bez dodatnog zahvaćanja dijelova okoline, odabire se *CSRT* algoritam koji se temelji na neprestanom učenju korelacijskih filtra uz ostvarivanje prosječno 30 ciklusa izračuna nove pozicije objekta u jednoj sekundi. U zadacima praćenja trajektorija, kada postoji potreba procjene brzina kao i sljedećeg položaja objekta koji se prati, koristi se *MedianFlow* algoritam, koji, ukoliko ne dolazi do nestajanja predmeta iz kadra ili preklapanja s drugim predmetima, uz korištenu opremu prosječno postiže visokih 125 ciklusa izračuna nove pozicije u jedinici sekunde. Ukoliko se predmet praćenja često prekriva drugim predmetima ili izlazi izvan okvira koje vidi kamera, odabire se *TLD* tragač koji je zbog svoje podjele algoritma praćenja u velikoj mjeri prilikom čestog izmicanja predmeta praćenja iz kadra sposoban isto i nastaviti. Prosječni broj izračuna nove pozicije tragača iznosi 25 ciklusa po sekundi. U situacijama kada se predmet giba relativno velikim brzinama, odabire se *MOSSE* algoritam koji je, radi korištenja optimiziranog algoritma za izračun minimalne izlazne sume kvadrata greške, sposoban ostvariti prosječni broja operacija od visokih 140 ciklusa u jedinici sekunde što mu omogućava praćenje relativno brzih objekata, ali je osjetljiv na gubitak predmeta iz vidnog

polja kamere. Postoji li mogućnost stvaranja specifičnog modela praćenog predmeta s njegovim težinama koji služe kao ulaz u neuronsku mrežu koja radi praćenje objekta te praćenje relativno sporih predmeta, najbolji izbor bilo bi korištenje *GOTURN* algoritma praćenja koji nikad ne analizira cijelu sliku već se obrađuju samo područja oko slike uz donošenje zaključka o smjeru gibanja te fokusiranje područja proučavanja na mjesto na kojemu se očekuje nova pozicija detektiranog objekta, kako bi se značajno ubrzao cijeli proces praćenja.

2.4. Algoritmi za prepoznavanje i praćenje u *ROS-u*

Prema kratkom objašnjenju arhitekture *ROS-a* iz poglavlja *1.3 Robotski operativni sustav (ROS)* vidljivo je da je, za korištenje razvijenih čvorova i algoritama koji su dostupni u okruženju *ROS* Online repozitorija, iste potrebno i preuzeti. Pošto je korišteni *Ubuntu* sustav verzije 18.04 koji podržava *ROS Melodic Morenia*, svi paketi koji su preuzeti kompatibilni su s tom verzijom *ROS*-a.

2.4.1. Paket find_object_2d

Paket *find_object_2d* nalazi se u kategoriji algoritama za prepoznavanje objekata i detekcije sustava. Algoritam u svom radu implementira *SURF, KAZE, SIFT, ORB, FAST. BRIEF* te ostale razvijene algoritme detekcije. Za lakši rad sa sustavom instaliranog paketa razvojni inženjeri su izradili grafičko sučelje pomoću kojega provodimo postupak označavanja novih objekata te pratimo informacije o uspješnosti detekcije prethodno označenih objekata. Paket *find_object_2d* preuzima se s *GitHub* repozitorija [24]. Glavni zadatak algoritma je pronaći predmete prethodno definiranih značajki na trenutnom kadru, odrediti njihov položaj te kao povratnu informaciju sustavu distribuirati koordinate središta detektiranih objekata u ravnini slike. Algoritam bi svoju primjenu mogao pronaći u zadacima s ravninskim rasporedom predmeta kao što je detekcija raznih predmeta u ravnini stola, detektiranju predmeta na pokretnoj traci te ostalim ravninskim zadacima zbog izostanka informacije o udaljenosti predmeta od središta kamere. Ukoliko se radi o zadacima gdje je cilj detektirati predmete koji su raspoređeni u prostoru, koristi se algoritam *find_objects_3d*, koji su detaljnije opisani u slijedećem odjeljku. Oba spomenuta algoritma, *find_objects_2d* kao i *find_objects_3d,* jednostavni su za korištenje upravo zbog intuitivnog grafičkog sučelja koje je prikazano u nastavku.

2.4.1.1. Detekcija položaja predmeta u 2D prostoru

Nakon preuzimanje potrebnog direktorija isproban je algoritam detekcije položaja predmeta u ravnini slike korištenjem samo *RGB* kamere. Za zadatke detekcije predmeta odabran je prethodno spomenuti paket radi relativno dobrih karakteristika kao što je mogućnost detekcije novih i praćenje otprije detektiranih objekata. Kompletna detekcija predmeta pokreće se korištenjem modificirane *.launch* datoteke prikazane u nastavku:

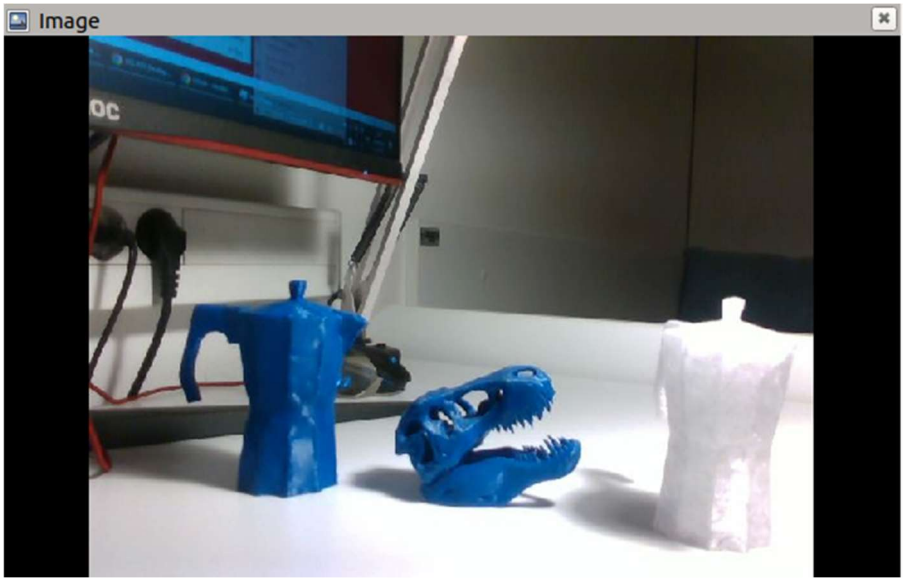
```
<?xml version="1.0"?>


<launch>
    <arg name="gui" default="true"/>
    <arg name="image_topic" default="/camera/color/image_raw"/>
    <arg name="objects_path" default="~/objects"/>
    <arg name="settings_path" default="~/.ros/find_object_2d.ini"/>

      <!-- Nodes -->
      <node name="find_object_2d" pkg="find_object_2d"
type="find_object_2d" output="screen">
            <remap from="image" to="$(arg image_topic)"/>
            <param name="gui" value="$(arg gui)" type="bool"/>
            <param name="objects_path" value="$(arg objects_path)"
type="str"/>
            <param name="settings_path" value="$(arg settings_path)"
type="str"/>
      </node>
</launch>
```

Launch datoteka služi za jednostavnije pokretanje *find_object_2d* čvora uz podešavanje tema na koji se čvor pretplaćuje za ispravan rad. Parametri koje je moguće podesiti u navedenoj *.launch* datoteci su sljedeći:

- *gui* – parametar koji stoji za pokretanje grafičkog sučelja, postavljeno na vrijednost *true* (*hrv. točno)*
- *image_topic* – parametar naziva teme za dobavu slike u *RGB* formatu na kojoj se provodi detekcija objekata, postavljen na temu */camera/color/image_raw*
- *objects_path* – parametar koji definira datoteku u kojoj se nalaze prethodno spremljeni objekti detekcije
- *settings_path* – parametar koji definira datoteku u kojoj su zapisane postavke korištenog detektora

Pokrenuti čvor objavljuje podatke na sljedeće teme:

- *objects* – tema na koju se objavljuju podaci o okvirima stvorenih oko detektiranih objekata. Parametri pozicije okvira zapisani su u obliku 3x3 homogene matrice
- *objectsStamped* – prikaz detektiranog objekta uz oznaku koja sadrži ime objekta. Korisno kada se istovremeno detektira veći broj objekata



Slika 24. Prikaz objekata nad kojima je testiran *find_object_2d* čvor

Slika 24. prikazuje 3 objekta koji su korišteni kao predmeti detekcije. Pošto je ovaj algoritam temeljen na prepoznavanju predmeta isključivo analizom slike dobivene kamerom, predmeti koji se detektiraju imaju različite karakteristike kako bi mogli donijeti zaključke o ponašanju algoritma u raznim uvjetima. Okolina detekcije predmeta namjerno nije odabrana kao idealna, već se u vidnom polju kamere također nalaze i brojni ostali predmeti koji potencijalno predstavljaju problem algoritmu detekcije. Osvjetljenje objekata je postavljeno tako da je srednji predmet (3D isprintana glava dinosaura) uvijek idealno direktno osvjetljen dok preostala dva predmeta (3D isprintana kuhala za kavu) zbog svoje geometrije uvijek imaju barem jednu plohu u sjeni. Model glave dinosaura odabran je zbog velikog broja različitih sitnih detalja koji bi trebali olakšati prepoznavanje i povećati jednoznačnost modela dok su modeli kuhala za kavu odabrani zbog velikog broja ravnih ploha i manjeg broj jedinstvenih detalja koji bi trebali otežati prepoznavanje i smanjiti stabilnost modela praćenja. Kuhalo s desne strane isprintano je s prozirnim filamentom kako bi se dodatno otežalo prepoznavanje značajki, smanjio kontrast objekta i okoline te kako bi se dodatno vidjela sposobnost detekcije korištenog algoritma. U

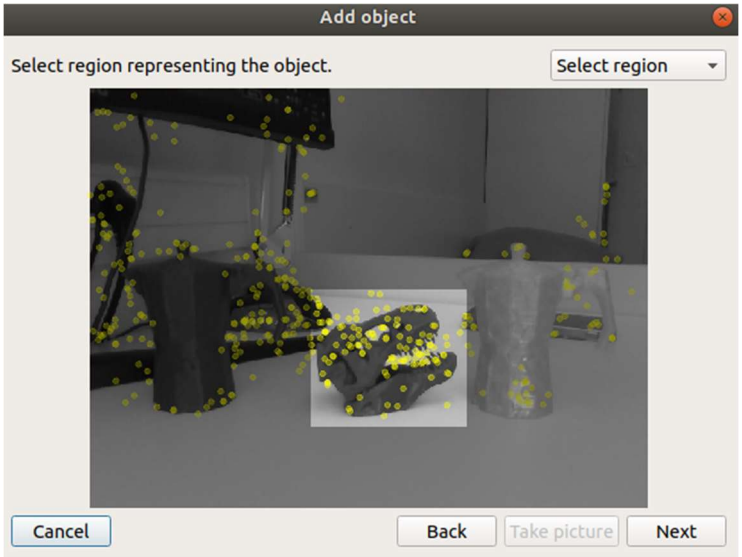
postavkama algoritma (*find-object/include/find_object/Settings.h*) moguće je mijenjati algoritme koji se koriste kao detektori za pronalazak predmeta.

```
        //List format : [Index:item0;item1;item3;...]
#if CV_MAJOR_VERSION < 3 || (CV_MAJOR_VERSION == 4 && CV_MINOR_VERSION <= 3) || (CV_MAJOR_VERSION == 3 &&
(CV_MINOR_VERSION < 4 || (CV_MINOR_VERSION==4 && CV_SUBMINOR_VERSION<11)))
#if CV_MAJOR_VERSION >= 3 // NONFREE=SURF, DEFAULT=SIFT
        PARAMETER_COND(Feature2D, 1Detector, QString, FINDOBJECT_NONFREE,
"9:Dense;Fast;GFTT;MSER;ORB;SIFT;Star;SURF;BRISK;AGAST;KAZE;AKAZE;SuperPointTorch" ,
"9:Dense;Fast;GFTT;MSER;ORB;SIFT;Star;SURF;BRISK;AGAST;KAZE;AKAZE;SuperPointTorch", "Keypoint detector.");
```

Gore prikazani kod dio je programske datoteke pisane u *C++* programskom jeziku koji je namijenjen za odabir detektora. Zbog najveće postignute robusnosti i stabilnosti prilikom testiranja spomenutog algoritma, odabire se algoritam *KAZE*.

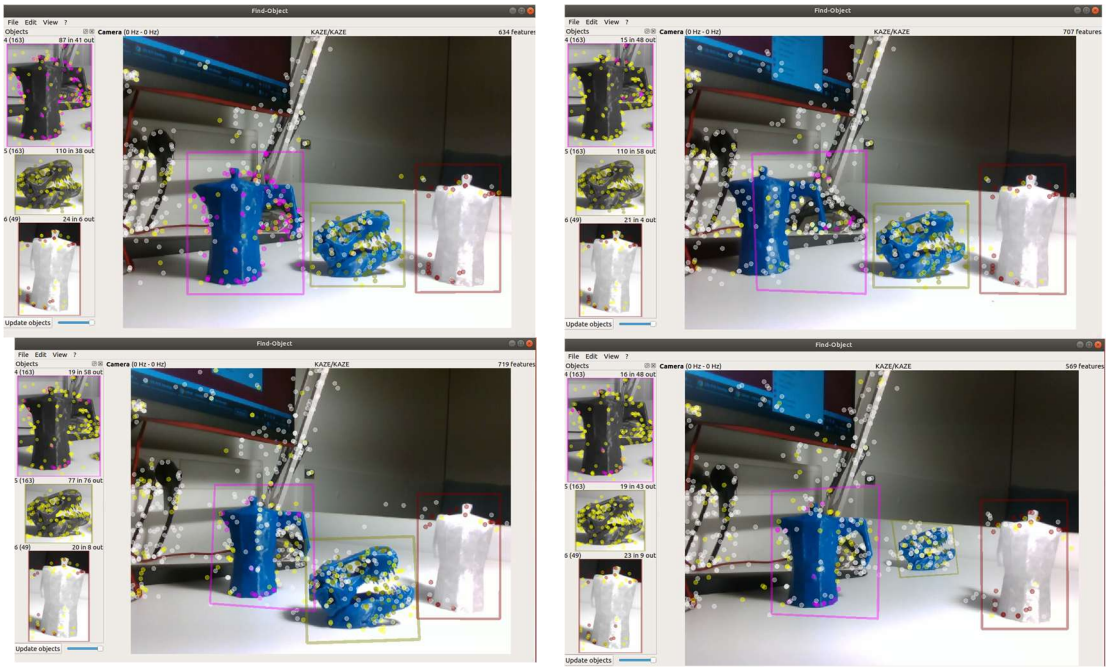
KAZE algoritam služi za detekciju i opisivanje dvodimenzionalnih značajki predmeta koji se detektira u nelinearnom prostoru. Algoritmi za prepoznavanje objekata koji koriste *Gaussovo* zamućivanje distordiraju prirodne rubove promatranog objekta te samim time smanjuju točnost takve metode. Nasuprot tim metodama, *KAZE* algoritam sprovodi detekciju i opis prostornih značajki uz pomoć postupka nelinearnog difuzijskog filtriranja koje metodi omogućavaju zamućenje slike radi pojednostavljenja postupka računanja ali uz očuvanje prirodnih granica detektiranih objekata te značajno bolju prepoznatljivost. Algoritam je pokazao značajno bolje rezultate i preciznost detekcije od ostalih spomenutih i isprobanih algoritama, ali je, zbog korištenja nelinearnog prostora, zahtjevniji za izračun od prethodno spomenutog *SURF* algoritma te se pokazao približno jednako zahtjevan kao gore spomenuti *SIFT* algoritam (25).

Zbog korištenja grafičkog sučelja proces označavanje objekata provodi se relativno jednostavno, što je prikazano na slici 25.



