

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2990

**Praćenje višestrukog broja objekata
jedne klase uz tehniku
re-identifikacije**

Dominik Vrbanić

Zagreb, rujan 2022.

**SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2990

Pristupnik: **Dominik Vrbanić (0036507932)**

Studij: Računarstvo

Profil: Računarska znanost

Mentor: prof. dr. sc. Stjepan Bogdan

Zadatak: **Praćenje višestrukog broja objekata jedne klase uz tehniku re-identifikacije**

Opis zadatka:

Diplomski rad trebao bi sadržavati kratki pregled najsuvremenijih tehnologija za praćenje više pokretnih objekata u slici te aktivnih izazova u tom području. Radom na zadatku potrebno je implementirati algoritam za praćenje više objekata za robotske sustave i pritom koristiti tehnologiju ponovnog prepoznavanja već viđenog objekta pomoću neuronskih mreža. Algoritam je potrebno prilagoditi za praćenje bespilotnih letjelica. Predloženi sustav potrebno je testirati u realnom okruženju.

Rok za predaju rada: 27. lipnja 2022.

Zahvaljujem mentoru prof. dr. sc. Stjepanu Bogdanu i mag. ing. Antonelli Barišić na pomoći pri izradi ovog diplomskog rada.

SADRŽAJ

1. Uvod	1
2. Detekcija i praćenje objekata	2
2.1. Detekcija objekata	2
2.1.1. Definicija	2
2.1.2. Postojeće metode	2
2.2. Praćenje objekata	5
2.2.1. Definicija	5
2.2.2. Postojeće metode	5
2.2.3. Aktivni izazovi	7
3. Sijamske mreže	9
3.1. Učenje u jednom pokušaju	9
3.2. Arhitektura sijamske mreže	10
4. Implementacija	12
4.1. Opis problema	12
4.2. Odabir i opis korištenog algoritma Deep SORT	12
4.2.1. Algoritam SORT	12
4.2.2. Algoritam Deep SORT	14
4.3. Priprema korištenih podataka	17
4.4. Testiranje	24
4.5. Rezultati i usporedba s algoritmom SORT	28
5. Zaključak	32
Literatura	33

1. Uvod

Danas se kamere nalaze na cestama, zgradama, semaforima, parkinzima, autima, kućama i raznim drugim mjestima. Gotovo se sve snima i prati. Međutim, računalno praćenje tih kamera je zahtjevno i skupo pa je još uvijek većina kamera praćena od strane ljudi. Ubrzanim razvojem računala, pa samim time i umjetne inteligencije i računalnog vida, dolazi se do boljih i jeftinijih rješenja. S druge strane, to je područje još uvijek nedovoljno razvijeno te još uvijek postoji mnogo izazova na čijim se rješenjima iz dana u dan radi i poboljšava.

Jedan od tih izazova je praćenje pokretnih objekata u slici. Praćenje objekata ima svoje primjene u programskoj podršci povezanoj sa sigurnošću, pronalasku prijetnji, kao poveznica između ljudi i računala, u metodama upravljanja, istraživanjima živih bića, medicini, autonomnoj vožnji, robotici i mnogim drugim područjima [18]. Robotika je jedna od grana inženjerske znanosti koja se aktivno bavi problemima detekcije i praćenja objekata. Roboti mogu imati kamere čija je uloga praćenje objekata, ali je čest problem i samo praćenje robota koje je i tema ovoga rada.

U drugom poglavlju ukratko su objašnjeni problemi detekcije i praćenja objekata te je napravljen kratak pregled njihovih najsuvremenijih tehnologija. Također su nabrojani neki od aktivnih izazova praćenja više pokretnih objekata u slici unutar područja umjetne inteligencije. U trećem poglavlju objašnjene su osnove konvolucijskih neuronskih mreža te je detaljnije objašnjena sijamska konvolucijska neuronska mreža koja će biti korištena u radu. Zatim je odabran jedan od najpopularnijih algoritama za praćenje objekata, Deep SORT, koji je implementiran i prilagođen problemu praćenja bespilotnih letjelica. Algoritam je testiran u realnom okruženju i uspoređen s njegovom jednostavnijom verzijom - algoritmom SORT.

2. Detekcija i praćenje objekata

Ulaskom u 21. stoljeće i ubrzanim razvojem računalne opreme (engl. *hardware*) i programske podrške (engl. *software*) mnogi se problemi pokušavaju riješiti uz pomoć računala. Čak i čovjeku najjednostavniji problemi, poput najobičnijeg prepoznavanja psa na slici, pokušavaju se riješiti na razne načine pomoću računala. Iako je taj problem čovjeku vrlo jednostavan i intuitivan, računalo nema svoj mozak kao čovjek te je potrebno koristiti umjetnu inteligenciju kako bi moglo riješiti slične probleme. Umjetna inteligencija je dakle sposobnost stroja da može razmišljati poput čovjeka.

Jedna od grana umjetne inteligencije je strojno učenje. Strojno učenje temelji se na ideji da sustav može samostalno učiti iz dobivenih podataka, donositi zaključke i kasnije odlučivati. Jedan od podskupa metoda strojnog učenja je duboko učenje. Metode dubokog učenja koriste duboke neuronske mreže, koje će u 3. poglavljtu biti ukratko opisane, kako bi iz podataka mogle donositi zaključke i odluke. S druge strane, u umjetnoj inteligenciji nalazimo i česte probleme detekcije i praćenja objekata koji su u nastavku objašnjeni.

