Razvoj i trendovi percepcije okoline kod autonomnih uređaja

Kržina, Elena

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Elena Kržina

RAZVOJ I TRENDOVI PERCEPCIJE OKOLINE KOD AUTONOMNIH UREĐAJA

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Elena Kržina

Matični broj: 0016135585

Studij: Baze podataka i baze znanja

RAZVOJ I TRENDOVI PERCEPCIJE OKOLINE KOD AUTONOMNIH UREĐAJA

DIPLOMSKI RAD

Mentor:

doc. dr. sc. Bogdan Okreša Đurić

Varaždin, kolovoz 2024.

Elena Kržina

Izjava o izvornosti

Izjavljujem da je ovaj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Računalni je vid danas bitno područje koje se istražuje desetljećima u svrhu pronalaska optimalnog načina na koji računalo može percipirati, raspoznati i razumjeti okolinu oko sebe. Ono je zahtjevno područje koje se kontinuirano razvija. U radu se opisuje računalni vid, njezina povijest i trendovi te se naznačuje što se sve smatra računalnim vidom i koje su njegove potencijalne koristi. Posebice se obrađuju suvremeni trendovi percepcije okoline kod autonomnih uređaja kao što su konvolucijske neuronske mreže (CNN) kao vrsta dubokih neuronskih mreža te vizualna odometrija (VO) i objašnjava se zašto se oni koriste. Naposlijetku se prikazuje implementacija prepoznavanja objekata i pozicije kamere u kombinaciji s računalnim vidom preko AI kamere koristeći CNN, OpenCV, Open3D i DepthAI u Pythonu.

Ključne riječi: konvolucijske neuronske mreže; umjetna inteligencija; percepcija; strojno učenje;

Sadržaj

1. Uvod

Umjetna je inteligencija znanost i inženjerstvo kreiranja inteligentnih strojeva i programa za postizanje različitih ciljeva baziranih na ljudskoj inteligenciji [1] [2]. U svijetu je današnjice vrlo važna i koristi se za brojne namjene, poput olakšavanja izvršavanja zadataka, uključivanja pametnog ponašanja u uređaje, samostalnog učenja iz danih podataka i samostalnog razvijanja komponenata, programa ili uređaja. Njezina se korist može prenijeti u razna područja od kojih su samo neka kibernetika, vizualna percepcija, otkrivanje značajki i uzoraka, kreiranje umjetnosti i tekstova, video igre, modeliranje i rad robota i agenata [3]. Ovaj se rad konkretno bavi pitanjem vizualne percepcije okoline za donošenje zaključaka o okolini uređaja perceptora.

Računalni je vid područje koje se istražuje već dugi niz godina za omogućavanje percpecije podataka, njihovog razumijevanja i rada s njima [4]. Složeno je područje i kao takvo ima nekoliko etapa razvoja od kojih svaka donosi nove zaključke i tehnologije. Fokus je ovog rada ispitati način funkcioniranja računalnog vida, ispitati njegov razvoj te proučiti novije trendove i tehnologije i kako one funkcioniraju. U tu je svrhu implementiran kod za razvoj konvolucijske neuronske mreže te kod koji demonstrira vizualni sustav uz AI RGB i stereo kamere koje primaju podatke iz okoline.

Rad je sastavljen od 7 glavnih dijelova. Prvo je poglavlje Uvod u kojem se čitatelja upoznaje s temom i u kojem se postavljaju temelji proučavanje teme. Dalje se navode metode i tehnike rada. U razradi je teme prvo poglavlje Računalni vid u kojem se predstavlja što on je i daje se pregled njegova razvoja te trendova. Sljedeće je poglavlje naziva Duboko učenje i tehnike percepcije prostora, gdje je fokus na predstavljanju načina rada dubokog učenja, specifično konvolucijskih neuronskih mreža uz opis tehnika za percepciju i navigaciju u okruženju. Sljedeće se poglavlje bavi implementacijom opisanih tehnologija. Na kraju se nalazi zaključak unutar kojeg se sažimaju najbitnije stavke rada te se daje pregled i osvrt na rad. Slijedi popis literature, popis tablica i slika te prilozi.

2. Metode i tehnike rada

Za izradu se diplomskog rada proučava pouzdana znanstvena literatura koja prvenstveno uključuje knjige i znanstvene časopise iz područja umjetne inteligencije i računalnog vida. Proučavali su se priručnici alata korištenih u izradi programskog koda za bolje razumijevanje njihovih mogućnosti. Proučavaju se dokumentacije korištenih biblioteka i mnogi načini kreiranja krajnjeg koda kako bi bio zadovoljavajući i prema potrebama poduzeća SICK Mobilisisa d.o.o., u suradnji s kojim je rad pisan. Također su korištene službene stranice korištenih alata.

Za kreiranje dijagrama, korištena je stranica Drawio [5]. Za implementaciju programskog koda korišten je programski jezik Python koji ima detaljno razvijene biblioteke za računalni vid i strojno učenje. Pritom se koriste sljedeće vanjske biblioteke:

- depthai [6];
- numpy;
- matplotlib;
- open3d;
- open3d _python;
- torch i torchvision;
- opencv_python;
- opencv_contrib _python;
- pandas;
- detectron2;
- pipreqs.

Za prikupljanje slika za izgradnju modela koristi se kamera mobitela te se rubne točke objekata za segmentaciju instanci označuju uz online aplikaciju VGG Image Annotator (VIA) [7] [8]. Za kreiranje modela i daljnje prepoznavanje objekata koristi se Detectron2 [9].

Prvi dio aplikacije vizualne odometrije kreiran je uz KITTI dataset [10]. Kamere koje se koriste u implementacijskom dijelu rada za primanje podataka iz okoline su Luxonis OAK-1 RGB kamera [11] i OAK-D-Lite Pro stereo kamera [12].

3. Računalni vid

U ovom se poglavlju opisuje računalni vid. Opisuju se najčešći oblici tehnika računalnog vida te se pruža pregled povijesnog razvoja računalnog vida kako bi se istakli bitni povijesni trenuci i trenutni trendovi u znanosti.

3.1. Pojam računalnog vida

U ovom se dijelu definira pojam računalnog vida te se ističu ciljevi njegova razvoja kao priprema za opis razvoja povijesti vida. Također se ističu njegove korisnosti u raznim područjima i nabrajaju se poznatije tehnike provođenja algoritama računalnog vida današnjice.

3.1.1. Definicija računalnog vida

Računalni je vid područje uz koje računala razumiju i upotrebljavaju slike [13]. Dio je područja umjetne inteligencije (engl. *artificial intelligence*) (AI) koje koristi strojno učenje i neuronske mreže za učenje računala za dobivanje korisnih informacija iz digitalnih slika, videa ili drugih vizualnih inputa [14]. Slike predstavljaju dvodimenzionalnu ili trodimenzionalnu prostornu reprezentaciju stvarnog svijeta [15].

Jedan je od primarnih ciljeva računalnog vida razumijevanje vizualnih scena, koje se sastoji od različitih zadataka kao što su prepoznavanje objekata prisutnih u promatranoj sceni, njihova segmentacija, lokalizacija u 2D i 3D prostoru, pružanje semantičkog opisa scene, prepoznavanje lica, detekcija akciji, procjena poza i druge [16]. Razumijevanje percipirane scene, radnja primjerenih akcija i učenje iz podataka okoline postižu se tako da se računalima pruža mogućnost percepcije slične ljudskoj [17].

Gleda li oko sebe, osoba jednostavno prepoznaje objekt ispred sebe i prepoznaje koliko je on otprilike udaljen, može li se uzeti u ruku, kako će osoba to učiniti, ima li to smisla učiniti i slično. Cilj je računalnog vida percipirati svijet na isti način kao i čovjek [13]. To znači znati o kojem se objektu radi kad se objekt nalazi u vidnom polju računala, znati njegove karakteristike i kako postupati njemu po potrebi korisnika.

No, iako se na početku razvoja ovog područja smatralo da je računalni vid problem koji će se jednostavno riješiti i koji će pridonijeti rješavanju težih zadataka poput promišljanja i planiranja na višim razinama, uspostavlja se da je računalni vid vrlo težak zadatak jer računalo nastoji opisati svijet u jednoj ili više slika kako bi rekonstruirala stvarnost, što je obrnuto od ljudske percepcije [4]. Nadalje, računala za vid trebaju razviti model, odnosno potreban je velik broj iteracija analize nad input slikama [14]. Dok je čovjek u stanju prepoznati sve objekte oko sebe jer uči o objektima i njihovim svojstvima od početka života, model će znati prepoznati samo one objekte za koje nosi informaciju u njegovu modelu koji je treniran na znatno manjem broju podataka od ljudi. Sve ostalo što se nalazi u percipiranoj okolini računalu je nepoznato. Za računalni je vid stoga vrlo bitan razvoj područja kognitivnih procesa u psihologiji kao i postignuća u drugim znanostima [13].

Slika 1: Detekcija objekata kategorije boce na policama u maloprodaji; preuzeto iz [18]

3.1.2. Primjeri korisnosti kod upotrebe računalnog vida

Iako je računalu potrebno napraviti i trenirati modele za razne operacije računalnog vida, dobro trenirani modeli mogu analizirati tisuće proizvoda i procesa u minuti s malim brojem pogrešaka [14]. Zbog velike količine promatranih objekata i veće brzine izračuna, ovo omogućuje postizanje znatno boljih performansa u radu od ljudi.

Čovjeku primjerice treba puno više vremena za izbrojiti prisutne proizvode na policama u dućanu nego što to treba računalu, posebice ako je količina proizvoda velika.

Na slici 1 su prikazani proizvodi u bocama na policama u maloprodaji, od kojih je svaki označen graničnim okvirom (engl. *bounding box*). U ovom slučaju, treniran model nije sasvim siguran jesu li proizvodi na policama boce, što se može zaključiti iz niskih postotaka sigurnosti koje model prikazuje na vizualu. Svakako, model će brže doći do ukupnog broja boca na slici nego što će čovjek. Što je model sigurniji u kategorizaciju objekata na slici, to je on precizniji i dati će bolje rezultate.

Slika 2 prikazuje segmentaciju instanci boci na policama u maloprodaji, pri čemu su pikseli svake instance bojani drugom bojom. Vidljiva je zadovoljavajuće precizna procjena broja boci koje su na policama. Za kreiranje slike korišten je priložen Python kod pod Prilog 1 te se koristio unaprijed trenirani Detectron2 model pri čemu se izdvojila klasa za segmentaciju boci te postojeća slika [9]. U kodu je postavljen prag preciznosti na 0.6 kako bi program označio i brojio boce samo ako je model barem 60% siguran da je prikazani objekt boca.

Prednost korištenja računalnog vida u ovoj situaciji mogla bi se povezati s optimizacijom rada u trgovini ili na skladištu. Budući da kod omogućuje brojanje instanci, ako kamera snimi da se broj instanci boca smanji za određenu količinu, tek tada mogla bi se poslati poruka radnicima za popunjavanje polica bez potrebe prethodnog provjeravanja. Čak u slučaju da iza fotografiranih boci postoje dodatne boce koje se iz trenutne pozicije ne vide, broj se neće mijenjati na manje jer će dobro trenirani model prepoznati sljedeću bocu iza, pa se obavijest za potrebom dodavanja novih boci ne bi poslala.

Slika 2: Segmentacija instanci kategorije boce na policama u maloprodaji; preuzeto iz [19]

Nadalje, trenirano računalo može uspješnije raspoznati uzorke u objektima kad su u pitanju male razlike među njima. Uzmimo za primjer prepoznavanje ispravnih od neispravnih mehaničkih dijelova poput zupčanika. Računalo s kvalitetno treniranim modelom ranije će raspoznati greške u proizvodnji nego čovjek koji bi možda proizvod trebao pregledati iz različitih kuteva ili koji nije dovoljno upućen u dizajn samog proizvoda. Modeli često koriste i filtre, sivu boju slika (engl. *grayscale*) te razne distorzije slika koje omogućuju bolji pronalazak značajki. Time će računalo ranije raspoznati problematičan dio proizvoda, kao što je primjerice teško uočljiva pukotina.

Tablica 1 u nastavku prikazuje današnje poznate tehnike za rad s podacima koje računalo percipira u svojoj okolini. Pruža se kratak opis svake tehnike i naznačuje gdje se može koristiti. Bitno je za naglasiti da su primjene svake tehnike brojne i da su naznačene samo neke.

3.2. Povijesni razvoj računalnog vida i njegovi trendovi

U nastavku se proučava povijesni razvoj računalnog vida te se vade zaključci iz priloženog. Opisuju se nastale tehnologije i razmišljanja etapa u razvoju računalnog vida. Na kraju se ističu najznačajnije tehnologije današnjice kao uvod u daljnji prikaz njihova rada.

Tablica 1: Česte tehnike računalnog vida i njihova primjena

Naziv	Opis	Primjenjivost
Klasifikacija objekata	Predviđa klasu kojoj neki objekt pripada te objekt dobiva labelu klase [14] [20]. Klase su predefinirane [20].	Određivanje postoji li objekt željenje klase u prostoru, baza za druge tehnike. [14]
Detekcija objekata	Povezana s klasifikacijom; porepoznaje određenu klasu objekta i označava je u mediju [14]. Objekt se označava graničnim okvirom (engl. *bounding box*) [20], odnosno vraća se lokacija i opseg objekta [21] [22].	Baza za rješavanje kompleksnijih zadataka kao segmentacije objekata, razumijevanje scene, labeliranje slika, detekcija događaja i aktivnosti, robotska vizija, autonomna vožnja, biometrija i sl. [20] [21].
Lokalizacija objekata	Pronalaženje instance određene kategorije objekta u slici prema graničnim okviru instance [23]. Omogućuje prepoznavanje i razumijevanje sadržaja slike [24].	Analiza slika i videa, praćenje objekata, proširena stvarnost i sl. [24].
Segmentacija objekta	Kreiranje maska prepoznatih objekata i označavanje svakog piksela objekta kao dio klase [25] [22]. Daje preciznije rezultate ekstrakcije [22]. Dijeli se na semantičku segmentaciju i segmentaciju instanci [26].	Medicinsko oslikavanje [20], oslikavanje prometa, segmentacija i brojanje objekata u skladištu i sl.
Podudaranje značajki	Pomaže pri povezivanju značajki dviju slika, gdje su značajke područja slike koja govore više o promatranom objektu. Značajke se detektiraju, formiraju se lokalni deskriptori te se oni usklađuju [20].	Dio vizualne odometrije [27] i simultane lokalizacije i mapiranja kao praćenje lokacije, mapiranje u robotici i proširenoj stvarnosti [28] te kalibraciji kamere i identificiranje objekata [20].
Rekonstrukcija scene	Digitalna 3D rekonstrukcija scene nekog objekta preko slika [20].	Struktura iz kretnje [27]; kreiranje modela arhitekture, objekata u prostoru i sl.

3.2.1. Počeci eksperimentiranja računalnim vidom (od 1959. godine)

Eksperimentiranje računalnim vidom započinje 1959. godine, kada neuropsiholozi Hubel i Wiesel mački prikazuju niz slika kako bi ispitali uzajamnu vezu slika i kako mozak reagira na njih. Znanstvenici su preko elektrode snimali reakcije mačjeg mozga na stimulus prikazan na projektoru [14] [29]. Eksperiment je služio za razumijevanje vizualnog korteksa [30].

Zaključak koji dobivaju je da mozak isprva reagira na rubove i linije. To znači da obrada slike započinje jednostavnim oblicima, dok se do složenijih oblika dolazi kombinacijama jednostavnih [14] [29]. Još jedan zaključak njihova istraživanja je da u mozgu postoje jednostavne ćelije koje reagiraju na točno određene linije i orijentacije na točno određenim pozicijama vidnog polja. Tim se reakcijama dalje koriste složene i hipersložene ćelije za otkrivanje složenijih oblika [31] [32]. Nije poznato što se događa u kasnijim slojevima [32].

Ovaj će rezultat kasnije imati velik utjecaj na oblikovanje tehnika računalnog vida korištenih u današnje vrijeme.

Otprilike u isto vrijeme razvija se prva tehnologija za digitalizaciju slika. 1963. godine računala omogućuju transformaciju 2D u 3D slike [14].

1956. je godine utemeljeno područje umjetne inteligencije na koledžu Dartmouth. Na radionici događaja predstavljena su istraživanja o mnogim temama kao što su dokazi matematičkih teorema, obrada prirodnih jezika, planiranje igara, računalni programi koji mogu učiti iz primjera i slično [33]. Utemeljenjem umjetne inteligencije kao znanosti omogućuje se šire i veće područje rada za novu znanost.

1960. se godine kreira model Perceptron, koji je predložen model za raspoznavanje uzoraka [34]. Perceptron je mreža sastavljena od umjetnih neurona i sinaptičkih veza, gdje svaki neuron definira utjecaj na drugi [32]. Ako je neka sinapsa pod naponom, a neuron A je aktivan, tada on pokušava aktivirati neuron B. Ako je sinapsa inhibitorna, a neuron A je aktivan, tada on pokušava potisnuti neuron B [32].

Svi ovi događaji postavljaju temelje za daljnji razvoj računalnog vida. Hubel i Wiesel postavili su temelj za razumijevanje vizualne percepcije u mozgu, koji će se kasnije koristiti za oponašanje funkcije mozga pri tehnikama računalnog vida. Digitalizacija slika omogućuje daljnji rad sa slikama na računalu, a stvaranje umjetne inteligencije kao znanstvenog područja omogućava daljnji razvoj područja u obliku formula, interesa, novih tehnika i tehnologija, matematičkih izračuna, modela i sličnog. Perceptron je prvi predloženi model za raspoznavanje uzoraka.

3.2.2. Počeci računalnog vida (od 1970-ih godina)

Pravi početak samog računalnog vida kao područja istraživanja kreće 1970-ih godina prošlog stoljeća, kao percepcijska komponenta programa koji bi oponašao ljudsku inteligenciju u robotima [4]. Kao što je prije naznačeno, u ovom se razdoblju vjerovalo da će se problem vizualnih inputa riješiti lagano prije no što se prijeđe na teže probleme kao što su promišljanje

i planiranje računala na višim razinama [4]. To zbog ranije navedenih razloga nije bilo tako, već je kreiran kompleksni problem koji se još i danas nastoji riješiti na što efikasniji način.

Neki od ranih pokušaja razumijevanja scene odnose se na ekstrakciju rubnih točaka iz slika i zaključivanja 3D strukture objekata iz topološke strukture 2D objekata. Javlja se i prvi stereo algoritam baziran na značajkama i algoritam optičkog toka baziran na intenzitetu [4]. 1974. godine kreira se i algoritam za optičko prepoznavanje znakova (engl. *optical character recognition*) (OCR) i za inteligentno raspoznavanje znakova (engl. *intelligent character recognition*) (ICR) [14].

OCR je pritom elektronički prijevod rukom napisanog, strojem ispisanog ili isprintanog teksta u strojno prevedene slike često korišten pri prepoznavanju i pretraživanju elektroničkih tekstova te objavi isprintanih tekstova u digitalnom obliku [35]. ICR je sličan OCR-u, no dizajniran je za prepoznavanje rukom pisanih znakova, pri čemu znakovi nikad nisu jednaki [36]. Budući da znakovi ne mogu biti jednaki, teži je za realizirati od OCR-a. Iako i OCR u definiciji sadrži rukom napisane tekstove, često se pod OCR podrazumijeva samo strojno napisan tekst [36].

Ovo je razdoblje značajno zbog početaka računalnog vida i raspoznavanja problema koji se javljaju prilikom rada s njime. Započinje se s izvođenjem nekih tehnika računalnog vida koje se koriste i danas, kao što su ekstrakcija točaka, OCR i ICR.

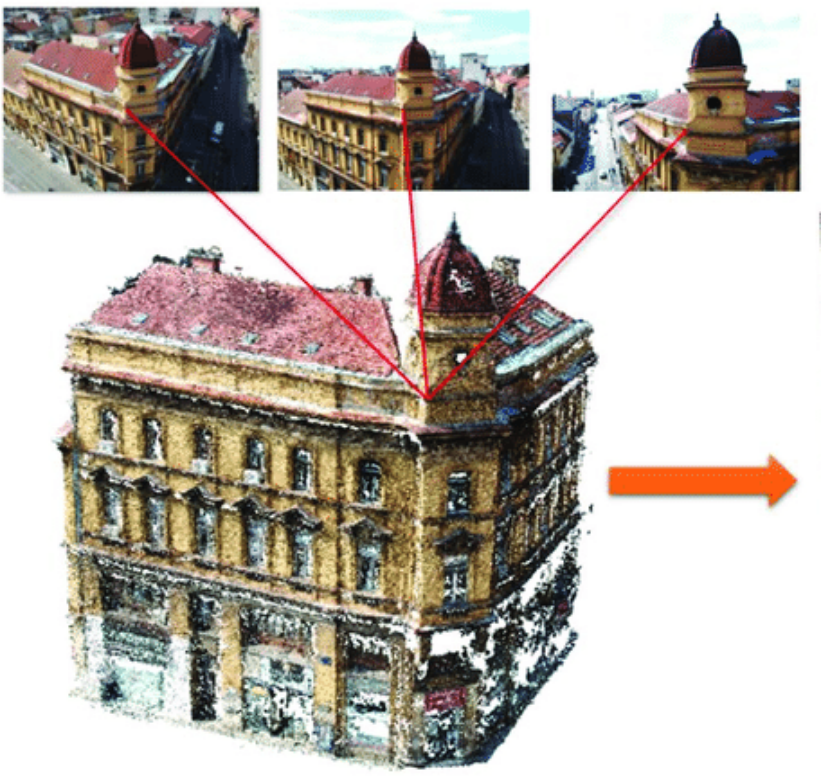
3.2.3. Daljnji razvoj računalnog vida (od 1980-ih godina)

1980-ih godina u fokusu je razvoj sofisticiranih matematičkih tehnika za izvođenje scena i slika [4]. 1982. David Marr uspostavlja da vid funkcionira na hijerarhijski način [14]. To znači da se neki dijelovi slike percipiraju i raspoznavaju ranije od drugih i da u vidu postoji višestruka obrada podataka. Izdaje algoritme za uređaje za prepoznavanje rubova, kutova, krivulja i sličnih oblika [14]. Njegov je rad vrlo značajan, no bio je veoma apstraktan bez potrebnih matematičkih modela koji bi se mogli koristiti u procesu učenja [29]. U isto vrijeme, japanski znanstvenik Kunuhiko Fukushima razvija mrežu stanica koje mogu prepoznavati uzorke bez obzira na promjene u poziciji [14] [29]. Kreiran model, Neocognitron, smatra se prvom dubokom mrežom [29] [37].

U ovom se razdoblju počinju koristiti slikovne piramide za spajanje slika i pretragu, tehnike oblikovanja kao oblikovanje iz sjenčanja, fotometrični stereo, oblikovanje iz teksture i oblikovanje iz fokusa [4]. 1984. godine, uspostavlja se da se većina algoritama oblikovanja može opisati istim matematičkim okvirom preko Markovljevog slučajnog polja [4].

1990-ih godina istraživanja se koncentriraju na problem SfM-a, odnosno strukture iz kretnje (engl. *Structure from Motion*) [4]. Ono je proces rekonstrukcije 3D strukture neke slike iz 2D niza slika uzetih iz različitih gledišta [38]. Na Slici 2 prikazan je primjer SFM-a. Model je kreiran putem slikanja zgrade iz različitih kuteva te njihovim kasnijim spajanjem.