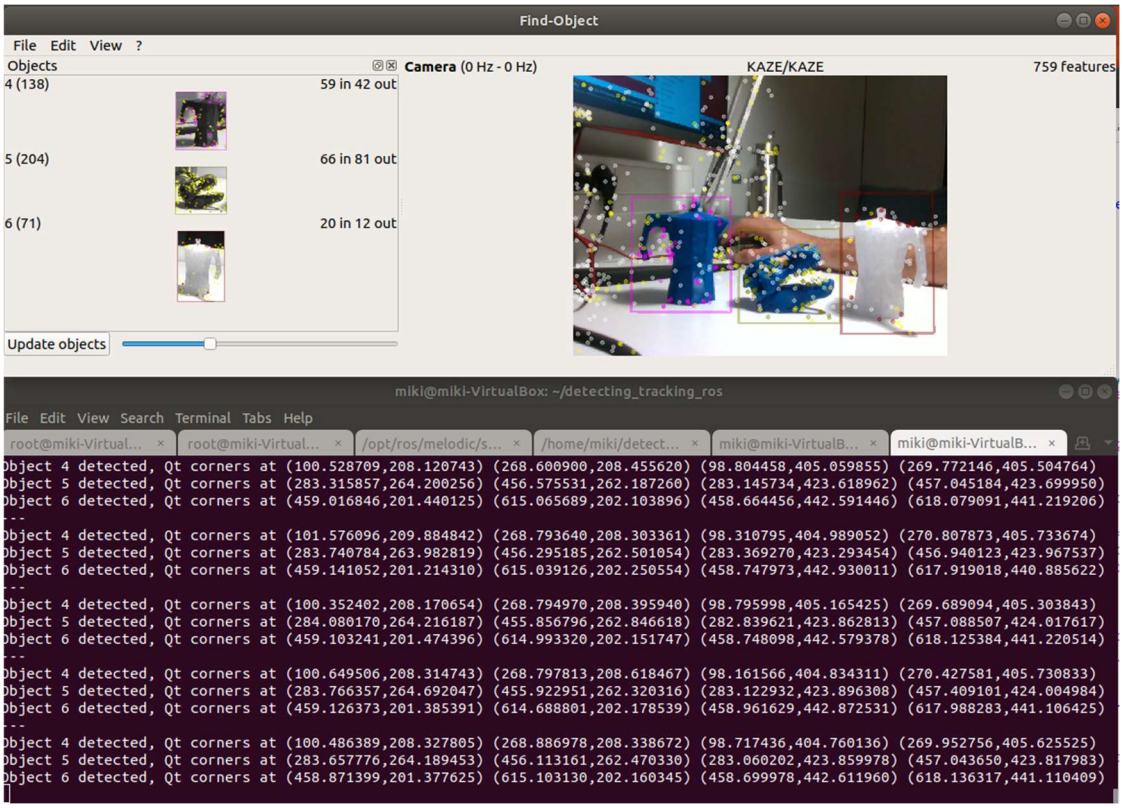
Slika 25. Postupak dodavanja novog objekta detekcije korištenjem čvora *find_object_2d*

Slika 26. prikazuje primjenu čvora *find_object_2d* za detekciju objekata koji su označeni koristeći grafičko sučelje pokrenuto istovremeno s čvorom iz koje je vidljivo ponašanje detektora pri detekciji 3 predmeta prilikom promjene pozicije i kuta osvjetljenja. Detektor relativno dobro iznova pronalazi predmet ukoliko se isti približava, udaljava ili čak rotira za određeni iznos kuta (predmet nije centralno simetričan pa veliki utjecaj za dobivanje uspješne detekcije ima orijentacija istoga). Kod predmeta 1 vidljivo je kako se, zbog označavanja dijela okoline prilikom inicijalizacije predmeta koji se prati kao područje detektiranog predmeta označava i dio stalka monitora te računalni miš, što dovodi do krivih rezultata detekcije prilikom relativnog pomicanja predmeta u odnosu na okolinu.



Slika 26. Prikaz objekata detektirani pomoću čvora *find_object_2d*

U paketu *find_object_2d* nalazi se skripta *print_objects_detected* koja se pretplaćuje na istoimeni čvor te omogućava dobavu koordinata sva četiri ruba okvira koji okružuje detektirani predmet. Slika 27. prikazuje koordinate detektiranih rubova predmeta ispisane u terminalu nakon pokretanja skripte *print_objects_detected*.



Slika 27. Prikaz koordinata detektiranih rubova skripte *print_objects_detected*

2.4.1.2. Detekcija položaja predmeta u 3D prostoru

Nakon testiranja ponašanja algoritma detekcije položaja predmeta u ravnini slike korištenjem samo *RGB* kamere, sustavu je nadodana informacija o dubinskoj slici te je isproban algoritam detekcije položaja predmeta u prostoru. Za zadatke detekcije predmeta odabran je prethodno spomenuti paket radi relativno dobrih karakteristika koje su spomenute u prethodnom odlomku. Kompletna detekcija predmeta pokreće se korištenjem modificirane *.launch* datoteke čiji kod je prikazan u nastavku:

```
<?xml version="1.0"?>
<launch>
      <arg name="object_prefix" default="object"/>
      <arg name="objects_path"  default=""/>
      <arg name="gui"           default="true"/>
      <arg name="approx_sync"   default="true"/>
      <arg name="pnp"           default="true"/>
      <arg name="tf_example"    default="true"/>
      <arg name="settings_path" default="~/.ros/find_object_2d.ini"/>

      <arg name="rgb_topic"         default="/camera/color/image_raw"/>
      <arg name="depth_topic"
default="/camera/depth/image_rect_raw"/>
      <arg name="camera_info_topic" default="/camera/color/camera_info"/>

      <node name="find_object_3d" pkg="find_object_2d"
type="find_object_2d" output="screen">
            <param name="gui" value="$(arg gui)" type="bool"/>
            <param name="settings_path" value="$(arg settings_path)"
type="str"/>
            <param name="subscribe_depth" value="true" type="bool"/>
            <param name="objects_path" value="$(arg objects_path)"
type="str"/>
            <param name="object_prefix" value="$(arg object_prefix)"
type="str"/>
            <param name="approx_sync" value="$(arg approx_sync)"
type="bool"/>
            <param name="pnp" value="$(arg pnp)" type="bool"/>
            <remap from="rgb/image_rect_color" to="$(arg rgb_topic)"/>
            <remap from="depth_registered/image_raw" to="$(arg
depth_topic)"/>
            <remap from="depth_registered/camera_info" to="$(arg
camera_info_topic)"/>
      </node>
            <node if="$(arg tf_example)" name="tf_example"
pkg="find_object_2d" type="tf_example" output="screen">
            <param name="object_prefix" value="$(arg object_prefix)"
type="str"/>
      </node>
</launch>
```

Parametri koje je potrebno podesiti u spomenutoj *.launch* datoteci, uz one spomenute u dijelu objašnjavanja rada *find_objects*_2d čvora su sljedeći:

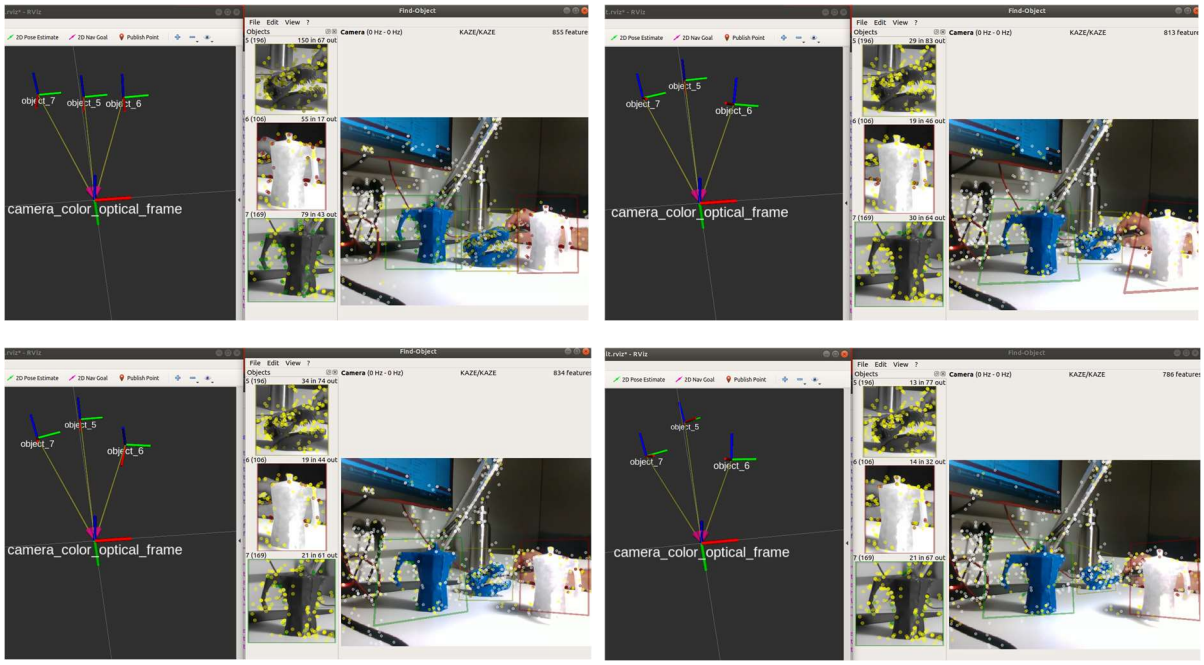
- *object_prefix* – parametar koji stoji za predodređeni naziv svakog novog predmeta koji se detektira, postavlja se na vrijednost *object*
- *camera_info_topic* – parametar koji čvoru daje podatke o korištenoj kameri, a postavlja se na vrijednost */camera/color/camera_info*
- *depth_topic* – parametar koji čvoru detektora pruža uvid u *point cloud* sliku područja s ciljem određivanja međusobnog položaja detektiranog objekta i kamere. Postavlja se na vrijednost */camera/depth/image_rect_raw*
- *rgb_topic* – parametar koji stoji za odabir teme koja dobavlja podatke o slici koju vidi kamera, a postavljena je na vrijednost */camera/color/image_raw*
- *tf_example* – parametar koji definira mjesto prikaza dobivenih transformacija detektiranih objekata u odnosu na referentni sustav kamere, postavlja se na vrijednost *screen*

Pokrenuti čvor objavljuje podatke na sljedeće teme:

- *objects* - tema na koju se objavljuju podaci o okvirima stvorenih oko detektiranih objekata. Parametri pozicije okvira zapisani su u obliku 3x3 homogene matrice uz prikaz transformacije detektiranog objekta u odnosu na referentni koordinatni sustav
- *objectsStamped* – prikaz detektiranog objekta uz oznaku koja sadrži ime objekta. Korisno kada se istovremeno detektira veći broj objekata

Slika 28. prikazuje čvor *find_object_3d* u radu uz dodatni prikaz u *RViz* grafičkom sučelju. Kao predmeti detekcije korišteni su predmeti kao i oni pod naslovom *2.4.1.1 Detekcija položaja predmeta u 2D prostoru* iz razloga navedenih u spomenutom poglavlju. Algoritam je, nakon provedenog postupka ručnog označavanja predmeta, prema položaju detektiranog objekta na *2D* slici odredio srednju vrijednost udaljenosti označenog okvira korištenjem *point claud* skupa podataka te istu prikazao u *RViz* grafičkom sučelju kao novi koordinatni sustav. Promjenom položaja detektiranih predmeta koordinatni sustavi im se zakreću te pomiču u odnosu na

referentni sustav kamere, što je bilo i za očekivati. Nakon što smo dobili koordinatni sustav detektiranog objekta, moguće je napraviti dinamičku transformaciju tog koordinatnog sustava u koordinatni sustav robota kako bi se robotu omogućilo lokaliziranje mete u stvarnom vremenu te kako bi bilo moguće isplanirati trajektorije mobilnog robota do željene pozicije, što je prikazano u poglavlju *4.Implementacija algoritma za prepoznavanje i praćenje na upravljačko računalo mobilnog robota.*



Slika 28. Prikaz parametara skripte *find_object_3d.launch*

3. PRAĆENJE I PREPOZNAVANJE MARKERA

3.1. Općenito o markerima

Poznavanje položaja nekog objekta u prostoru igra vrlo važnu ulogu, kako u zadacima virtualne stvarnosti, računalnog vida tako i u mobilnoj robotici. Najlakši i najprimjenjiviji način definiranja položaja nekog objekta u prostoru izvodi se korištenjem standardnih markera. Proces se temelji na uspoređivanju (pronalaženju podudarnosti) između piksela dobivenih vizijskim sustavom i njihove prethodno implementirane 2D projekcije. Najpopularnija skupina markera koji se koriste u zadacima definiranja položaja nekog objekta u prostoru su, zbog svoje robusnosti i stabilnosti, fiducijalni (povjerljivi) markeri kvadratnog oblika. Glavna prednost korištenja takvih markera je dobivanje dovoljne količine podataka korištenjem isključivo slike s kamere te dobivanje dovoljne količine podataka da se omogući određivanje relativne pozicije detektiranog markera u odnosu na koordinatni sustav robota. Također, unutar svakog fiducijalnog markera nalazi se zapis izražen binarnom kodifikacijom koja služi za raspoznavanje različitih klasa markera te također doprinosi i poboljšanju robusnosti cijelog modela detekcije i praćenja. Prilikom korištenja markera treba voditi računa o prostornom položaju te rotaciji markera u odnosu na kameru, osvjetljenju kao i rezoluciji korištene kamere kamere jer svi ti parametri određuju uspješnost provedene detekcije. Algoritmi prepoznavanja i praćenja markera su radi svoje metode i upotrebe standardnih objekata detekcije manje zahtjevni za sustav od algoritama za detekciju ostalih predmeta. Najčešće su korišteni u zadacima kalibracije, lokalizacije kao i u zadacima navigacije i mapiranja prostora.

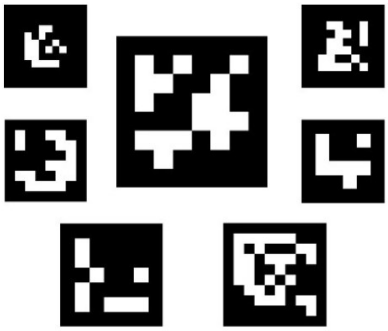
Najrasprostranjenije vrste fiducijalnih markera su *AprilTag* i *ArUco* markeri. *AprilTag* markeri se koriste od strane *NASA-e* za gore spomenute zadatke. Diče se daleko većom vjerojatnosti detekcije na veće udaljenosti od ostalih tipova markera uzimajući u obzir isti *hardware* na kojem se testiranje provodi zbog korištenja optimiziranijeg algoritma. Markeri koji se koriste za detekciju ne moraju nužno biti kvadratnog oblika veće je moguće generirati markere drugih oblika, primjerice okrugle markere. Korištenjem *AprilTag* markera dobiva se mogućnost korištenja paketa više oznaka kombiniranih u jednu, što zbog korištenje više oznaka u različitim orijentacijama povećava robusnost sustava određivanja položaja te značajno eliminira probleme dvosmislenosti orijentacije detektiranog predmeta. Nedostatak korištenja *AprilTag* markera je izostanak *OpenCV* implementacije, što nam ostavlja mogućnost implementacije samo u *ROS*

okruženju. Druga često korištena vrsta markera su *ArUco* markeri. Njihovo postavljanje se provodi vrlo jednostavno uz postizanje relativno visoke razine pouzdanosti detektiranih objekata korištenjem već predefiniranih parametara. Nedostaci korištenja *ArUco* markera su značajnija rotacijska višeznačnost ukoliko se marker nalazi na relativno velikoj udaljenosti od kamere kao i složeniji algoritam detekcije koji koristi značajno veće procesorske resurse sustava na kojemu se primjenjuje. U ovom radu su radi postojanja *OpenCV* implementacije korišteni *ArUco* markeri, čiji se detaljniji opis nalazi u nastavku.

3.2. *ArUco* markeri

3.2.1. Općenito o ArUco markerima

ArUco marker je sintetički kvadratni marker sastavljen od binarne matrice koja služi kao njegov identifikator te širokog crnog ruba koji omogućuje bržu detekciju i ranije otkrivanje istoga na slici. Pošto svaki marker u prostoru može biti pronađen u položaju koji je zakrenut od početnog, svaki rub je određen binarnom kodifikacijom kako bi se omogućilo određivanje izvorne orijentacije markera. Svaki od *ArUco* markera pripada nekom skupu (rječniku) koji je nužno uključiti u *Python/C++/ROS* okruženje kako bi sustav dobio pristup informacijama o skupini markerima koje je potrebno detektirati. Slika 29. prikazuje sedam *ArUco* markera raznih kategorija i veličina.



Slika 29. Primjeri raznih veličina *ArUco* markera [19]

Rječnik *ArUco* markera klasificira se prema dvama glavnim svojstvima markera: veličini rječnika i veličini markera. Veličina rječnika parametar je koji definira broj markera od kojih je sastavljen, a određuje se prema broju kombinacija koje je moguće ostvariti s zadanim brojem bitova. Drugo svojstvo je veličina markera, koje je izravno vezano za prvo svojstvo, a stoji za broj bitova uzduž obje osi u ravnini detekcije markera.

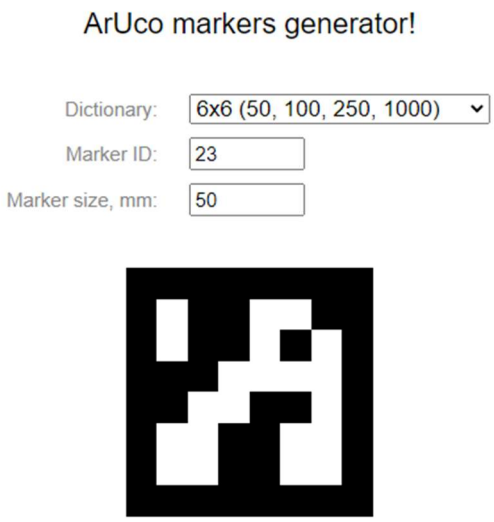
3.2.2. Generiranje željenog tipa ArUco markera

Prije traženja i praćenja markera potrebno je odabrati unaprijed definirani rječnik markera iz *ArUco* modula te generirati željeni marker. Generiranje markera može se izvršiti putem funkcije *drawMarker*() u *Python-u* kao i u *C++* programskom jeziku ili putem *online* generatora markera s grafičkim sučeljem. Koji god način generiranja se koristi potrebno je podesiti sljedeće parametre. Prvi parametar je definicija rječnika iz kojeg se uzimaju gotovi *ArUco* markeri s kojima se u nastavku radi usporedba. Drugi parametar je odgovarajući identifikacijski broj markera koji se mora nalaziti u rasponu definiranog rječnika. Treći parametar koji je potrebno unijeti prilikom izrade *ArUco* markera stoji za veličinu vodeće dimenzije markera u pikselima (visinu i širinu markera pošto se radi o markerima kvadratnog oblika). Ukoliko se marker generira preko *drawMarker*() funkcije, četvrti parametar određuje naziv dobivene slike dok peti parametar stoji kao parametar širine crnog obruba markera.

Kod u nastavku prikazuje poziv funkcije za generiranje *ArUco* markera iz rječnika *dictionary*, identifikacijskog broja 23, vodećih stranica veličine 200 piksela imena *markerImage,* širine crnog ruba od jednog internog bita.

```
cv::Mat markerImage;
cv::Ptr<cv::aruco::Dictionary> dictionary = cv::aruco::getPredefinedDictionary(cv::aruco::DICT_6X6_250);
cv::aruco::drawMarker(dictionary, 23, 200, markerImage, 1);
cv::imwrite("marker23.png", markerImage);
```

Slika 30. prikazuje online generator markera čije grafičko sučelje omogućava brzu i jednostavnu izradu markera iz istog rječnika kao i prethodna skripta pisana u *C++* jeziku.