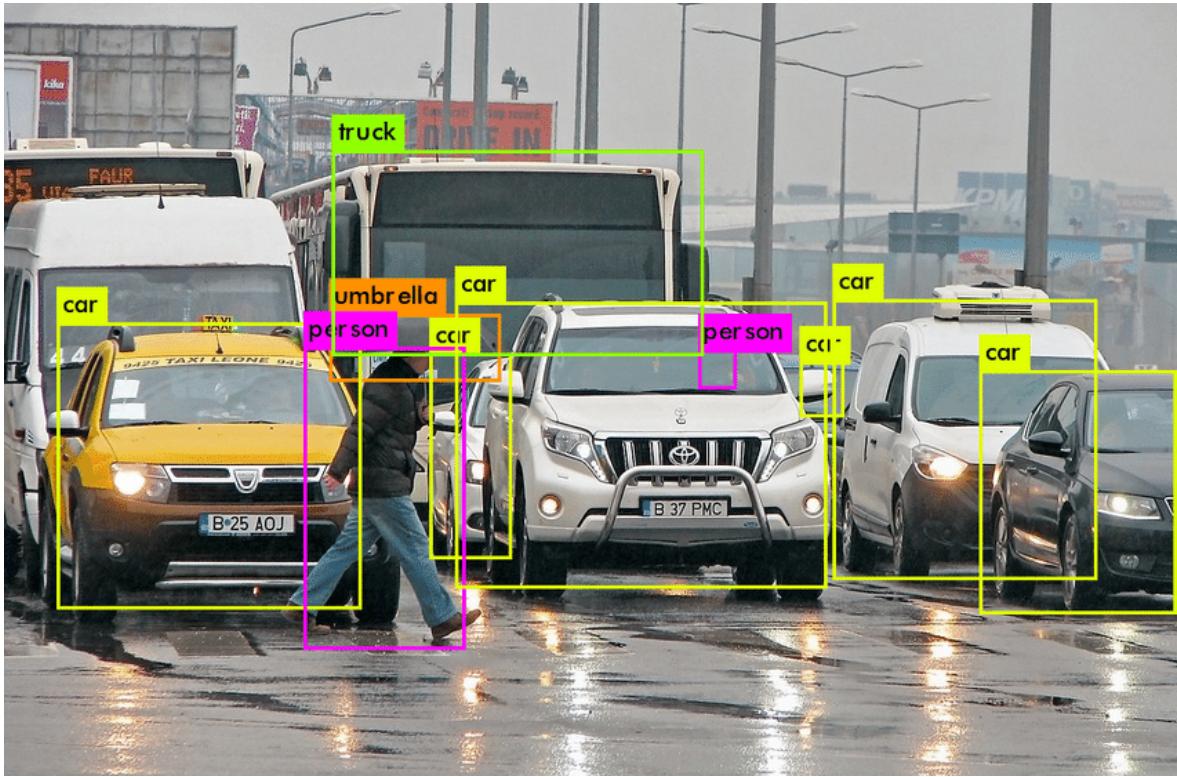
2.1. Detekcija objekata

2.1.1. Definicija

Problem detekcije objekata definira se kao zadatak pronađenja objekta određene klase unutar slike ili videozapisa. Primjer se nalazi na slici 2.1 na kojoj je prikazana cesta s vozilima te pješački prijelaz preko kojeg prelazi čovjek. Ljudskom mozgu vrlo je jednostavno prepoznati da se na slici nalaze auti, autobus, čovjek koji u ruci drži kišobran te svi ostali detalji. Međutim, računalu taj problem i nije toliko intuitivan već je potrebno stvoriti i istrenirati model koji će moći donositi zaključke poput čovjeka.

2.1.2. Postojeće metode

U zadnjih desetak godina dolazi do ubrzanog napretka računala te njihovih procesora i grafičkih kartica, ali i kamera čija kvaliteta slike je već usporediva s ljudskim okom. Strojno učenje je također eksponencijalno napredovalo na području detekcije objekata koja se samim



Slika 2.1: Primjer detekcije objekata [16]

time sve više koristi u svakodnevnim aktivnostima (medicina, autonomna vožnja, vojna programska podrška...). S obzirom na sav spomenuti napredak, algoritmi su toliko napredovali da se danas mogu koristiti i u stvarnom vremenu. Trenutno postoje dvije metode za detekciju objekata - tradicionalne metode obrade slike i moderne metode koje koriste duboko učenje [4].

Tradicionalne metode obrade slike

Tradicionalne metode počele su se koristiti od početka razvoja problema detekcije objekata u slici koje se pojavilo početkom 21. stoljeća.

Jedan od prvih i poznatijih algoritama bio je onaj koji su 2001. godine predložili Paul Viola i Michael Jones [17]. Njihov se rad temeljio na problemu detektiranja ljudskih lica iako se može prilagoditi za bilo koji drugi objekt, odnosno klasu. Iako ima osjetno manju točnost, algoritam je brz i robustan pa se i danas koristi u situacijama kada je korištena procesorska snaga manja. Algoritam uvodi 3 ključna noviteta. Prvi je reprezentacija slika koja se zove *integral image* koja omogućuje brzo izvlačenje značajki iz slika. Drugi je algoritam učenja koji se temelji na *AdaBoost* i odabire mali broj kritičnih vizualnih značajki iz većeg skupa i daje iznimno učinkovite klasifikatore. Posljednji novitet je kaskada koja omogućuje da se

pozadinska područja oko objekta ranije odbace kako bi se više procesorske snage koristilo na područja nalik na tražene objekte.

Još jedna od popularnijih tradicionalnih metoda je histogram orijentiranih gradijenata (HOG). Njega su popularizirali 2005. godine Navneet Dalal i Bill Triggs u svom radu [9]. U radu se proučava problem detekcije ljudi na slici. Nakon pregleda postojećih deskriptora temeljenih na rubovima i gradijentima, eksperimentalno je pokazano da mreže deskriptora histograma orijentiranih gradijenata (HOG) značajno nadmašuju postojeće skupove značajki za ljudsko otkrivanje.