Kreiraju se faktorizacijske tehnike za ortografske kamere i počinje se koristiti algoritam za potpunu optimizaciju u fotogrametriji [4]. Istražuju se stereo algoritmi koji kreiraju 3D površine, razvijaju se tehnike produkcije 3D volumetričkih opisa uz tehnike praćenja i rekonstrukcije



Slika 3: Primjer SfM-a; preuzeto iz [39]

glatkih kontura i napreduju algoritmi praćenja za primjerice praćenje lica i tijela [4].

Javljaju se i tehnike za segmentaciju slika te statističke metode učenja, od kojih je prva Eigenface za prepoznavanje lica [4]. Princip rada Eigenface je transformiranje lica u mali skup značajnih karakteristika, zvanih eigenface, koji su input za učenje modela. Projicira se nova slika u prostor eigenfacea, te se osoba prepoznaje i klasificira usporedbom svoje pozicije s drugim poznatim individuama [40].

Prije je glavna paradigma prepoznavanja objekata bila bazirana na geometrijskim reprezentacijama [21]. Prepoznavanje se objekata primjerice svodilo na fokusiranje na specifične objekte koji imaju relativno rigidnu prostornu strukturu, odnosno lokacije željenih objekata unaprijed su poznate [21]. Primjer ovakvog pristupa je prepoznavanje lica [21]. Za oči su, nos i usta na licu poznate prostorne koordinate. Iako svaka osoba ima različiti položaj ovih dijelova lica, objekti se uvijek nalaze na lokaciji na kojoj ih računalo može predvidjeti. Problem je u ovom pristupu to što je prepoznavanje objekata ovisno o položaju značajki koje se promatraju. Pri izobličenju ili uz specifični kut, objekt ne bi bio prepoznat jer se značajka ne bi nalazila na mjestu na kojem bi je računalo očekivalo.

Također se pojačava interakcija s računalnom grafikom, posebice u području modeliranja i renderiranja na bazi slika [4].

David Marr dokazuje da vid radi na hijerarhijski način, odnosno uspostavlja da se neki oblici prepoznaju ranije od drugih i postoji složen, višestruki proces prepoznavanja u vidu. Ovaj je zaključak ključan za daljnji razvoj računalnog vida. Također postavlja algoritme i podlogu za daljnji rad s ovim znanjem, dok Kunuhiko Fukushima razvija prvu duboku mrežu. U radu

Slika 4: Eigenface; preuzeto iz [41]

se s računalnim vidom dalje koriste Markovljeva slučajna polja koja su vrlo bitna za današnje algoritme zbog sljedećih razloga [42]:

- omogućuju sistematski razvoj algoritama temeljen na čvrstim principima umjesto heuristike,
- olakšavaju izvođenje kvantitativnih mjera uspješnosti za karakteriziranje učinkovitosti algoritama za analizu slika,
- uključuju se različite prediktivne kontekstualne informacije ili ograničenja na kvantitativan način i
- algoritmi su često lokalni i skloni paralelnoj implementaciji na hardverskoj razini.

Iako je danas rijeđe korištena, SfM je tehnika koja se koristi u algoritmima poput SLAM-a [43], koji je korišten od primjerice autonomnih uređaja za iscrtavanje nepoznatog okruženja dok prati svoju lokaciju, bez potrebe za GPS-om ili drugim uređajem za mjerenje lokacije [44].

Isto su tako vrlo bitne tehnike 3D opisa, praćenja i rekonstrukcija kontura, praćenja lica i tijela te tehnike za kreiranje semantičke segmentacije koje se koriste i usavršavaju i danas.

3.2.4. Učenje pogonjeno podacima (2000-te godine)

2000-tih se godina ovog stoljeća fokus učenja računalne vizije prebacuje na podatke, odnosno učenje je pogonjeno podacima kao inputima u algoritme [4]. Do 2000. je godine fokus znanosti na prepoznavanju objekata [14]. Javlja se pojam komputacijska fotografija u koju spadaju tehnike poput renderiranja, hvatanja svjetlosnog polja, hvatanje slika visokog dinamičnog raspona i šivanja kako bi se priznalo povećano korištenje ovakvih tehnika u svakodnevnoj digitalnoj fotografiji [4].

2000. je godine standardiziran način označavanja vizualnih podataka [14], a 2001. se godine kreira prva aplikacija za prepoznavanje lica [14].

Nadalje, zbog velike količine djelomično ili potpuno označenih podataka na internetu i zbog javljanja snažnijih računala, razvijaju se sofisticirane tehnike strojnog učenja za problem računalnog vida [4]. Od 2005. započinje dugotrajni projekt u kojem se kreiraju i održavaju serije *benchmark* skupova podataka. Projekt završava 2012. godine [22].

Prebacivanje procesa učenja na podatke vrlo je značajno za sljedećih nekoliko godina razvoja računalnog vida, posebice uz početak razvoja projekta za kreiranje i održavanje serije *benchmark* skupova podataka zbog buduće mogućnosti razvoja konvolucijskih neuronskih mreža i potrebnih snažnih skupova podataka za njihovu propagaciju u računalnom vidu. Događa se standardizacija označavanja vizualnih podataka za olakšanu širu upotrebu te se kreira prva aplikacija prepoznavanja lica.

3.2.5. Označeni skupovi podataka i revolucija računalnog vida (2010-te godine)

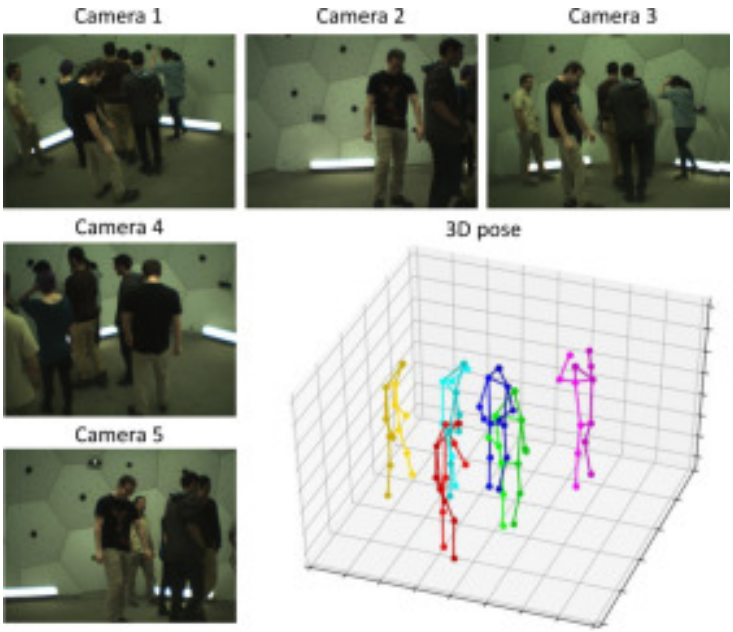
Označeni skupovi podataka čine potpuni preokret u razvoju algoritama za prepoznavanje slika, uklanjanje šuma i optički tok [4]. 2010. godine postaje dostupan ImageNet, skup podataka visoke kvalitete koji sadrži milijune označenih slika za tisuće objektnih klasa [4] [14].

Osim ImageNet-a, još neki od kvalitetnih označenih skupova podataka su Microsoft COCO i LVIS. Svi oni pružaju i pouzdane metrike za praćenje napretka učenja prepoznavanja i segmentacije modela, koji su detaljnije objašnjeni u ostatku rada [4]. Ovi veliki modeli omogućuju istraživačima rješavanje realističnijih i kompleksnijih problema [26]. Svaki model traži veliku količinu podataka za uspješno treniranje, a ozbiljniji problemi traže visoku preciznost. Stoga skupovi podataka poput ImageNet-a i ostalih čine veliki preokret u korištenju računalnog vida u problemskom području.

Stvara se velik broja modela za duboko učenje (engl. *Deep Learning*) (DL) te konvolucijskih neuronskih mreža (engl. *Convolutional Neural Networks*) (CNN) [14]. Ipak, precizan i točan rezultat vrlo je ovisan o kvaliteti točnih oznaka na podacima, koje je teško naći u velikom broju [26]. Zato je potrebno osmisliti metode koje smanjuju poteškoće s označavanjem skupova podataka [26].

Dramatično se razvijaju računalne mogućnosti zbog razvoja grafičkih kartica (GPU), koje su specijalizirane za matematičke i geometrijske računice [4] [21] [26], odnosno za vrlo brzo množenje matrica [33]. Dubinsko učenje uvelike ovisi o ogromnim količinama podataka za treniranje, kao i o razvoju grafičkih procesnih jedinica [21].

30. rujna 2012. izvodi se ImageNet natjecanje za prepoznavanje objekata u računalnom vidu [29]. Za natjecanje je kreiran model AlexNet koji značajno smanjuje postotak pogreške u prepoznavanju slika. Model se oslanja na učenje uz GPU i pobjeđuje natjecanje, nakon čega se područje računalnog vida razvija i povećava [4] [14] [29]. Nakon objave pobjedničkog časopisa, napredak u razvoju CNN-a ubrzano raste do mjere da su one trenutno jedina arhitektura koja se uzima u obzir pri prepoznavanju objekata i pri njihovoj segmentaciji [4]. Uvjeti također omogućuju daljnje razvijanje prepoznavanja akcija, razumijevanje videa i strukturalno-regresijskih zadataka kao što je prepoznavanje ljudske pozicije u stvarnom vremenu [4]. Duboko učenje



Slika 5: Primjer algoritma prepoznavanja ljudske pozicije tijela; preuzeto iz [45]

danas je svake godine popularnije zahvaljujući upravo CNN arhitekturi, čemu i doprinosi njezina sveprisutnost, te je postala najčešće korištena metoda za računalnu viziju današnjice [26] [46] .

Razvijaju se kamere s mogućnošću prepoznavanja dubine, odnosno stereo kamere, razni AI algoritmi na pametnim telefonima, aplikacije koje koriste proširenu stvarnost s prepoznavanjem ljudske pozicije u stvarnom vremenu. Razvija se vizualna odometrija (VO) i SLAM algoritam za izgradnju preciznih 3D mapa i autonomni let primjerice dronova kroz kompliciranije scene poput šuma [4].

Iako se računalni vid cijelo vrijeme nadograđuje i razvija, još je uvijek teško dizajnirati precizan, dostupan i efikasan sustav za prepoznavanje koji bi mogao odgovarati ljudskoj mogućnosti percepcije okoline, koja prema procjenama kognitivnih znanstvenika sadrži ukupno 30000 vizualnih kategorija [21]. Računalni se vid razvija već dugi niz godina te na njega veoma utječu uspjesi iz različitih područja znanosti poput kognitivnih znanosti, umjetne inteligencije, računalne grafike, računarstva i sličnih. Kao takvo područje, računalna se percepcija ne razvija jednoznačno, već se tehnologije stvaraju kako se dolazi do novih spoznaja.

U radu će se dalje obrađivati duboko učenje i specifično konvolucijske neuronske mreže kako bi se objasnio princip rada ove danas izrazito bitne arhitekture koja se koristi za objektnu klasifikaciju, prepoznavanje objekata, segmentaciju instanci, semantičku segmentaciju, [4], medicinsko slikanje, obrada zvuka, generiranje podataka [47], pa i za ICR [36]. Također se i objašnjava i implementira rad s vizualnom odometrijom kao potreba za projekt u SICK Mobilisisu.

4. Duboko učenje i tehnike percepcije prostora

U ovom se poglavlju opisuju i objašnjavaju tehnike dubokog učenja uz fokus na CNN, čija je implementacija prikazana u kasnijem dijelu rada. Nakon CNN-a, opisuje se i objašnjavaja tehnika percepcije prostora vizualna odometrija, čiji se primjeri također nalaze u implementacijskom dijelu rada.

4.1. Duboko učenje

U ovom se dijelu rada opisuje i objašnjava proces dubokog učenja. Navode se načini rada dubokog učenja te se u detalje opisuju konvolucijske neuronske mreže.

4.1.1. Pojam dubokog učenja

Duboko je učenje podskup generalnog polja umjetne inteligencije koji se zove strojno učenje na temelju učenja iz primjera [48]. Umjesto da se računalu daje algoritam koji mu kaže kako napraviti neku akciju, računalo je usredotočeno na stvaranje velikog modela putem kojeg računalo evaluira sve daljnje ulazne podatke [33] [48]. Računalo trenira model kako bi dobro prilagođeni model probleme mogao riješiti ekstremno točno [33].

Duboko učenje omogućuje prepoznavanje uzoraka u različitim razinama apstrakcije uz minimalni ljudski doprinos [26] [49] [50]. Može raspoznavati informacije prema prostornim, spektralnim, i temporalnim informacijama iz okoline na niskoj razini apstrakcije poput kutova, linija i rubova, do visoke razine koju čine čitavi objekti [26]. Duboko je učenje danas često korištena tehnika za rad s podacima u obilju. Koriste je različite tvrtke za analizu internih podataka, za analizu tržišta, pri obradi slika, prepoznavanja lica i slično. Na potražnju za analizom vizualnih podataka uvelike utječe rasprostranjenost društvenih mreža i mobitela [21]. Koristi se za velik broj zadataka, kao što su primjerice prepoznavanje objekata, promjena prepoznavanja, semantička segmentacija objekata, analiza vremenskih serija, segmentacija instanci objekata, registracija slika, klasifikacija objekata, prepoznavanje ljudskih pozicija i slično [26].

4.1.2. Način rada dubokog učenja

Duboko je učenje složen proces baziran na radu ljudskog mozga i procesa učenja u njemu. Za razumijevanje načina rada dubokog učenja, isprva se objašnjava proces učenja ljudskog mozga te se prelazi na opis i objašnjenje procesa učenja u računalima.

4.1.2.1. Proces učenja u mozgu

Temeljni se gradivni element ljudskog mozga naziva neuron [48]. I sitni dio mozga sadrži preko 10000 neurona, gdje svaki od njih ima vezu s 6000 drugih neurona [51]. Neuroni su optimizirani da primaju informaciju od drugih neurona, obrade tu informaciju na jedinstven

način te se rezultat šalje dalje drugim stanicama mozga [48].

Svaki je neuron umrežen s velikim brojem drugih neurona preko veza koje se nazivaju sinapse [32]. Neuroni djeluju paralelno i preko sinapsi šalju informacije koje se dobivaju obradom informacija unutar jedinici [32]. Vjeruje se da obradu podataka vrše neuroni, a da se ljudsko sjećanje, odnosno memorija, nalazi u sinapsama, odnosno u načinu na koji su neuroni međusobno povezani i na koji utječu na druge [32].

Neuron svoj input dobiva od dijela koji se zovu dendriti, pri čemu je svaka od veza dinamično pojačana ili smanjena ovisno o tome koliko se često koristi i jačina svake veze određuje doprinos inputa na neuronov output [48]. Pritom su neki su od neurona osjetilni i primaju vrijednosti iz okoline, koje se onda proslijeđuju drugim neuronima koji nad njima vrše dodatnu obradu te se vrijednosti dalje šire mrežom. Postoje i izlazni neuroni, koji donose konačnu odluku o radnji i provode akcije kroz aktuatore, kao što je izgovaranje riječi [32].

Tijekom rada, svaki neuron u tijelu stanice zbraja ponderirane jačine odnosno aktivacije svih neurona koji su s njime povezani preko sinapsa i ako je ukupna aktivacija veća od nekog određenog praga, neuron se aktivira, odnosno "pali" (engl. *fire*) [48]. Neuron svoju aktivacijsku vrijednost šalje svim neuronima s kojim je povezan [32] [48]. Njegov izlaz odgovara vrijednosti te aktivacije [32].

Ljudski je mozak stoga građen od mnoštva međusobno povezanih jedinica obrade neurona, koji primaju vrijednosti iz okoline i proslijeđuju ih drugim neuronima na daljnju obradu. Svaki neuron tijekom rada zbraja aktivacije svih njime povezanih neurona i aktiviraju se ovisno o rezultatu, što će biti jednako izlazu.

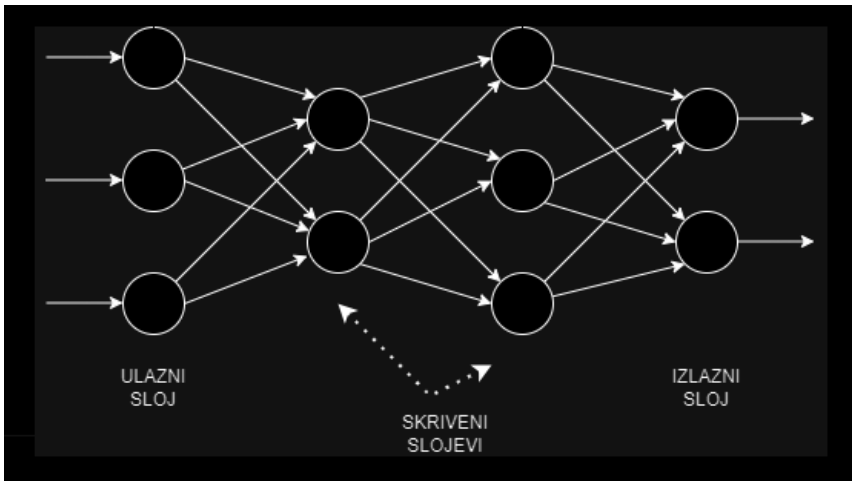
4.1.2.2. Proces učenja u računalima

Nakon opisa procesa učenja u mozgu, cilj je izgradnja takvog sustava ili modela koji razumije kako funkcionira ljudski mozak i iz njega može proizvesti takav apstraktni, matematički i računalni opis koji možemo računalno implementirati za potrebe računalnog vida [32]. Model koji opisuje proces dubokog učenja u računalima zove se model umjetnih neuronskih mreža.

Umjetne su neuronske mreže (engl. *artificial neural networks*) (ANN) sustavi za obradu računanja inspirirani biološkim živčanim sustavom, gdje se mreža sastoji od velikog broja međusobno povezanih čvorova za kolektivno učenje iz inputa kako bi se dobio što optimalniji output [32] [52]. Na slici je u nastavku prikazana bazična struktura neuronske mreže.

Svaki čvor na slici reprezentira jedan neuron. Vidljivo je da ni jedan neuron nije izdvojen iz mreže neurona, već da su svi povezani. Snaga je neuronskih mreža interakcija velikog broja skupa jednostavnih neurona [33]. Također je vidljivo da su neuroni organizirani u slojeve, odnosno nisu svi na istoj razini. Vrste slojeva koje određuju značenje i rad neurona su ulazni, izlazni i skriveni slojevi [33].

Lijevi čvorovi na slici reprezentiraju neurone za input, odnosno ulazne slojeve. Ulazni sloj je jedini sloj koji ne sadrži obradu podataka, već služi isključivo za unos mjerenih podataka u neuronsku mrežu [33].



Slika 6: Struktura umjetne neuronske mreže; izrađeno prema [53]

Kao prvi se korak u procesu učitavaju inputi n, najčešće u obliku višedimenzionalnog vektora [48] [52].

$$x1, x2, ..., xn$$

Zadnji su slojevi u mreži izlazni slojevi ili slojevi outputa. Svi slojevi između ulaznih i izlaznih slojeva čine skrivene slojeve. Pritom kad promatramo duboko učenje, najmanji mogući broj skrivenih slojeva mora biti jednak ili veći od dva [33]. Dubina se neuronske mreže mjeri brojem skrivenih slojeva uz dodavanje izlaznog sloja [33]. To znači da ako neka neuronska mreža sadrži tri skrivena sloja neurona, dubina joj je, uz izlazni sloj, jednaka četiri. Učenje s više skrivenih slojeva teže je i sporije zbog većeg broja obrade, smetnje zbog velikog broja parametara i propagiranje greške unatrag za ažuriranje svih prethodnih slojeva [32].

Izlaz iz svakog neurona je vrijednost podataka pohranjenih u dotičnoj memorijskoj lokaciji svakog neurona [33].

Proces učenja kod umjetnih neuronskih mreža svodi se na rad skrivenih slojeva. Skriveni slojevi dobivaju input podatke i na temelju njih odlučuju i mjere stohastičku promjenu unutar sebe kojom utječu na završni output [52]. Ovi se neuroni nazivaju skrivenima jer odgovaraju skrivenim značajkama koje se izravno ne opažaju, ali su definirani na temelju opaženog iz inputa [32]. Svaki sljedeći neuron uzima ulaz, koji je izlaz iz procesnog neurona ili osjetilnog, neobrađenog, i preslikava ga u jednu izlaznu vrijednost [33].

Veze su među neuronima usmjerene i svaka zasebna veza sadrži neku povezanu težinu, koja je broj koji utječe na to kako neuron obrađuje informacije koje pristižu preko nje [33]. Svaki je input u neuron pomnožen s odgovarajućom težinom [52]. Tijekom procesa se treniranja modela te težine ažuriraju kako bi smanjile pogrešku primjera [32]. Što je manja pogreška u izlazu, to je izlaz bliži stvarnosti [32].

$$w1, w2, ..., wn$$

U preslikavanju ulaza u neuron u njegov izlaz, provode se dva koraka [33]:

- računanje težinskog zbroja ulaza u neuron i prosljeđivanje funkciji za preslikavanje
- propuštanje rezultata kroz aktivacijsku funkciju

Isprva se računa težinski zbroj ulaza u neuron [33]. Izmjereni se inputi zbrajaju te se kreira logit za neuron $z = \sum_{i=0}^{n} wixi$. U mnogim slučajevima, logit također sadrži pristranost (engl. *bias*), koji je neka konstanta b. Logit se prenosi kroz funkciju f kako bi se kreirao output $y = f(z)$ koji se širi do drugih neurona [48].

Rezultat se tog zbroja proslijeđuje sljedećoj funkciji koja rezultat preslikava u konačnu izlaznu vrijednost iz dotičnog neurona [33]. Ova se izlazna vrijednost naziva aktivacijska vrijednost [33]. Funkcija učenja umjetnih neuronskih mreža je stoga [48]:

$$y = f(x \cdot w + b)$$

Rezultat tog zbroja se dalje propušta kroz aktivacijsku funkciju za preslikavanje ulaza u izlaze. One mogu biti različitih oblika [33]. ReLu i tangens hiperbolični [46] primjeri su nelinearne aktivacijske funkcije, dok postoje i linearne funkcije, koje output određuju pomoću jednadžbe pravca [54]. Pritom neuroni mogu imati različite aktivacijske funkcije, no općenito su svi neuroni unutar određenog sloja istog tipa funkcije [33].

Izlazna vrijednost se dalje proslijeđuje sljedećem sloju, koji definira složeniju značajku u odnosu na jednostavnije uzorke prepoznate u prethodnom primjeru [32]. Izvlačenjem se značajki također provodi redukcija dimenzionalnosti koja pomaže sa smanjenjem vremena i resursa prilikom treniranja modela. Iako značajke koje se promatrajumogu biti brojne, željenih je značajki općenito manje, i njih koristimo za račun izlaza [32]. Ako je neuron u izlaznom sloju, interpretacija značenja izlazne vrijednosti ovisi o zadaći koju neuron modelira [33].

Iz ovog je vidljivo da se i dubokim neuronskim mrežama možemo koristiti određenom vrstom hijerarhije. Neuroni u ranijim slojevima izvlače jednostavnije oblike, dok se jednostavniji kombiniraju i tvore složenije u kasnijim slojevima obrade. Kretanju prema kraju mreže, značajke su sve složenije i apstraktnije, dok se ne dođe do cijele tražene klase [32].