Slika 30. Prikaz online generatora *ArUco* markera [28]

3.3. Prepoznavanje i praćenje *ArUco* markera

Pošto zadatak detekcije i praćenja markera zahtjeva povezani rad više sustava, u ovom radu prikazan je postupak detekcije markera korištenjem *OpenCV* biblioteka implementiranih u *Python-u* te uz primjenu razvijenih algoritama u *ROS* programskom okruženju.

Detekcija markera je proces koji se izvršava u dva glavna koraka. Prvi korak je prelazak algoritama detekcije po cijeloj slici s ciljem traženja svih objekata koji svojim oblikom nalikuju na marker. To se provodi na način da se traže svi kvadratni oblici prisutni na slici, što algoritmu detekcije uvelike olakšava i ubrzava proces jer su to oblici koje je relativno lako detektirati korištenjem metoda i algoritama detekcije koji su spomenuti u prethodnim poglavljima. Nakon detekcije svih kandidata, dijelovi slike na kojima su detektirani kandidati markera se izdvajaju i provlače kroz određene filtere kako bi se otklonio šum te ostale nepravilnosti nastale prilikom procesa dobave slike ili vanjskih smetnji. Drugi korak detekcije je, nakon određivanja i izdvajanja svih potencijalnih kandidata, očitavanje i analiziranje sadržaja izdvojenih objekata. Cilj tog koraka je izdvojiti kvadratne predmete od markera te očitavanje binarnog zapis središta markera s ciljem njihove identifikacije. Najprije se dijelovi slike koje smo izdvojili u prethodnom koraku prebacuju u kanonski oblik kako bi se omogućilo iščitavanje sadržaja binarnog zapisa izrađenog primjenom crnih i bijelih kvadrata. Posljednji dio algoritma detekcije je provjera pripadnosti očitanih markera skupinama rječnika koji su definirani na početku.

Nakon provedenog postupka detekcije potrebno je odrediti poziciju središta detektiranog markera. Za dobivanje što točnijih rezultata potrebno je načiniti kalibraciju kamere kako bi se dobili parametri distorzije slike te kako bi sustav računalnog vida dobio što vjerniju sliku okoline i davao što vjernije izlazne podatke. U slučaju korištenja *Intel RealSense* kamera nije potrebno raditi kalibraciju ukoliko se nisu radile modifikacije na optici (primjerice mijenjala leća ili slično), što je provjereno mjerenjem udaljenosti dobiveno korištenjem *ROS* sustava i algoritma detekcije markera te ručnim mjerenjem uz pomoć metra te je utvrđena preciznost do maksimalne moguće mjerene ovom metodom provjere (+/- 2 mm što je za mobilnu robotiku i više nego zadovoljavajuće).

3.4. Primjena algoritama za praćenje i prepoznavanje *ArUco* markera korištenjem *OpenCV* biblioteke

U ovom odlomku rada prikazana je primjena razvijenih algoritama računalnog vida za detektiranje i praćenje *ArUco* markera na slici u stvarnom vremenu čija se dobava u sustav vrši korištenjem prethodno spomenute *RealSense* kamere.

Izvršavanje algoritama za detekciju i praćenje markera moguće je uz korištenje dodatnih modula koji se ne nalaze u osnovnoj instalaciji *Python* distribucije. Za iste je potrebno instalirati *OpenCV-contrib-python* modul proširenja koji sadrži sve datoteke i biblioteke potrebne za rad s markerima. Nakon sprovedene instalacije paketa dobiva se pristup kodovima, funkcijama i klasama od kojih je nekolicina izravno upotrjebljena za ispravan rad detekcije i praćenja markera. Kao prvi korak u proceduri otkrivanja markera, potrebno je specificirati *ArUco* rječnik iz kojega se preuzimaju značajke markera koje se uspoređuju s trenutnim kadrovima dobivenih korištenjem kamere. Funkcija inicijalizacije rječnika naziva se *cv2.aruco.Dictionary_get*() koja kao ulazni parametar zahtjeva podatak o nazivu rječnika korištenog markera. Sljedeća korištena funkcija u postupku je *cv2.aruco.DetectorParameters_create*() koja služi za stvaranje i prilagođavanje parametara prema kojima se naknadno provoditi detekcija. Posljednja funkcija iz biblioteke *OpenCV* korištena u algoritmu detektiranja markera nosi naziv *cv2.aruco.detectMarkers*() koja kao ulaze zahtjeva parametar ulazne slike ili videa, rječnik korištenih markera te prethodno stvorenu listu parametara dok kao izlaz funkcije dobivamo listu pozicija vrhova markera, identifikacijski broj markera te listu odbijenih kandidata koji nalikuju na marker ali ne sadržavaju unutarnji kod ili je isti nemoguće očitati. Proučavani kod napisan je u programskom jeziku *Python* te su u njega implementirane gore navedene funkcije uz dodane petlje za provjeru prisutnosti markera, dijelove koda koji obrađuju sliku te otklanjaju šum nanesem nesavršenostima kamere te dijelove koda zadužene za vizualizaciju rezultata detektiranih markera.

Slika 31 grafički prikazuje detektirane *ArUco* markere uz prikaz ID broja svakog od njih uz korištenje gore spomenutog koda. Na slici je vidljiv veći broj tamnih kvadratnih oblik ali je algoritam uspješno izdvojio samo željene markere

Slika 31. Prikaz detektiranih *ArUco* markera [26]

Primjenom takvog algoritma, analizom međusobnih udaljenosti bridova kao i relativnog položaja bridova detektiranog markera u ravnini slike, moguće je dobiti udaljenost i orijentaciju detektiranog markera u odnosu na sustav korištene kamere što je i načinjeno korištenjem paketa *aruco_ros* u nastavku ovog rada.

3.5. Primjena algoritama za praćenje i prepoznavanje *ArUco* markera korištenjem *ROS*-a i *aruco_ros* čvora

U sljedećim poglavljima prikazani su i opisani paketi potrebni za pravilan rad algoritma detekcije i praćenja *ArUco* markera.

3.5.1. Paket *ddynamic_reconfigure*

Prvi od paketa koji je potrebno instalirati je paket naziva *ddynamic_reconfigure*.[20] On je zapravo *C++* ekstenzija *dynamic_reconfigure* paketa kojemu je glavna zadaća omogućiti krajnjem korisniku promjenu parametara kao što su imena, vrste, rasponi vrijednosti te ostale značajke čvorova uživo, tj. bez potrebe za njihovim gašenjem. Paket *ddynamics_reconfigure* proširuje mogućnosti prethodno spomenutom čvoru tako što mu dodaje mogućnost promjene parametara čvorova u stvarnom vremenu bez potrebe za pisanjem *cfg* (*engl. configuration, hrv. konfiguracijskih*) datoteka. Taj paket, iako se ne koristi izravno, zahtjevan je od strane *aruco_ros* paketa jer je isti korišten u programskom kodu čvora *aruco* detektora koji je spomenut u nastavku ovog poglavlja.

3.5.2. Paket *realsense-ros*

Sljedeći paket koji je potrebno instalirati za uspješno izvršavanje zadatka nosi naziv *realsense-ros* [21]. Paket sadrži sve potrebne datoteke i skripte za uspješnu uporabu *Intel RealSense* kamere serije *SR300* i modula za praćenje *T265* te kamera iz serije *D400* kao što je kamera korištena u zadatku.

Nakon provedene instalacije, dobiva se paket u kojemu se nalaze *.launch*, *.cpp* i *.py* datoteke potrebne za ispravan rad kamere. Najbitnija datoteka koja se izravno koristi iz preuzetog direktorija nalazi se u paketu naziva *realsense2_camera* a nosi naziv *rs_camera.launch.* Napisana je za istovremeno pokretanje svih skripti koje će nam omogućiti dobavu slike s *Intel RealSense* kamere u *ROS* okruženje dok je glavni zadatak paketa stvaranje čvora kamere koji će omogućiti daljnju distribuciju slike. Parametre koje je moguće mijenjati u spomenutoj *.launch* datoteci su sljedeći:

- *serial_no* – parametar koji pridružuje serijski broj priključenoj kameri. Ukoliko se parametar ne odabere, serijski broj se generira nasumično
- *usb_port_id* – parametar koji povezuje priključeni uređaj s korištenim *USB* priključkom

- *device_type* – parametar koji služi za odabir uređaja iz liste svih dostupnih uređaja po nazivu modela, u ovom slučaju taj parametar se postavlja na vrijednost *d435*
- *rosbag_filename* – koristi se za objavljivanje teme s *rosbag* datoteke
- *initial_reset* – u slučaju nepravilnog gašenja uređaja potrebno je resetirati programsku memoriju uslijed novog pokretanja čvora kamere, što se postiže postavljanjem ovog parametra na vrijednost *true*
- *reconnection_timeout* – parametar koji određuje potrebno vrijeme nakon neuspjelog spajanja za ponovni pokušaj uspostavljanja veze s uređajem
- *align_depth* – parametar koji određuje korištenje dubinske slike
- *filters* – dodatne radnje provedene na dobivenoj slici scene, odabiru se sljedeći, odvojeni zarezom:
    - *colorizer* – parametar koji, ukoliko se postavi na vrijednost *true*, umjesto 16 bitne sive slike prikazuje sliku u boji
    - *pointcloud* – parametar koji uključuje objavljivanje parametre oblaka točaka na istoimenu temu
    - *hdr_merge* – parametar koji omogućuje stvaranje informacije o dubini na temelju spajanja dva uzastopna kadra snimljeni različitim iznosima ekspozicije i pojačanja
- *enable_sync* – parametar koji omogućava usklađivanje slanja informacija s različitih senzora kako bi svi podaci bili slani u jednakim vremenskim intervalima
- *tf_prefix* – parametar koji određuje ime prikazano ispred svakog koordinatnog sustava. U ovom slučaju ovaj parametar se postavlja na vrijednost *camera_*
- *base_frame_id* – određuje naziv koordinatnog sustava prema kojemu se izvršavaju sve statičke transformacije, što je u ovom slučaju koordinatni sustav robota naziva *robot_link*
- *odom_frame_id* – parametar koji definira koordinatni sustav koji je korišten pri radu s kamerom. Ovaj parametar postavlja se po standardnoj *ROS* konvenciji – X stoji za ravno, Y za lijevo dok Z stoji za gore

Datoteka za pokretanje također nudi promjenu velikog broja ostalih parametara, koji su svi navedeni u [21].

Nakon pokretanja *.launch* datoteke kamere stvaraju se teme na koje će čvor objavljivati podatke. Neke od važnijih tema na koje će čvor kamere objavljivati su:

- */camera/color/camera_info* – tema na koju kamera objavljuje podatke o slici u boji, primjerice razlučivost, frekvenciju isporučene slike te ostale informacije potrebne za rad s istom
- */camera/color/image_raw* – tema na koju se objavljuje trenutna slika koju vidi kamera
- */camera/color/metadata* – tema na koju se objavljuju podaci o korištenom *RGB* senzoru te sustavu obrade podataka kao što su verzija paketa, opisi koda te ostali parametri važni pri korištenju paketa kamere
- */camera/depth/camera_info* – tema koja pruža informacije o dubinskoj slici
- */camera/depth/image_rect_raw* – nefiltrirani i neobrađeni (sirovi) oblik dubinske slike dobiven pomoću laserskih sustava fiksno montiranih na konstrukciju kamere
- */camera/depth/metadata* – tema na koju se objavljuju podaci o dubinskom senzoru te sustavu obrade podataka kao što su verzija paketa, opisi koda te ostali parametri važni prilikom upotrebe paketa kamere
- */camera/extrinstic/depth_to_color* – tema na koju spomenuti čvor objavljuje podatak slike u boji generirane na temelju dobivene dubinske slike
- */camera/extrinstic/depth_to_infra1* – tema koja objavljuje podatke o infracrvenoj slici dobivenoj s lijevog infracrvenog projektora (*engl. imager*)
- */camera/extrinstic/depth_to_infra2* –tema koja objavljuje podatke o infracrvenoj slici dobivenoj s desnog infracrvenog projektora
- */camera/infra1/camera_info* – informacije o lijevom infracrvenom projektoru
- */camera/infra1/image_rect_raw* – nefiltrirani i neobrađeni oblik infracrvene slike s lijevog senzora
- */camera/infra2/camera_info* – informacije o desnom infracrvenom projektoru
- */camera/infra2/image_rect_raw* – nefiltrirani i neobrađeni oblik infracrvene slike s desnog senzora
- */camera/gyro/imu_info* – tema na koju se objavljuju podaci o mjerenoj inerciji dobiveni pomoću žiroskopa (*engl. Inertial Measurement Unit*)

- */camera/gyro/metadata* – tema na koju se objavljuju meta podaci o korištenom žiroskopu
- */camera/gyro/sample* – tema na koju se objavljuju uzorci signala s žiroskopa u određenom vremenskom intervalu, koji se određuje u prethodno spomenutoj *.launch* datoteci
- */camera/accel/imu_info* – tema na koju se objavljuju podaci o mjerenom ubrzanju
- */camera/accel/metadata* – meta podaci korišteni u procesu određivanja ubrzanja
- */camera/accel/sample* – tema na koju se objavljuju uzorci signala o ubrzanju u određenom vremenskom intervalu, koji se također definira u prethodno spomenutoj *.launch* datoteci
- */diagnostics* – tema na koju se objavljuju obavijesti i poruke uslijed pogrešaka i nepravilnosti u radu algoritma kamere

3.5.3. Paket *aruco_ros*

Sljedeći paket koji je potrebno instalirati kako bi se zadatak detekcije *ArUco* markera mogao uspješno izvršiti je *aruco_ros* [22] programski paket koji u sebi sadrži sve datoteke i rječnike koji se koriste prilikom detektiranja i praćenja markera. Paket je razvijen i usavršen od strane grupe programera *Ava* sa Sveučilišta u Cordobi.

Za zadatke detekcije i praćenja *ArUco* markera odabran je prethodno spomenuti paket radi dobrih karakteristika kao što je mogućnost detekcije novih i praćenje otprije detektiranih markera velikom brzinom koja se postiže na račun optimalnog algoritma pisanog u *C* jeziku. Algoritam također sadrži dijelove koda koji služe za izbjegavanje dvosmislenosti i lažnih detekcija te mogućnost implementacije naprednijeg način praćenja objekata korištenjem standardnih ploča s više markera kao i mogućnost implementacije stereo-vizije, sve to radi povećanja preciznosti metode te ostalih značajki navedenih kao prednosti [22]. Kompletna detekcija *ArUco* markera pokreće se korištenjem *.launch* datoteke koja dolazi s instaliranim paketom uz dodatne modifikacije naziva tema na koji se čvor mora pretplatiti za uspješan rad. Programski kod datoteke za pokretanje čvora detektora nalazi se u nastavku:

```
<?xml version="1.0"?>
<launch>

    <arg name="markerId"        default="26"/>
    <arg name="markerSize"      default="0.1"/>    <!-- in m -->
    <arg name="eye"             default="left"/>
    <arg name="marker_frame"    default="aruco_marker_frame"/>
    <arg name="ref_frame"       default="camera_link"/>  <!-- leave empty
and the pose will be published wrt param parent_name -->
    <arg name="corner_refinement" default="LINES" /> <!-- NONE, HARRIS,
LINES, SUBPIX -->


    <node pkg="aruco_ros" type="single" name="aruco_single">
        <remap from="/camera_info" to="/camera/color/camera_info" />
        <remap from="/image" to="/camera/color/image_raw" />
        <param name="image_is_rectified" value="True"/>
        <param name="marker_size"        value="$(arg markerSize)"/>
        <param name="marker_id"          value="$(arg markerId)"/>
        <param name="reference_frame"    value="$(arg ref_frame)"/>   <!--
frame in which the marker pose will be refered -->
        <param name="camera_frame"       value="stereo_gazebo_$(arg
eye)_camera_optical_frame"/>
        <param name="marker_frame"       value="$(arg marker_frame)" />
        <param name="corner_refinement"  value="$(arg corner_refinement)"
/>
    </node>

</launch>
```

Parametri koje je moguće podesiti u spomenutoj *.launch* su sljedeći:

- *marker_id* – definira identifikacijski broj markera iz kategorije *OriginalArUco* kojega postavljamo na vrijednost 26
- *markerSize* –stoji za veličinu vodeće dimenzije markera (širina i visina) te služi za izračun relativne udaljenosti i orijentacije markera u odnosu na kameru. Postavlja se na vrijednost 0.1m
- *eye* -parametar koji se koristi pri upotrebi stereo-vizije i implementacije druge kamere u sustav, definira položaj postavljene kamere
- *marker_frame* –služi za određivanje koordinatnog sustava detektiranog markera. U ovom slučaju postavlja se na vrijednost *aruco_marker_frame*
- *ref_frame* – inicijalizira referentni koordinatni sustav od kojega se računa pozicija markera. Pošto se algoritam koristi u zadacima vezanim za kretanje mobilnog robota, kao referentni koordinatni sustav odabire se referentni sustav robota naziva *robot_link*