Metode temeljene na dubokom učenju

Područje detekcije objekata dolazi u svoju drugu fazu 2014. godine i traje sve do danas. Ono se temelji na metodama dubokog učenja te se dijeli na dvije vrste algoritama - jednostupanjske (engl. *one-stage*) i dvostupanjske (engl. *two-stage*). Obje metode podijeljene su u dva dijela. Prvi je pronalazak objekata u slici ili videozapisu, a drugi je klasifikacija pojedinih objekata.

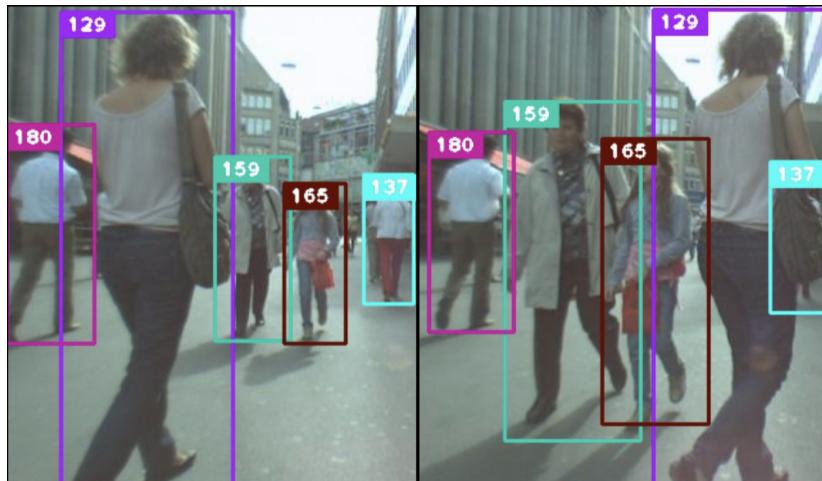
Dvostupanjske metode imaju ta dva koraka odvojena. Prvo se približna područja objekata predlažu pomoću dubokih značajki, a zatim se te značajke koriste za stvaranje *bounding boxova* oko objekata te njihovu klasifikaciju. Te metode zbog svoje veće kompleksnosti daju veću točnost, ali su samim time i sporije od jednostupanjskih. Neki od najpopularnijih primjera dvostupanjskih metoda su *RCNN* i *PPNet*, *Fast RCNN*, *Faster RCNN*, *Mask R-CNN* i *G-RCNN*.

S druge strane, jednostupanjske metode kombiniraju dva spomenuta koraka u jedan. Oni predviđaju *bounding boxove* preko slika bez koraka prijedloga regije kao što je to bio slučaj u dvostupanjskim. Tim pojednostavljenjem postiže se napredak u brzini izvođenja, ali također dolazi i do smanjenja točnosti. Takvi se algoritmi zbog svoje brzine mogu koristiti u stvarnom vremenu. Najpoznatiji primjeri su *YOLO* (engl. *You Only Look Once*), *SSD* (engl. *Single-Shot Detector*), *RetinaNet*, *YOLOv3* i *YOLOv4* [1].

2.2. Praćenje objekata

2.2.1. Definicija

Problem praćenja objekata nešto je manje popularan problem u usporedbi s detekcijom. On se definira kao problem praćenja detektiranih objekata u uzastopnim slikama (engl. *frames*) unutar videozapisa. Za kvalitetnu upotrebu algoritma za praćenje, prvo je potrebno napraviti kvalitetnu detekciju objekata pa zatim u obzir uzeti prostornu i vremensku komponentu. One se uzimaju kako bi se isti objekt, odnosno njegov *bounding box* mogao povezati u uzastopnim slikama. Iako još uvijek aktivno istraživačko područje, praćenje objekata koristi se u raznim svakodnevnim zadacima.



Slika 2.2: Primjer praćenja objekata u uzastopnim slikama [21]

2.2.2. Postojeće metode

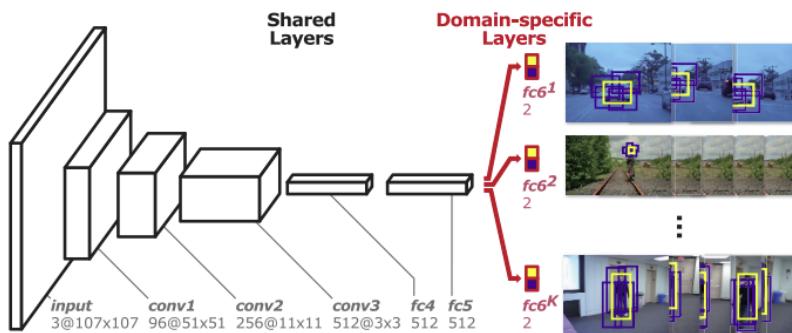
Glavna podjela odnosi se na količinu praćenih objekata. Moguće je pratiti jedan objekt (*Single Object Tracking - SOT*) ili više njih (*Multiple Object Tracking - MOT*). Za SOT su dovoljni i jednostavniji algoritmi jer se *bounding box* traženog objekta ručno označava u prvoj slici pa se onda taj objekt nastavlja tražiti u idućima. Iz tog razloga može se pratiti i bez klasifikacijskog modela. S druge strane, MOT algoritmi prate svaki pojedinačni traženi objekt unutar videozapisa. Objekti s istim identitetom pokušavaju se povezati u uzastopnim slikama.