Još jedan bitan aspekt uspješnosti neuronskih mreža je da sve probleme rješava veći broj neurona istovremeno, odnosno problemi se rješavaju koristeći strategiju podijeli pa vladaj [33]. Postavljanjem težina unutar mreže, veći problem dekomponira se na manje probleme koji se raspoređuje po neuronima. Težine se mijenjaju s obzirom na uspjeh ili neuspjeh modela i na kraju se dekomponirani problem kombinira u krajnje rješenje [33]. Prema tome, pri prepoznavanju rukopisa, prvo će se primijetiti ravnije i jednostavnije linije na pojedinim slojevima, a kasnije se u obzir uzima iskrivljenost zapisi, ukošenost slova, specifične mrlje pri pisanju i slično.

Postoje razne vrste dubokog učenja. Neke od tih su povratne neuronske mreže (engl. *Recurrent Neural Networks*) RNN u kojima su neuroni povezani s neuronima idućeg i trenutnog ili prijašnjeg sloja [32], grafovske neuronske mreže (engl. *Graph Neural Networks*) GNN za grafovske podatke, konvolucijske neuronske mreže (engl. *Convolutional Neural Networks*)

CNN, generativne suparničke mreže (engl. *Generative Adversarial Networks*) GAN i drugi. U radu se posebno obrađuje CNN.

4.1.3. Nadzirano učenje, nenadzirano učenje i učenje nagrađivanjem

Postoje tri ključne paradigme za neuronske mreže: nadzirano učenje, nenadzirano učenje i učenje nagrađivanjem.

4.1.3.1. Nadzirano učenje

Učenje je nadzirano kad je unaprijed poznata željena izlazna vrijednost [32]. Postoje inputi koji su unaprijed označeni (engl. *labeled*), koji trebaju biti ciljevi odnosno rezultati učenja [52]. To je primjerice objekt koji se prepoznaje na slici ili očekivani životni vijek čovjeka prema životnim navikama. Cilj je nadziranog učenja sveukupno smanjenje modela klasifikacije grešaka kroz točan račun vrijednosti atributa, pri čemu slike sadrže labele [26]. Svaki skup podataka treba navesti vrijednost ciljne značajke za svaki njegov primjer [33]. Algoritam učenja labele može koristiti kao pomoć u procesu učenja jer to omogućuje usporedbu izlaza iz neurona s ciljnim vrijednostima navedenim u skupu podataka i može mjeriti razliku među njima koju koristi za procjenu ugođenosti funkcijskih težina [33]. Nadzirano je učenje najčešći oblik učenja [33] i najčešće se koristi za prepoznavanje uzoraka u slikama [52].

Kao primjer možemo uzeti algoritam prepoznavanja lica koji je treniran za specifične osobe, prema biometrijskim specifikacijama. Algoritam strojnog učenja na slici prepoznaje značajke poput čela, kose, očiju, nosa i ustiju određene osobe s njezina profila. Međutim, algoritam treba prepoznati je li na danoj slici Alisa ili Bob, koji su jedine dvije klase koje algoritam trenira. Algoritam za Boba kaže da je on Alisa sa sigurnošću od 28%, što nije velik postotak, ali algoritam svakako osobu nije dobro klasificirao. No, umjesto da algoritam čeka na ispravljački input čovjeka pri njegovu treniranju, on provjerava labelu koja je slici dodijeljena te nastoji ispraviti pogrešku mijenjanjem težina koje je postavio nad vezama neurona.

U radu s CNN-om u implementaciji koristi se nadzirano učenje. Za segmentaciju instanci, potrebno je koristiti inpute koji sadrže labele jer labele prepoznaju instance, odnosno omogućavaju brojanje instanci, a ne samo označavanje semantičkih objekata [26, str 2.].

4.1.3.2. Nenadzirano učenje

Nenadzirano nema oznake pri učenju i uspjeh se određuje s time može li mreža povećati ili smanjiti asociranu funkciju troška [52]. Umjesto provjere labeli, algoritam kreira funkcije preslikavanja u tzv. klastere gdje su podaci u klasteru sličniji od onih u drugom klasteru [33]. To znači da se iz podataka izvlače određena svojstva, oni se grupiraju prema sličnosti i izdvajaju. Nenadzirani algoritmi često nemaju definirane klastere ili su oni labavo definirani, a rad često započinju pogađanjem početnog združivanja primjeraka podataka i podešavaju funkcije po potrebi [33].

Kao primjer se može spomenuti segmentacija odjevnih predmeta prema kupcima. Al-

goritam nenadziranog učenja početno raspodjeli različite odjevne predmete u klastere te ih nastoji grupirati prema sličnosti. To može činiti prema primjerice godišnjem dobu u kojem se odjevni predmeti najčešće kupuju, prema dobi za koju je odjevni predmet namijenjen ili prema namijenjenom rodu. Još je jedan primjer segmentacija videa na multimedijskim stranicama, gdje se korisniku preporučuju videozapisi koji su segmentirani prema njegovom uobičajenom ukusu. Mogu se dijeliti u klastere prema duljini videa, količini pogleda, temi koju video obrađuje i slično.

4.1.3.3. Učenje nagrađivanjem

Učenje nagrađivanjem (engl. *reinforcement learning*) oblik je strojnog učenja koji je najprikladniji za zadatke poput upravljanja robotima ili igranja igara [33]. Agent uči politiku ponašanja kako bi maksimizirao nagrade iz okruženja, a to radi tako da poduzima akcije temeljene na trenutnim opažanjima i vlastitom unutarnjem stanju [33]. To radi na licu mjesta tako da pokušava raditi različite akcije, počevši s nasumičnima, i postepeno svoje akcije ažurira ovisno o nagradama. Ako je nagrada pozitivna, povećava se vjerojatnost da će agent koristiti određenu vrstu akcije, dok se pri negativnim bodovima slabi značaj akcije [33].

Primjer ove vrste učenja bilo bi učenje računala kako igrati neku videoigru poput Zmijica (engl. *Snake*). Zmijom se upravlja dok ona hvata hranu, pri čemu raste u dužini te se zmija ne smije sudariti sa zidovima igrališta ili repom. Definira se sustav nagrađivanja za računalo. Pozitivni se bodovi daju ako zmija pronađe hranu i ako igrač preživljava određeni broj koraka u igri, dok se oduzimaju bodovi ako se zmija sudari sa zidom. Računalo se isprva kreće nasumičnim kretnjama (lijevo, desno, gore i dolje) te ažurira svoje postupke, odnosno mijenja vjerojatnosti što će učiniti u kojem trenutku, prema pozitivnim ili negativnim bodovima koje skuplja u igri. Svaka iduća iteracija igre trebala bi dovoditi do boljeg igranja igre.

4.1.4. Konvolucijske neuronske mreže

Konvolucijske su neuronske mreže (CNN) mreže dubokog učenja u računalnom vidu koje su specijalizirane za prepoznavanje značajka slika [54]. Digitalna je slika pritom skup piksela u vrijednostima od 0 do 255 koji su formirani u matricu, pri čemu vrijednosti piksela označavaju svjetlinu i boju (engl. *hue*) [54]. U povijesti je razvoja računalnog vida spomenut eksperiment s mačkama Hubela i Wiesela 1959. godine koji donosi određene bitne zaključke. U eksperimentu se uspostavlja da se neki dijelovi promatranog objekta prepoznaju ranije od drugih, kao što su primjerice ravne linije i jednostavni oblici, dok se složenije karakteristike uočavaju kasnije. Nadalje, dijelovi su mozga specijalizirani za prepoznavanje različitih oblika. Tako neki dijelovi mozga prepoznaju samo horizontalne, a neki samo vertikalne linije i slično [31] [48] . Ove ideje znatno utječu na razvoj CNN arhitekture za duboko učenje u vizualnim inputima [48].

CNN mreže analogne su ANN-u u tome da su sastavljene od međusobno povezanih neurona koji se samooptimiziraju tijekom procesa učenja [52]. Glavna je razlika između ANN i CNN da se CNN primarno koristi u prepoznavanju uzoraka unutar slika [52] zbog postizanja

odličnih rezultata u njihovoj digitalnoj obradi [26]. Njihova je struktura prilagođena specifičnim značajkama podataka iz tog problemskog područja kako bi se smanjilo vrijeme treniranja i kako bi se poboljšala točnost generiranog modela [33].

ANN-ovi koriste manje dimenzije [52]. Za sliku koja je veličine 64X64, te je u boji, broj težina samo za jedan neuron na prvom sloju bitno se povećava do 12288 [52]. Problemi koji se ovdje javljaju su limitirana računalna snaga i vrijeme treniranja modela [52]. Što je veći broj neurona prisutan u modelu, vrijeme treniranja modela je dulje te je potrebno više računalne snage. Nadalje, veći broj parametara treniranja povlači veću vjerojatnost prenaučenosti (engl. *overfitting*) [52], što se događa zbog toga što se model iscrpno trenira na danim inputima, izvlačući što više informacija iz njih, a kasnije ih ne može generalizirati na nove informacije u testiranju [55].

Kako bi se izbjegli ovi problemi, CNN se organizira u 3 dimenzije: visinu, širinu i dubinu [52]. Prema tome, input u algoritam je:

$$h * w * c,$$

gdje je h visina, w širina, a c dubina boje slike, dok je output:

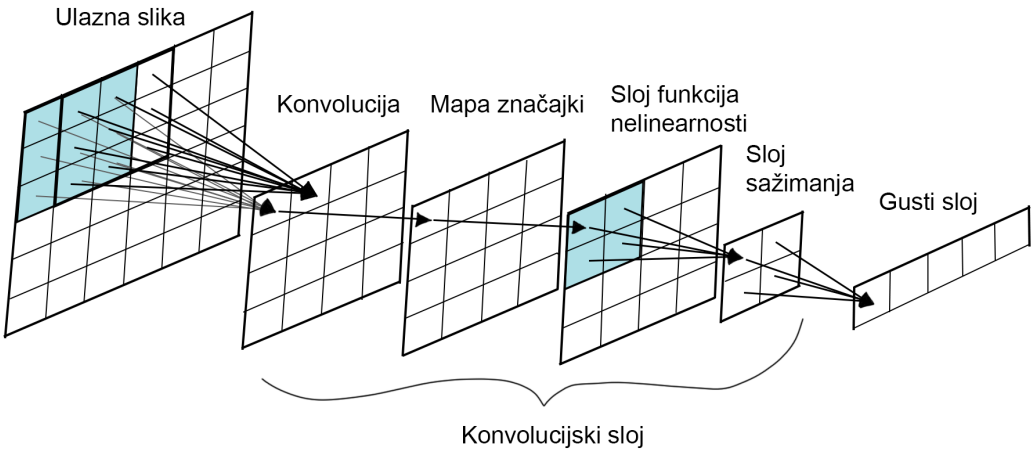
$$1 * 1 * n,$$

gdje je n jednak broj mogućih klasa pri klasificiranju objekata [52]. To ujedno znači da je ulaz u CNN algoritam tenzor trećeg reda, dok je izlaz iz algoritma vektor duljine n s postocima predviđanja n klasa.

Osnovni je cilj stoga stvaranje mreže u kojoj neuroni u ranim slojevima izvlače osnovne lokalne vizualne značajke dane slike kao što su krivulje i linije, a oni neuroni u kasnijem dijelu mreže, kao i kod ljudskog mozga, kombiniraju ove značajke u složenije [33] [54]. Lokalna je vizualna značajka pritom značajka čiji je opseg ograničen na samo mali segment slike, u skupu susjednih grafičkih točaka [33]. Prepoznavanje će se objekata stoga svoditi na to da raniji neuroni prepoznaju jednostavne linije, a fokus je kasnijih prepoznavanje kompliciranijih oblika načinjenih od kombinaciji jednostavnijih. Ako je zadatak algoritma segmentirati ljude na slici, prvo će prepoznati ravne linije koje čine udovi, glavu i osnovne crte lica, kako bi čovjeka razlikovao od okoline, a nakon toga će algoritam prijeći na preradu detalja kako bi mogao segmentirati osobu.

Kako bi se stvorila takva mreža, potrebno je kreirati takvu funkciju otkrivanja značajki koja na jasan način može prepoznati postoji li željena lokalna vizualna značajka na slici ili ne, bez obzira na kojem se ona mjestu nalazi [33]. Kako se ne bi mijenjala s obzirom na translaciju, potrebno je kreirati mrežu za otkrivanje invarijantnih značajki gdje neuroni [33]:

- međusobno dijele težine,
- svaki neuron istražuje drugi dio slike i



Slika 7: Prikaz strukture konvolucijske mreže; izrađeno prema [33, str. 168]

- repectivna područja svih neurona zajedno pokrivaju cijelu sliku.

CNN je sastavljen od višeslojne strukture od koje svaki sloj ima samostalan zadatak čiji se rezultat predaje sljedećem sloju na obradu. Sastoji se od tri različitih sloja [33] [52] [56]:

- konvolucijski slojevi (engl. *convolution layers*)
- slojevi združivanja (engl. *pooling layers*)
- potpuno povezani slojevi (engl. *fully-connected layers*)

Slika 7 prikazuje korake procesuiranja podataka u konvolucijskim neuronskim mrežama.

S lijeve je strane prikazan input u CNN. Input sadrži vrijednosti piksela slike [52]. Na slici su na inputu označena dva receptivna područja veličine $3 * 3$, pri čemu se dijelovi područja preklapaju. Hiperparametar koji upravlja veličinom prostora koji se preklapa naziva se duljina koraka [33]. Ova su područja dijelovi slike, odnosno inputa, što znači da sadrže vrijednosti piksela slike. Veće vrijednosti duljine koraka označavaju manje preklapanje između susjednih područja [33].

Prvi sloj koji obrađuje ovaj input naziva se konvolucijski sloj. Na slici ga reprezentira matrica pokraj ulaza u CNN. On predstavlja sloj neurona koji pretražuje cijelu sliku i traži lokalne značajke, pri čemu svaki neuron ima različito receptivno područje, odnosno promatra drugi dio slike, i koji na svoje ulaze primjenjuju istu matricu težina [33]. Ta se matrica težina naziva jezgra (kernel) [33]. Nju se može zamisliti kao filter koji se slijedno pomiče od početka slike (matrice) do njezina kraja. Kako se kreće, provjerava postoji li preklapanje, odnosno pogodak u vrijednosti matrice koja bi odgovarala pronađenom objektu ili dijelom objekta [48]. Ako postoji preklapanje, ono se označava, te se generira dvodimenzionalna reprezentacija slike koja pokazuje kernelov odgovor na svaku spacijalnu poziciju u slici [52] [54]. Rezultat se naziva mapa značajki, koja u sebi sadrži pozicije uočenih značajki na slici [33] [48].

Konvolucijski je sloj sastavljen od takvih filtera koji reprezentiraju kombinacije veza koje se repliciraju preko cijelog inputa, a neuron se u mapi značajki aktivira samo ako filter koji doprinosi njegovoj aktivnosti detektira prikladnu značajku na potrebnoj lokaciji u prošlom sloju [48].

Funkcija se mape značajke može prikazati na sljedeći način [48]:

$$m^k ij = f((W * x)ij + b^k),$$

gdje je m^k k-ta mapa značajki u sloju m, W označuje vrijednosti težina, a b^k pristranost (engl. *bias*) koja je jednaka za sve neurone mape značajki. Funkcija slijednog pomicanja kernela uz množenje težina odgovarajućih neurona ulaznim vrijednostima grafičkih točaka i zbrajanje takvih umnožaka naziva se konvolucijom [33].

Za ispitivanje postojanja značajki i provjeru postoji li neka značajka u prostoru ili ne, koriste se aktivacijske funkcije. Output filtra kernela daje se matematičkoj funkciji koja se zove aktivacijska. Najčešća aktivacijska funkcija za ekstrakciju značajki u CNN-u zove se ReLu (*rectified linear unit*) [54], koja primjenjuje neku nelinearnu aktivacijsku funkciju poput sigmoide ili tangensa hiperboličnog na output aktivacije koji proizvodi prijašnji sloj [52].

Prolazeći kroz input, za svaku se vrijednost kernela računa skalarni produkt i mreža tako uči prepoznati jezgre koje se aktiviraju prilikom primjećivanja neke specifične značajke. Taj se proces naziva aktivacija [52]. Na kraju procesa, svaka jezgra ima odgovarajuću aktivacijsku mapu koja će se naslagati uz dimenziju dubine kako bi se formirao potpuni volumen outputa iz konvolucijskog sloja [52]. Dubina filtra pritom odgovara vrijednostima dubine inputa kako bi filter mogao kombinirati naučene informacije svih značajki [48].

Bitno je za naglasiti da filtri ne djeluju samo na jednoj mapi značajka, nego na cijelim volumenu mapa [48]. Recimo da sa slike želimo prepoznati mačku. Neka postoje mape značajke za oči, njušku, veličinu, uši, rep, krzno i slično. Mačka se prepoznaje samo ako možemo prikupiti dokaze o tome da postoje sve ostale značajke na prikladnim mjestima slike. Ako se na slici pojavi primjerice pas, koristan model neće označiti psa kao mačku jer nije prepoznao sve potrebne značajke poput repa, veličine i očiju. Dubina volumena outputa sloja ekvivalentna je broju filtera u tom sloju jer svaki filter vraća svoj vlastiti odsječak [48].

Mapa značajki dalje ulazi u *pooling* sloj, koji se još naziva slojem funkcija nelinearnosti. *Pooling* sloj smanjuje veličinu konvolirane značajke na razini dimenzija te je njegova uloga smanjenje potrebne računalne snage za rad s modelom, a tako i ujedno smanjuje vrijeme treniranja modela [52] [54], koje je za model koji obrađuje slike, zbog svojih veličina i kompleksnosti, mnogo dulje od treniranja modela jednodimenzionalnih podataka. Osim toga, pomaže pri ekstrakciji glavnih karakteristika koje su neovisne o poziciji i rotaciji inputa [54]. Sloj se često izvodi primjenom ispravljačke aktivacijske funkcije na svaku stavku mape značajki [33].

CNN često odbacuje informacije o lokaciji kako bi se širila sposobnost mreže da klasificira slike, što se postiže u sloju združivanja [33]. Ulaz je ažurirana mapa značajki [33]. Postoje dvije vrste združivanja. To su srednje združivanje (engl. *average pooling*) i maksimalno združi-

vanje (engl. *max pooling*) [54]. Obje tehnike funkcioniraju na istom principu. Uzima se kernel veličine $n * n$ koji se giba kroz matricu i čita polja matrice na kojima se nalazi. Pri maksimalnom objedinjavanju, kernel bira najveću vrijednost pročitanih vrijednosti i postavlja se u poziciju za output, dok se pri prosječnim objedinjavanju uzima prosjek svih dobivenih vrijednosti [54]. Tako sloj računa i sumira statistiku okružujućih outputa za mijenjanje mreže u određenim dijelovima [54].

Na kraju, pomoću potpuno povezanih slojeva klasificiraju se spljoštene aktivacije (engl. *flattened activations*) [46]. Sloj se nalazi na zadnjem dijelu mreže i dobiva input zadnjeg sloja objedinjavanja ili sloja konvolucijskog outputa koji se spljošti. To znači da se "odmotavaju" sve vrijednosti outputa u vektor [54]. Oni izvode iste zadatke kao i standardni ANN [52].

U implementacijskom primjeru CNN-a u nastavku korišten je MaskCRNN. MaskCRNN je vrsta CNN-a čije se izvođenje sastoji od dva koraka [57]:

- generiranja kandidata za regije interesa (engl. *region of interests*) (ROI) i
- klasifikacije i segmentacije dijelova.

Ovaj algoritam na svaku regiju interesa slike dodaje granu za predviđanje segmentacijske maske paralelno s granom za klasifikaciju i određivanje graničnih okvira [58]. MaskRCNN potreban je zbog toga što je za zadatak, osim prepoznavanja, potrebno i segmentirati željene dijelove slike istovremeno kako bi se precizno označili jedinstveni objekti željene klase.

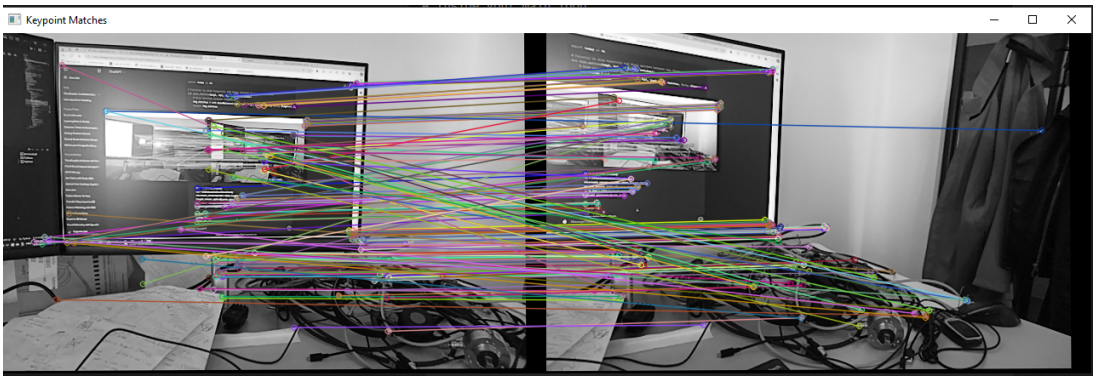
4.2. Percepcijske i navigacijske metode autonomnih uređaja

U ovom se dijelu rada opisuju percepcijske i navigacijske metode autonomnih uređaja vizualna odometrija i SLAM za kasniji prikaz njihove funkcije u implementacijskom dijelu rada.

4.2.1. Vizualna odometrija

Vizualna je odometrija (VO) proces za procjenu vlastita kretanja (egomocije) nekog agenta, kao što je autonomni automobil ili robot, koji koristi input samo jedne ili više kamera povezano s uređajem za određivanje njegove putanje [27] [59]. Ona se konkretno koristi za procjenu pozicije kamere koja se poboljšava uz dodatne optimizacijske tehnike [60]. Vizualna se odometrija koristi tako da procjenjuje položaj autonomnog uređaja inkrementalno, uz ispitivanje promjena koje potakne kretanje kamere [59] [61].

Problem vraćanja relativne pozicije kamere od slika kamere u 3D prostoru se u računalnom vidu zove struktura iz kretnje (engl. *structure from motion*) (SfM), koja je nastala 1990-ih godina prošlog stoljeća [27]. VO zapravo je podvrsta SfM koja određuje položaj kamere preko sekvencijalnih slika inputa, dok se struktura preko kretnje može izvoditi na sortiranim ili nesortiranim slikama [27].



Slika 8: Prikaz podudaranja ključnih točaka

Kako bi vizualna odometrija mogla biti efektivna, potrebno je imati dovoljno osvjetljenja u okruženju te scena treba biti statička uz dovoljno izdvojivih značajki kako bi program mogao procijeniti položaj kamere, odnosno uređaja [27] [59]. Faktori koji utječu na VO su primjerice neravan teren, direktni prodor sunca, sjena i dinamične promjene okruženja kao puhanje vjetra [59]. Bez ovih uvjeta postoji problem prepoznavanja ključnih točaka iz dobivenih slika. Pri snimanja s kamerom, ovo je problem jer kamera uzastopno hvata slike pri kretanju. Ako postoji snažan prodor svjetlosti, algoritam vizualne odometrije ne može hvatati značajnu količinu ključnih točaka ili prouzroči dezorijentaciju zbog krivih pretpostavka. Zbog toga se ona najčešće izvodi u kontroliranim uvjetima, kao što je u zatvorenim prostorijama ili dobrim atmosferskim uvjetima [59].