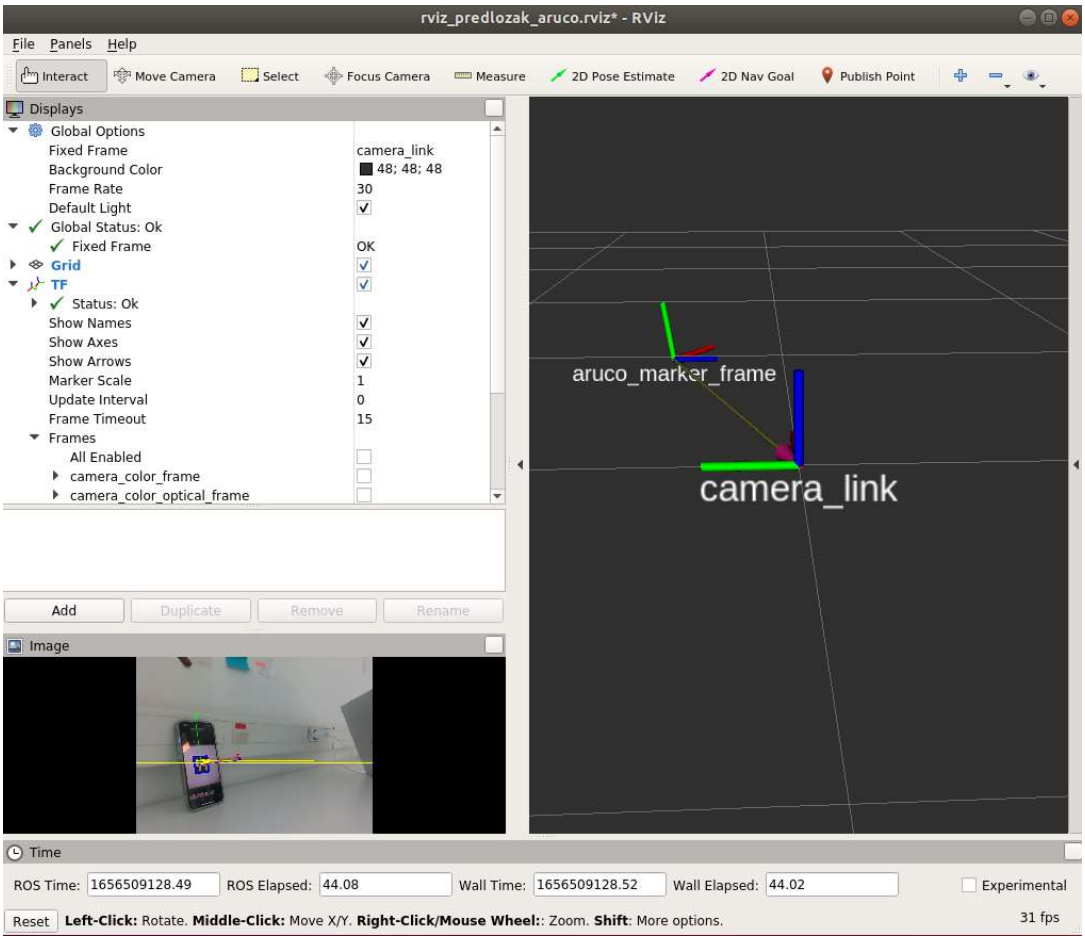
- *corner_refinement* – određuje vizualni prikaz bridova detektiranog markera. U ovom slučaju marker je prikazan linijama, što je definirano parametrom *LINES*
- */camera_info* – parametar koji uzima podatke o kameri, kao što je rezolucija korištenog senzora kamere, maksimalni broj scena u sekundi (*engl, frames per second*, *FPS*), postavlja se na vrijednost */camera/color/camera_info*
- */image* – parametar o trenutnoj slici u boji dobivenoj s kamere, postavlja se na */camera/color/image_raw*

Nakon izvršavanja naredbe pokreće se čvor koji kao glavni ulazni parametar uzima sliku dobivenu uz pomoć kamere, a zadatak mu je provođenje algoritama detekcije i praćenja markera. Čvor također stvara teme na koje se moguće pretplatiti i primati ili objavljivati informacije na iste, od kojih valja spomenuti sljedeće:

- */aruco_single/debug* – tema koja služi za otkrivanje pogrešaka u fazi postavljanja čvora detektora
- */aruco_single/marker* – tema na koju se objavljuju podaci o detektiranim markerima na trenutnoj sceni dobivenoj s kamere
- */aruco_single/pixel* – podatak o lokaciji na slici gdje je otkriven marker odabranog rječnika
- */aruco_single/pose* – tema na koju se objavljuju informacije o položaja izračunatog središta markera. Sastavljena je od podataka o poziciji i orijentaciji detektiranog markera u odnosu na referentni koordinatni sustav kamere.
- */aruso_single/position* – na ovu temu se objavljuje pozicija središta detektiranog markera u odnosu na referentni koordinatni sustav
- */aruco_single/result* – ova tema sadrži podatke o svim markerima koji se nalaze na slici kamere te se ista može prikazati kao slika korištenjem *RViz* sučelja (što je prikazano u sljedećem odjeljku)
- */aruco_single/result/compressed* – komprimirani podaci o svim markerima detektirani na slici kamere
- */aruco_single/transform* – tema na koju se objavljuje dinamička transformacija koordinatnog sustava kamere u odnosu na referentni koordinatni sustav

3.5.4. Alat za 3D vizualizaciju RViz

RViz je grafičko sučelje implementirano u osnovnu verziju *ROS* sustava koje korisniku omogućava brzo prikazivanje brojnih opće korištenih tema iz svijeta robotike. U ovom zadatku *RViz* se koristi za prikaz koordinatnog sustava robota, kamere, *ArUco* markera, njihovih međusobnih transformacija te slike s kamere montirane na robotu.



Slika 32. Prikaz *RViz* sučelja prilikom detekcije *ArUco* markera

Slika 32. prikazuje *RViz* sučelje prilikom detekcije *ArUco* markera uz dodatni prikaz koordinatnih sustava markera i kamere te transformacije između ta dva sustava. Algoritam prepoznavanja *ArUco* primijenjen na upravljačko računalo mobilnog robota prikazan je u nastavku rada.

4. IMPLEMENTACIJA ALGORITMA ZA PREPOZNAVANJE I PRAĆENJE NA UPRAVLJAČKO RAČUNALO MOBILNOG ROBOTA

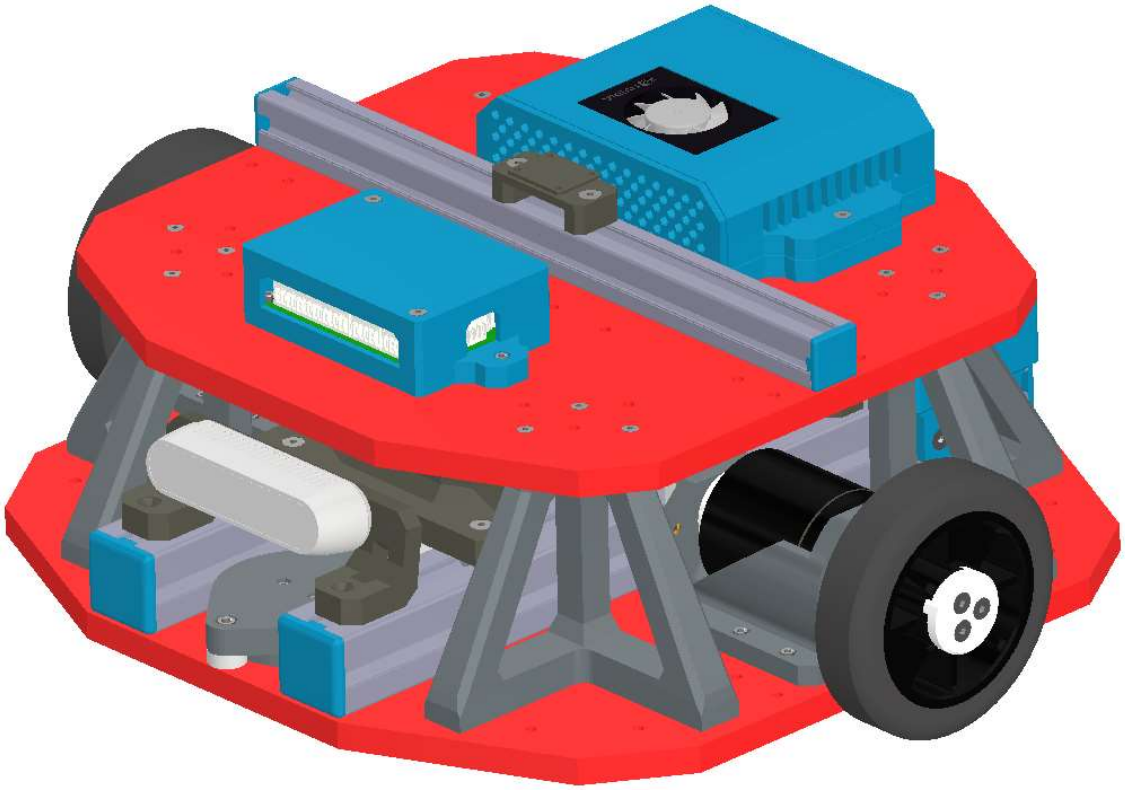
Kako bi se kretanje mobilnog robota uspješno izvršavalo, potrebno je odrediti algoritam po kojemu će robot izvršavati detekciju i praćenje. U ovom radu detektor je, zbog najveće stabilnosti u radu i pouzdanosti u skoro svim uvjetima slabijeg osvjetljenja, odabran algoritam detekcije i praćenja *ArUco* markera opisan u poglavlju *3.2 ArUco markeri*. Kako bi se na mobilnom robotu omogućili svi sustavi koji su potrebni za rad na isti je potrebno instalirati sve pakete navedene u tom poglavlju. Robot također treba na sebi imati postavljenu kameru spomenutu na početku rada ili kameru sličnih svojstva, ali je u slučaju korištenja neke druge kamere potrebno instalirati pakete prilagođene toj korištenoj kameri.

4.1. Istraživački mobilni robot izrađen u laboratoriju *CRTA*

Kao sustav na kojemu je provedeno testiranje algoritama i podešavanje parametara izabran je mobilni robot izrađena u *CRTA-i*. Robot pripada skupini robota s diferencijalnim pogonom te se sastoji od dva usporedna upravljana kotača te dodatna dva slobodno rotirajućih kotača koji su dodani radi stabilnosti te osiguravaju nesmetano kretanje po ravnoj podlozi. Tablica karakteristika mobilnog robota nalazi se u prilogu (I). Slika 33. prikazuje mobilnog robota na kojem su testirani algoritmi praćenja i detekcije.

Slika 33. Mobilni robot iz CRTA-e

Slika 34. prikazuje *CAD* model koji vjerno prikazuje korištenog robota.



Slika 34. Prikaz *CAD* modela korištenog mobilnog robota

4.1.1. Glavne komponente mobilnog robota

Kao što je spomenuto u prethodnim poglavljima, upravljački sustav mobilnog robota sastoji se od tri glavna dijela: *low level* upravljanja koje se provodi na mikrokontroleru *Teensy 4.0*, *high level* upravljanja provođenog na mikroprocesoru *NVIDIA Jetson Xavier NX* te *Odrive* upravljača. U nastavku poglavlja su opisane glavne komponente mobilnog robota koje su zaslužne za njegov rad.

4.1.1.1. NVIDIA Jetson Xavier NX razvojni komplet

Najvažnija komponenta iz perspektive upravljanja procesom gibanja mobilnog robota je mikroprocesorsko računalo *NVIDIA Jetson Xavier NX* opremljeno Linux operativnim sustavom *Ubuntu 18.04 – Bionic Beaver*. Na njega je instaliran ROS inačice *Melodic Morenia*. Korišten razvojni komplet mikroprocesorskog računala prikazan je na slici 35.

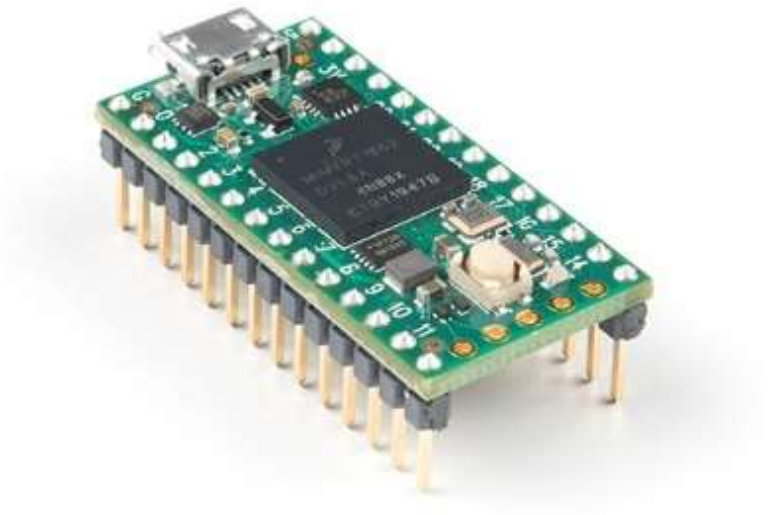
Slika 35. ***NVIDIA Jetson Xavier NX razvojni komplet*****[6]**

Spomenuto mikroprocesorsko računalo odabrano je zbog svoje relativno niske cijene, zadovoljavajućih performansi i mogućnosti *Ethernet*, *Wi-Fi*, *Bluetooth* i *USB* komunikacije. Detaljne karakteristike mikro-računala dane su u prilogu (II). Na spomenutom mikroprocesorskom računalu izvršava se robotski operativni sustav na kojemu se provode algoritmi prepoznavanja i praćenja te provoditi izračuni referenci položaja te potrebne brzine pogonskih motora kotača za postizanje željenog položaja koja se dalje prosljeđuje *low level* kontroleru.

4.1.1.2. Mikrokontroler Teensy 4.0

Kao *hardware* zadužen za rad s aktuatorima i senzorima korišten je mikrokontroler *Teensy 4.0* koji je s *Jetson* mikroprocesorom povezan korištenjem *USB port-a* na mikroračunalu i serijske komunikacije. Mikrokontrolerska pločica zadužena je za upravljanje robota na najnižoj razini što u pravilu znači upravljanje motorima prema referencama dobivenim iz nadređenog *high level* upravljača te očitavanja trenutnog položaja s enkodera koje je nemoguće ostvariti

korištenjem isključivo mikroračunala zbog nemogućnosti postizanja zadovoljavajućih brzina na istomu. Na slici 36. prikazan je mikrokontrolerska pločica *Teensy 4.0*.



Slika 36. *Teensy* 4.0 mikrokontroler [7]

Mikroračunalo *Teensy* odabrano je kao upravljački uređaj prije *Arduino UNO* ili *MEGA* zbog mogućnosti postizanja brzina izvršavanja nekoliko redova većih od *Arduino* pločica što se preslikava na daleko bolje performanse u zadacima čitanja podataka s senzora te slanja referenci. Programiranje i prevođenje programa može se također provoditi uz pomoć software-a *Arduino IDE*, što olakšava i ubrzava proces postavljanja sustava i otklanjanje pogrešaka. Detaljne karakteristike računala dane su u prilogu (III).

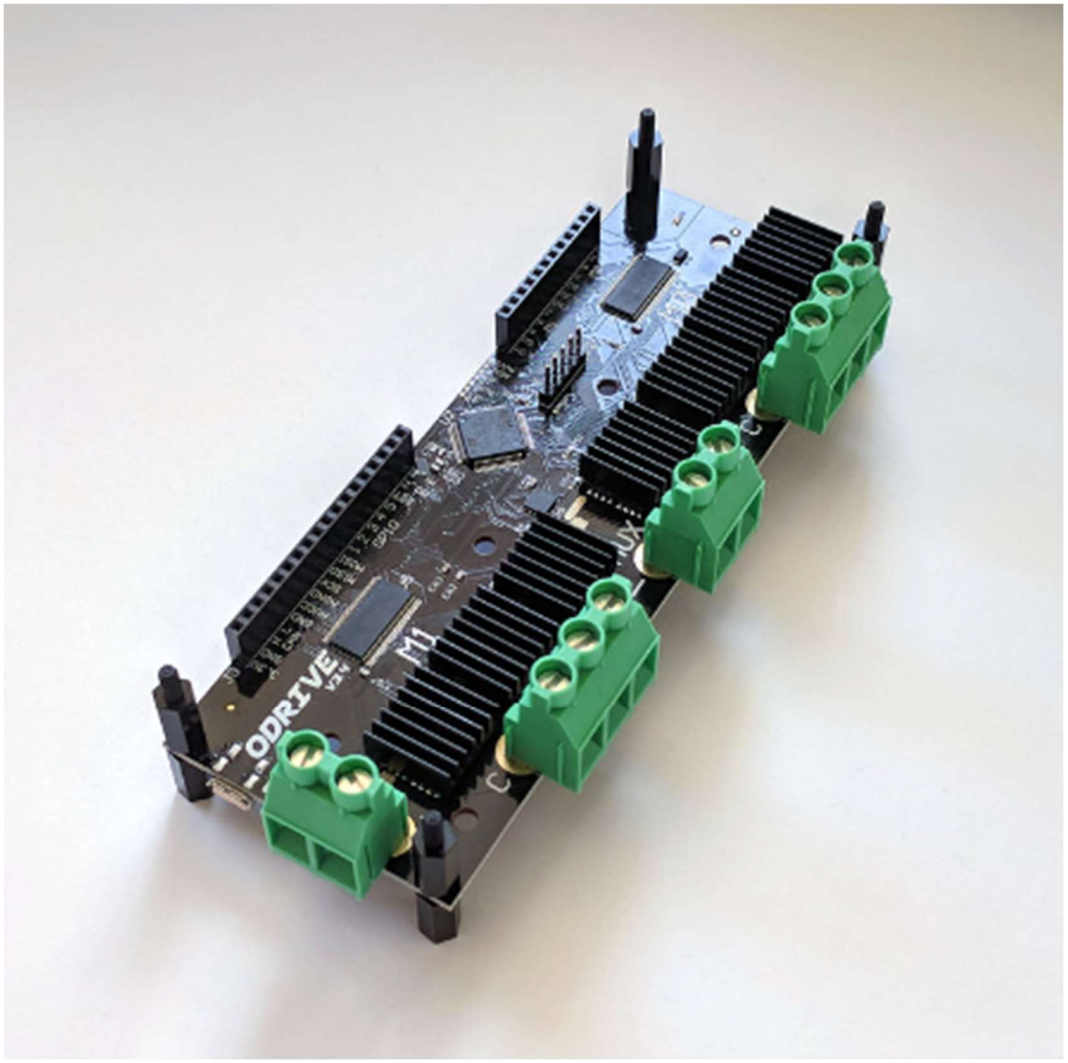
4.1.1.3. odrive V3.6

Odrive je uređaj koji se koristi za precizno upravljanje motorima bez četkica korištenjem petlje povratne veze. Ulazni parametar u algoritam *Odrive-a* je informacija o trenutnom položaju ili brzini očitana uz pomoć prethodno spomenutih enkodera, sustava kamera ili neke treće metode određivanja položaja mobilnog robota. Kao izlazni podatak iz algoritma *Odrive*-a dobivaju se reference prema kojima se zakreću korišteni motori. Motorima se, pomoću spomenutog *Odrive* sustava može upravljati na sljedeće načine:

- *goto* – upravljanje pozicijom motora prema modelu planera putanje

- upravljanje pozicijom uz dodatno korištenje filtera i interpolacija
- upravljanje brzinama dobivenih od upravljača visoke razine
- upravljanje vrijednostima momenata na motorima koje je potrebno postići

Komunikacija između *Odrive* sustava s upravljačkim sustavom visoke razine provodi se korištenjem standardnog *UART* protokola (*engl. universal asynchronous reciever/transmitter, hrv. univerzalni asinkroni prijamnik/predajnik*). Slika 37. prikazuje *Odrive V3.6* uređaj koji se koristi za regulaciju brzine vrtnje i/ili položaja *BLDC* motora.