Problem praćenja objekata pojavio se, kao i problem detekcije objekata, ulaskom u 21. stoljeće i od onda je predloženo mnogo metoda i ideja za poboljšanje točnosti i učinkovitosti modela za praćenje. U početku su aktualne bile klasične metode koje su koristile pristupe

strojnog učenja poput k-najbližih susjeda ili stroja potpornih vektora. Te su metode dobre u predviđanju traženog objekta, ali zahtijevaju ručno izvlačenje značajki izgleda. Suprotno njima, danas se više koriste metode temeljene na dubokom učenju koje samostalno obavljaju izvlačenje značajki i neke od najpoznatijih opisane su u nastavku.

MDNet

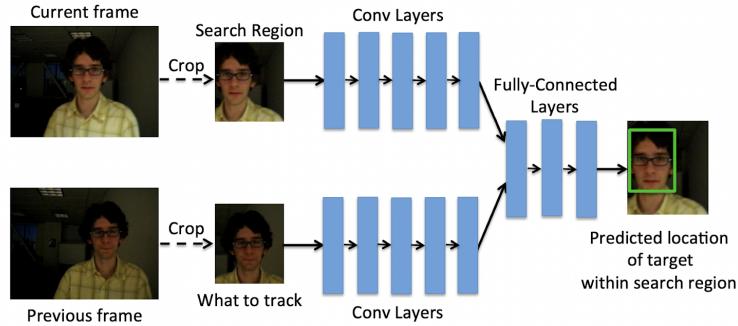
MDNet (*Multi-Domain Net*) koristi se za rješavanje praćenja jednog objekta (SOT) korišteњem konvolucijskih neuronskih mreža. Sastoji se od dvije faze. Prva faza je predtreniranje. Algoritam je u toj fazi treniran na više različitih videozapisa s anotiranim objektom kako bi naučio vizualne i prostorne značajke. Drugi je korak online vizualno praćenje. Nakon što se obavi predtreniranje, slojevi specifični za domenu uklanjuju se i mreži ostaju samo zajednički slojevi koji se sastoje od naučenih prikaza. Tijekom odlučivanja na njih se dodaje sloj binarne klasifikacije [15].



Slika 2.3: Arhitektura mreže algoritma MDNet [15]

GOTURN

GOTURN je algoritam koji koristi offline treniranje neuronske mreže pa može postići veće brzine u usporedbi s online treniranim modelima te se iz tog razloga može kvalitetno koristiti u stvarnom vremenu. Algoritam koristi jednostavnu mrežu i traži povezanost između kretanja objekta te njegovog izgleda kako bi se mogao koristiti i za praćenje novih, već ne viđenih objekata. Mreža prima dva parametra - jedan je područje interesa iz trenutne slike, a drugi je objekt iz prethodne slike te se oni međusobno uspoređuju [11].



Slika 2.4: Arhitektura mreže algoritma GOTURN [15]

Recurrent YOLO - ROLO

ROLO koristi dvije neuronske mreže - jedna od njih je konvolucijska neuronska mreža koja izvlači prostorne podatke sa slike, a druga je LSTM (Long short-term memory) koja pronađe trajektorije praćenog objekta. Za svaku sliku, prostorna informacija iz CNN je izvučena te proslijeđena u LSTM koji zatim predviđa lokaciju traženog objekta [2].

Tracktor++

Tracktor++ je online algoritam za praćenje objekata koji se temelji na detekciji. Ideja je da se za svaku sliku predviđa lokacija objekta računanjem regresije *bounding boxova*. Za detekciju se u originalnoj objavi koristi Faster RCNN [2]. Algoritam ima veliku točnost, ali zbog brzine od 3 fps ne može se koristiti u stvarnom vremenu.

Osim spomenutih algoritama, u 4. poglavlju detaljnije su objašnjena još dva popularna algoritma SORT i Deep SORT koji su testirani i čiji su rezultati uspoređeni u kontekstu ovog rada.

2.2.3. Aktivni izazovi

Rješavanjem problema praćenja objekata nailazi se na mnoge izazove. Jedan od najčešćih izazova su okluzije. Samo praćenje određenog objekta i nije toliko zahtjevno u situacijama u kojima je objekt vidljiv u svakoj slici. Međutim, kada dođe do situacije u kojoj se ispred praćenog objekta nađe neki drugi objekt na slici, tada se u jednostavnijim algoritmima pretpostavlja da je praćeni objekt nestao iz slike. Nakon što okluzija prestane i prethodno praćeni objekt se vrati u sliku, tada se taj objekt predstavlja kao novi te dobiva drugačiji identitet.

Jedno od rješenja za sprječavanje navedenog problema je izvlačenje značajki izgleda praćenih objekata pa se oni mogu i nakon okluzije identificirati kao već viđeni objekti na temelju izgleda.



Slika 2.5: Primjer okluzije na slici desno [14]

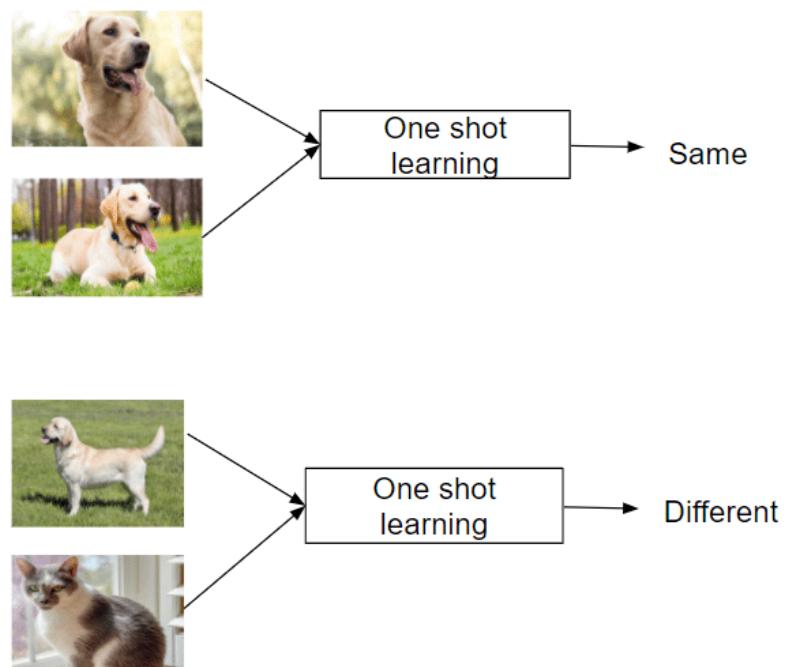
Idući česti izazov je pozadinski šum na slici. Taj se izazov javlja u većini ostalih problema unutar strojnog učenja pa tako nije izbjegnut niti u praćenju objekata. Do problema dolazi kod izvlačenja značajki praćenog objekta. U situacijama s gušćom okolinom objekta uče se i redundantni podaci i šumovi oko objekta što može značajno pokvariti brzinu i optimalnost mreže. Taj se problem rješava kvalitetnim skupom podataka za treniranje u kojem nema puno redundantnih podataka niti šumova.