Vizualna je odometrija povoljna i alternativna vrsta odometrije koja je točnija od tehnika poput GPS, INS, odometrije kotača i sonarnih lokalizacija [27]. Za određivanje pozicije autonomnog uređaja postoje tri različita pristupa [59]:

- pristup temeljen na značajkama,
- pristup temeljen na izgledu i
- pristup hibridnih značajki.

U radu se koristi pristup temeljen na značajkama. Ovaj se pristup bazira na tome da je u svakoj slici, odnosno u svakom okviru kamere, moguće pronaći i izvaditi značajke kao što su linije i krivulje. Uzimaju se dva susjedna okvira kamere te se značajke uspoređuju i prate, putem čega je moguće procijeniti kretanje uređaja [59]. Značajke se nazivaju točke interesa, odnosno ključne točke. Te se točke interesa mogu pratiti uz pomoć vektora koji opisuju lokalnu regiju oko ključnih točaka [60]. Na slici u nastavku se za primjer prikazuju ključne točke i njihovo podudaranje između dva različita kadra. Za izradu slike korištena je Luxonis stereo kamera.

Tradicionalne strategije prepoznavanja rubova i kutova bile su Moravec i Harris detektori, dok se danas za prepoznavanje kuteva koriste tehnike poput SIFT, SURF, ORB i BRISK-a [60]. U radu se za prepoznavanje kuteva koristi SIFT tehnika, koji sadrži 4 koraka [62] [63]:

- detekcija ekstrema u skaliranom prostoru putem funkcije Gaussovih razlika,

- detekcija ključnih točaka uz eliminaciju nevažnih točaka,
- dodjela orijentacije svakoj ključnoj točki i
- stvaranje neovisnih deskriptora koji sadrže informacije o području oko ključne točke.

Algoritam tako generira ključne točke putem k najbližih susjeda i vraća i deskriptore tih točaka [62].

Računanje pozicije omogućuje izračun Euklidske udaljenosti između dviju ključbnih točaka, a pomak se dobiva računanjem vektora brzine između identificiranih parova točaka [59] [63] [64]. Za račun preklapanja značajnih točaka potrebno je računati K najbližih susjeda kako bi se generirali deskriptori značajki [59].

Tehnike koje koriste pristup temeljen na značajkama osim o ranije navedenim značajkama ovise i o teksturi slike za pronalazak značajnih točaka [60]. To znači da ako su u pitanju okružja koja imaju relativno stalnu, odnosno sličnu teksturu, algoritam teže pronalazi značajne točke koje bi mogle odrediti trenutnu poziciju kamere. To mogu biti okruženja poput jednoličnih ureda u kojima je teško pronaći značajne točke zbog relativne sličnosti primjerice zidova, stolova i slično ili pješčana podloga, s pijeskom koji ne bi imao dovoljno razlika u svakom pojedinačnom okviru za određivanje pozicije uređaja. Još je jedan problem zanošenje, pri kojem se pozicija kamere mijenja neovisno o realnom svijetu. To je zbog toga što se u vizualnoj odometriji putanja kamere određuje inkrementalno i greške koje se događaju od okvira do okvira preko vremena se sakupljaju. U tome je slučaju potrebno dodati tzv. *bundle adjustment*, što je lokalna optimizacijska tehnika koja se izvodi preko zadnjih m pozicija kamera [27]. *Bundle adjustment* okvir minimizira projekciju grešaka kroz 3D točke slike i predviđenih točaka koristeći poziciju kamere, intrinzični parametar kamere i parametar distorzije [60].

Shema se vizualne odometrije može implementirati preko različitih vrsti kamera kao što su mono, stereo i RGB-D [60]. U radu se koristi tehnika koja koristi stereo kameru za određivanje dubine slika, što daje precizniji rezultat pozicije.

Algoritmi vizualne odometrije koji su bazirani na značajkama pružaju pouzdane podatke, no skloni su određenom gubitku mogućih informacija. Tehnike bazirane na izgledu omogućuju gustu rekonstrukciju podataka jer u obzir ne uzimaju samo značajke iz prostora, već cijele slike, no skloni su gubicima. Hibridni algoritmi kombiniraju ova dva pristupa [60].

4.2.2. Simultana lokalizacija i mapiranje

Simultana lokalizacija i mapiranje (engl. Simultanous Localization and Mapping) (SLAM) metoda je metoda kojoj je cilj dobiti globalnu i konzistentnu procjenu putanje nekog agenta, što znači da je potrebno konstantno praćenje okruženja [27]. Originalno je predložena za autonomnu kontrolu u robotici [65] [66]. Omogućuje iscrtavanje nepoznatog okruženja dok uređaj prati svoju vlastitu putanju [44]. Nakon inicijalnog predstavljanja, korisnost se SLAM algoritma proširila te se danas koristi za računalni vid, 3D modeliranje, proširenu stvarnost i kod samovozećih automobila [65].

Primjer uređaja koji bi mogao koristiti SLAM algoritam je rumba. Robot se kreće po podlozi gdje mjeri svoju poziciju i kreira putanju kojom se kreće. U tim slučaju, potrebno mu je znati na kojoj je poziciji i kuda se kreće. Ako se kreće nasumičnim putanjama, čišćenje sobe postaje problem jer bi se robot mogao kretati po putanjama na kojima je već bio i tako potencijalno čistiti u nedogled. No, ako uređaj mapira svoje ranije lokacije odnosno pozicije, znati će na kojim je mjestima već bio te se njima nije potrebno ponovno kretati. Vraćanje robota na već posjećeno mjesto zove se zatvaranje petlje (engl. *loop closure*) [27].

Postoje mnoge vrste SLAM algoritama. LIDAR SLAM najpreciznija je SLAM metoda koja primarno koristi laserske ili daljinske senzore, a postoji i multi-senzorski SLAM koji kao input koristi razne uređaje kako bi povećao preciznost i robusnost SLAM algoritma [67]. Još je jedan algoritam vizualni SLAM (VSLAM), koji mapira svoje okruženje samo putem vizualne kamere [68]. VSLAM mogu koristiti obične kamere, stereo kamere i RGB-D kamere za određivanje pozicije samo putem vizualnog inputa [67]. VSLAM je vrlo sličan vizualnoj odometriji jer obje tehnike procjenjuju poziciju senzora. Vizualni SLAM sadrži vizualnu odometriju kao komponentu, no dodaje globalnu optimizaciju mape [65].

5. Prikaz rada tehnologija dubokog učenja i percepcije prostora uz upotrebu AI kamera

U ovom se dijelu rada opisuje i prikazuje implementacija i rezultati implementacije Python programskog koda koji prikazuje konvolucijsku neuronsku mrežu segmentacije instanci i vizualnu odometriju. Fokus je dijela na segmentaciji instanci u stvarnom vremenu zbog prevladavajućeg trenda na području računalnog vida te se pruža teoretski pregled u segmentaciju instanci za bolje razumijevanje implementacije. Za potrebe SICK Mobilisisa također je kreiran program vizualne odometrije.

5.1. Segmentacija instanci u stvarnom vremenu putem RGB kamere

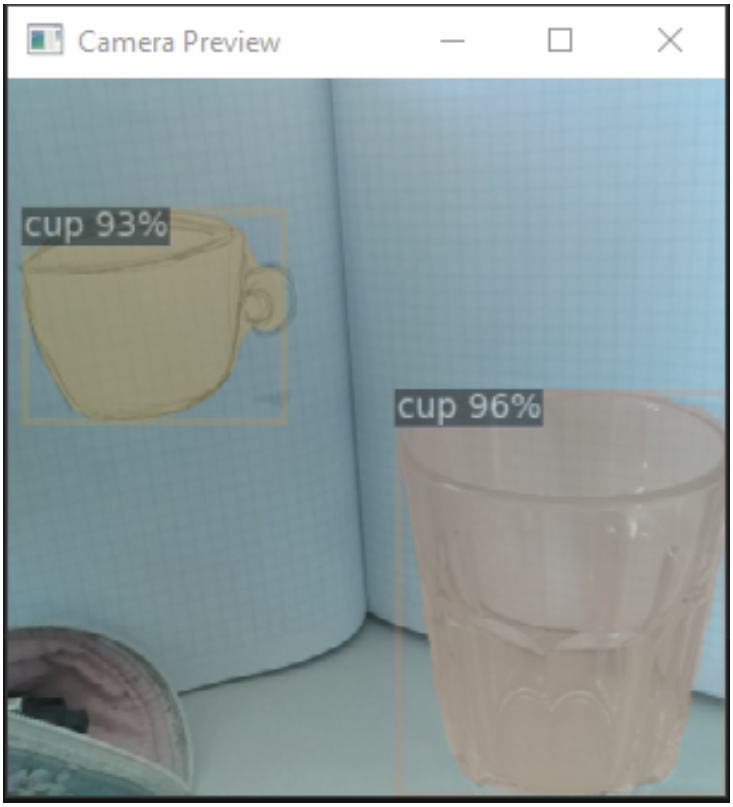
U ovom se dijelu rada opisuje kreiranje Python programa za segmentaciju instanci modela. Za njegovu realizaciju, potrebne su sljedeće biblioteke, čiji je opis generiran uz "pipreqs" biblioteku:

- depthai==2.24.0.0
- detectron2==0.6
- numpy==1.24.3
- opencv_contrib_python==4.9.0.80
- opencv_python==4.9.0.80

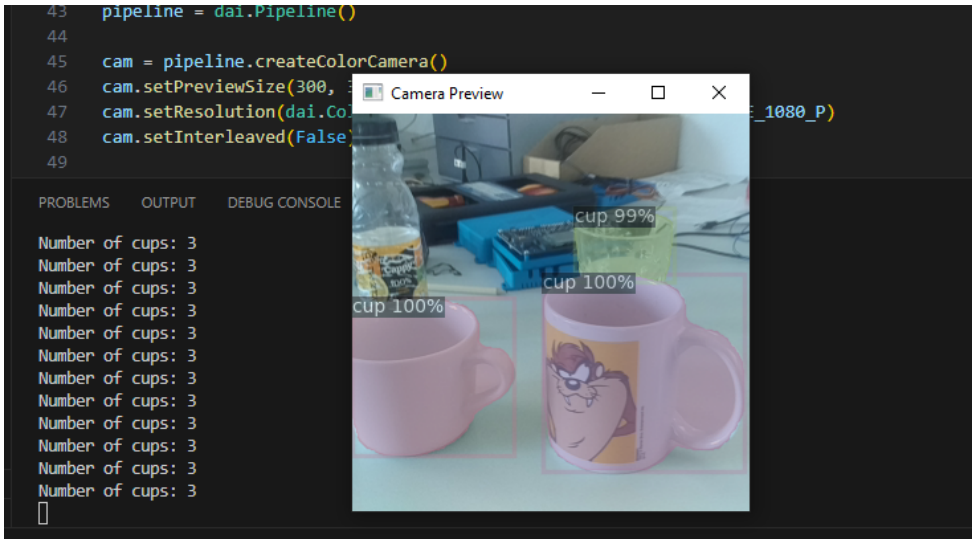
U ovom se dijelu rada kreira model koji raspoznaje TDC-E uređaje za potrebe percepcije okoline i djelovanja na okruženje. Pri tome je potrebno prikupiti slike TDC-E uređaja, označiti rubne točke uređaja za formiranje poligona za masku objekta, spremanje poligona u JSON datoteku te treniranje samog modela.

Kreiran se program izvodi u stvarnom vremenu (engl. *real-time*) i za slike inputa koristi se Luxonis OAK-1 RGB kamera. RGB kamera je kamera koja sadrži standardni CMOS senzor koji omogućuje hvatanje bojanih objekata [69]. Korištena kamera ima sposobnost provođenja umjetne inteligencije, no nije namijenjena za segmentaciju instanci, zbog čega se prepoznavanje odvija na računalu preko Python programa. Prebacivanjem odgovornosti prepoznavanja i segmentiranja objekata na računalo, postiže se ubrzanje kamere, odnosno povećanje broja okvira po sekundi (engl. *frames per second*).

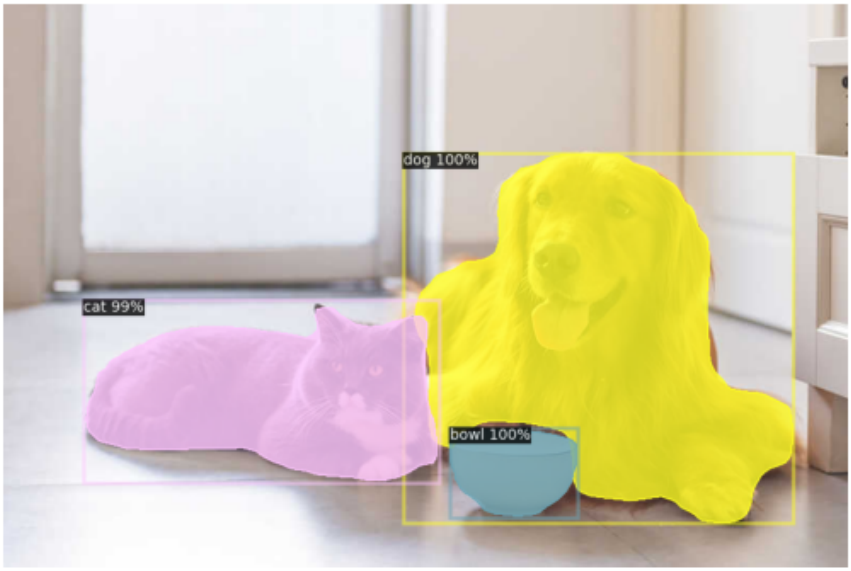
Prije implementacije glavnog rješenja, implementiran je i kod kojim se segmentiraju različiti objekti na slici i preko kamere uz već postojeći Detectron2 model. U nastavku je prikazano nekoliko slika ekrana kreirane aplikacije čiji je kod sličan krajnjem proizvodu ovog dijela implementacije, stoga se ne obrađuje zasebno.



Slika 9: Vizualizacija instanci segmentiranih šalica u stvarnom vremenu



Slika 10: Segmentiranje i brojanje instanci šalica u stvarnom vremenu



Slika 11: Primjer segmentacije na slici bez naznačenih kategorija; preuzeto iz [70]

Pritom je u kodu specificira tip instance koja se želi segmentirati. Pokretanjem modela nad slikama za segmentaciju bez specifikacije rednog broja objekta koji se želi segmentirati, program će segmentirati svaku klasu koju prepoznaje unutar slike.

5.1.1. Uvod u segmentaciju instanci

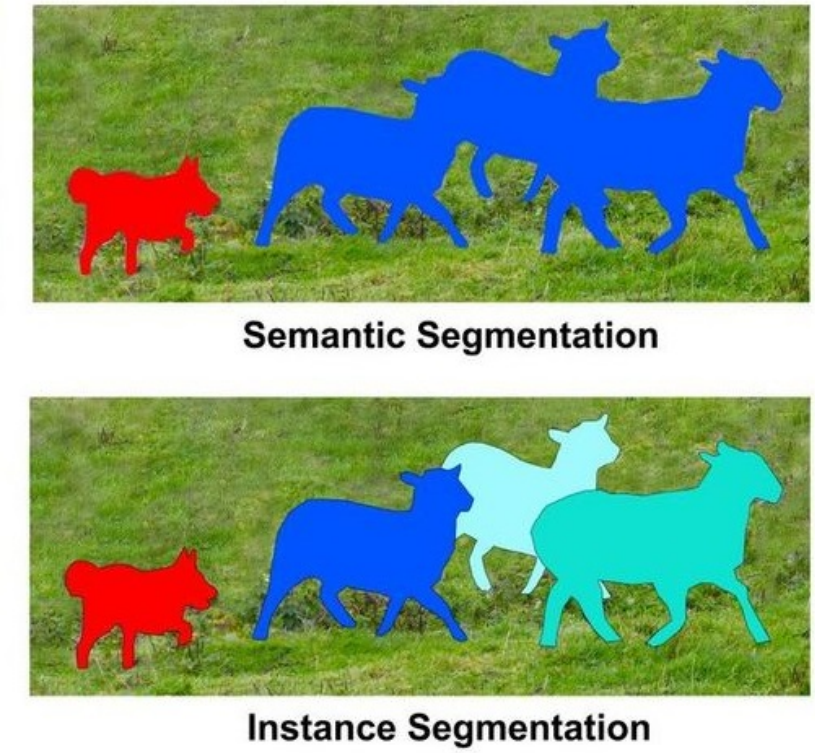
Zbog novije i kompleksnije metode segmentiranja instanci te zbog potrebe za označavanjem svakog piksela prepoznatog objekta za točniju percepciju objekata u prostoru, za implementaciju konvolucijske neuronske mreže u Pythonu koristi se segmentacija instanci.

Označavanje semantičkih objekata u sceni traži da se svaki piksel slike imenuje nekom kategorijom, poput čaše, ptice, mačke, neba i slično [22] [71]. U usporedbi s detekcijom daje preciznije rezultate ekstrakcije podataka odnosno informacija iz scene [71]. Detekcija odnosno prepoznavanje objekata nad objektom kreira granični okvir, koji prikazuje granične točke objekta koji se prepoznaje. Segmentacija pruža preciznije označavanje objekta jer označava svaki zasebni piksel, a ne samo rubne točke danog objekta.

Tehnike se segmentacije dubokog učenja dijele u dvije kategorije [26]:

- semantička segmentacija (engl. *semantic segmentation*)
- segmentacija instanci (engl. *instance segmentation*)

U semantičkoj segmentaciji, prepoznaje se i označava samo klasa objekata, dok se u



Slika 12: Razlika semantičke segmentacije i segmentacije instanci; preuzeto iz [72]

segmentaciji instanci označava svaka pojedinačna instanca [26]. Segmentacija instanci objekta traži razlikovanje drukčijih instanci iste objektne klase [21]; svaki se piksel asocira s posebnom fizičkom instancom objekta [71]. Drugim riječima, semantička segmentacija ne uzima u obzir individualne instance i veći broj objekata iste klase označava kao jedan identitet, dok segmentacija instanci raspoznaje svaki zaseban objekt kao svoj vlastiti identitet. Slika u nastavku predstavlja ovu razliku vizualno. Uz razvoj se CNN-a interes za segmentaciju povećao [56].

Semantička segmentacija daje klasifikaciju cijele scene s dijelovima informacija o kategoriji, lokalizaciji i obliku objekata scene [73]. Funkcionira tako da se identificiraju i grupiraju pikseli koji su semantičko pripadni, odnosno iste su klase [74]. Cilj je semantičke segmentacije dodjela svakog piksela u slici nekoj klasi [21] [56] [58].

Ograničenje semantičke segmentacije je nerazlikovanje instanci unutar iste klase, odnosno ne razdvajaju se objekti, tako da nije moguće dobiti točan broj objekata unutar slike [26]. Ovaj problem moguće je riješiti uz segmentaciju instanci, koja dopušta jedinstveno razumijevanje, prebrojavanje i analizu svakog objekta [26].

Algoritmi segmentacije instanci dijele se na dva glavna pristupa [26]:

- isprva segmentacija (engl. *segmentation-first*), gdje se prvo uspostavljaju kandidati segmenata te nakon toga slijedi klasifikacija
- isprva instance (engl. *instance-first*), uz paralelno procesiranje segmentacije i klasifikacije

Algoritmi tipa isprva instance fleksibilniji su i izravni i omogućuje se dobivanje graničnog okvira i segmentacijske maske istovremeno. Jedan od modela ovog tipa instanciranja je *Mask-Region-based Convolutional Neural Network* (Mask R-CNN), koji se koristi u nastavku rada.

Za segmentaciju instanci, kreiranje modela svodi se na poligonsko označavanje individualnih objekata, gdje u proces učenja mreže ulaze i oznake i slike. Oslanja se na lokalizaciju značajki za kreiranje maska [57]. Izazovna je zbog toga što je potrebno ispravno detektirati objekt i istovremeno se traži precizna segmentacija svake instance, odnosno kombinira prepoznavanje objekata sa semantičkom segmentacijom [58].

Dvije najpopularnije procedure za označavanje objekata su *Creating Common Object in Context* (COCO) i *Pascal Visual Object Classes* (VOC) [26]. COCO, koji je korišten u programu, u setu podataka sadrži 91 kategoriju čestih objekata [22]. Točna je i brza segmentacija instanci i danas izazovni problem [75].

Za potrebe projekta SICK Mobilisisa, bitno je označiti svaki piksel objekta, ali i znati točan broj individualnih objekata. Primjer potencijalnog korištenja programa je robotska ruka koja hvata zaseban TDC-E uređaj kako bi ga stavio na stol na kojem nije dovoljan broj uređaja. Program u tom slučaju treba prepoznati broj instanci uređaja na stolu i ako uređaja nema dovoljno, robot uzima konkretno jedan uređaj iz skupine uređaja, pri čemu je označen konkretan, jedinstveni objekt kako bi ga položio na stol.

5.1.2. Kreiranje i treniranje modela za segmentaciju instanci

U ovom se dijelu kreira vlastiti model koji prepoznaje instance TDC-E uređaja na temelju segmentacije instanci i CNN-a. Da bi model bio koristan, on mora biti u suglasnosti s pravim svijetom [33].

5.1.2.1. Prikupljanje podataka za treniranje modela

Za početak je kreiranja modela potrebno prikupiti slike koje će poslužiti za treniranje modela. Za konvolucijske neuronske mreže (CNN), bitno je odabrati dobre reprezentante objekata koji će postati instance modela. To znači odabrati slike koje željeni objekt prikazuju jasno, uz različite udaljenosti, različite kutove objekta i različite položaje u kojima se objekt može naći. To je zato što su slike dvodimenzionalne, dok je objekt trodimenzionalan. Za određivanje veličine i za njegovo točno segmentiranje iz bilo kojeg kuta, model je potrebno trenirati takvim slikama koje će omogućiti potpunije razumijevanje objekta. Primjerice, objekt se fotografira uz druge, nepovezane objekte, kako model ne bi naučio samo prepoznati što objekt je, već i što on nije. Kreiranje većeg broja klasa unutar modela također pospješuje ovaj cilj. Također je dobro po mogućnosti objekte postaviti u različita okruženja; na primjer, ako se objekt slika samo na stolu, računalo će naučiti da je objekt vezan uz stol te će model teže prepoznavati objekt u drugom prostoru. To je zato što se prepoznavanje objekata često oslanja na kontekstualne informacije, tako da je bitno da skupovi podataka sadrže objekte u prirodnom okolišu [22].

Što je u modelu više podataka, modeli imaju veću količinu informacija koje mogu izvući

Slika 13: Primjer slike koja ulazi u treniranje modela segmentacije instanci; preuzeto iz [72]

iz konteksta i algoritam učenja prepozna i mapira veći broj značajki iz danih matrica. Osim većeg broja slika, pomaže i veći broj instanci željenih objekata u slikama. Stoga se fotografira što veći broj slika na kojim se nalaze TDC-E uređaj koji će poslužiti kao klasa s kojom će model trebati povezati piksele u slici. Također se fotografiraju slike na kojima je označen veći broj TDC-E uređaja posloženih na različite načine, kao primjerice jedan na drugome, u svrhu kasnijeg boljeg prepoznavanja uređaja.