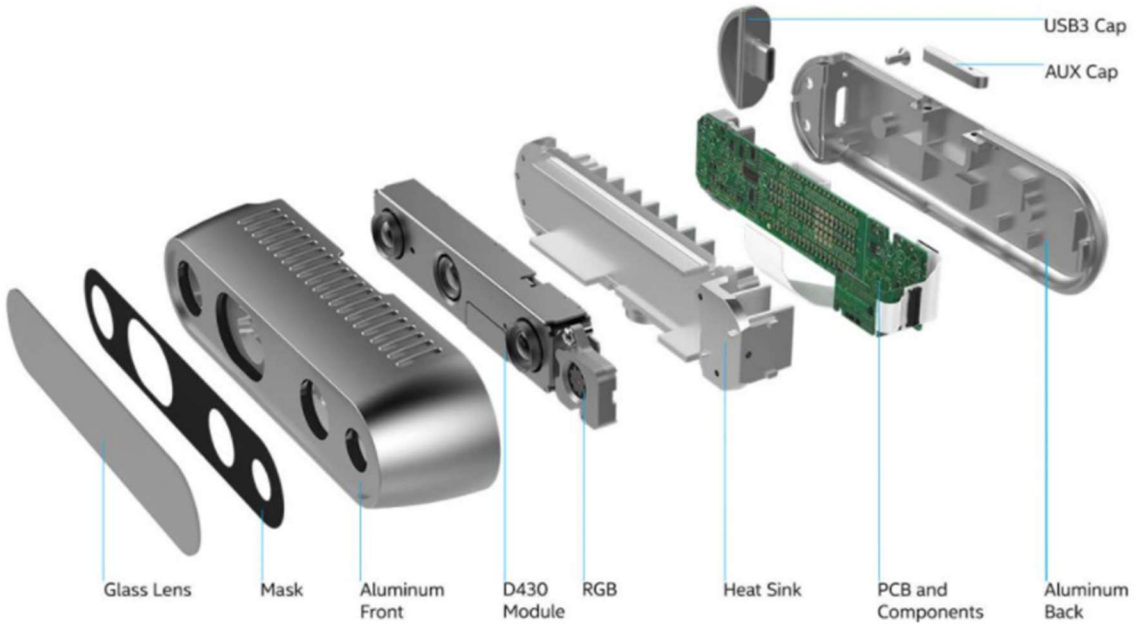


Slika 37. Prikaz korištenog *Odrive 3.6* sustava [31]

Detaljne karakteristike kao i korisnički vodič [31].

4.1.1.4. Intel RealSense D435i kamera

Kao uređaj za prikupljanje informacija o stanju okoliša u kojem je smješten robot korištena je *Intel Realsense D435i* dubinska kamera. Prije početka upotrebe potrebno je instalirati *Intel Realsense SDK* (*engl. software development kit, hrv. programski razvojni komplet*) koji je zadužen za obradu i prikaz slike na mikroprocesoru dok se ista s mikroprocesorom spaja putem *USB* 3.1 tip C priključka. Slika 38. prikazuje segmente *RealSense D435i* kamere.



Slika 38. *Realsense D435i* kamera [8]

U zadacima iz području mobilne robotike kao što su kretanje te snalaženje u prostoru, s ciljem što kvalitetnijeg i preciznijeg izvršavanja zadatka, potrebno je voditi brigu o parametrima kao što su kvaliteta dobivene slike, brzina osvježavanja kadra (*engl. frame rate*) te kut vidnog polja kamere. U prilogu (IV) nalazi se tablica parametara korištene kamere.

4.1.1.5. APS 5065 Outrunner motor

Kao pogonski članovi mobilnog robota korištena su dva motora bez četkica kompanije *Alien Power System* koji pri naponu od 33.6 V mogu pojedinačno postići moment iznosa 8,5 Nm. Detaljnije karakteristike korištenog motora dane su u prilogu (V).

Slika 39. prikazuje korišteni *BLDC* (*engl. brushless direct current, hrv. istosmjerni motor bez četkica*) motor *APS* 5065.

Slika 39. *APS* 5065 *BLDC* motor[29]

4.1.1.6. CUI AMT10E2-V enkoder

Za određivanje pozicije mobilnog robota koriste se *CUI AMT10E2-V* inkrementalni enkoderi s gustoćom od 20480 *CPR* (*engl. cycles per revolution, hrv. tikova po okretaju*) što nam omogućava vrlo precizno određivanje pozicije mobilnog robota uz prethodno postavljanje na slobodni kraj osovine pogonskih motora. Osnovne mehaničke karakteristike enkodera dane su u prilogu (VI). Potpuni opis enkodera nalazi se u korisničkom vodiču [30].

Slika 40. prikazuje korišteni enkoder *CUI AMT10E2-V* te kabel koji služi za povezivanje istoga s *Odrive* uređajem.

Slika 40. prikaz korištenog enkodera CUI AMT10E2-V [30]

4.2. Pokretanje potrebnih čvorova

4.2.1. Launch datoteka za pokretanje čvorova mobilnog robota

S ciljem uspostavljanja komunikacije između sustava visoke i niske razine pokreće se *.launch* datoteka naziva *robot_startup.launch*. Nakon pokretanja skripte uspostavlja se veza između *Jetson Xavier NX* mikrokontrolera te *Teensy 4.0* mikroprocesora. Treba napomenuti da se u spomenutoj datoteci nalazi i naredba za pokretanje čvora *DS4* upravljača koji omogućava upravljanje robotom zadavanjem referenci brzina korištenjem *Sony DS4* upravljača.

```
<?xml version="1.0"?>
<launch>

    <node name="teensy_T4_node" pkg="rosserial_python"
type="serial_node.py">
    <param name="port" type="string" value="/dev/ttyACM0"/>
    <param name="baud" type="int" value="115200"/>
    </node>

    <!--<include file="/home/zic95/catkin_ws/src/ps4-ros/launch/ps4.launch"
/> -->

</launch>
```

Nakon pokretanja *.launch* datoteke za pokretanje čvora serijske komunikacije robota stvaraju se teme na koje se objavljuju podaci. Teme na koje će čvor robota objavljivati su:

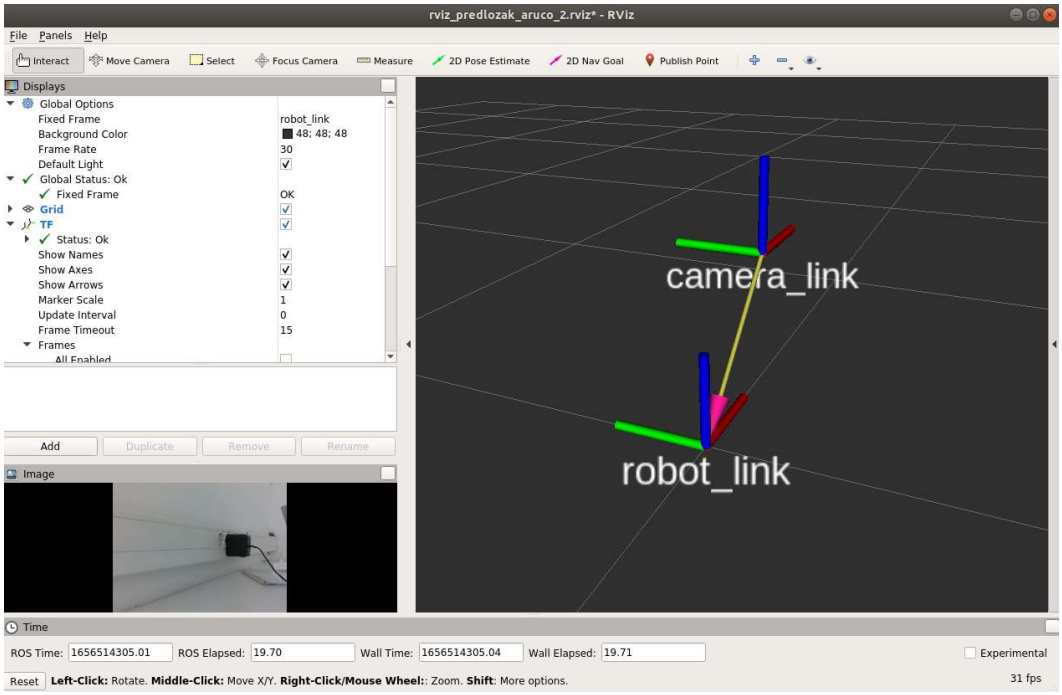
- */cmd_vel* – ova tema služi za razmjenu informacija o brzinama koje se moraju ostvariti na robotu. U ovom radu će na spomenutu temu podatke o referencama brzina objavljivati čvorovi regulatori brzina.
- */diagnostic* – daje informacije o procesima koji se izvršavaju prilikom serijske komunikacije te se koristi se prilikom otkrivanja pogrešaka u radu robota
- */odom_encoder* – tema koja se koristi za dobivanje povratne informacije o položaju mobilnog robota koji je određen iz podataka dobivenih očitavanjem enkodera montiranih na osovine motora
- */odrive_voltage* – daje povratnu informaciju o napunjenosti *LiPo 5S* baterije koja se koristi za pogon *BLDC* motora koji pokreću robota

4.2.2. čvor adding_frame_robot_link

Radi kasnije simulacije kretnji potrebno je izraditi koordinatni sustav robota. Iz opće prakse, radi univerzalnosti i poštivanja označavanja u robotici, nadjenut će mu se ime *robot_link*. Stvaranje koordinatnog sustava izvršava se pokretanjem programskog koda pisanog u *Python-u* naziva *add_robot_link.py* [32.] a prikazuje stvaranje, rotaciju i translaciju koordinatnog sustava *robot_link* u odnosu na koordinatni sustav kamere.

4.2.3. čvor RViz

Radi potrebe za vizualizacijom vitalnih dijelova sustava pokreće se čvor *RViz* čije sučelje je prethodno uređeno i spremljeno kao predložak. Slika 41. prikazuje *RViz* sučelje u kojem je referentni koordinatni sustav kamere naziva *camera_link* zamaknut u odnosu na bazni koordinatni sustav robota naziva *robot_link* prema modelu stvarnog robota. Kao nepomični koordinatni sustav u grafičkom sučelju odabire se referentni koordinatni sustav robota jer se prema njemu određuju sve naknadne transformacije potrebne za definiranje pozicije mobilnog robota u odnosu na detektirani marker.



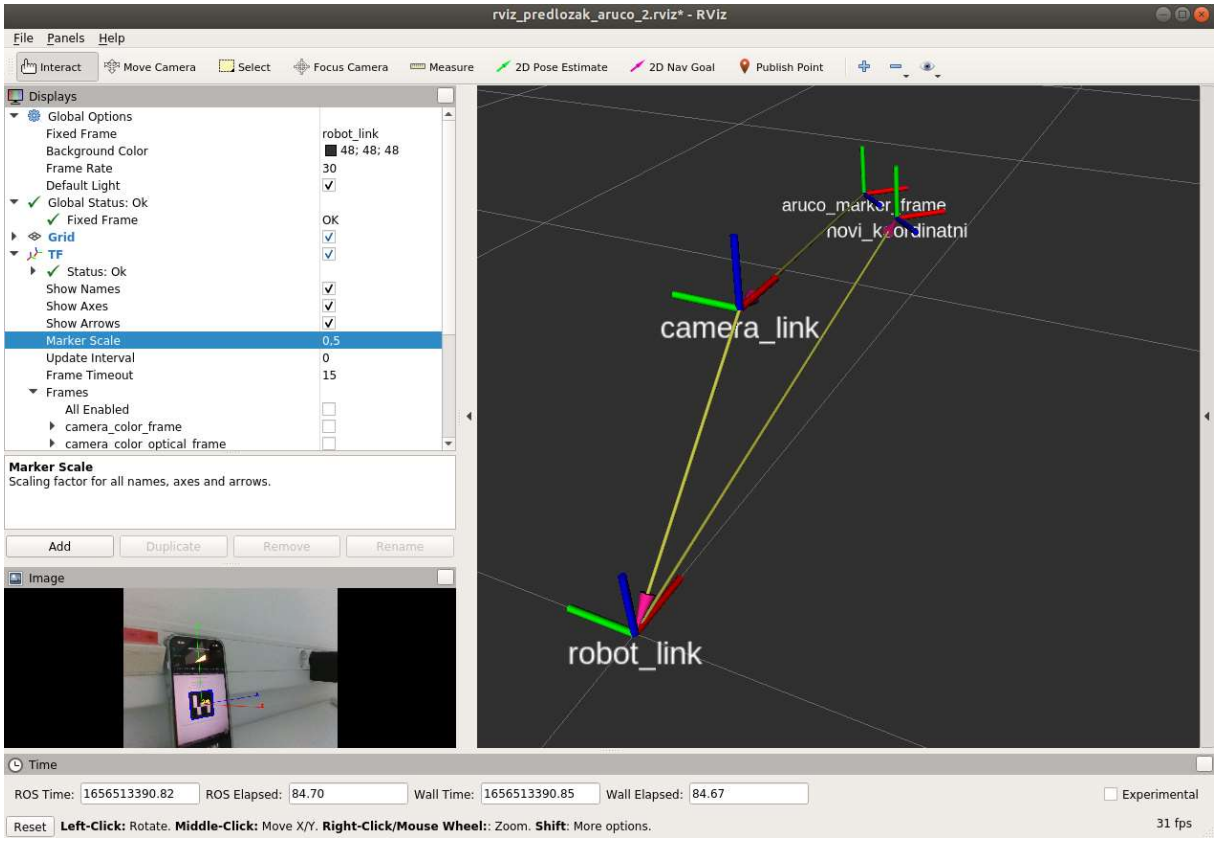
Slika 41. Prikaz predloška *RViz* sučelja sustava mobilnog robota

4.2.4. čvor static_tf_broadcaster

Sljedeći čvor koji je potrebno pokrenuti zadužen je za stvaranje statičke transformacije između koordinatnog sustava robota *robot_link* i koordinatnog sustava detektiranog markera *aruco_marker_frame*. Nakon što se u vidnom polju robota pojavi marker, čvor detekcije istom tom markeru određuje poziciju u odnosu na referentni koordinatni sustav mobilnog robota *robot_link* te se ista odmiče okomito na marker za 400mm, stvarajući tako točku prilaska mobilnog robota u zadacima pozicioniranja. Čvor je pisan u programskom jeziku *Python*, koji se aktivira pokretanjem skripte *static_trans_marker.py* [32.]

Nakon pokretanja čvora izvršena je statička transformacija koordinatnog sustava markera u odnosu na koordinatni sustav baze. Novo nastali koordinatni sustav nosi naziv *novi_koordinatni*, a poslužit će kao nova točka prilaska mobilnom robotu u nastavku ovog rada.