Još jedan od izazova predstavlja pomična kamera ili gibajući objekt u slici. Iako mnogi algoritmi koriste izvlačenje značajki izgleda iz praćenih objekata, svejedno može doći do lošijih rezultata zbog velikih pomaka kamere ili objekta unutar videozapisa. Objekt se može približavati ili udaljavati pa na slici izgleda manje, odnosno veće. Također je moguće da se objekt okreće pa iz različitih kuteva izgleda drugačije. Rješenje tog izazova je također robusniji skup podataka za treniranje koji u obzir uzima različite veličine i poglede na traženi objekt.

3. Sijamske mreže

3.1. Učenje u jednom pokušaju

U današnje doba postoji obilje dostupnih podataka koji se između ostalog koriste i u raznim metodama dubokog učenja. Neuronske mreže treniraju se za razne specifične slučajeve. Klasične neuronske mreže zahtijevaju veliku količinu podataka za treniranje. Upravo iz tog razloga postoje situacije kada one ipak nisu dovoljno dobre zbog manjka podataka za treniranje. Jedan od takvih slučajeva je npr. evidencija dolazaka na posao unutar firme. Ideja je da se osoba na ulasku u firmu automatski prepozna pomoću kamere na ulazu koja snima lice osobe. U toj situaciji bilo bi potrebno istrenirati model koji će razlikovati relativno malen broj različitih lica. Za prepostaviti je da firma ne posjeduje velik broj fotografija lica svake od osoba. Za ovaku vrstu problema kao rješenje se javlja učenje u jednom pokušaju (engl. *one-shot learning*).



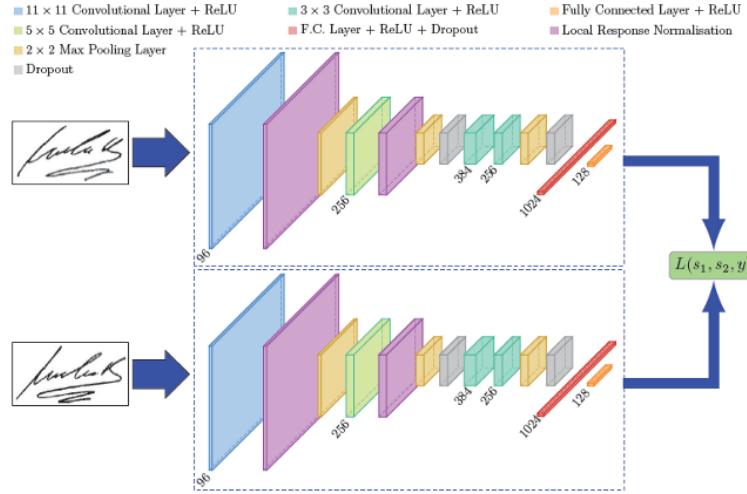
Slika 3.1: Primjer tehnike učenja u jednom pokušaju [5]

Učenje u jednom pokušaju je tehnika u kojoj za treniranje konvolucijske neuronske mreže nije potreban ogroman broj podataka kako bi naučila značajke. Dakle, na problem se ne gleda kao klasifikacijski problem. Umjesto toga gradi se funkcija sličnosti koja uspoređuje po dvije slike na temelju izvučenih značajki izgleda te odlučuje dolazi li do podudaranja. Funkcija vraća brojčanu vrijednost koja predstavlja sličnost između dvaju objekata. Ako je dobivena vrijednost veća ili jednaka zadanom pragu, objekti se smatraju jednakim, a u slučaju kada je vrijednost manja od praga objekti se smatraju različitim.

3.2. Arhitektura sijamske mreže

Kao praktično rješenje za učenje u jednom pokušaju nameće se arhitektura sijamskih mreža. Njih su prvi put spomenuli Bromley i ostali u radu iz 1993. godine [7] u kojem se opisuje algoritam za verifikaciju potpisa. Sijamske mreže poput ostalih konvolucijskih mreža također uzimaju slike te enkodiraju njihove značajke. Do razlike dolazi u obradi tih enkodiranih vrijednosti. Klasične konvolucijske neuronske mreže optimiziraju parametre kako bi se pomoću njih određeni objekt mogao klasificirati. S druge strane, sijamske mreže treniraju se s ciljem da kasnije ne klasificiraju objekt nego rezultiraju brojčanom vrijednošću koja pokazuje koliko su objekti međusobno slični, odnosno različiti. Sijamske mreže sastoje se od dviju jednakih neuronskih mreža koje dijele parametre i težine te koje su na kraju spojene s jednakom funkcijom. Svaka od mreža uzima po jednu sliku i preslikava ju u euklidski prostor. Zatim se te dvije vrijednosti uspoređuju na temelju značajki izgleda i odlučuje se predstavljaju li one isti objekt ili ne. Kod treniranja se kao funkcija gubitka uzima trostruki gubitak (engl. *triplet loss*). Funkcija uzima po 3 slike. Prva slika je referentni primjer (engl. *anchor - a*) koji se uspoređuje s pozitivnim (engl. *positive - p*) i negativnim (engl. *negative - n*) uzorkom. Slike se odabiru nasumično na način da se odaberu po dvije slike (*anchor i positive*) iz jedne klase, i jedna slika iz druge klase (*negative*). Cilj je da se različitost između referentne i pozitivne slike treba smanjiti, a razlika između referentne i negativne slike treba povećati. Također, može se koristiti i hiperparametar *margin* koji se dodaje u funkciju. On određuje kolika bi trebala biti minimalna razlika između dviju različitosti [10]. Formula za gubitak je oblika:

$$\mathcal{L} = \max(d(a, p) - d(a, n) + margin, 0), \quad (3.1)$$



Slika 3.2: Primjer arhitekture sijamske mreže [12]

Unatoč jednostavnosti i potrebi za malim brojem podataka za treniranje, sijamske mreže također imaju i svoje nedostatke. Specifična sijamska mreža može se koristiti samo za određeni problem. Npr. u nastavku rada istrenirana je mreža za praćenje bespilotnih letjelica te se ona može koristiti samo za taj zadatak. Kada bi se zadatak preformulirao i bilo bi potrebno pratiti npr. zrakoplove, mrežu bi bilo potrebno retrenirati s kompletno drugaćijim skupom podataka za treniranje.

4. Implementacija

4.1. Opis problema

Cilj ovoga rada je implementirati algoritam za praćenje više objekata za robotske sustave i pritom koristiti tehnologiju ponovnog prepoznavanja već viđenog objekta pomoću neuron-skih mreža. Algoritam je potrebno prilagoditi za praćenje bespilotnih letjelica te je predloženi sustav potrebno testirati u realnom okruženju. Očekivani rezultat rada je videozapis u kojem su *bounding boxovima* označene bespilotne letjelice te se uz svaki *bounding box* nalazi redni broj koji označava identitet letjelice.

4.2. Odabir i opis korištenog algoritma Deep SORT

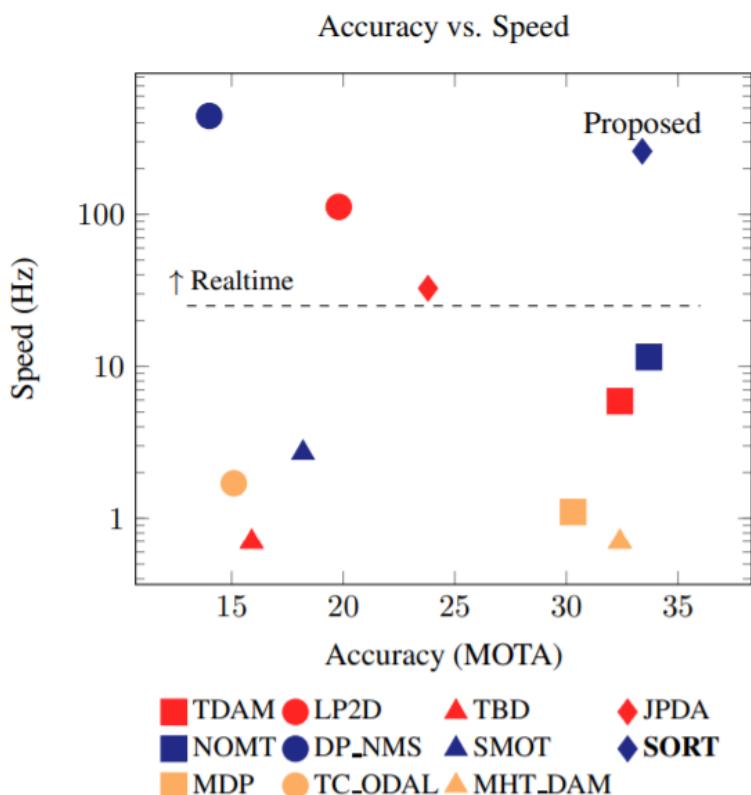
Za rješavanje navedenog problema odabran je algoritam Deep SORT [21] - složenija verzija originalnog algoritma SORT. Oba algoritma u nastavku su opisana. Kod je korišten s javno dostupnog GitHub repozitorija algoritma Deep SORT [20]. Za korištenje algoritma potrebna je sijamska mreža. Za treniranje i testiranje sijamske mreže korišten je također javno dostupan kod s GitHub repozitorija koji je napravljen iznad originalnog repozitorija [13]. Kod sadrži funkciju za treniranje sijamske mreže uz dobivene ulazne podatke te kod za testiranje mreže na skupu podataka koji je izdvojen iz skupa podataka za treniranje. Osim toga, sadrži i funkciju za testiranje cijelog algoritma Deep SORT koja na ulaz prima istreniranu sijamsku mrežu i ulazni videozapis s već dobivenim detekcijama.

4.2.1. Algoritam SORT

Algoritam SORT, punog naziva Simple Online Real-time Tracking, jest algoritam koji pripada grupi algoritama temeljenih na detekciji. Rad algoritma temelji se na osnovnim tehnikama povezivanja podataka i procjene stanja te koristi jednostavne, ali efektivne algoritme. Algoritam SORT osnova je za složeniji algoritam Deep SORT koji dodatno implementira informaciju o izgledu objekta kako bi riješio glavne probleme algoritma SORT koji su opisani u nastavku.