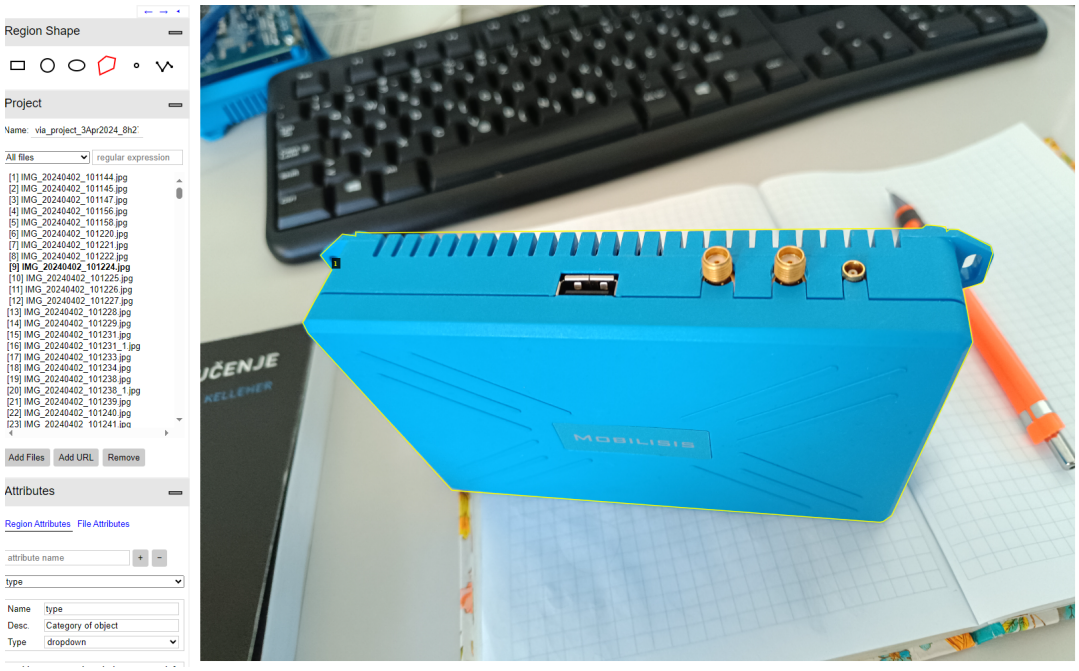
Ukupno u treniranje i validiranje modela ulazi 504 slika, od kojih se 52 slike odvajaju za skup validacije, dok se ostatak postavlja u direktorij za treniranje modela. Fotografije u pravilu sadrže veći broj TDC-E uređaja u slikama te je ukupno označeno 729 instanci TDC-E uređaja. To se i ispisuje na početku treniranja modela, što je parametar koji je zapisan u JSON datoteci čije se kreiranje opisuje u nastavku.

Kreira se mapa "img", unutar kojih se kreiraju direktoriji "val" i "train". Ovo su direktoriji u koje se postavljaju slike. Nakon što se fotografije podjele u potrebne direktorije treniranja i validacije, pri čemu se slike biraju nasumično, potrebno je kreirati dokument koji će služiti za označavanje čitave maske objekta.

5.1.2.2. Označavanje rubnih točaka objekata

U ovom dijelu kreira se JSON datoteka u kojoj su naznačene pozicije rubnih točaka poligona koji se crtaju nad slikama. Za samo crtanje poligona odabran je VGG Image Annotator [7] [8].

Kako bi se kreirala kategorija instanci, označava se kartica "type" i pod "Attributes" dodaje se novi tip koji se naziva "tdce". U aplikaciju se postavljaju posebno trening set i posebno validacijski set, te se slike pojedinačno označavaju. Na slici u nastavku moguće je vidjeti način označavanja objekata sa slike. TDC-E uređaj označen je žutom rubnom crtom, koja je sastavlja



Slika 14: Označavanje poligona za masku instance objekta TDC-E

od točaka koje se ručno postavljaju na svaki rub poligona uređaja. Krajnji je rezultat poligon nad objektom čije se koordinate bilježe u JSON datoteku.

Na kraju je potrebno spremiti JSON datoteku koja sadrži pojedinosti i koordinate točaka postavljene tijekom označavanja. Proces je potrebno odraditi posebno za set treniranja i posebno za set validacije. JSON se datoteke spremaju pod istim nazivom, no svaka u direktorij u koji pripada. Nastaju dvije JSON datoteke, od kojih svaka zasebno sadrži podatke označavanja o svakoj slici seta podataka. U nastavku se nalazi dio JSON strukture za set validacije. Konkretno su prikazani podaci za samo jednu sliku u setu, no datoteka je sastavljena od skupa parametara svake slike.

Nakon završenog procesa označavanja svakog TDC-E objekta, prelazi se na Python kod za treniranje modela segmentacije instanci.

5.1.2.3. Treniranje modela u Pythonu

Za početak, postavljaju se globalne varijable "OUTPUT_DIR", koja sugerira na direktorij u koji će se novokreirani model spremiti, te "IMG_FOLDER", koji je krovni direktorij za ranije spomenute direktorije treniranja "train" i validacije "val". Model segmentacije instanci trenira se uz Detectron2 biblioteku, koja traži provjeru je li ime funkcije main i prebacivanje u novu funkciju.

Prije treniranja modela i validiranja modela, model se treba registrirati. To se radi uz kodni isječak 2. Kod prolazi kroz svaku datoteku u direktorijima za treniranje i validaciju te se spremaju skupovi u DatasetCatalog i MetadataCatalog. Ovaj korak je obavezan pri treniranju modela uz Detectron2.

Tada se postavlja konfiguracija treniranja modela, koja je najbitniji dio za određivanje

Slika 15: Označavanje poligona za masku instance objekta TDC-E s dva objekta u slici

```
{"IMG_20240402_101200.jpg2619966":{"filename":"IMG_20240402_101200.jpg",
"size":2619966,
"regions":[{"shape_attributes":{"name":"polygon",
"all_points_x":[616,544,529,905,930,961,2905,2939,2958,3648,
3651,3574,3626,3629,3533,3465,3368,812,684,526,575],
"all_points_y":[1969,1798,1757,905,871,861,756,750,759,1499,
1521,1735,1813,1844,1941,1962,1962,2224,2227,2149,2006]},
"region_attributes":
    {"name":"not_defined",
    "type":"TDC-E",
    "image_quality":{
        "good":true,
        "frontal":true,
        "good_illumination":true}}}],
        "file_attributes"
    {"caption":"",
    "public_domain":"no",
    "image_url":""}
},
```

Isječak koda 1: Primjer JSON objekta označene slike

```
def do_training():
    for d in ["train", "val"]:
        DatasetCatalog.register("tdce_" + d, lambda d=d:
        ↪  get_dicts(os.path.join(IMG_FOLDER, d)))
        MetadataCatalog.get("tdce_" + d).set(thing_classes=["tdce"])
```

Isječak koda 2: Registracija novog seta podataka

```
cfg = get_cfg()
cfg.OUTPUT_DIR = OUTPUT_DIR
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/
mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("tdce_train",)
cfg.DATASETS.TEST = ()
cfg.MODEL.WEIGHTS =
↪  model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.DATALOADER.NUM_WORKERS = 4
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.WEIGHT_DECAY = 0.0001
cfg.SOLVER.MAX_ITER = 1000
cfg.SOLVER.WARMUP_ITERS = 100
cfg.SOLVER.WARMUP_FACTOR = 1.0 / 3
cfg.INPUT.RANDOM_FLIP = "horizontal"
cfg.INPUT.RANDOM_FLIP = "vertical"
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
cfg.MODEL.DEVICE = "cpu"
```

Isječak koda 3: Postavljanje konfiguracije za treniranje

hiperparametara modela, odnosno parametara koji postavlja programer i koji se ne mijenjaju pod utjecajem modela.

Konfiguracija se dohvaća u prvoj liniji koda u isječku koda 3. Konfiguracija isprva postavlja output direktorij budućeg modela na direktorij definiran na početku programa. Trećom se linijom koda dohvaća konfiguracijska datoteka unaprijed treniranog Mask R-CNN modela. Postavlja se "TRAIN" parametar koji se odnosi na ranije registrirani dataset za treniranje. Sedma se linija koda odnosi na transferno učenje, gdje se dobivaju težine ranije treniranih modela za bolje i brže učenje.

Dalje se postavlja broj radnika koji paralelno vode proces učenja, stopa učenja, broj klasi, distorzija slike uz pomoć okretanja slike i drugi. Za stopu se učenja smatra da je najbitniji hiperparametar pri treniranju CNN-ova jer o njemu ovisi veličina koraka i gradijent pri propagaciji treniranja modela unatrag, a osim toga je i povezana s drugim parametrima kao normalizacijom ili veličinom serije (engl. *batch size*). Premala stopa učenja dovodi do usporenog učenja, a prevelika dovodi do oscilirajućih performansi [76]. Tijekom kreiranja različitih modela, stopa je učenja hiperparametar koji se često mijenja kako bi se pronašao optimalniji način njegova učenja. Za krajnji odabrani model, model 3, stopa učenja je postavljena na 0.00025 jer je primijećen trend pri treniranju drugih modela da ova stopa učenja daje najbolje rezultate.

Postavlja se i 100 iteracija zagrijavanja kako bi se premostilo početno učenje koje je sklono nepoželjnim rezultatima, što se uključuje jer se u prvih 3 modela primjećuje nestabilnost u učenju koje se prikazuje na kasnijim grafovima performansi modela. Također nije dostupna grafička kartica te se MODEL.DEVICE postavlja na CPU. Garfičke radnje su uz CPU mnogo sporije, tako da su trening modela i njegova primjena također sporiji.

Bitno je za naglasiti kako se većina prikazanih hiperparametara mijenja kako se kreiraju novi modeli. Na kraju je kreirano 6 korisnih modela čiji se rezultati vizualno prikazuju u nastavku.

```
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=True)
trainer.train()

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
cfg.DATASETS.TEST = ("tdce_val", )
```

Isječak koda 4: Početak treniranja modela i postavljanje testnog dataseta

```
[04/04 08:26:51 d2.utils.events]:  eta: 0:28:09  iter: 119  total_loss: 1.156
↪  loss_cls: 0.2511  loss_box_reg: 0.5857  loss_mask: 0.3065  loss_rpn_cls:
↪  0.003839  loss_rpn_loc: 0.005684    time: 5.0869  last_time: 5.7989  data_time:
↪  0.0015  last_data_time: 0.0014   lr: 6.6295e-05
```

Isječak koda 5: Output pri treniranju modela

Kako bi se model spremio, potrebno je postaviti točno ime modela koje se nadovezuje na ime direktorija. Model se i težine učitavaju pri pozivanju programa koji radi s kreiranim modelom. Trening započinje uz liniju "trening.train()", gdje objekt za treniranje pokreće samu funkciju treniranja. U prikazanom kodu za početak treniranja naznačeno je da trener nastavlja ili učitava prijašnju sesiju treniranja. Ako je ovaj dio postavljen na False, trener trening započinje ispočetka. Inače se trening nastavlja na prekidu, odnosno program prepoznaje već istrenirani model i prelazi na sljedeći dio koda.

Treniranje daje output sličan onome u kodnom isječku 5. Ovo je log koji prikazuje korisne informacije o modelu i njegovu treniranju. Prikazuje informacije poput preostalog vremena treniranja, broj iteracije u treningu, greške u treningu i slično. Svaka iteracija treninga ispisuje se u konzolu. Idealno treniranje modela je ono gdje se greške sa svakom iteracijom smanjuju.

Za bolji se vizualni prikaz i za usporedbu s ostalim modelima kasnije koristi TensorFlow.

Na kraju treniranja, uključena je evaluacija rezultata te se kod provjerava čitanjem kreiranog modela, težini modela i nasumičnih slika postavljenih u zasebni registrirani direktorij "tdce_val". Za evaluaciju rezultata koristi se COCO evaluator. U kodu se postavlja konfiguracija testnog seta podataka na naznačeni direktorij.

U zadnjem dijelu programa prelazi se na vizualizaciju rezultata. Iz naznačenog se direktorija nasumično biraju tri slike. Otvara se OpenCV2 vizualizacija koja prikazuje rezultat maskiranja objekata.

```
evaluator = COCOEvaluator("tdce_val", cfg, False, output_dir=cfg.OUTPUT_DIR)
val_loader = trainer.build_test_loader(cfg, "tdce_val")
inference_on_dataset(trainer.model, val_loader, evaluator)

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
cfg.DATASETS.TEST = ("tdce_val", )
```

Isječak koda 6: Evaluacija rezultata

```
predictor = DefaultPredictor(cfg)
dataset_dicts = get_dicts(os.path.join(IMG_FOLDER, "val"))

random_images = random.sample(dataset_dicts, 3)

for d in random_images:
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    tdce_instances = outputs["instances"][outputs["instances"].pred_classes == 0]
    num_tdce_instances = len(tdce_instances)
    print(f"For image {d['file_name']}: {num_tdce_instances}")

    v = Visualizer(im[:, :, ::-1], metadata=MetadataCatalog.get("tdce_val"),
    ↪  scale=0.2, instance_mode=ColorMode.IMAGE_BW)
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2.imshow("Visualization", v.get_image()[:, :, ::-1])
    cv2.waitKey(0)

cv2.destroyAllWindows()
```

Isječak koda 7: Vizualna provjera rezultata

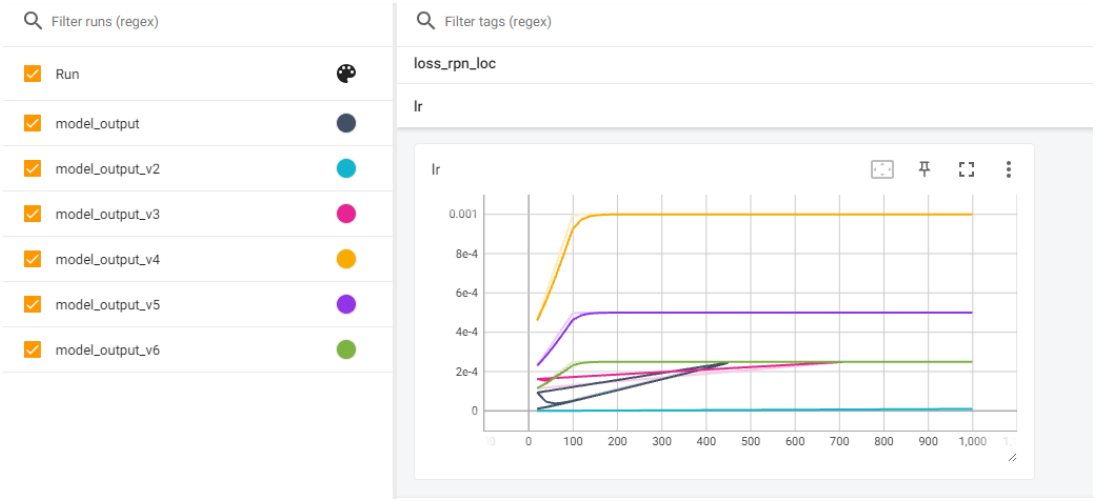
5.1.2.4. Praćenje treniranja modela i njegovi rezultati

Tijekom treniranja se model prati kako bi se izbjeglo neželjeno trošenje resursi ako model ne uči dobro te kako bi se lakše uspostavljale razlike između treniranih modela. Na taj su se način određivali najpovoljniji hiperparametri za treniranje. Hiperparametri koji su se za modele mijenjali su stopa učenja, dodavanje distorzija, dodavanje faze i faktora zagrijavanja, broj iteracija i druge. Tablica u nastavku prikazuje razlike između odabranih modela.

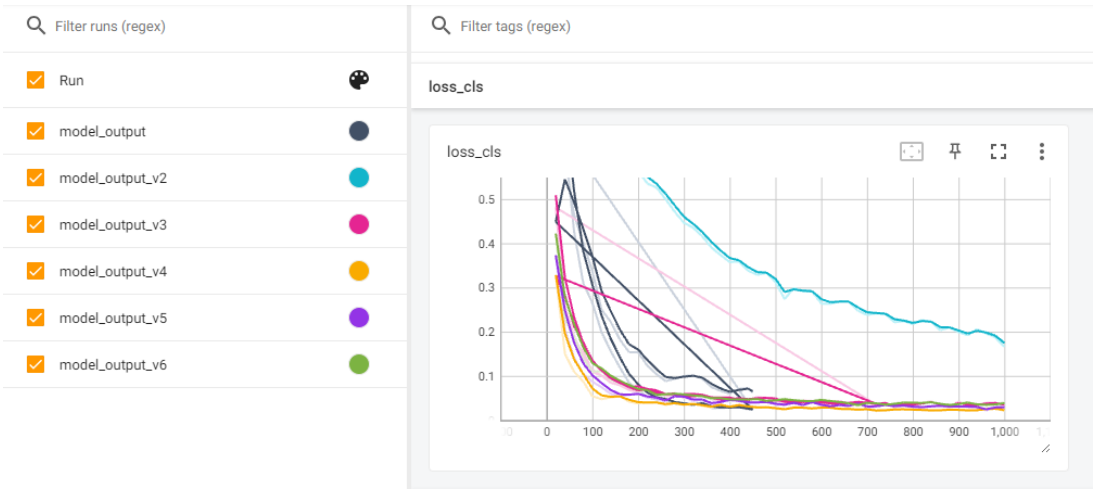
Tablica 2: Usporedba hiperparametara treniranja modela

Stopa učenja	Broj iteracija	Regulacija	Zagrijavanje	Okretaji slika
0.00025	450	-	-	-
0.00001	1000	-	-	-
0.00025	1000	pad težine (0.0001)	100 (1/3)	horizontalni
0.001	1000	pad težine (0.0002)	100 (1/3)	horizontalni
0.0005	1000	pad težine (0.0001)	100 (1/3)	horizontalni + vertikalni
0.00025	1000	pad težine (0.0001)	100 (1/3)	horizontalni + vertikalni

Iz tablice je vidljivo kretanje stope učenja. Tijekom treniranja, primjećeno je da je za konkretan set i za povezane hiperparametre najpovoljnije učenje sa stopom učenja od 0.00025. Sporije ili brže učenje dovodi do manje preciznosti. Svi modeli osim prvog treniraju se na 1000 iteracija, s time da modeli 3,4,5 i 6 provode i period zagrijavanja gdje se stopa učenja spušta za njezinu trećinu. Izvodi se regulacija težine podataka kako bi se sprječilo pretreniranje od 0.0001 ili 0.0002 od 3. modela nadalje. Modeli 3, 4, 5 i 6 također uključuju augmentaciju slika, odnosno nasumične se slike okreću horizontalno ili horizontalno i vertikalno.



Slika 16: Vizualizacija stope učenja Mask R-CNN modela

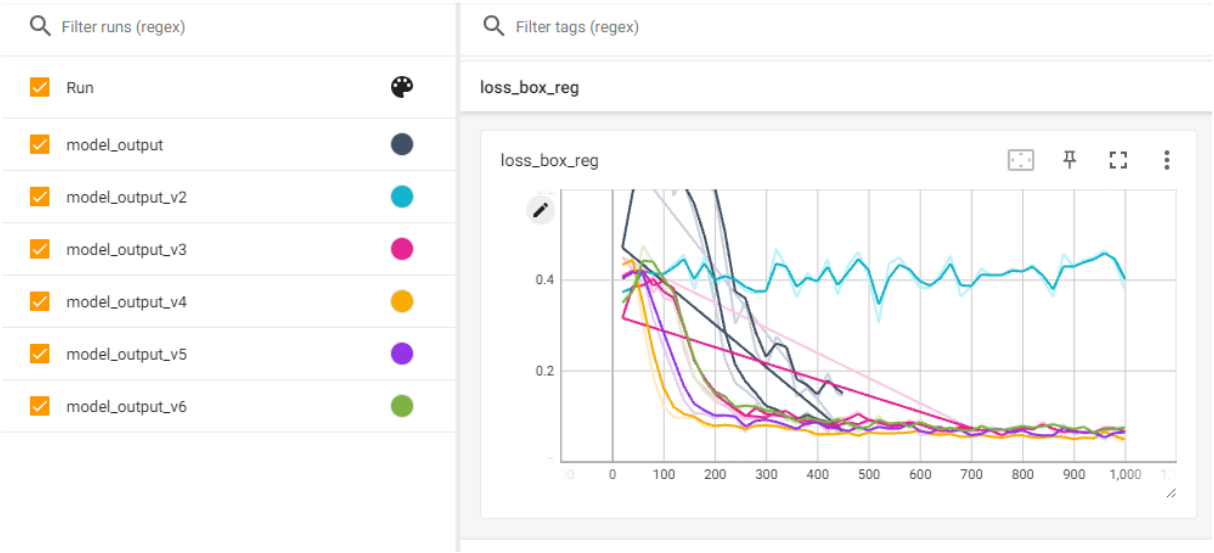


Slika 17: Vizualizacija gubitka klasifikacije Mask R-CNN modela

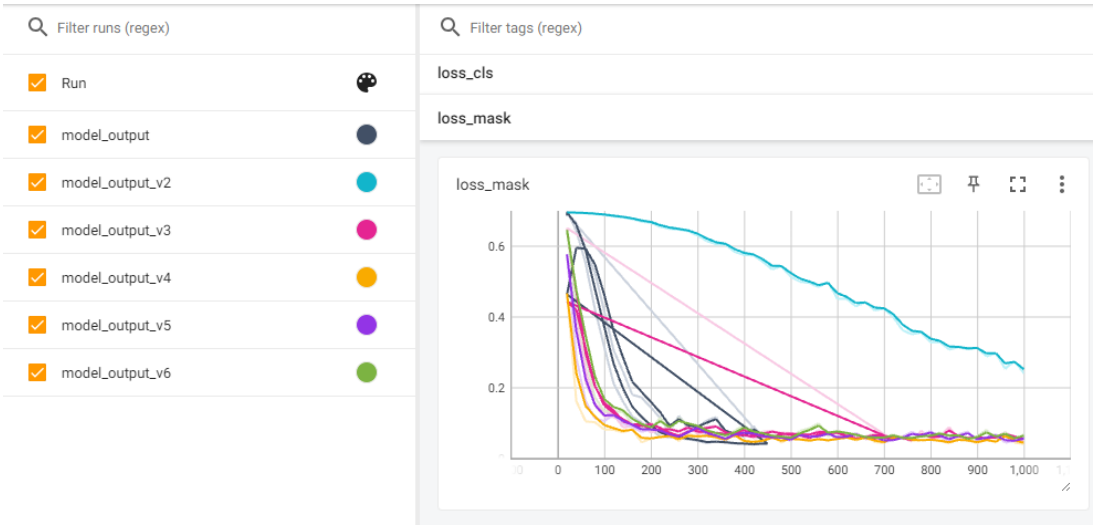
U nastavku su prikazane slike ekrana s Tensorflow-a. One prikazuju tijek treniranja 6 Detectron2 modela uz informaciju kako su napredovali zasebno i u usporedbi s drugim modelima.

Za prvi je, treći i šesti model stopa učenja jednaka 0.00025, za drugi je postavljena 0.00001, četvrta je stopa 0.001 i peta 0.0005, pri čemu modeli od 3 na dalje imaju postavljen period zagrijavanja. Prvi model sadrži samo 450 etapa treniranja, za razliku od ostalih modela koji provode 1000 iteracija treniranja, stoga je crna linija na grafikonu kraća od drugih. Također je treniranje provedeno dva puta te je vidljivo vraćanje linije na početak. Treći model također se trenira dva puta zbog prekida u međuvremenu. Drugi model ima vrlo malenu vrijednost stope učenja te je učio premalo za količinu obrađenih podataka, dok daljnji modeli rade efikasnije. Eskperimentira se s različitim vrijednostima, no 0.00025 je stopa učenja koja pokazuje najefektivnije rezultate na bazi inputa i hiperparametara.