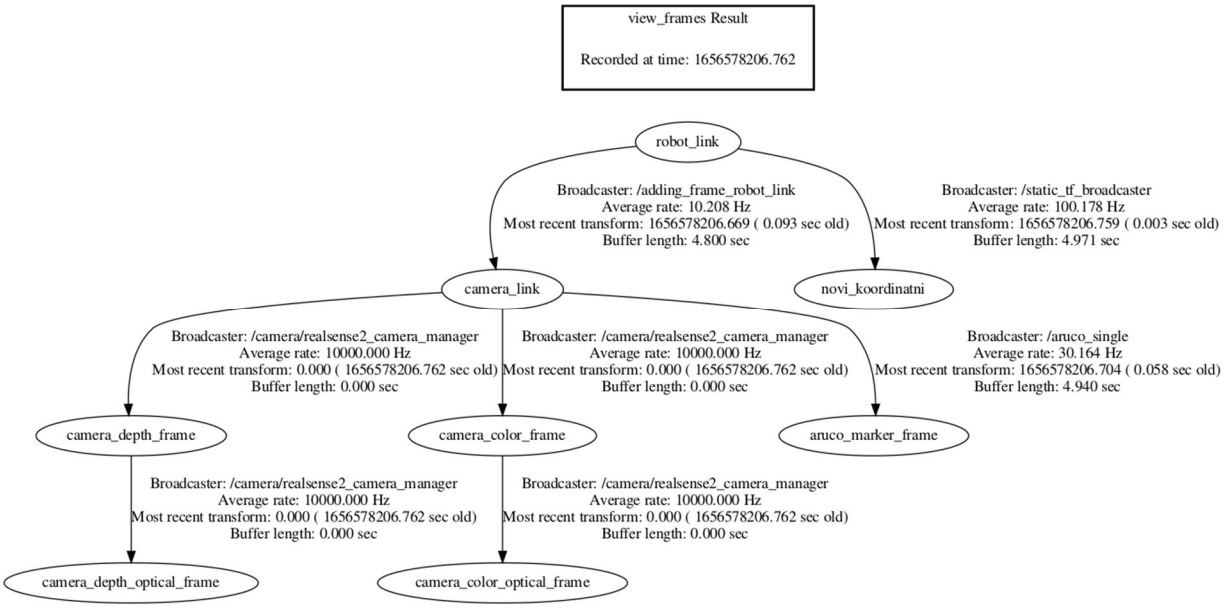
Slika 42. prikazuje scenu iz *RViz* grafičkog sučelja nakon pokretanja svih dosad navedenih čvorova za dodavanje svih dosad spomenutih koordinatnih sustava kao i grafički prikaz transformacija između njih.



Slika 42. Prikaz *RViz* sučelja nakon pokretanja čvora *static_transformation.py*

4.2.5. čvor view_frames

Radi dobivanja što boljeg dojma o međusobnim vezama koordinatnih sustava pokreće se čvor naziva *view_frames* koji je integriran unutar paketa *tf*. Pokrenuti čvor provodi slušanje svih aktivnih koordinatnih sustava mobilnog robota u trajanju od 5 sekundi te nakon tog vremena prikazuje rezultate u formi grafa uz pružanje informacije o transformacije o nazivu odašiljača svake pojedine transformacije (*engl. broadcaster*). U grafu je prikazana nekolicina koordinatnih sustava koji nisu direktno upotrijebljeni za rješavanje dobivenog zadatka (*camera_depth_frame*, *camera_color_frame*, *camera_depth_optical_frame*, *camera_color_optical_frame*), ali su stvoreni pri pokretanju čvora kamere te se mogu iskoristiti prilikom nadogradnji na postojećem sustavu. Slika 43. prikazuje sve aktivne koordinatne sustave mobilnog robota.

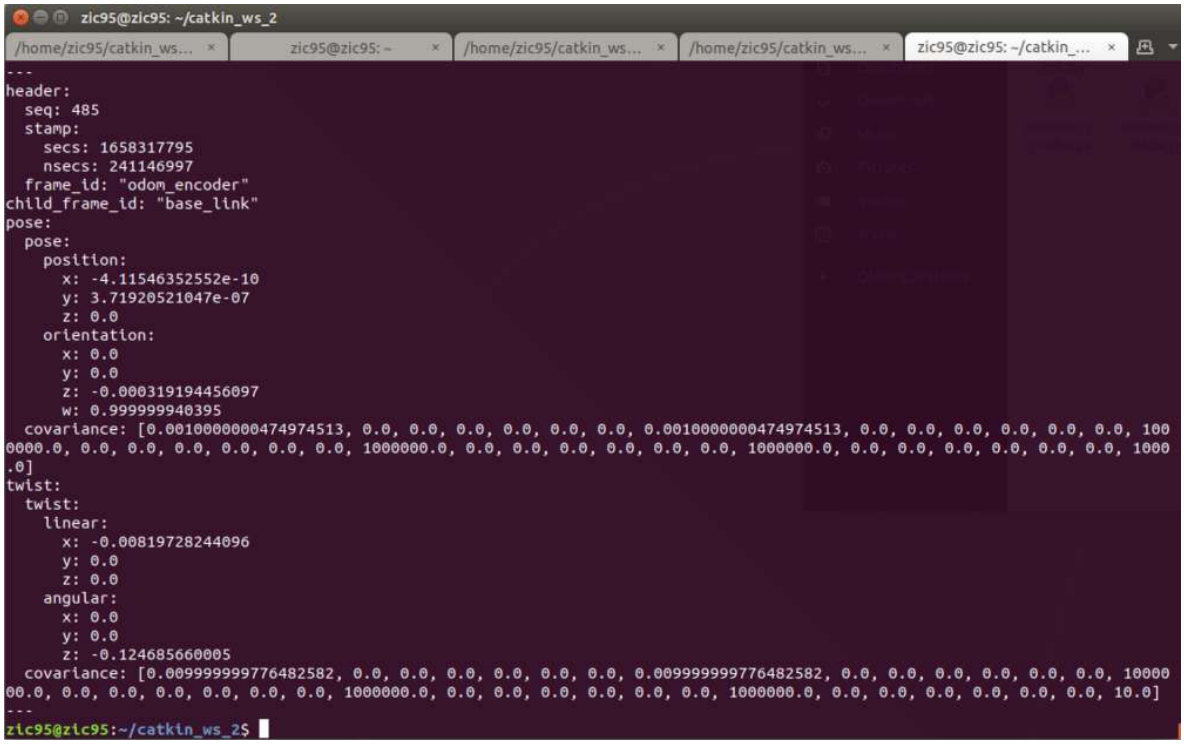


Slika 43. Prikaz koordinatnih sustava dobivenih čvorom *view_frames*

4.2.6. čvor odometry mobilnog robota

Odometrija mobilnog robota dobiva se putem dva enkodera montirana na osovine kotača. Tema na koju se objavljuje sastoji se od dva tipa poruka. Prvi tip poruke je pozicija koordinatnog sustava baze robota naziva *base_link* u odnosu na koordinatni sustav enkodera naziva *odom_encoder*. Kutovi koji se objavljuju na temu pozicije zapisani su u *quaternion* obliku te se, radi lakšeg podešavanja i intuitivnijeg rada, prilikom pisanja koda regulatora pretvaraju u Eulerove kutove. Na temu *Twist* objavljuju se brzine kretanja izmjerene pomoću enkodera koje mogu služiti kao povratna informacija regulatoru brzine o postignutoj brzini. U idealnom svijetu sa idealnim regulatorima i bez gubitaka energija i informacija poslana brzina bila bi jednaka ostvarenoj dok u stvarnom to nije tako radi ograničene rezolucije enkodera, proklizavanja i varijacija u iznosima promjera kotača prilikom kretanja robota stoga tema *Twist* može uvelike pomoći prilikom izrade i podešavanja preciznih regulatora brzine.

Slika 44. prikazuje objavljene vrijednosti na temu odometrije nakon poziva *ROS* servisa za resetiranje odometrije. Vidimo da te vrijednosti nisu egzaktno na nuli, sve zbog različitih šumova te realnih mehaničkih i elektroničkih komponenti.



Slika 44. Prikaz teme odometrije mobilnog robota naziva *odom_encoder*

4.3. Čvorovi za upravljanje referencama brzina

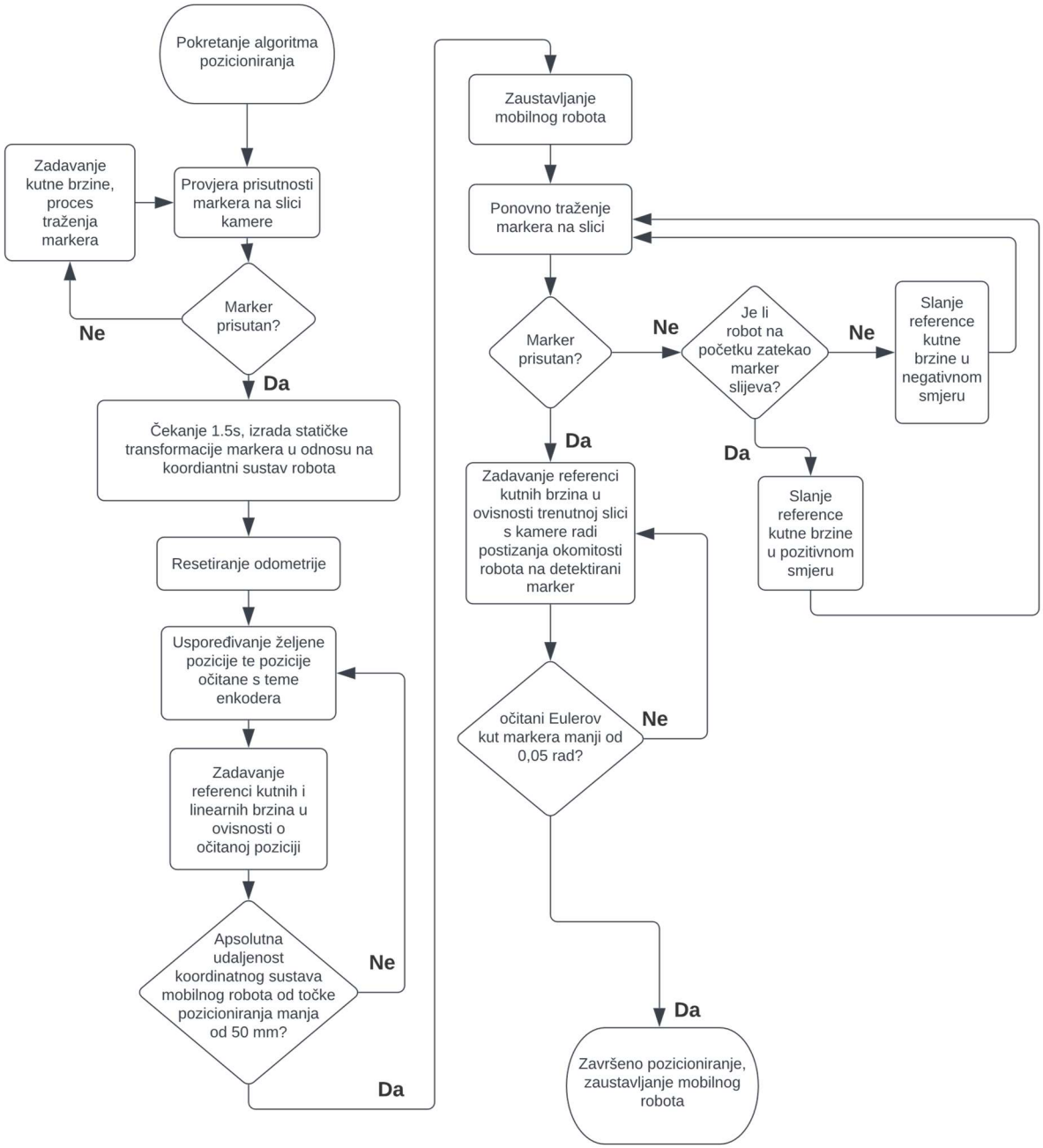
Praćenje objekata u ovom radu izvedeno je u dvije varijante: pozicioniranje i čisto praćenje (*visual servoing*). Pozicioniranje se sastoji od traženja markera u okruženju mobilnog robota, stvaranje statičke transformacije po nailasku na marker, odlazak u točku određenu statičkom transformacijom (postizanje položaja) te zakretanje mobilnog robota s ciljem postizanja paralelnosti robota i detektiranog markera (postizanje orijentacije). Takva metoda upravljanja koristi se u zadacima kada robota želimo postaviti u prostoru u točku određenu pozicijom markera, primjerice postavljanje mobilnog robota na mjesto ispod palete s ciljem zahvaćanja iste ili postavljanje mobilnog robota u neku točku u prostoru gdje se nalazi punjač na kojega se robot spaja. Zadatak praćenja se također sastoji od traženja markera u robotskom okruženju, ali umjesto izrade statičke transformacije čvor čistog praćenja konstantno provjerava položaj markera u odnosu na robota te u ovisnosti o njihovom međusobnom položaju zadaje reference brzina na temu */cmd_vel*.

4.3.1. čvor speed_controller_positioning

Nakon gore pokrenutih čvorova dobivamo potrebnu povezanost s temama koje nam omogućuju izvršavanje zadatka planiranja kutne i linearne brzine mobilnog robota prema detektiranom cilju. Izvor [32.] sadrži kod čvora upravljača brzine mobilnog robota u zadacima pozicioniranja. Čvor se pretplaćuje na temu odometrije mobilnog robota naziva /*odometry* iz koje dobiva poziciju robota očitavanjem inkrementalnih enkodera kontinuirano provođenje transformacije između koordinatnog sustava robota te koordinatnog sustava načinjenog translacijom od stvarnog koordinatnog sustava *ArUco* markera. Nakon provedenih izračuna, čvor regulatora brzine objavljuje potrebnu brzinu robota na temu /*cmd_vel* na koju se pretplaćuje čvor zadužen za kretanje robota te prema kojoj se robot na posljetku i giba.

Pošto se kretanje robota izvodi u ravnini poda isto je podijeljeno na četiri glavna dijela. Prvi dio gibanja odnosi se na rotiranje mobilne platforme ukoliko na slici kamere nije pronađen marker. U drugom dijelu gibanja podešava se kut zakreta mobilnog robota tako da se postigne usmjerenost robota prema markeru te da se omogući treći dio gibanja koji se odnosi na linearno gibanje mobilnog robota prema markeru. Posljednji dio gibana odnosi se na podešavanje orijentacije mobilnog robota tako da se postigne okomitost s detektiranim markerom. Za regulaciju kuta zakreta korišten je regulator koji prati graf funkcije *tangensa hiperbolnog m*noženog s pojačanjem dok je za pravocrtno gibanje korišten isključivo *P* regulator množen s

faktorom udaljenosti. Orijentacija robota u odnosu na marker određena je pomoću dinamičke transformacije koja se dobiva iz trenutne slike kamere radi izbjegavanja pogreške koja se akumulirala prilikom očitavanja enkodera te eventualnih proklizavanja kotača. Koeficijenti P regulatora iz priloga koji su namijenjeni za pravocrtno i kutno gibanje podešeni su u fazi testiranja algoritama na vrijednosti kojima je postignuto najpovoljnije ponašanje robota. Slika 45. prikazuje dijagram toka u zadatku pozicioniranja mobilnog robota.



Slika 45. Dijagram toka algoritma pozicioniranja

4.3.2. Launch datoteka za jednostavnije pokretanje čvorova za pozicioniranje

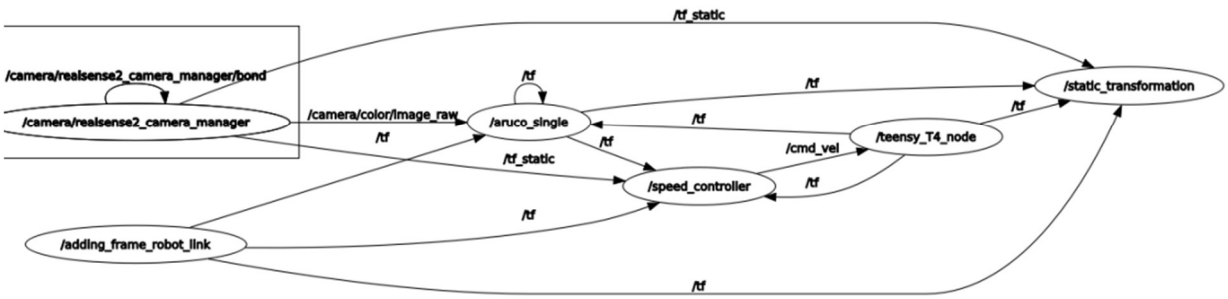
S ciljem jednostavnijeg pokretanja čvorova zaduženih za zadatak pozicioniranja, napisan je kod *marker_positioning.launch* [32.]

Pokretanjem *.launch* datoteke pokreću se čvorovi za transformaciju koordinatnog sustava *robot_link*, grafičko sučelje za prikaz okruženja robota, čvor koji radi statičku transformaciju koordinatnog sustava markera te čvor regulatora brzine. Zbog potrebe za rotacijom mobilnog robota prilikom traženja markera naredba za resetiranje odometrije nije moguće smjestiti u *launch* datoteku već je istu potrebno resetirati prilikom ostvarivanja statičke transformacije pomoću poziva *ROS* servisa naziva */resetEncoderOdom*.

Kod *static_trans_marker.py* [32.] prikazuje modificirani čvor statičke transformacije koji nakon izvršavanja poziva resetiranje teme odometrije kako bi se omogućilo praćenje trenutne pozicije mobilnog robota.

4.3.3. Pregled čvorova u zadatku pozicioniranja

U zadacima pozicioniranja prema referencama dobivenih analizom slike s kamere i detekcije markera na istoj, uz prethodno pokretanje svih čvorova koji su zaslužni za ispravan rad cijelog sustava robota, dobiva se mreža pretplatnika/pošiljatelja uz definirane smjerove komunikacija i teme preko kojih se razmjenjuju podatci. Slika 46. prikazuje odnos između pretplatnika i pošiljatelja koji je potrebno ostvariti i među kojima je potrebno omogućiti komunikaciju za izvršavanje zadataka pozicioniranja u odnosu na postavljeni *ArUco* marker.