SORT je dizajniran za praćenje objekata u stvarnom vremenu. To se postiže rekurzivnim načinom rada gdje su potrebne detekcije samo iz prošle i trenutne slike. Ideja je da se koristi Kalmanov filter za procjene stanja i Mađarski algoritam za povezivanje pronađenih objekata u uzastopnim slikama. Mađarski algoritam provodi se nad matricom troška pridruživanja u kojoj se nalazi sličnost između svake detekcije i svakog procijenjenog novonastalog *bounding boxa*. Ta se vrijednost računa kao *intersection-over-union* (IoU) njihovih *bounding boxova*.

Zahvaljujući ovom jednostavnom pristupu, brzina algoritma SORT čak je i do 20 puta veća od brzina sličnih *state-of-the-art* algoritama. Kvaliteta rezultata također je dovoljno dobra u usporedbi s većinom sličnih algoritama. Omjer točnosti i brzine vidi se na Slici 4.1.



Slika 4.1: Omjer točnosti i brzine poznatih algoritama [6]

Međutim, jednostavnost algoritma sa sobom vuče i negativne strane koje u pojedinim problemima mogu biti izraženije. Algoritam SORT donosi relativno velik broj netočnih promjena identiteta objekata, odnosno isti objekti u uzastopnim slikama identificirani su kao različiti. Do tog problema dolazi jer se povezivanje odvija kvalitetno samo kad je nesigurnost procjene stanja niska. Stoga, SORT ima problema u slučajevima kada objekt prolazi kroz mnogo okluzija te kada se mijenja položaj promatrača, odnosno kamere. Iz tog razloga razvijen je algoritam Deep SORT koji rješava navedene probleme.

4.2.2. Algoritam Deep SORT

Kako bi se prethodno navedeni problemi riješili, u Deep SORT-u se povezivanje odvija nad rezultatima dobivenim kombinacijom pokreta (kao i u algoritmu SORT) i izgleda. Dodaje se sijamska konvolucijska neuronska mreža (CNN) koja je prethodno istrenirana za ponovnu identifikaciju određenih objekata. Integracijom ove mreže povećava se robusnost prema ne-pronađenim objektima te okluzijama, dok cijeli sustav ostaje jednostavan za implementaciju, efikasan te iskoristiv u stvarnom vremenu.

Algoritam Deep SORT može se podijeliti u četiri osnovne komponente koje su u nastavku detaljno objašnjene [21].

Praćenje i procjena stanja

U ovome dijelu nema značajnih razlika u odnosu na osnovni algoritam SORT. S obzirom na realnu situaciju u kojoj je to najčešće slučaj, i na kvalitetan skup podataka za treniranje, moguće je prepostaviti da kamera nije kalibrirana te da je u pokretu. Scenarij je predstavljen kao 8-dimenzionalno stanje $(u, v, g, h, x', y', g', h')$ koje se sastoji od središta pripadnog *bounding boxa* (u, v), omjera njegovih stranica (g), njegove visine (h) te pripadnih brzina unutar koordinata slike (x', y', g', h') [3].

Procjena položaja određenog objekta u nekoj slici određuje se kao kombinacija detekcije u toj slici i procjene dobivene modelom linearne konstantne brzine u odnosu na prošlu sliku. Ukoliko detekcija nije pronađena tada se u obzir uzima samo rezultat dobiven modelom linearne konstantne brzine. Detekcija sama po sebi nije dovoljna jer u mjeranim podacima uvijek postoji određeni šum koji se dodavanjem izračunatih podataka smanjuje. Korišteni algoritam zove se Kalmanov i pretpostavlja da je razdioba podataka Gaussova te da su pomaci između uzastopnih mjerjenja linearni pa će jedino u takvim uvjetima rezultati biti idealni.

Svaki dobiveni objekt iz postupka Kalmanovog filtera k sadrži pripadnu varijablu a_k u kojoj se nalazi broj proteklih uzastopnih slika od posljednjeg uspješnog povezivanja s novonastalom procjenom. Vrijednost varijable a_k povećava se u svakom koraku predikcije Kalmanovog filtera, a postavlja se na 0 prilikom svakog uspješnog povezivanja s novonastalom procjenom. Također postoji zadana varijabla A_{max} koja predstavlja maksimalnu dopuštenu vrijednost varijable a_k . Ukoliko vrijednost varijable a_k za neki od objekata prijeđe zadalu vrijednost A_{max} tada se taj objekt smatra nestalim iz slike i briše se iz skupa objekata. Svaki novo detektirani objekt smatra se privremenim objektom sve dok ne bude pronađen i povezan s procjenom u 3 uzastopne slike. Ukoliko ne bude pronađen u bilo kojem od početne 3 slike, taj se objekt također smatra nestalim i izbrisanim.

Problem povezivanja

Za povezivanje procijenjenih položaja Kalmanovog filtera i novonastalih detekcija potrebno je imati kvantitativan problem i algoritam za njegovo rješavanje. Za kvantificiranje povezivanja koristi se metrika udaljenosti koja je opisana u nastavku, a za algoritam se uzima već korišteni Mađarski algoritam. U odnosu na osnovni algoritam SORT, Mađarski algoritam u Deep SORT-u rješava problem u kojem su kombinirane informacija u pokretu i informacija o izgledu objekata.