Gubitak se klasifikacije odnosi na gubitak pri procjeni klase objekta i odnosi se na njegovu klasifikaciju [77]. Vidljiv je vrlo malen gubitak klasifikacije pri većini modela. Osim kod modela 2, čiji je gubitak klase ispod 0.2, ostali modeli imaju gubitak klasa manji od 0.05.



Slika 18: Vizualizacija gubitka regresije okvira Mask R-CNN modela



Slika 19: Vizualizacija gubitka maske Mask R-CNN modela

Regresija se okvira odnosi na gubitak graničnih okvira pri treniranju modela i o njemu ovisi točnost okvira oko objekta pri predviđanju objekata [77]. Vidljivo je da većina treniranih modela imaju vrlo malen gubitak na graničnim okvirima, ispod 0.1. Drugi je model izuzetak, čija ga mala stopa učenja i početna preciznost dovodi do toga da je gubitak graničnih okvira na kraju treninga jednak otprilike 0.4.

Gubitak je maske segmentacijski dio treniranja i određuje preciznost maske [77]. Što je manji gubitak, to je maska preciznija. Na grafikonu generirano putem Tensorflowa vidljivo je da je gubitak maske vrlo malen za većinu modela; nalazi se ispod 0.1. Model 2 ukazuje na gubitak maske od 0.25, što je visoka vrijednost.

Mask R-CNN odnosi se na ukupnu preciznost modela i Tensorflow prikazuje tri grafa:

- graf preciznosti (engl. *accuracy*)

Slika 20: Vizualizacija rezultata Mask R-CNN modela

- graf lažnog negativa (engl. *false negative*)
- graf lažnog pozitiva (engl. *false positive*)

Lažni su negativi i pozitivi pojmovi u statistici koji ukazuju na pogrešno određene hipoteze. Hipoteze trebaju predstavljati realni svijet [78]. Ako je hipoteza koju smo postavili istinita te je mi prihvaćamo, ona je istinita pozitiva. Ovo se događa kad model istinito predviđa masku ili objekt ili granični okvir. Ako je hipoteza lažna, a prihvaća se, rezultat je lažan pozitiv. To se događa ako model primjerice umjesto TDC-E uređaja na bilo koji označi mobitel, koji nije dio iste klase. Lažan je negativ kad je hipoteza istinita, no odbijena. Ovo se događa kad na slici postoji TDC-E uređaj, ali prediktor ga ne prepozna te tako i ne označi ni na koji način. U donjem je desnom dijelu matrice istinit negativ. On se događa kad na slici nema TDC-E uređaja te prediktor prepozna da se ne radi o uređaju, odnosno prepozna da ga nema te i ne obavlja označavanje, odnosno aktivacije [78].

Tablica 3: Veza hipoteze i stvarnosti; prema [78]

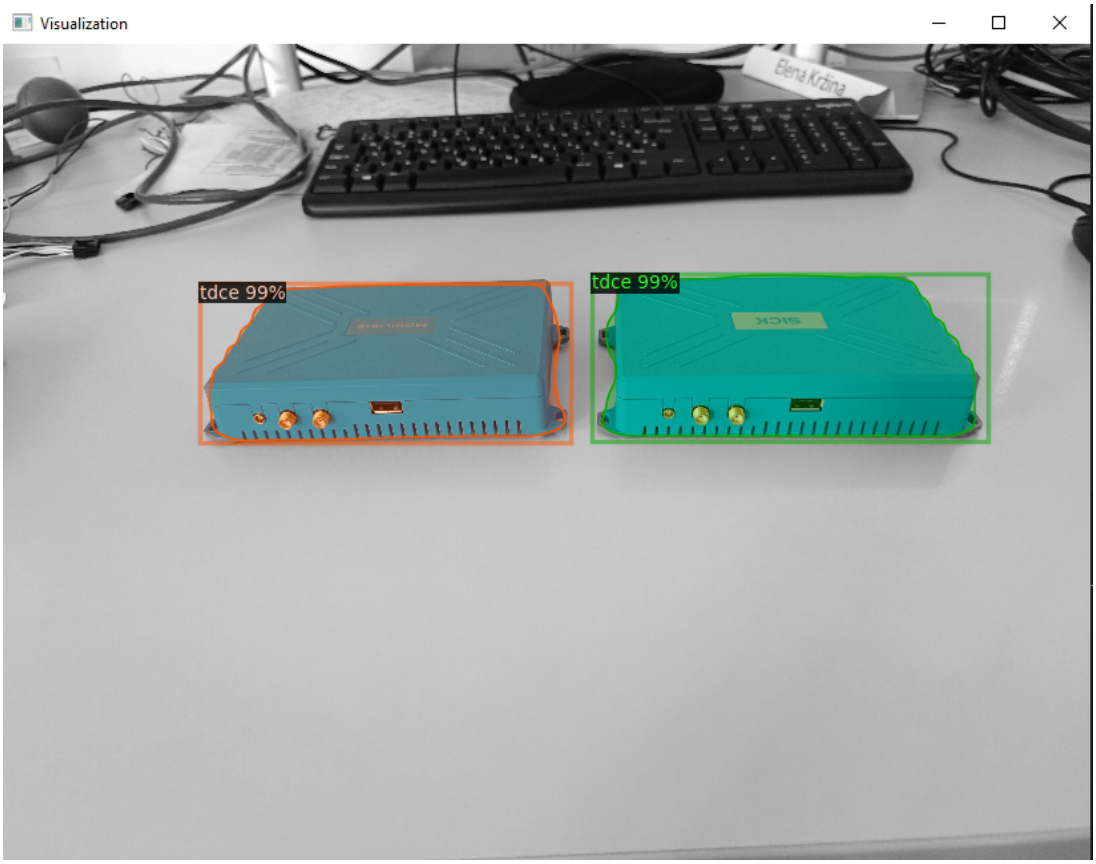
	Istinita hipoteza	Lažna hipoteza
Hipoteza prihvaćena	Istinit pozitiv	Lažan pozitiv
Hipoteza odbijena	Lažan negativ	Istinit negativ

Četiri od šest modela zadovoljavajući su prema vrijednostima iz grafova, pri čemu je prvi model kraće treniran od ostalih te se ne vidi njegov razvoj u kasnijim iteracijama, a drugi je manje precizan od ostalih zbog nedovoljno visoke stope učenja. Za navedenih je 4 modela ukupna preciznost iznad 0.97 prema treniranju. Lažni su negativi na kraju treniranja između 0.017 i 0.022, a lažni pozitivi su oko 0.05.

Najbolji rezultat postižu model broj 3 i broj 6, što se provjerava modelima i evaluacijom. Za daljnji je rad izabran model broj 3 zbog najbolje evaluacije testnih podataka u Python kodu, koja ukupno iznosi 94.699%.

```
[04/08 11:33:17 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|   AP   |  AP50  |  AP75  |  APs  |  APm  |  APl   |
|:------:|:------:|:------:|:-----:|:-----:|:------:|
| 94.699 | 99.816 | 99.816 |  nan  |  nan  | 94.699 |
```

Isječak koda 8: Evaluacija modela 3



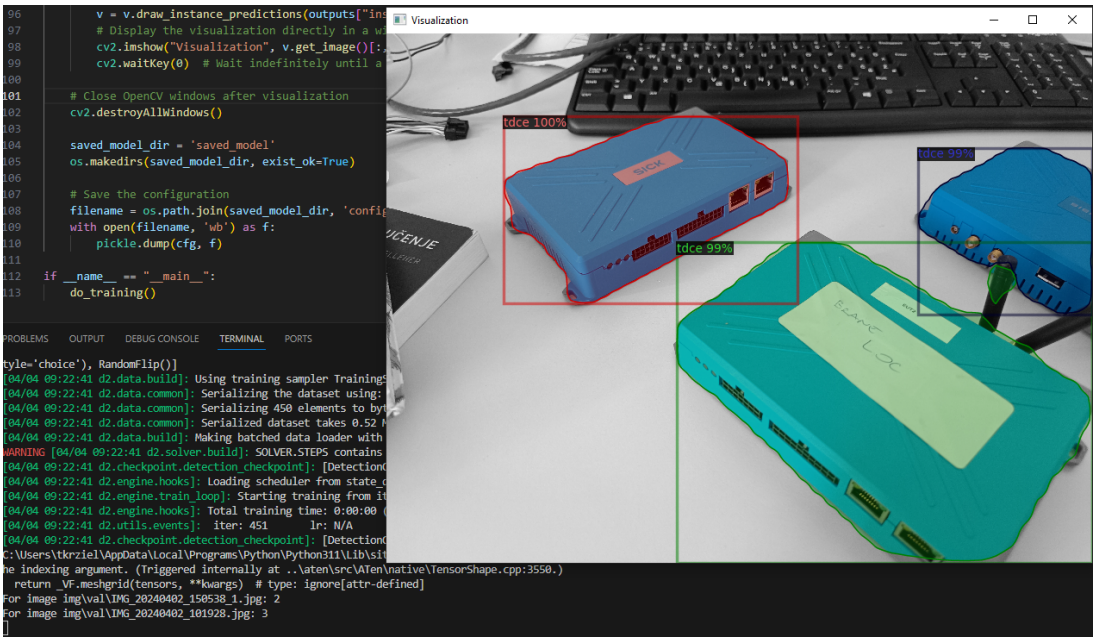
Slika 21: Rezultat predviđanja dva uređaja preko slike

Postoci testne evaluacije su za modele 1, 2, 3, 4, 5 i 6 redom 92.901%, 69.623%, 94.699%, 92.710%, 92.712% i 93.176%. Većina odabranih modela imaju preciznost veću od 90%, pri čemu drugi nema dovoljnu stopu učenja te nije zadovoljavajući, a prvi ima manji broj etapa treniranja te stoga nije toliko pouzdan kao drugi modeli čija se preciznost mjeri na 1000 etapa odnosno iteracija treniranja. Kako je evaluacijski, odnosno testni rezultat modela 3 najbolji, on se uzima za krajnji model koji se koristi u daljnjem projektu.

Na daljnjim su slikama prikazani rezultati predviđanja TDC-E klase uz segmentaciju svakog piksela na slici. Maske instanci prikazane su uz Python kod pomoću OpenCV2 biblioteke.

5.1.3. Segmentacija instanci uz upotrebu kreiranog Detectron2 modela i RGB kamere

Za segmentaciju u realnom vremenu na računalo se preko USB porta povezuje OAK-1 Lite Luxonis kamera. Kreiran je kod za spajanje kamere s input i output vezama prema



Slika 22: Rezultat predviđanja tri uređaja preko slike

potrebama DepthAI-ja. Uključuje se i prozor za vizualizaciju. Postavlja se rezolucija, veličina, veza strujanja i drugi parametri koji su potrebni za hvatanje i primanje digitalnih slika okvira s uređaja.

Dalje se postavlja prediktor, varijabla koja predviđa maske objekta TDC-E klase. Hvata se konfiguracija koja je spremljena u model 3 te se dohvaća .pkl datoteka koja je kreirana tijekom treniranja modela. U njoj se nalazi konfiguracija treniranog modela koja se učitava.

Glavni dio programa nalazi se u beskonačnoj while petlji koja prekida rad na zahtjev korisnika pritiskom na tipku ESC. Petlja dohvaća trenutni okvir kamere te se u varijablu "instances" spremaju instance predviđene prediktorom. Segmentacija se prikazuje u vizualnom oknu tako da se provede funkcija "show _segmentation _masks" te OpenCV prikazuje segmentiran okvir uz funkciju "cv2.imshow". Na pritisak se tipke ESC zatvaraju svi prozori.

Prikaz segmentiranih maski ostvaruje se tako da se instance filtriraju prema klasi 0, koja je TDC-E klasa te se prikazuju samo one maske gdje je model s 96-postotnom sigurnošću može odrediti prepoznavanje i segmentiranje objekta TDC-E klase. Trenutno u programu nema potrebe postavljati ograničenje na broj TDC-E uređaja koji se trebaju nalaziti na nekom mjestu, ali svakako je dodana "print" linija koja će u svakom danom trenutku, odnosno u svakom okviru, ispisati koliko uređaja je vidljivo na zaslonu. Segmentiraju se predviđene instance i prikazuje se BGR format okvira za OpenCV vizualizaciju. Okvir se vraća za prikaz u glavnu petlju.

Na donjim slikama prikazan je rezultat programa. Vidljiv je TDC-E uređaj, maska koju Python program određuje u stvarnom vremenu gdje RGB kamera snima poziciju radnog stola te je vidljiv broj TDC-E uređaja prilikom generiranja svakog kadra.

```
pipeline = dai.Pipeline()
cam = pipeline.createColorCamera()
cam.setPreviewSize(300, 300)
cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
cam.setInterleaved(False)

# creating a link out for preview
xout = pipeline.createXLinkOut()
xout.setStreamName("preview")
cam.preview.link(xout.input)

# set new dai device
device = dai.Device(pipeline)
# max four displayed segmentations
outputQueue = device.getOutputQueue(name="preview", maxSize=4, blocking=False)

cv2.namedWindow("Camera Preview", cv2.WINDOW_NORMAL)
```

Isječak koda 9: Postavljanje parametara kamere

```
def setup_predictor():
    saved_model_dir = 'model_output_v3/saved_model'
    config_file = os.path.join(saved_model_dir, 'config.pkl')
    with open(config_file, 'rb') as f:
        cfg = pickle.load(f)

    predictor = DefaultPredictor(cfg)
    return predictor, cfg
```

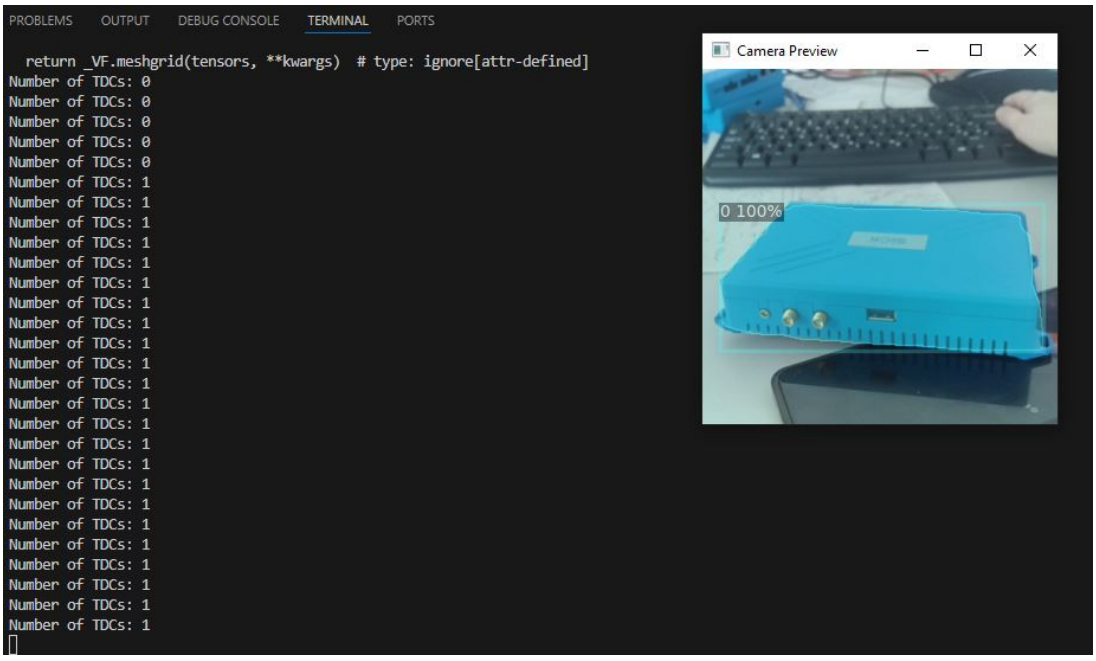
Isječak koda 10: Postavljanje prediktora

```
while True:
    inPreview = outputQueue.get()
    frame = inPreview.getCvFrame()
    instances = predictor(frame)

    segmented_frame = show_segmentation_masks(frame, instances['instances'], cfg)
    cv2.imshow("Camera Preview", segmented_frame)

    if cv2.waitKey(1) == 27:
        break

cv2.destroyAllWindows()
```

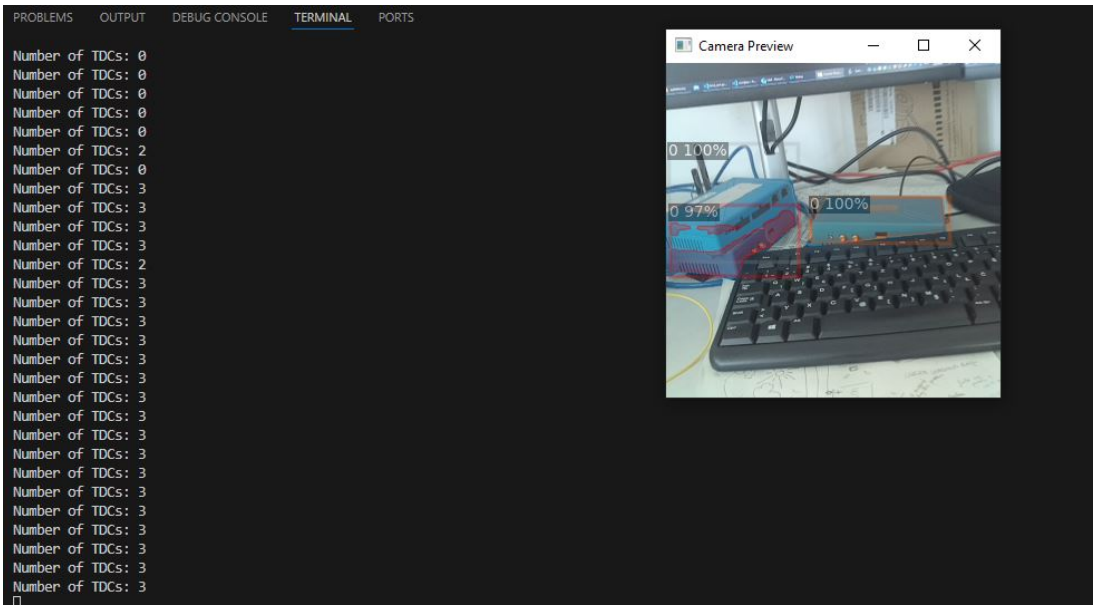
Isječak koda 11: Prikazivanje segmentiranih okvira

```python
def show_segmentation_masks(frame, instances, cfg):
    instances = instances.to("cpu")

    filtered_instances = instances[instances.scores > 0.96]  # confidence score
    filtered_instances = filtered_instances[filtered_instances.pred_classes == 0]

    num_tdc = len(filtered_instances)
    print(f"Number of TDCs: {num_tdc}")

    visualizer = Visualizer(frame[:, :, ::-1],
    ↪  MetadataCatalog.get(cfg.DATASETS.TEST[0]), scale=1.0,
    ↪  instance_mode=ColorMode.SEGMENTATION)
    visualized_output = visualizer.draw_instance_predictions(filtered_instances)
    visualized_frame = visualized_output.get_image()[:, :, ::-1]

    frame = cv2.addWeighted(frame, 0.5, visualized_frame, 0.5, 0)

    return frame
```

Isječak koda 12: Predviđanje maski



Slika 23: Predviđanje jednog TDC-E uređaja u stvarnom vremenu



Slika 24: Istovremeno označen veći broj TDC-E uređaja u stvarnom vremenu

5.2. Implementacija vizualne odometrije u stvarnom vremenu uz pomoć stereo kamere

U ovoj se sekciji prikazuje implementacija *real-time* aplikacije vizualne odometrije. Izrađena su dva Python programa koji objašnjavaju način rada vizualne odometrije i njezinih komponenata. Prva prikazana aplikacija kreirana je uz pomoć sirovog KITTI dataseta, koji sadrži set sukcesivnih slika putem kojih se određuju značajke za kreiranje putanje, prave putanje kretanja te kalibraciju kamere. Program uzima set podataka i generira mapu kretanja vozila samo na temelju vizualnog prepoznavanja značajki sa slika.

Kreiranje se druge aplikacije temelji na prvoj izrađenoj aplikaciji. Uzima većinu postavljenih parametara te se umjesto čitanja iz seta podataka ugrađuje čitanje iz Luxonis kamere, specifično čitanje s lijeve kamere uz određivanje kalibracije. Tako kreirana aplikacija izvodi istu funkcionalnost kao i ona koja koristi KITTI dataset, no uzima u obzir real-time podatke te ih obrađuje na mjestu.

Modul "pipreqs" generira sljedeće biblioteke potrebne za rad s aplikacijom:

- depthai==2.24.0.0
- matplotlib==3.7.2
- numpy==1.24.3
- open3d==0.18.0
- open3d _python==0.3.0.0
- opencv _contrib _python==4.9.0.80
- opencv _python==4.9.0.80

```
def load_images_and_poses(kitti_path):
    left_images = sorted(glob.glob(os.path.join(kitti_path, 'image_0', '*.png')))
    right_images = sorted(glob.glob(os.path.join(kitti_path, 'image_1', '*.png')))
    calib_file = os.path.join(kitti_path, 'calib.txt')
    with open(calib_file, 'r') as f:
        calib_data = f.read()
    return left_images, right_images, calib_data

def main():
    kitti_path =
    ↪ r'C:\Users\tkrziel\Projects\my_ai_cam\KITTI_train\KITTI\data\dataset\sequences\00'
    left_images, right_images, ground_truth_poses =
    ↪ load_images_and_poses(kitti_path)
    filename = "points_experiment.txt"

    if not left_images or not right_images or not len(left_images) ==
    ↪ len(right_images):
        print("Error loading images or unequal image pairs.")
        return
```

Isječak koda 13: Učitavanje potrebnih datoteka

- pandas==2.2.2

5.2.1. Vizualna odometrija uz KITTI dataset

U ovom se dijelu rada opisuje implementacija koda vizualne odometrije uz KITTI dataset [10]. Cilj je izrade aplikacije što korektnije iščitavati kretanje uređaja samo putem čitanja slika preuzetih iz KITTI dataseta i ucrtavati putanju kretanja uređaja na vizualni okvir uz njegovo spremanje pri ulazu. Spremanje će omogućiti njegovo čitanje i pregledavanje u kasnijem vremenskom trenutku. Ova je aplikacija također podloga za kreiranje aplikacije koja će putem kamere učitavati slike i iz njih odrediti putanju kretanja u stvarnom vremenu.

Na početku se programa učitavaju putanje potrebnih datoteka. To su datoteka u kojoj se nalaze KITTI slike lijeve i desne strane u sivoj boji te datoteka za kalibraciju kamere. Slike pritom prikazuju sukcesivne okvire u video snimci gdje je pomak između njih jedan okvir. Lijeva slika je prvi okvir u videu, a desna je okvir poslije prvog. Tako se određuje razlika između njih te se može zabilježiti relativna kretnja kamere.

Ova aplikacija prikazuje dva dijaloška okvira, od kojih pri prikazuje slike koje se obrađuju, dok desni prikazuje crtanu putanju. Na taj način, odmah se provjerava ima li crtana putanja smisla i kreće li se ona na isti način kao što to čini automobil koji prenosi kameru u setu podataka. Za to su korištena vizualizacijska okna Open3D. Također, odmah se kreira set za linije koji će se popunjavati točkama kako aplikacija određuje nove točke lokacija.