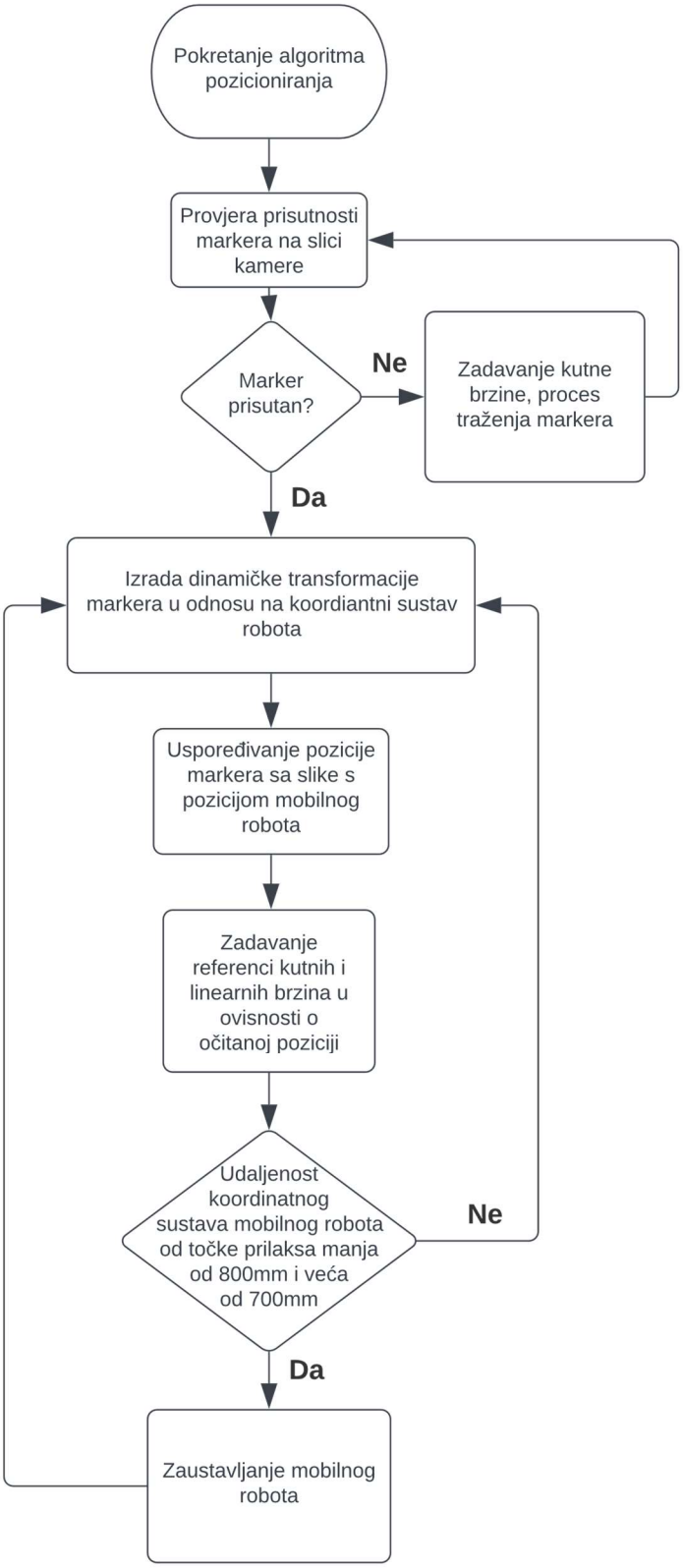
Slika 46. Prikaz pretplatnika/pošiljatelja u zadacima pozicioniranja

4.3.4. čvor speed_controller_following

Nakon pokretanja gore navedenih čvorova postiže se potrebnu povezanost s temama koje nam omogućuju izvršavanje zadatka planiranja kutne i linearne brzine mobilnog robota prema objektu u pokretu. Izvor [32.] sadrži kod čvora upravljača brzine mobilnog robota u zadacima slijeđenja objekata na koje je pričvršćen marker. Čvor se ne pretplaćuje na temu odometrije mobilnog robota već konstantno radi transformaciju između koordinatnog sustava robota i koordinatnog sustava detektiranog *ArUco* markera. Nakon provedenih izračuna u ovisnosti o iznosu spomenute transformacije čvor regulatora brzine objavljuje potrebnu brzinu robota na temu */cmd_vel* na koju je čvor zadužen za kretanje robota pretplaćen te prema kojoj se giba.

Kretanje mobilnog robota u zadacima čistog slijeđenja podijeljeno je na dva glavna dijela. Prvi dio kretanja odnosi se na rotiranje mobilnog robota oko svoje osi ukoliko ne dolazi do detekcije markera na slici kamere. Drugi dio kretanja odnosi se na kombinaciju translacije i rotacije kako bi se stvorila što prirodnija putanja robota do detektiranog markera.

Za regulaciju kuta zakreta također korišten je regulator koji prati graf funkcije *tangensa hiperbolnog* množenog s pojačanjem dok je za pravocrtno gibanje korišten isključivo *P* regulator množen s faktorom udaljenosti. Koeficijenti P regulatora iz priloga koji su namijenjeni za pravocrtno i kutno gibanje podešeni su u fazi testiranja algoritama. Slika 47. prikazuje dijagram toka u zadacima slijeđenja *ArUco* markera.



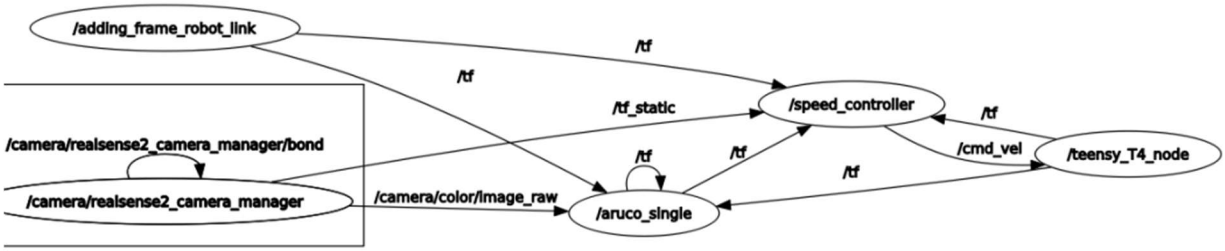
Slika 47. Dijagram toka algoritma slijeđenja markera

4.3.5. Launch datoteka za jednostavnije pokretanje čvorova za praćenje

S ciljem jednostavnijeg pokretanja čvorova zaduženih za zadatak praćenja napisan je kod *follow_marker.launch* [32.]. Izvršavanjem *.launch* datoteke pokreću se čvorovi za transformaciju koordinatnog sustava *robot_link*, grafičko sučelje za prikaz okruženja robota te čvor regulatora brzine.

4.3.6. Pregled čvorova u zadatku slijeđenja

U zadacima slijeđenja prema referencama dobivenih kontinuiranim praćenjem i analiziranjem slike dobivene kamerom, uz prethodno pokretanje svih čvorova koji su zaslužni za ispravan rad cijelog sustava robota, dobiva se mreža pretplatnika/pošiljatelja uz definirane smjerove komunikacija i teme preko kojih se razmjenjuju podatci. Slika 48. prikazuje odnos između pretplatnika i pošiljatelja koji je potrebno ostvariti i među kojima je potrebno omogućiti komunikaciju za izvršavanje zadataka slijeđenja *ArUco* markera.

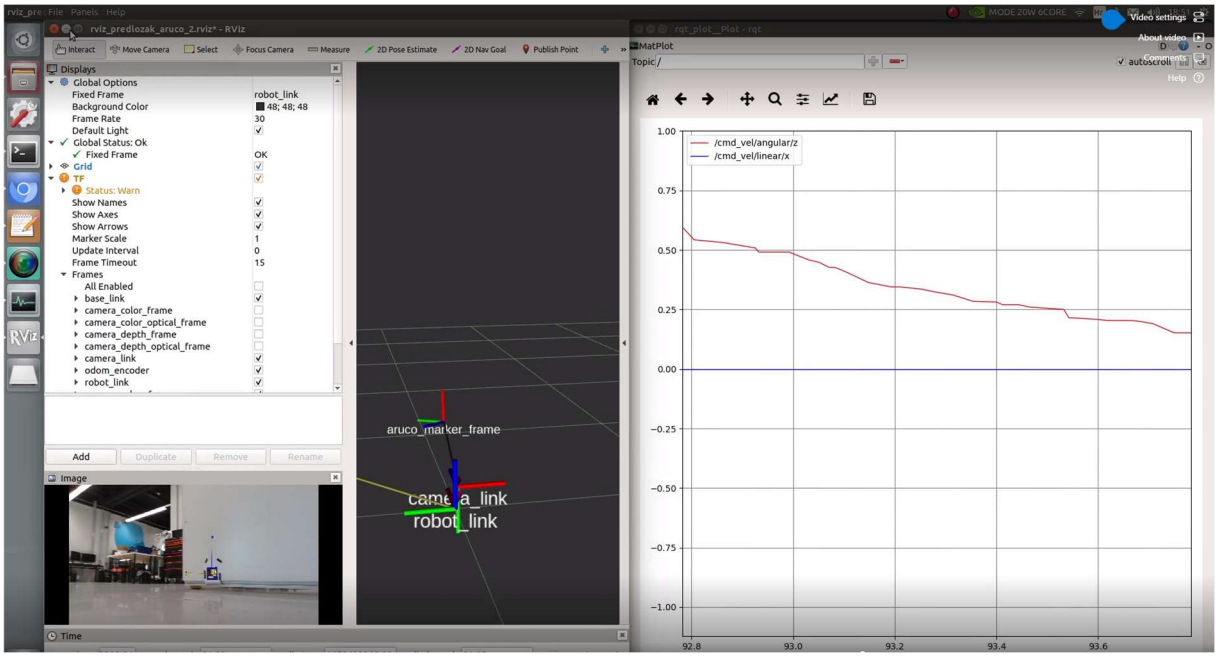


Slika 48. Prikaz pretplatnika/pošiljatelja u zadacima slijeđenja

4.4. Vizualizacija referenci brzine teme *cmd_vel*

S ciljem dobivanja predodžbe o brzinama koje čvor *speed_controller* objavljuje na temu */cmd_vel* iste je potrebno grafički prikazati. Za grafičko prikazivanje u *ROS* programskom okruženju najčešće se koristi gotov čvor *rqt_plot* koji se nalazi u osnovnoj verziji korištene inačice *ROS*-a.

Pošto je algoritam regulatora brzine pisan za diferencijalnog robota namijenjen za kretanje u ravnini poda te kao parametre brzine imamo samo translaciju uzduž *x* te rotaciju oko *z* osi dok su svi ostali parametri koje se objavljuju na temu */cmd_vel* jednaki nuli. Slika 49. prikazuje *RViz* sučelje prilikom detekcije markera (lijevi dio slike) i dijagram linearnih i kutnih brzina koje se objavljuju na temu */cmd_vel* (desni dio). Brzine na slici prikazuju reference koje regulator brzine šalje robotu da bi postigao željeni prostorni položaj i orijentaciju u odnosu na detektirani marker (referenca linearne brzine prikazana je plavom dok je referenca kutne brzine prikazana crvenom bojom).



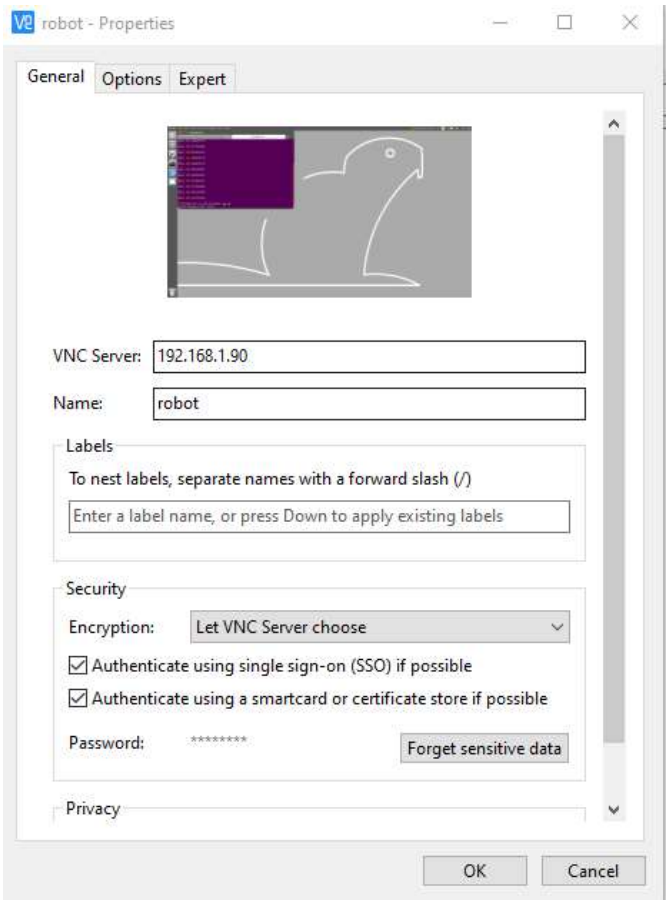
Slika 49. Prikaz reference brzine objavljene na temu */cmd_vel* pomoću *rqt_plot* čvora

4.5. Spajanje na *ROS* sustav montiran na mobilnom robotu

Zbog pojma mobilnosti robota, s ciljem lakšeg programiranja i testiranja rada robota potrebno je na neki način ostvariti vezu između upravljačkog računala mobilnog robota koje je

u pokretu i stacionarnog računala. Veza se može ostvariti na dva načina. Prvi način je udruženi rad *ROS* sustava na više računala spojenih preko iste mreže od kojih jedan preuzima ulogu *Mastera* te upravlja komunikacijom između svih čvorova dok sva ostala računala služe isključivo za pokretanje i izvršavanje čvorova. [28].

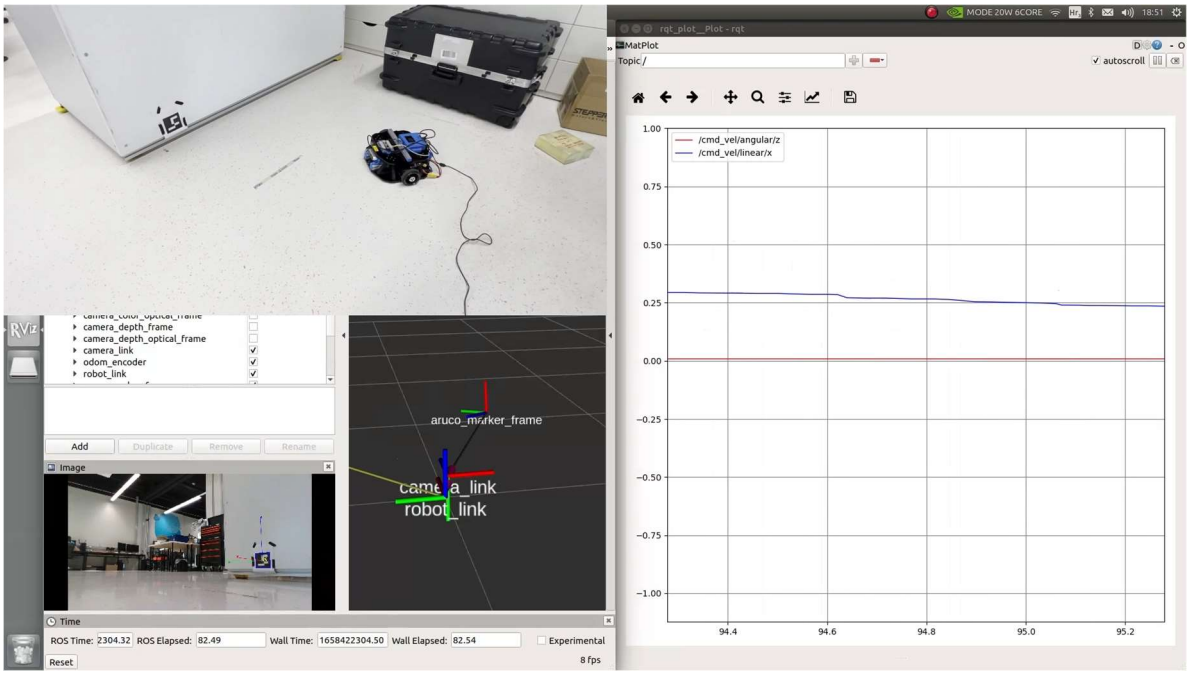
Drugi način, tj. onaj koji je korišten u ovom radu je upotreba programa za daljinsko upravljanje računalima spojenih na istu mrežu. Jedan od tih programa je i *VNC viewer* koji se postavlja na brz i jednostavan način te osigurava relativno dobru vezu računala domaćina s upravljačkim računalom robota. Slika 50. prikazuje mrežne parametre mobilnog robota koje je potrebno podesiti kako bi se ostvarila veza između računala domaćina i računala mobilnog robota.