Također se postavlja algoritam za podudaranje značajki na slikama, Kalmanov filter za filtriranje pozicija te potrebna prazna polja i varijable. Kalmanov je filter vrlo značajan zbog poboljšanja točnosti rezultata procjene vizualne odometrije, odnosno zbog smanjenja srednje greške kvadrata svih vrijednosti [59].

Program prolazi preko svih slika danog dijela dataseta i ispisuje preostali broj potrebnih

```
sift = cv2.SIFT_create()
kalman_filter = KalmanFilter()

vis_images = o3d.visualization.Visualizer()
vis_images.create_window(width=800, height=800)

vis_line = o3d.visualization.Visualizer()
vis_line.create_window(width=800, height=800)

path_line = o3d.geometry.LineSet()

poses = []
points = []
prev_gray_left = None
prev_kp = None
prev_des = None
```

Isječak koda 14: Kreiranje i postavljanje parametara

koraka u prepoznavanju. Čita se slika i provjerava se je li ona prva u setu podataka. Ako je, program preskače rad sa slikom zbog toga što su za računanje putanje potrebne dvije sukcesivne slike.

Dalje se dobivaju ključne točke i deskriptori slike. Za detektor je ključnih točaka odabran SIFT (*Scale Invariant Feature Transform*) algoritam zbog visoke preciznosti, iako su se razmatrali i drugi algoritmi, kao što su ORB i SURF. Nakon toga, provjerava se podudaranje samih ključnih točaka između trenutnih i prošlih deskriptora. U kodu, to se čini putem FLANN algoritma za podudaranje značajki koji se nalazi u "feature _matches.py".

Također je implementirana provjera broja podudaranja, odnosno parova, kako bi se izbjeglo kreiranje točaka na slici koje nisu dovoljno značajne kako bi se mogle smatrati ispravnima. U protivnom, može se dogoditi da se generiraju netočne točke.

Dalje se generira esencijalna matrica i oporavljena pozicija kamere. Kao parametre uzimaju točke trenutne i sljedeće slike, fokalnu udaljenost i glavne točke kamere (cx i cy). Esencijalna matrica također traži vjerojatnost događaja i prag. Druga funkcija vraća oporavljenu relativnu rotacijsku matricu danih pogleda, translacijsku matricu i masku koja pokazuje na unutarnje točke [80].

Točke se pozicije ispravljaju i izglađuju za točniji prikaz na vizualizaciji te se prikazuju točke putanje svakom sljedećom iteracijom slike, pri čemu se svaka nova pozicija kamere dodaje na kraj putanje. Istovremeno se pokazuju slike čija se pozicija trenutno određuje, a brzina programa omogućuje sekvencijalni video prikaz. Na taj način, odmah je moguće obavljati provjeru kreće li se crtana putanja u dobrom smjeru.

Nakon svakog crtanja u vizualizaciju, ona se prebriše kako bi nastala animacija, odnosno kako bi se kreirale i prikazale nove točke.

U nastavku se prikazuju slike ekrana kreirane aplikacije. Prikazano je crtanje putanje uz prikaz sukcesivnih slika koje program izračunava te rezultat putanje bez i s koordinatnim prostorom.

```
for frame_idx, (left_img_path, right_img_path) in enumerate(zip(left_images,
↪  right_images)):
    if exit_program:
        break

    print(f"Processing frame {frame_idx}/{len(left_images)}")

    gray_left = cv2.imread(left_img_path, cv2.IMREAD_GRAYSCALE)
    if gray_left is None:
        print(f"Failed to load image {left_img_path}")
        continue

    if frame_idx == 0:
        prev_gray_left = gray_left
        prev_kp, prev_des = sift.detectAndCompute(prev_gray_left, None)
        print("Initialized first frame")
        continue

    kp, des = sift.detectAndCompute(gray_left, None)
    if prev_des is None or des is None:
        print("No descriptors found")
        prev_gray_left = gray_left
        prev_kp = kp
        prev_des = des
        continue
```

Isječak koda 15: Prolazak po slikama i prepoznavanje značajki

```
matches = fm.flann_match(prev_des, des, k=2)
    if len(matches) < 5:
        print("Not enough good matches found!")
        prev_gray_left = gray_left
        prev_kp = kp
        prev_des = des

def flann_match(des1, des2, k=2):
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)
matcher = cv2.FlannBasedMatcher(index_params, search_params)

des1 = np.asarray(des1, np.float32)
des2 = np.asarray(des2, np.float32)

matches = matcher.knnMatch(des1, des2, k=k)
matches = sorted(matches, key=lambda x: x[0].distance)
return matches
```

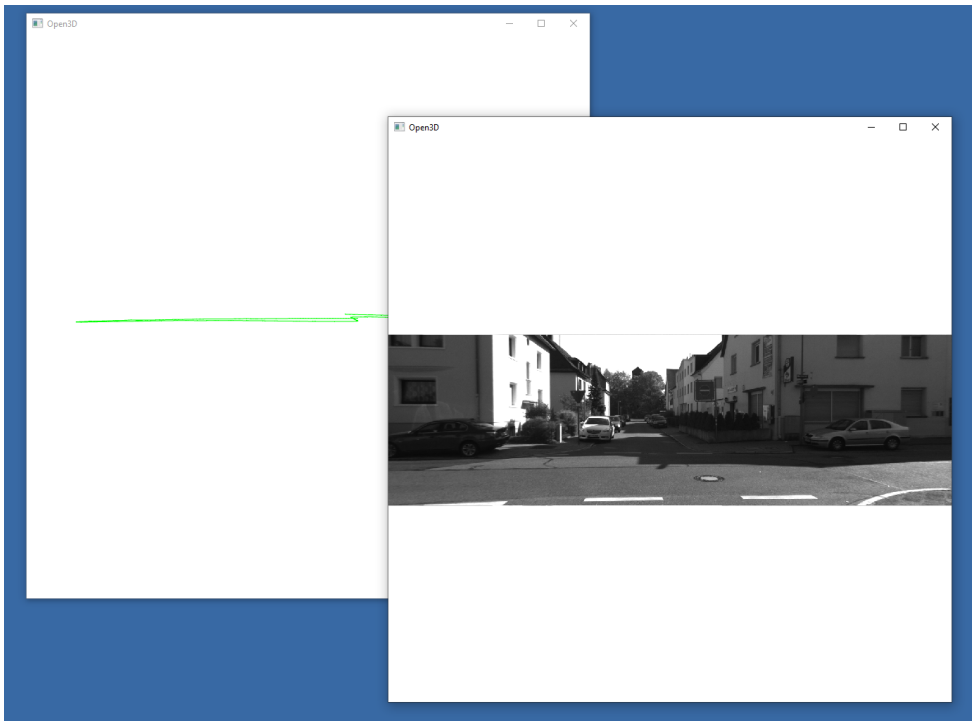
Isječak koda 16: FLANN podudaranje; izrađeno prema [79]

```
E, mask = cv2.findEssentialMat(dst_pts, src_pts, focal=718.856, pp=(607.1928,
↪  185.2157), method=cv2.RANSAC, prob=0.999, threshold=1.0)
if E is None or mask is None:
    print("Essential matrix computation failed!")
    prev_gray_left = gray_left
    prev_kp = kp
    prev_des = des
    continue

_, R, t, mask = cv2.recoverPose(E, dst_pts, src_pts, focal=718.856, pp=(607.1928,
↪  185.2157))
```

Isječak koda 17: Esencijalna matrica i oporavljena kamera

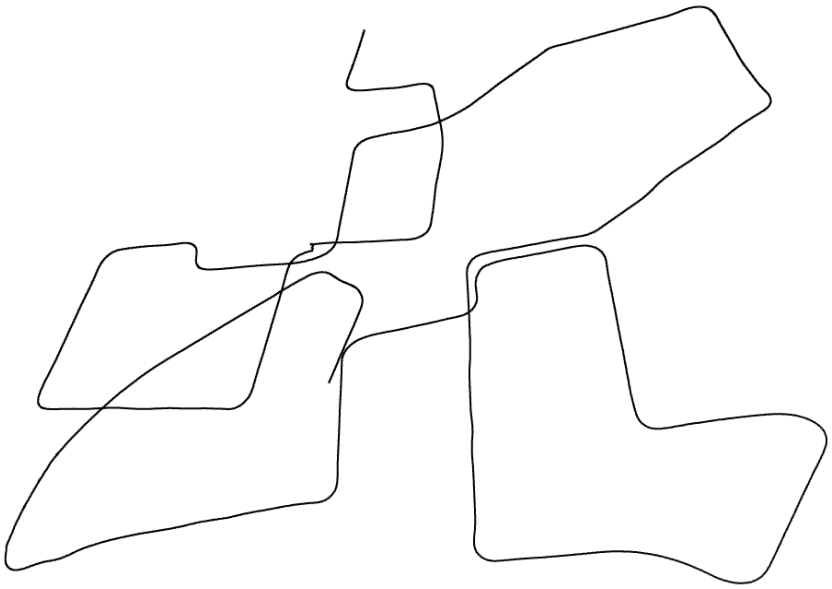
```
if len(points) > 1:
        lines = [[i, i + 1] for i in range(len(smoothed_points) - 1)]
        path_line.points = o3d.utility.Vector3dVector(smoothed_points)
        path_line.lines = o3d.utility.Vector2iVector(lines)
        path_line.paint_uniform_color([0, 1, 0])

        vis_line.clear_geometries()
        vis_line.add_geometry(path_line)
        vis_line.poll_events()
        vis_line.update_renderer()

        left_image = o3d.geometry.Image(cv2.cvtColor(gray_left, cv2.COLOR_GRAY2RGB))
        vis_images.clear_geometries()
        vis_images.add_geometry(left_image)
        vis_images.poll_events()
        vis_images.update_renderer()
```

Isječak koda 18: Obnavljanje vizualizaciji i prikaz rezultata

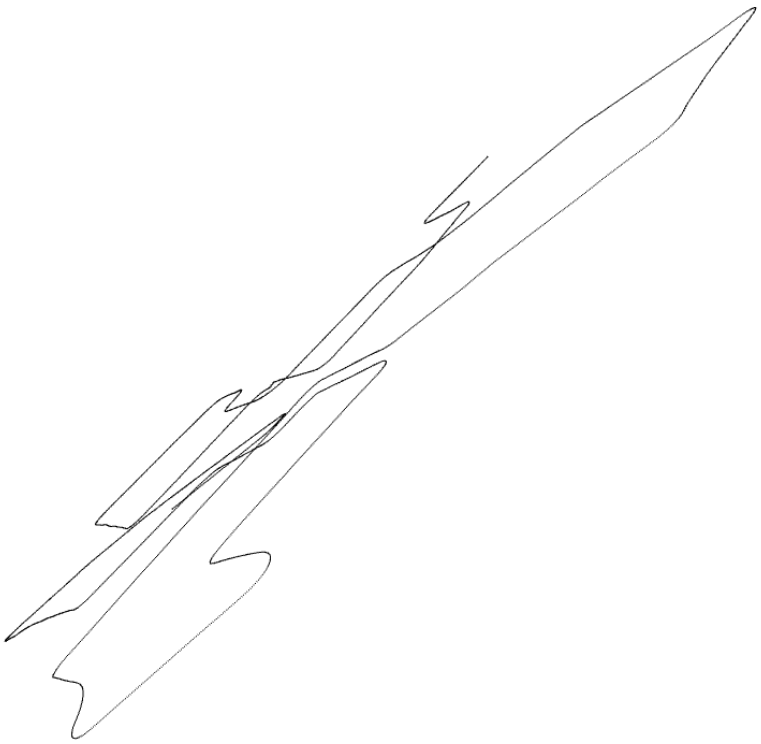


Slika 25: Prikaz generiranja putanje uz KITTI dataset

Slika 26: Prikaz generiranja putanje pod kutem uz KITTI dataset



Slika 27: Generirana putanja



Slika 28: Generirana putanja pod kutem

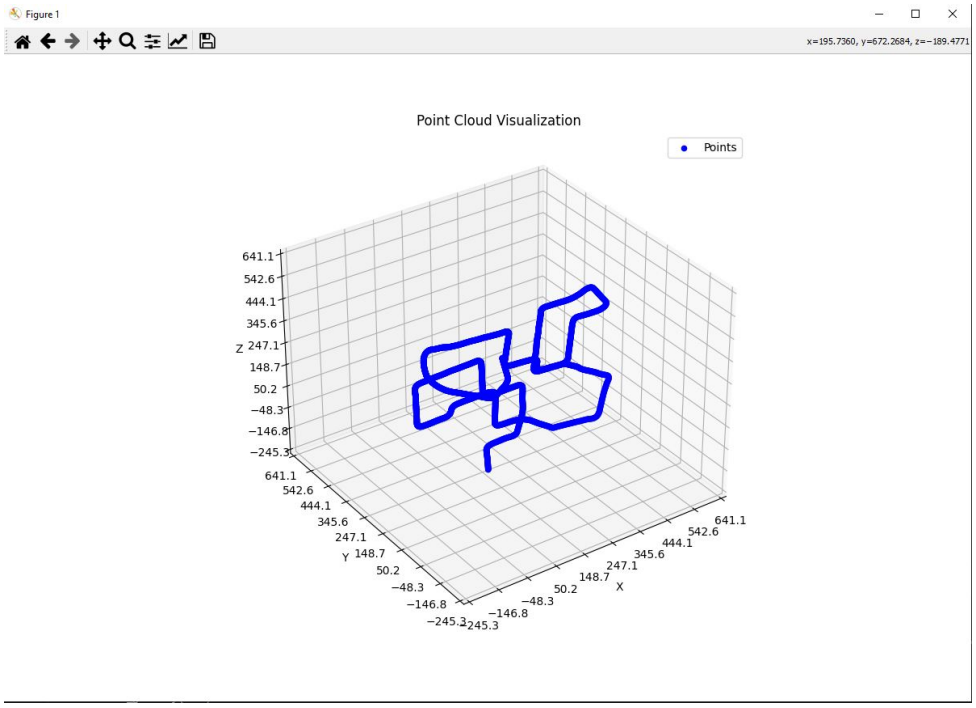
Kad je obrađen posljednji par slika danog dijela dataseta ili pritiskom signala Ctrl+C, prozori se zatvaraju i sprema se .txt datoteka koja u sebi sadrži generirane pozicije točaka. Točke i generiranu putanju moguće je vidjeti unosom u program koji može prikazati oblake točaka. Za prikaz putanje korištena je kreirana Python skripta koja se može pronaći pod nazivom Prilog 2 u prilozima.

5.2.2. Vizualna odometrija uz podatke stereo kamera u stvarnom vremenu

Ovaj je dio baziran na prvim dijelu implementacije vizualne odometrije i koristi većinu promatranih parametara i koncepata. Glavna je razlika u tome da se podaci koji se učitavaju u program generiraju preko Luxonis stereo kamere, da se oni obrađuju u stvarnom vremenu te da je potrebno generirati kalibraciju kamere, za razliku od kod KITTI dataseta, gdje je kalibracija unaprijed poznata.

Za provođenje snimanja kamere, isprva je potrebno ispravno postaviti Luxonis kameru. To je moguće kreiranjem cjevovoda, postavljanjem lijeve i desne mono kamere te kamere za dubinu.

Dalje se s DepthAI uređajem ulazi u glavni dio programa, gdje se kreiraju svi potrebni objekti i varijable. To su objekt za određivanje ključnih točaka SIFT, Kalmanov filter, objekt za



Slika 29: Prikaz puta kamere u prostoru

```python
def define_pipeline():
    pipeline = dai.Pipeline()

    monoLeft = pipeline.create(dai.node.MonoCamera)
    monoRight = pipeline.create(dai.node.MonoCamera)
    monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
    monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
    monoLeft.setBoardSocket(dai.CameraBoardSocket.CAM_B)
    monoRight.setBoardSocket(dai.CameraBoardSocket.CAM_C)

    stereo = pipeline.create(dai.node.StereoDepth)
    stereo.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
    stereo.initialConfig.setConfidenceThreshold(200)
    stereo.setLeftRightCheck(True)
    stereo.setExtendedDisparity(False)
    stereo.setSubpixel(True)
    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    xout_depth = pipeline.createXLinkOut()
    xout_depth.setStreamName("depth")
    stereo.depth.link(xout_depth.input)

    xout_rectified_left = pipeline.createXLinkOut()
    xout_rectified_left.setStreamName("rectified_left")
    stereo.rectifiedLeft.link(xout_rectified_left.input)

    xout_rectified_right = pipeline.createXLinkOut()
    xout_rectified_right.setStreamName("rectified_right")
    stereo.rectifiedRight.link(xout_rectified_right.input)

    return pipeline
```

Isječak koda 19: Postavljanje parametara Luxonis kamere; izrađeno prema [81]

```
fx_left, _, cx_left, cy_left = get_camera_calib_data()

sift = cv2.SIFT_create()
kalman_filter = KalmanFilter()
vis = o3d.visualization.Visualizer()
vis.create_window(width=1200, height=1200)
vis.get_render_option().point_size = 1.0
path_line = o3d.geometry.LineSet()
vis.add_geometry(path_line)

poses = []
points = []
frame_idx = 0
prev_gray_left = None
prev_kp = None
prev_des = None

with dai.Device(define_pipeline()) as device:
        qs = []
        qs.append(device.getOutputQueue(name="depth", maxSize=4, blocking=False))
        qs.append(device.getOutputQueue(name="rectified_left", maxSize=4,
        ↪  blocking=False))
        qs.append(device.getOutputQueue(name="rectified_right", maxSize=4,
        ↪  blocking=False))
```

Isječak koda 20: Postavljanje potrebnih varijabli i objekti

vizualizacijsko okno i putanja crtanja iz Open3D biblioteke koja se postavlja na vizualizaciju. Kreira se polje koje će sadržati sve pozicije kamere i polje u koje se spremaju generirane točke. Varijable "frame_idx" služi za provjeru broja dohvaćenog okvira jer se prvi okvir ne može uspoređivati s njegovim prošlim, a varijable s vrijednošću "None" naznačuju prijašnju lijevu sliku, ključne točke i deskriptore odnosno okruženje točaka.

Postavlja se DepthAI uređaj i polja za spremanje poslanih poruka sa specifičnih kanala kamere.

Kreira se zasebna Python datoteka koja sadrži klasu HostSync, koja je implementirana za sinkronizaciju dijelova stereo kamere.

HostSync se unosi u glavni program i program ulazi u beskonačnu petlju u kojoj prikuplja poruke, odnosno slike koje kamera sprema u program. Ako dođe poruka, slika se dodaje u red slika i hvata se okvir ispravljene lijeve *grayscale* kamere i okvir dubine. Provjerava se ima li se okvir uspoređivati s drugim okvirom ili je on prvi, gdje usporedba nije moguća. SIFT algoritam se koristi za računanje ključnih točaka i deskriptora, nakon čega se provjerava je li program skupio dovoljan broj točaka.

Daljnji je kod jednak. FLANN algoritam se koristi za usporedbu slika pri čemu je K postavljen na 2 i računaju se točke izvora i odredišta.

Potrebno je dohvatiti kalibraciju stereo kamere. U esencijalnoj matrici, parametar "fx_left" koristi se kao fokalna udaljenost, dok se parametri "cx_left" i "cx_right" koriste kao glavne točke kamere. Pritom je bitno za naglasiti da se za input slika koristi lijeva kamera, tako da se uzima i kalibracija lijeve kamere.

```
class HostSync:
    def __init__(self):
        self.arrays = {}

    def add_msg(self, name, msg):
        if name not in self.arrays:
            self.arrays[name] = []

        self.arrays[name].append({"msg": msg, "seq": msg.getSequenceNum()})

        synced = {}
        for name, arr in self.arrays.items():
            for obj in arr:
                if msg.getSequenceNum() == obj["seq"]:
                    synced[name] = obj["msg"]
                    break

        if len(synced) == 3:  # depth, rectified_left, rectified_right
            for name, arr in self.arrays.items():
                self.arrays[name] = [obj for obj in arr if obj["seq"] >=
                ↪  msg.getSequenceNum()]
            return synced
        return False
```

Isječak koda 21: HostSync klasa; preuzeto iz [81]

Računa se pozicija kamere i postavlja se Kalmanov filter. Provjerava se postoji li dovoljna količina dobivenih poziciji. Uzima se zadnja pozicija kamere iz liste "poses" i inicijalizira se nova matrica veličine $4 * 4$. Gornja lijeva submatrica nove pozicije postavlja se na vrijednost nove dobivene pozicije "R" i ona se pljošti s translacijskim vektorom "t", koji predstavlja translaciju između okvira kamere. Na kraju se stara pozicija množi s novom pozicijom kako bi se saznala trenutna pozicija kamere.

Ako ne postoji dovoljno poziciji, odnosno ako ne postoji barem jedna pozicija, kreira se nova pozicija čija se gornja lijeva submatrica postavlja na vrijednost "R" i ona se pljošti translacijskim vektorom.

Pozicija se dodaje listi "poses" i prijašnje se vrijednosti slike, ključnih točaka i deskriptora ažuriraju s novima kako bi se njima mogla uspoređivati sljedeća slika koju kamera hvata.

Prije prikaza i spremanja kreiranih točaka, računa se i provjerava mapa dubine dohvaćena na početku svake iteracije petlje. Ako su dubine i izvora i odredišta valjane, odnosno veće su od nule, dubina je valjana. Ako nije, zbog mogućih grešaka, okvir se preskače. Nadalje, točke se filtriraju za kreiranje blaže putanje.

Vizualizacija se ažurira prema duljini točaka i rezultati se prikazuju na jednak način kao i u ranijem programu. U nastavku je prikazan rad aplikacije. Za testiranje aplikacije prolazi se po unutrašnjem prostoru SICK Mobilisisa d.o.o. Pritiskom se signala Ctrl+C izlazi iz programa i sve se pozicije spremaju u tekstualnu datoteku koja se po potrebi može učitati u drugi program.

Dobiveni su rezultati dobri, kamera prepoznaje kuteve, skretanja i duljinu hodanja u smjeru, a također prepoznaje dizanje i spuštanje kamere. Na slici je 30 vidljiv je zaokret kamere prema izlazu iz ureda. Zaokret se iscrtava u stvarnom vremenu.

```
sync = HostSync()
while not exit_program:
    for q in qs:
        new_msg = q.tryGet()
        if new_msg is not None:
            msgs = sync.add_msg(q.getName(), new_msg)
            if msgs:

                gray_left = msgs["rectified_left"].getCvFrame()
                depth_map = msgs["depth"].getFrame()

                # skip first frame
                if frame_idx == 0:
                    prev_gray_left = gray_left
                    prev_kp, prev_des = sift.detectAndCompute(prev_gray_left, None)
                    frame_idx += 1
                    continue

                # sift keypoints and descriptors
                kp, des = sift.detectAndCompute(gray_left, None)
                if prev_des is None or des is None:
                    prev_gray_left = gray_left
                    prev_kp = kp
                    prev_des = des
                    continue

                if len(prev_kp) < 2 or len(kp) < 2:
                    print("Not enough keypoints detected")
                    prev_gray_left = gray_left
                    prev_kp = kp
                    prev_des = des
                    continue
                    # ...

    if exit:
            print("Saving points and closing...")
            save_points_to_txt(points)
            vis.destroy_window()
            break
```

Isječak koda 22: Glavna petlja programa

```
def get_camera_calib_data():

    camera_matrix = np.array([[818.56884766, 0, 664.12939453],
                              [0, 812.00598145, 382.4239502],
                              [0, 0, 1]], dtype=np.float32)

    fx_left = camera_matrix[0, 0]
    fy_left = camera_matrix[1, 1]
    cx_left = camera_matrix[0, 2]
    cy_left = camera_matrix[1, 2]
    return fx_left, fy_left, cx_left, cy_left
```

Isječak koda 23: Kalibracija stereo kamere putem DepthAI biblioteke

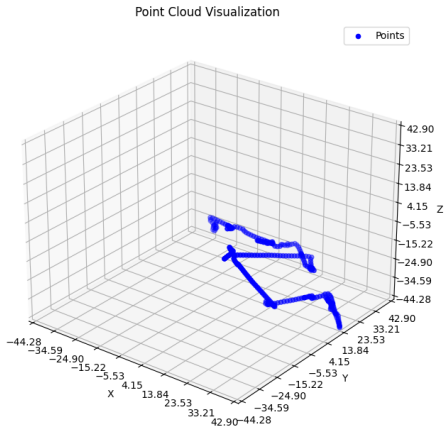
```
if len(poses) > 0:
    last_pose = poses[-1]
    new_pose = np.eye(4)
    new_pose[:3, :3] = R
    new_pose[:3, 3] = t.flatten()
    new_pose = last_pose @ new_pose
else:
    new_pose = np.eye(4)
    new_pose[:3, :3] = R
    new_pose[:3, 3] = t.flatten()

    poses.append(new_pose)
    prev_gray_left = gray_left
    prev_kp = kp
    prev_des = des
```

Isječak koda 24: Ažuriranje poziciji kamere

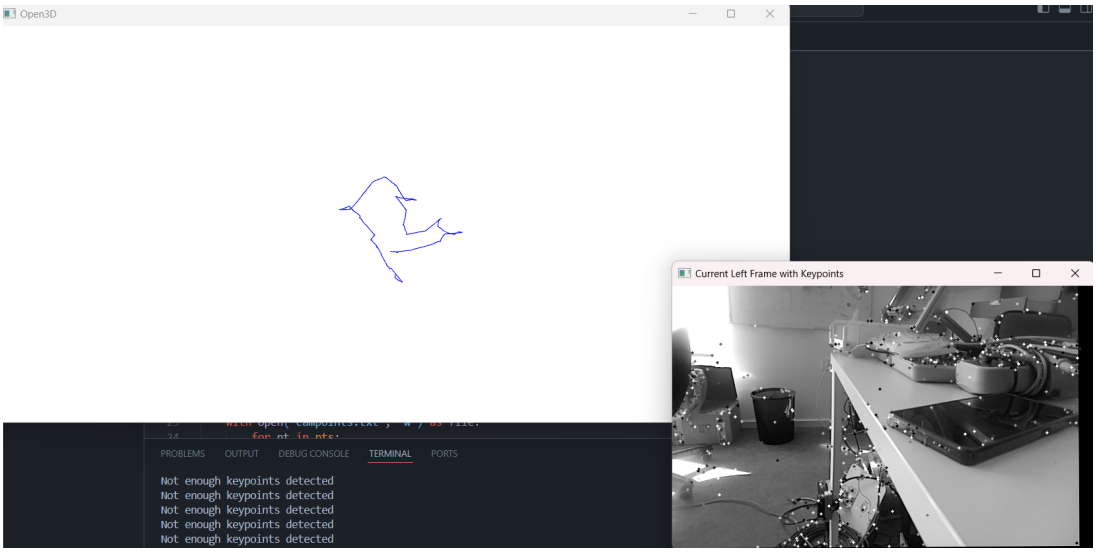
```
src_pts_depth = np.array([depth_map[int(pt[0][1]), int(pt[0][0])] for pt in
↪  src_pts])
dst_pts_depth = np.array([depth_map[int(pt[0][1]), int(pt[0][0])] for pt in
↪  dst_pts])
valid_depth = (src_pts_depth > 0) & (dst_pts_depth > 0)

if not np.any(valid_depth):
    print("No valid depth points found for validation")
    continue

filtered_poses = moving_average_filter(poses, window_size=WINDOW_SIZE)
```

Isječak koda 25: Provjera dubine i filtriranje

Slika 30: Crtanje putanje zaokreta u stvarnom vremenu



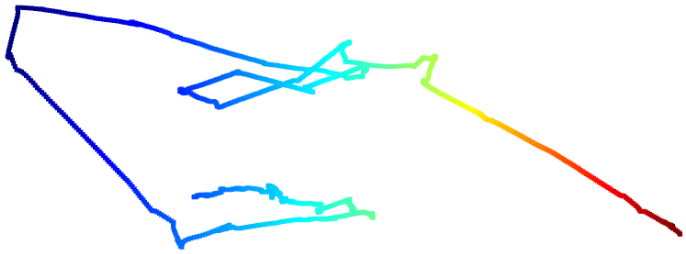
Slika 31: Prikaz putanje u stvarnom vremenu u prostoru



Slika 32: Prikaz putanje u stvarnom vremenu u prostoru

Na slici se 31 nalazi putanja koja se uzima nakon gibanja kamere i povezanog laptopa prema gore. Dobro se prepoznaju kutevi jer ima dovoljni broj značajka koje program raspoznaje.

Na zadnjim se slikama nalazi provedeni put oko uredi SICK Mobilisis d.o.o. Kako se putanja povećava, linije se u vizualizaciji smanjuju kako bi se prikazala čitava putanja. Program raspoznaje kretnje u prostoru i većinski dobro određuje okrete i gibanje u prostoru, no program teže raspoznaje putanju u mračnijim i jednoličnijim prostorima gdje je količina ključnih točaka manja. Također je potrebno napomenuti da se kamera ne giba konzistentnom brzinom, već promjenjivom prema hodu. Na zadnjoj se slici, gdje su pozicije kamere označene oblakom točaka, jasno vide dijelovi u kojima je programu lakše raspoznati putanju prema pravim kutevima i ravnim linijama. Raznoliko, dobro osvijetljeno okruženje s puno posebnih značajka za prepoznavanje pridonosi preciznosti programa. Primjer je korištenja ovakve aplikacije viličar koji treba postaviti palete na platformu u sobi s kontroliranim uvjetima. U svakom je trenutku



Slika 33: Prikaz putanje u stvarnom vremenu u prostoru

vidljivo gdje se on nalazi bez potrebe za GPS-om, već samo uz korištenje vizualne odometrije, što omogućuje i njegovo kretanje u prostorima gdje se GPS teško učitava.

6. Zaključak

Računalni je vid vrlo bitan aspekt umjetne inteligencije u računalima jer omogućuje rad s podacima u prostoru, odnosno omogućuje percepciju okoline u kojoj se računalo ili uređaj nalazi. Značenje podataka može se uspostaviti tehnikama poput prepoznavanja objekata, njihovom klasifikacijom, semantičkom segmentacijom ili segmentacijom instanci, vizualnom odometrijom, oblakom točaka i sličnim tehnologijama.

U ovom je radu implementirana segmentacija instanci za određivanje broja TDC-E uređaja u vidnom polju RGB kamere. U svakom se okviru kamere provjerava sadrži li okolina traženi objekt te se označava svaki piksel objekta u slučaju njegova postojanja. Program je i u mogućnosti brojanja koliko objekata je pronađeno, što je obilježje segmentiranje instanci. Predloženi model je odlične preciznosti od 94.699%. TDC-E se uređaj fotografira iz različitih kuteva i uz različite predmete kako bi model naučio što je, ali i što nije TDC-E. Implementirana je i vizualna odometrija putem stereo kamere za dohvaćanje dubine slika s kamere i crtanja njezina kretanja. Ona omogućuje mapiranje kretanja uređaja, odnosno nekog agenta, kako bi se uspostavila njegova relativna lokacija i kako bi se ona ucrtavala u svakom trenutku.

Povijesni razvoj i analiza trendova percepcije okoline prikazuje stalan rad na algoritmima računalnog vida i dokazuje važnost daljnjeg istraživanja ovog područja. Istraživanje računalnog vida uvelike je potaknulo razvoj dubokog učenja i konvolucijskih neuronskih mreža, koje su i dalje najznačajnija tehnologija dubokog učenja današnjice. Dok se CNN najčešće koristi za percepciju i razumijevanje okoline, vizualna je odometrija jedna od suvremenijih tehnologiji koja se koristi za percepciju i praćenje kretanja okoline. Prikazani su suvremeni algoritmi odnosno tehnologije i postavlja se naglasak na brojne mogućnosti koje pružaju. Duboko je učenje korišteno u raznolikim područjima kao što su medicina, video igre, umjetnost, odometrija, promet, robotika, sigurnost, agrikultura, manufaktura, transport i druga, što ukazuje na potrebu njegovog daljnjeg razvoja.

Popis literature

[1] J. McCarthy i dr., „What is artificial intelligence," 2007.

[2] F. A. PK, „What is artificial intelligence?" *Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do*, sv. 65, 1984.

[3] A. Pannu, „Artificial intelligence and its application in different areas," *Artificial Intelligence*, sv. 4, br. 10, str. 79–84, 2015.

[4] G. Stockman i L. G. Shapiro, *Computer vision*. Prentice Hall PTR, 2001.

[5] draw.io, *draw.io*, Accessed: 2024-08-02, n.d.

[6] luxonis, *DepthAI: Embedded Machine learning and Computer vision api*, Software available from luxonis.com, 2020.

[7] A. Dutta, A. Gupta i A. Zisserman, *VGG Image Annotator (VIA)*, Version: 2.0.12, Accessed: May 13 2024, 2016.

[8] A. Dutta i A. Zisserman, „The VIA Annotation Software for Images, Audio and Video," *Proceedings of the 27th ACM International Conference on Multimedia*, serija MM '19, Nice, France: ACM, 2019., ISBN: 978-1-4503-6889-6/19/10. DOI: 10.1145/3343031.3350535.

[9] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo i R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[10] A. Geiger, P. Lenz, C. Stiller i R. Urtasun, „Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[11] luxonis, *OAK-1 Lite: 13 MP RBG camera*, Lightweight version of OAK-1 device employing a different, 13Mpix camera module, 2020.

[12] luxonis, *OAK-D Pro: Stereo camera with Edge AI*, Stereo Camera with Edge AI capabilities from Luxonis and OpenCV, 2020.

[13] M. Gosta, „Računalni prostorni vid," *Hrvatska agencija za poštu i elektroničke komunikacije (HAKOM)*,

[14] IBM, *What is computer vision?* 2024.

[15] R. M. Haralick i L. G. Shapiro, „Glossary of computer vision terms.," *Pattern Recognit.*, sv. 24, br. 1, str. 69–93, 1991.

[16] A. Voulodimos, N. Doulamis, A. Doulamis i E. Protopapadakis, „Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, sv. 2018, 2018.

[17] N. Sebe, *Machine learning in computer vision*. Springer Science & Business Media, 2005., sv. 29.

[18] viso.ai, *Product Detection*, https://viso.ai/application/product-detection/, 2023.

[19] K. Britland, *Bottles of soft drinks in a UK supermarket stock photo*, Alamy, 2014.

[20] V. Labs, *What Is Computer Vision? [Basic Tasks & Techniques]*, Accessed 13 May 2024.

[21] L. Liu, W. Ouyang, X. Wang i dr., „Deep Learning for Generic Object Detection: A Survey," *International Journal of Computer Vision*, sv. 128, str. 261–318, 2020. DOI: 10.1007/s11263-019-01247-4.

[22] T.-Y. Lin, M. Maire, S. Belongie i dr., „Microsoft COCO: Common Objects in Context," *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele i T. Tuytelaars, ur., Cham: Springer International Publishing, 2014., str. 740–755, ISBN: 978-3-319-10602-1. DOI: 10.1007/978-3-319-10602-1_48.

[23] P. W. Code, *Object Localization*.

[24] Encord, *Object Localization Definition*, Accessed: May 13 2024.

[25] F. Perazzi, A. Khoreva, R. Benenson, B. Schiele i A. Sorkine-Hornung, „Learning Video Object Segmentation From Static Images," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[26] O. L. F. d. Carvalho, O. A. de Carvalho Junior, A. O. d. Albuquerque i dr., „Instance segmentation for large, multi-channel remote sensing imagery using mask-RCNN and a mosaicking approach," *Remote Sensing*, sv. 13, br. 1, str. 39, 2020.

[27] D. Scaramuzza i F. Fraundorfer, „Visual odometry [tutorial]," *IEEE robotics & automation magazine*, sv. 18, br. 4, str. 80–92, 2011.

[28] A. Tourani, H. Bavle, J. L. Sanchez-Lopez i H. Voos, „Visual slam: What are the current trends and what to expect?" *Sensors*, sv. 22, br. 23, str. 9297, 2022.

[29] R. Demush. „A Brief History of Computer Vision (and Convolutional Neural Networks)," HackerNoon Mobile. (2019.), adresa: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3.

[30] D. H. Hubel i T. N. Wiesel, „Early exploration of the visual cortex," *Neuron*, sv. 20, br. 3, str. 401–412, 1998.

[31] D. H. Hubel i T. N. Wiesel, „Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, sv. 148, br. 3, str. 574, 1959.

[32] E. Alpaydin, *Strojno učenje*. Mate d.o.o., 2021., 205 str., ISBN: 978-953-246-460-3.

[33] J. D. Kelleher, *Duboko učenje*. MIT press, 2019.

[34] F. Rosenblatt i dr., *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books Washington, DC, 1962., sv. 55.

[35] A. Singh, K. Bacchuwar i A. Bhasin, „A survey of OCR applications," *International Journal of Machine Learning and Computing*, sv. 2, br. 3, str. 314, 2012.

[36] R. Ptucha, F. P. Such, S. Pillai, F. Brockler, V. Singh i P. Hutkowski, „Intelligent character recognition using fully convolutional neural networks," *Pattern recognition*, sv. 88, str. 604–613, 2019.

[37] K. Fukushima, „Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, sv. 36, br. 4, str. 193–202, 1980.

[38] J. L. Schonberger i J.-M. Frahm, „Structure-from-motion revisited," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016., str. 4104–4113.

[39] M. Stepinac i M. Gasparovic, „A Review of Emerging Technologies for an Assessment of Safety and Seismic Vulnerability and Damage Detection of Existing Masonry Structures," *Applied Sciences*, sv. 10, 7. 2020. DOI: 10.3390/app10155060.

[40] M. Slavković i D. Jevtić, „Face recognition using eigenface approach," *Serbian Journal of electrical engineering*, sv. 9, br. 1, str. 121–130, 2012.

[41] GeeksforGeeks, *ML: Face recognition using eigenfaces (PCA algorithm)*, Accessed: May 13 2024, 2021.

[42] S. Z. Li, „Markov random field models in computer vision," *Computer Vision—ECCV'94: Third European Conference on Computer Vision Stockholm, Sweden, May 2–6 1994 Proceedings, Volume II 3*, Springer, 1994., str. 361–370.

[43] *What Is SLAM (Simultaneous Localization and Mapping) - MATLAB & Simulink*, MATLAB & Simulink, Accessed: May 13 2024.

[44] E. Technologies, *SLAM 3D Mapping*, https://www.exyn.com/slam-3d-mapping, Accessed: 2024-05-27.

[45] J. Wang, S. Tan, X. Zhen i dr., „Deep 3D human pose estimation: A review," *Computer Vision and Image Understanding*, sv. 210, str. 103 225, 2021.

[46] N. Audebert, B. Le Saux i S. Lefèvre, „Segment-before-Detect: Vehicle Detection and Classification through Semantic Segmentation of Aerial Images," *Remote Sensing*, sv. 9, br. 4, str. 368, 2017. DOI: 10.3390/rs9040368.

[47] MATLAB & Simulink, *What is a convolutional neural network?: 3 things you need to know*, Accessed: May 13 2024.

[48] N. Buduma, *Fundamentals of Deep Learning*, M. Loukides i S. Cutt, ur. O'Reilly Media, Inc., 2017., 283 str., ISBN: 978-1-491-92561-4.

[49] Y. LeCun, Y. Bengio i G. Hinton, „Deep learning," *nature*, sv. 521, br. 7553, str. 436–444, 2015.

[50] J. Schmidhuber, „Deep learning," *Scholarpedia*, sv. 10, br. 11, str. 32 832, 2015.

[51] R. Restak, „The secret life of the brain," 2001.

[52] K. O'Shea i R. Nash, „An Introduction to Convolutional Neural Networks," *arXiv:1511.08458 [cs]*, 2. 12. 2015. arXiv: 1511.08458.

[53] A. Dongare, R. Kharde, A. D. Kachare i dr., „Introduction to artificial neural network," *International Journal of Engineering and Innovative Technology (IJEIT)*, sv. 2, br. 1, str. 189–194, 2012.

[54] D. Bhatt, C. Patel, H. Talsania i dr., „CNN variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, sv. 10, br. 20, str. 2470, 2021.

[55] C. F. G. D. Santos i J. P. Papa, „Avoiding overfitting: A survey on regularization methods for convolutional neural networks," *ACM Computing Surveys (CSUR)*, sv. 54, br. 10s, str. 1–25, 2022.

[56] L. Ma, Y. Liu, X. Zhang, Y. Ye, G. Yin i B. A. Johnson, „Deep learning in remote sensing applications: A meta-analysis and review," *ISPRS journal of photogrammetry and remote sensing*, sv. 152, str. 166–177, 2019.

[57] D. Bolya, C. Zhou, F. Xiao i Y. J. Lee, „Yolact: Real-time instance segmentation," *Proceedings of the IEEE/CVF international conference on computer vision*, 2019., str. 9157–9166.

[58] K. He, G. Gkioxari, P. Dollár i R. Girshick, „Mask r-cnn," *Proceedings of the IEEE international conference on computer vision*, 2017., str. 2961–2969.

[59] M. O. Aqel, M. H. Marhaban, M. I. Saripan i N. B. Ismail, „Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, sv. 5, str. 1–26, 2016.

[60] S. Poddar, R. Kottath i V. Karar, „Evolution of visual odometry techniques," *arXiv preprint arXiv:1804.11142*, 2018.

[61] J. Campbell, R. Sukthankar, I. Nourbakhsh i A. Pahwa, „A robust visual odometry and precipice detection system using consumer-grade monocular vision," *Proceedings of the 2005 IEEE International Conference on robotics and automation*, IEEE, 2005., str. 3421–3427.

[62] OpenCV, *OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)*, `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`, Accessed: 2024-05-22, 2020.

[63] D. G. Lowe, „Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, sv. 60, str. 91–110, 2004.

[64] D. Nistér, O. Naroditsky i J. Bergen, „Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Ieee, sv. 1, 2004., str. I–I.

[65] T. Taketomi, H. Uchiyama i S. Ikeda, „Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ transactions on computer vision and applications*, sv. 9, str. 1–11, 2017.

[66] R. Chatila i J. Laumond, „Position referencing and consistent world modeling for mobile robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, IEEE, sv. 2, 1985., str. 138–145.

[67] M.
bibinitperiod Simulink, *What is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink*, https://www.mathworks.com/discovery/slam.html, Accessed: 2024-05-27.

[68] FARO, *What is SLAM?* https://www.faro.com/en/Resource-Library/Article/What-is-SLAM, Accessed: 2024-05-27.

[69] IGI Global. „What is RGB camera (RGB-cam).” (), adresa: https://www.igi-global.com/dictionary/mobile-applications-for-automatic-object-recognition/60647#google_vignette.

[70] E. Chen, *British Shorthair Cats and Golden Retriever*, https://www.dreamstime.com/stock-photo-british-shorthair-cats-golden-retriever-indoor-shooting-image98053512, Accessed: 2024-06-05.

[71] L. Mou i X. X. Zhu, „Vehicle Instance Segmentation From Aerial Image and Video Using a Multitask Learning Residual Fully Convolutional Network,” *IEEE Transactions on Geoscience and Remote Sensing*, sv. 56, br. 11, str. 6699–6711, 2018. DOI: 10.1109/TGRS.2018.2841808.

[72] M. Walia. „Semantic segmentation vs. instance segmentation: Explained.” Accessed May 14 2024. (2024.), adresa: https://blog.roboflow.com/difference-semantic-segmentation-instance-segmentation/.

[73] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez i J. Garcia-Rodriguez, „A review on deep learning techniques applied to semantic segmentation,” *arXiv preprint arXiv:1704.06857*, 2017.

[74] Y. Guo, Y. Liu, T. Georgiou i M. S. Lew, „A review of semantic segmentation using deep neural networks,” *International journal of multimedia information retrieval*, sv. 7, str. 87–93, 2018.

[75] J. Dai, K. He i J. Sun, „Instance-Aware Semantic Segmentation via Multi-Task Network Cascades,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[76] J. Raitoharju, „Convolutional neural networks,” *Deep learning for robot perception and cognition*, Elsevier, 2022., str. 35–69.

[77] P. Potrimba, „What is Mask R-CNN? The Ultimate Guide.,” *Roboflow Blog*, 2024., Accessed: 16 May 2024.

[78] *Exploring our Fluid Earth*, Practices of Science: False Positives and False Negatives, Accessed: 16 May 2024.

[79] OpenCV, *OpenCV: Feature Matching*, https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html, Accessed: 2024-05-22.

[80] OpenCV, *OpenCV: Camera Calibration and 3D Reconstruction*, https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html, Accessed: 2024-05-22.

[81] Luxonis, *RGB-D projection Demo*, Accessed: May 13 2024, 2022.

Popis slika

Popis tablica

Popis isječaka koda

Prilozi

1. Prilog 1

```python
import os
import cv2
import numpy as np
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.data import MetadataCatalog
from detectron2.utils.visualizer import Visualizer
from detectron2.structures import Instances

def predict_image(image_path):
    # Load Mask R-CNN model config
    cfg = model_zoo.get_config("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml", trained=True)
    cfg.MODEL.DEVICE = "cpu"
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.6
    predictor = DefaultPredictor(cfg)
    im = cv2.imread(image_path)

    outputs = predictor(im)

    instances = outputs["instances"].to("cpu")
    bottle_instances = [instances[i] for i in range(len(instances)) if instances.pred_classes[i] == MetadataCatalog.get(cfg.DATASETS.TEST[0]).thing_classes.index("bottle")]
    bottle_instances = Instances.cat(bottle_instances)
    num_bottle_instances = len(bottle_instances)
    v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TEST[0]), scale=0.5)
    v = v.draw_instance_predictions(bottle_instances)

    image_with_text = v.get_image()[:, :, ::-1]
    image_with_text = np.array(image_with_text)

    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(image_with_text, f"Number of bottle instances: {num_bottle_instances}", (10, 30), font, 1, (0, 0, 0), 2, cv2.LINE_AA)

    return image_with_text

if __name__ == "__main__":
    image_path = "supermarket.jpg"
    predicted_image = predict_image(image_path)

    cv2.imshow("Predicted Image", predicted_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Isječak koda 26: Segmentacija instanci boca i njihovo brojanje u Pythonu

2. Prilog 2

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

points_file = "points_experiment.txt"
points = np.loadtxt(points_file)

min_range = np.min(points)
max_range = np.max(points)

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(points[:, 0], points[:, 1], points[:, 2], c='b', marker='o',
↪  label='Points')

# same steps for axis
ax.set_xlim([min_range, max_range])
ax.set_ylim([min_range, max_range])
ax.set_zlim([min_range, max_range])

# ticks
ticks = np.linspace(min_range, max_range, num=10)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_zticks(ticks)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Point Cloud Visualization')

plt.legend()
plt.show()
```

Isječak koda 27: Kod za prikaz kreirane putanje vizualnom odometrijom