Slika 50. Prikaz mrežnih parametara korištenog mobilnog robota

4.6. Testiranje razvijenih algoritama u stvarnom okruženju

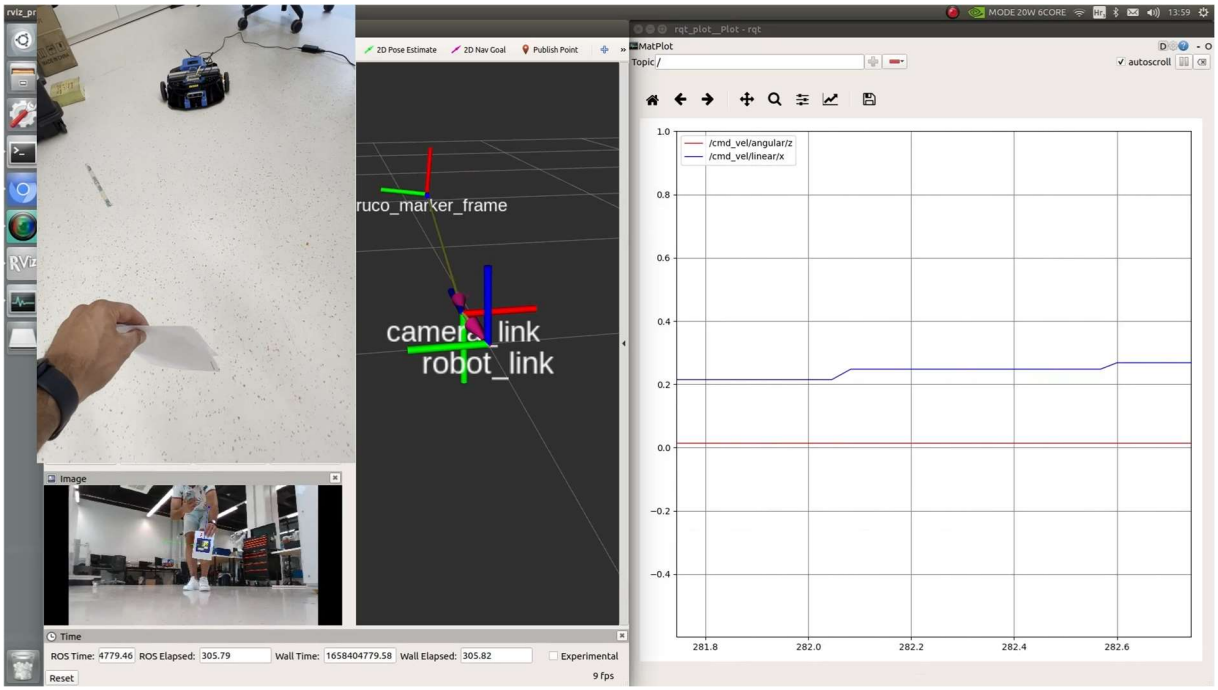
U prethodnim poglavljima prikazani su svi čvorovi koje je potrebno pokrenuti na mobilnom robotu kako bi se uspješno ostvarila interakcija programskog i izvršnog dijela. U nastavku rada nalazi se osvrt na izvršavanje algoritama praćenja i pozicioniranja proveden u realnom okruženju na spomenutoj opremi. Na slici 51. nalazi se prikaz okoline u kojoj se izvršava algoritam pozicioniranja, slika dobivena kamerom prema kojoj se provode algoritmi detekcije markera, pregled koordinatnih sustava te prikaz referenci linearnih i kutnih brzina koje se šalju na pogonske motore.



Slika 51. Provođenje algoritama pozicioniranja na mobilnom robotu

Nakon provedenog testiranja algoritma, dolazi se do zaključka da je algoritam sposoban odrediti stvarnu poziciju markera te robota pozicionirati u željenu točku u prostoru ukoliko je prostorija dobro osvjetljena te marker vidljiv na slici. Nakon provedenog većeg broja testiranja programa utvrđeno je da robot postiže prostornu poziciju s točnošću od +/- 5 cm, što je za mobilnu robotiku zadovoljavajuće, a posljedica je postizive oštrine markera na slici u koja ovisi o međusobnoj udaljenosti markera i robota, proklizavanja kotača te pogrešaka u očitavanju odometrije mobilnog robota. Video rada robota u zadatku pozicioniranja na lokaciju u prostoru određenu postavljenim *ArUco* markerom [33.].

Na slici 52. prikazana je okolina u kojoj se izvršava algoritam praćenja markera, slika dobivena s kamere prema kojoj se provode algoritmi praćenja, pregled koordinatnih sustava te prikaz referenci linearnih i kutnih brzina koje se šalju na pogonske motore.

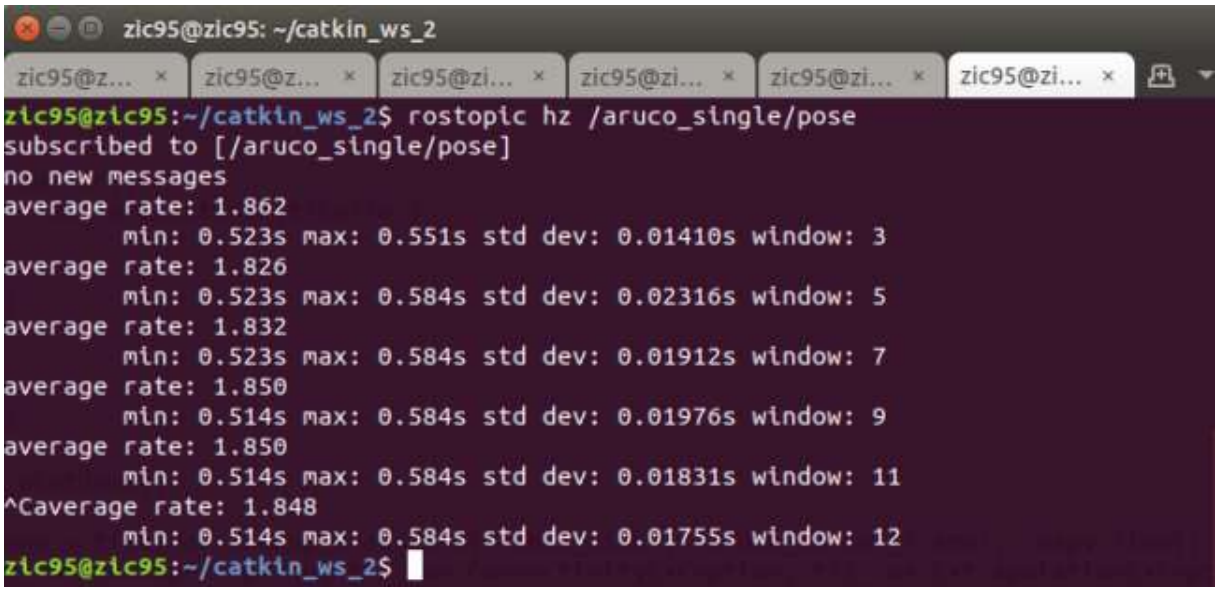


Slika 52. Provođenje algoritama praćenja na mobilnom robotu

Prilikom testiranja rada algoritma praćenja markera, utvrđeno je da algoritam izvršava svoju zadaću ukoliko je prostorija dovoljno osvjetljena kako bi marker uslijed provođenja algoritma bio potpuno vidljiv na slici koja se dobavlja putem kamere montirane na robotu. Video rada robota u zadatku slijeđenja objekta koji je označen *ArUco* markerom [34.]

Algoritmi detekcije i praćenja markera zahtjevni su za provođenje jer u svakoj novoj sceni dobivenoj sa slike kamere traže ključne točke markera kako bi se markeru odredio prostorni položaj (poziciju i orijentaciju) u odnosu na kameru pomoću koje se vrši akvizicija slike. Zbog ograničenja *hardware-a* te istovremeno pokrenute sve čvorove, algoritmima za detekciju i praćenje markera značajno padaju performanse. Prilikom testiranja čvorova vezanih uz detekciju i praćenje markera koje je provedeno na virtualnoj mašini na računalu s *Intel* i5 procesorom osme generacije te 16 GB *RAM* memorije algoritam je bez problema davao rezultate o detektiranom markeru s prosječnom frekvencijom od 15 *Hz*. Nakon prebacivanja i pokretanja svih čvorova na upravljačko računalo mobilnog robota koji su istome potrebni za provođenje gore opisanih radnji detektiranja i praćenja objekata, upravljačko računalo je u

jedinici vremena bilo primorano izvršavati i pogoniti više čvorova te se brzina osvježavanja pozicije markera značajno smanjila, na frekvenciju približno jednaku 2 *Hz* (Slika 53.). Tako mala frekvencija osvježavanja pozicije pokazala se kao problem u zadacima čistog slijeđenja markera pri povećanju brzine kretanja robota ili bržeg pomicanja markera, sve zbog nemogućnosti praćenja transformacija dovoljno velikom brzinom da bi regulator brzine mogao uspješno i pravilno reagirati, dok ta mala brzina osvježavanja u zadacima pozicioniranja ne predstavlja značajniji problem jer robot ne provodi kontinuiranu provjeru slike kamere već se lokalizira u prostoru putem odometrije. Iako nije postignuta velika brzina osvježavanja, algoritam detekcije markera pokazao se dovoljno pouzdan za upotrebu u zadacima praćenja implementiranih na mobilnog robota za razliku od ostalih gore spomenutih algoritama koji su postigli značajno manje brzine osvježavanja u jedinici sekunde.



Slika 53. Prikaz brzine osvježavanja pozicije detektiranog *ArUco* markera

5. ZAKLJUČAK

Osnovna ideja ovog rada bila je upoznati se s najpoznatijim sustavom za programiranje robota današnjice, *ROS-om* te sagledati širu sliku i glavnu primjenu algoritama računalnog vida. *ROS* sadrži vrlo složenu arhitekturu kao i velik broj pravila što od apsolutnog početnika zahtjeva određeno vrijeme za adaptaciju, ali nakon duljeg rada s sustavom stvara se šira slika o metodama rada sa sustavom uz dobivanje uvida u brojne mogućnosti istog. Arhitektura sustava *ROS* uz brojna pravila omogućava podjelu poslova među timovima različitih područja rada te tako projekt može biti podijeljen na više glavnih zadataka koji se, po završetku, mogu jednostavno združiti u funkcionalnu cjelinu. Najprije je opisana osnovna podjela algoritama računalnog vida fokusiranih na detekciju i praćenje značajki i objekata. Potom se, zbog najveće pouzdanosti, preciznosti i robusnosti u radu, uvodi pojam markera te se u nastavku detaljno objašnjavaju uz prikaz implementacije detekcije i praćenja na *ROS-u*. Posljednji dio rada prikazuje podešavanje sustava mobilnog robota za zadatak detekcije i praćenja objekata kako bi mu se, uz upotrebu markera, omogućilo održavanje potrebnog prostornog položaja u odnosu na objekt koji prati (pozicije i orijentacije). Zbog manje pouzdanosti ostalih metoda detekcije i praćenja u zadacima slijeđenja objekata na iste je potrebno pričvrstiti marker, što ispada jednostavno, iskoristivo i robusno rješenje.

Nakon uređenja upravljačkog računala dobiva se funkcionalni sustav upravljanja visoke razine (*engl. high level control*) koji je uz primjenu spomenutih čvorova zadužen za praćenje okoline i donošenje odluke o smjeru i brzini kojom robot nastavlja svoju kretnju. Također je u sklop sustava upravljanja visoke razine implementiran regulator brzine. Pošto zadatak ovog rada nije izrada globalnog planera trajektorija u sustavu nisu korišteni lokalni planeri već je izrađeni *P* regulator brzine koji šalje reference brzine sustavu upravljanja niske razine (*engl. low level control*). Glavna zadaća tog sustava, zbog mogućnosti postizanja višestruko većih frekvencija naspram sustava upravljanja visoke razine, je aktivno provođenje upravljanja motorima te očitavanje podataka sa senzora (u ovom slučaju enkodera) koji u grani povratne veze robotu daju povratnu informaciju prema kojima sustav donosi odluke.

Cilj ovog završnog rada bio je stvaranje šire slike o brojnim mogućnostima rada s kompleksnim sustavima kao što je mobilni robot, proučavanje koncepata *ROS* sustava, suočavanje s brojnim problemima i nedostacima koje takvi sustavi donose te uspješno otklanjanje istih. Cijeli proces izrade završnog rada stvorio je širu sliku o robotskim sustavima te je doprinio širenju sveopćeg inženjerskog znanja.

LITERATURA

[1] https://www.geeksforgeeks.org/, 20.01.2022.

[2] https://www.pyimagesearch.com/, 20.01.2022.

[3] https://www.turtlebot.com/, 20.07.2022.

[4] http://wiki.ros.org/ROS/Concepts, 22.01.2022.

[4.1] http://wiki.ros.org/Master , 22.01.2022.

[5] Braninir Ćaran: Diplomski rad, Zagreb 2021

[5.1] https://www.cs.columbia.edu/~allen/F19/NOTES/icckinematics.pdf, 25.01.2022.

[6] https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit , 20.07.2022.

[7] https://www.pjrc.com/ , 24.01.2022.

[8] https://www.intelrealsense.com/, 24.01.2022.

[9] Jason M. O'Kane: A Gentle Introduction to ROS, 2013.

[10] https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html, 29.01.2022.

[11] https://www.analyticsvidhya.com/blog/2021/06/feature-detection-description-and-matching-of-images-using-opencv/, 01.02.2022.

[12] https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html, 01.02.2022.

[13] https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html, 02.02.2022.

[14] https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html, 02.02.2022.

[15] https://www.analyticsvidhya.com/blog/2021/06/feature-detection-description-and-matching-of-images-using-opencv/, 03.02.2022.

[16] https://learnopencv.com/histogram-of-oriented-gradients/, 04.02.2022.

[17] https://pyimagesearch.com/2018/07/30/opencv-object-tracking/, 06.02.2022.

[18] https://broutonlab.com/blog/opencv-object-tracking, 06.02.2022.

[19] https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html, 10.02.2022.

[20] https://github.com/pal-robotics/ddynamic_reconfigure, 12.02.2022.

[21] https://github.com/IntelRealSense/realsense-ros.git, 15.02.2022.

[22] https://github.com/pal-robotics/aruco_ros, 15.02.2022.

[23] https://github.com/introlab/find-object.git, 16.02.2022.

[24] https://github.com/introlab/find-object.git, 20.02.2022.

[25] European Conference on Computer Vision -EECV 2012

[26] https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/, 22.02.2022.

[27] https://razbotics.wordpress.com/2018/01/23/ros-distributed-systems/, 24.02.2022.

[28] https://chev.me/arucogen/25.05.2022.

[29] https://alienpowersystem.com/shop/brushless-motors/50mm/aps-5065-outrunner-brushless-motor-60kv-1800w/#tab-additional_information, 30.06.2022.

[30] https://odriverobotics.com/shop/20480-cpr-encoder-with-odrive-cable, 30.06.2022.

[31] https://odriverobotics.com/shop/odrive-v36 30.06.2022.

[32] Programski kodovi za čvorove i *.launch* datoteke- https://github.com/fmiklauzic/ArUco_marker_tracker_using_RealSense-435i

[33] Video rada robota u zadatku pozicioniranja - https://youtu.be/AuXp9xBGEu0

[34] Video rada robota u zadatku slijeđenja markera - https://youtu.be/91eNYTEUGuE

PRILOZI

I. Tablica karakteristika mobilnog robota

Broj pogonskih kotača	2
Vrsta pogonskih kotača	Ispunjena guma
Promjer pogonskih kotača [mm]	95
Širina pogonskih kotača [mm]	24
Broj slobodno rotirajućih kuglica	2
Vrsta slobodno rotirajućih kotača	Aluminijska kuglica
Promjer slobodno rotirajućih kotača [mm]	19
Prijenosni omjer	1:1
Broj enkodera	2
Međusobni razmak kotača [mm]	36,5

II. Karakteristike Jetson Xavier NX mikroračunala [6]

GPU	NVIDIA Volta architecture with 384 NVIDIA CUDA® cores and 48 Tensor cores
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6 MB L2 + 4 MB L3
DL Accelerator	2x NVDLA Engines
Vision Accelerator	7-Way VLIW Vision Processor
Memory	8 GB 128-bit LPDDR4x @ 51.2GB/s
Storage	microSD (not included)
Video Encode	2x 4K @ 30 | 6x 1080p @ 60 | 14x 1080p @ 30 (H.265/H.264)
Video Decode	2x 4K @ 60 | 4x 4K @ 30 | 12x 1080p @ 60 | 32x 1080p @ 30 (H.265) 2x 4K @ 30 | 6x 1080p @ 60 | 16x 1080p @ 30 (H.264)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E (WiFi/BT included), M.2 Key M (NVMe)
Display	HDMI and display port
USB	4x USB 3.1, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	103 mm x 90.5 mm x 34.66 mm

III. Karakteristike *Teensy 4.0* mikroupravljača [7]

CPU	ARM Cortex-M7 at 600 MHz
Radna memorija	1024KB
Pohrana	2048KB
Komunikacija	2 USB priključka, oba 480 Mb/s
Napajanje	5V
Ukupni broj digitalnih U/I priključaka	40
Broj PWM priključaka	31

IV. Karakteristike *Intel RealSense D435i* kamere [8]

Primjena	Unutanja/vanjska
Tehnologija snimanja dubine	Aktivni IR Stereo
Tehnologija senzora slike	Globalni okidač, veličina *pixela* 3um x 3um
Polje dubinske slike (FOV)-(Širina x Visina)	87° x 58° (+/- 3°)
Izlazna rezolucija i frame rate	Do 1280 x 720 i 90 *fps*
Minimalna mjerena udaljenost	0,105m
Maksimalna mjerena udaljenost	10m+, ovisna o uvjetima scene i osvjetljenju
Preciznost dubinske slike	<2% pri udaljenosti od 2 m
RGB rezolucija	Do 1920 x 1080 – 2 MP
RGB FOV (Š x V x D)	69 x 42 x 77 (+/- 3°)
Format	10-bit RAW
Procesor obrade slike	*Intel RealSense Vision Processor* D4

V. Tablica karakteristika korištenog *BLDC APS* 5065 motora (29)

VI. Tablica mehaničkih karakteristika korištenog enkodera CUI AMT10E2-V [30]

MECHANICAL

parameter	conditions/description	min	typ	max	units
motor shaft length		9			mm
motor shaft tolerance			NOM +0/-0.015		mm
weight	AMT10E2 AMT10E3		20.5 14.0		g g
axial play				±0.3	mm
rotational speed (at each resolution)	480, 960, 1000, 1250, 2000, 2500, 2560, 5120			3000	RPM
	120, 240, 250, 312.5, 500, 625, 640, 1280			6000	RPM

ENVIRONMENTAL

parameter	conditions/description	min	typ	max	units
operating temperature[1]		-40		100	°C
humidity	non-condensing			95	%
vibration	20~500 Hz, 1 hour on each XYZ			10	G
shock	11 ms, ±XYZ direction			50	G
RoHS	yes				

Note: 1. Encoders with operating temperature of -40~125°C are available as a custom order