Informacija o pokretu izražena je kao Mahalanobisova udaljenost između predviđenih položaja dobivenih Kalmanovih filterom i novonastalih detekcija. Izraz:

$$d^{(1)}(i, j) = (\mathbf{d}_j - \mathbf{y}_i)^\top \mathbf{S}_i^{-1} (\mathbf{d}_j - \mathbf{y}_i), \quad (4.1)$$

prikazuje zadani problem gdje $(\mathbf{y}_i, \mathbf{S}_i)$ označava projekciju i -tog objekta dobivenog Kalmanovim filterom u mjerljivi prostor, a \mathbf{d}_j označava detekciju u obliku *bounding boxa*. Korištenjem navedene metrike uzima u obzir nesigurnost procjene stanja tako što mjeri koliko standardnih devijacija je detekcija udaljena od lokacije objekta. Također, dodavanjem pravogova na dobivene vrijednosti moguće je ukloniti malo vjerojatna povezivanja. Vrijednost varijable:

$$b^{(1)}(i, j) = 1 |d^{(1)}(i, j) \leq t^{(1)}| \quad (4.2)$$

postavlja se na 1 kada je povezivanje i -tog objekta i j -te detekcije dopustivo. Mahalanobisov prag za 4-dimenzionalni prostor u kojem se odvija mjerjenje iznosi $t^{(1)} = 9.4877$.

S druge strane, informacija o izgledu izražena je preko dubokog deskriptora izgleda. Za svaki detektirani *bounding box* \mathbf{d}_j računa se deskriptor izgleda \mathbf{r}_j . Za svaki objekt k čuva se informacija o posljednjih 100 povezanih deskriptora izgleda u varijabli R_i . Ova metrika računa najmanju kosinusovu udaljenost između i -tog objekta iz Kalmanovog filtera i j -te detekcije po formuli:

$$d^{(2)}(i, j) = \min\{1 - \mathbf{r}_j^\top \mathbf{r}_k^i | \mathbf{r}_k^i \in R_i\}, \quad (4.3)$$

i u ovoj metrići dodan je prag kojim se uklanjaju malo vjerojatna povezivanja:

$$b^{(2)}(i, j) = 1 |d^{(2)}(i, j) \leq t^{(2)}|. \quad (4.4)$$

Dvije se metrike kombiniraju iz razloga što je svaka od njih povoljna u nekom slučaju. Mahalanobisova udaljenost prigodna je za situacije u kojima se traži mogući položaj objekta u kratkim vremenskim razmacima. S druge strane, kosinusova udaljenost korisna je kada se traži objekt u dužim vremenskim razmacima uzrokovanim okluzijama.

Dvije metrike kombinirane su težinskim zbrojem te je dobivena formula:

$$C(i, j) = \lambda \cdot d^{(1)}(i, j) + (1 - \lambda) \cdot d^{(2)}(i, j), \quad (4.5)$$

gdje se povezivanje smatra mogućim ako se nalazi unutar oba zadana praga. Parametar λ određuje koja će se metrika više uzeti u obzir ovisno o zadanim ulazima i traženom rješenju.

Kaskada

Algoritam sam po sebi ima neke nelogičnosti. Ti se problemi rješavaju odgovarajućom kaskadom. Jedan od problema je taj da ako je objekt nevidljiv zbog okluzije u više uzastopnih slika, naknadne predikcije Kalmanovog filtera povećavaju pogrešku kod lociranja objekta. Mahalanobisova udaljenost kontra intuitivno favorizira veću grešku ako se dva pronađena objekta natječu za istu detekciju jer efektivno smanjuje udaljenost standardnih devijacija. Kako bi se taj problem riješio uvodi se kaskada kojom se prioritet daje češće viđenim objektima.

U početku se zadaju skup pronađenih objekata T i skup detekcija D te već spomenuta vrijednost varijable A_{\max} . Zatim se računa težinski zbroj $C(i, j)$ po zadanoj formuli i miču se malo vjerojatna povezivanja ovisno o tome jesu li ispunila zadani prag. Inicijaliziraju se pomoći skupovi M (početno prazan skup povezanih detekcija) i U (skup nepovezanih detekcija u kojem se početno nalaze sve detekcije iz D). Nakon toga iterira se po vrijednosti starosti pronađenih objekata n koja uzlazno ide od vrijednosti 1 do zadane vrijednosti A_{\max} . Za svaku vrijednost n uzima se podskup T_n u kojem se nalaze svi objekti iz skupa T koji nisu bili povezani s detekcijom u zadnjih n slika. Zatim se odvija postupak povezivanja, objašnjen u prethodnom potpoglavlju, nad skupovima T_n i nepovezanim detekcijama iz skupa U . Ažuriraju se skupovi nepovezanih detekcija U i povezanih detekcija M . Naposljetku se izvršava već spomenuti *intersection-over-union* (IoU) nad nepotvrđenim i nepovezanim objektima čija vrijednost starosti n iznosi 1.

Navedena kaskada daje prioritet pronađenim objektima čija je starost n manja, odnosno češće viđenim objektima.

Duboki deskriptor izgleda

Najveća razlika u odnosu na algoritam SORT je korištenje dubokog deskriptora izgleda. Za uspješnu primjenu algoritma Deep SORT, potrebna je kvalitetna ugradnja značajki izgleda.

Taj se korak odvija offline, prije upotrebe algoritma u stvarnom vremenu. Za izvlačenje značajki izgleda koristi se sijamska mreža koja je trenirana na velikom skupu podataka za ponovnu identifikaciju određenog objekta, a čija je struktura navedena u idućem poglavlju.

4.3. Priprema korištenih podataka

Prvi korak bio je pripremiti skup podataka za treniranje sijamske mreže. Za treniranje mreže korištena su dva manja skupa podataka. Dva su skupa kombinirana i slike posložene u foldere koji predstavljaju klase. Svaka je klasa jedna vrsta drona slikana iz različitih kutova i udaljenosti. Za stvaranje takvog skupa podataka za treniranje napravljene su kratke skripte dostupne na GitHub rezitoriju [19].

Prvi od dvaju skupova podataka sastoji se od 9000 slika bespilotnih letjelica te njihovih anotacija u YOLO formatu

```
< class_name > < center_x > < center_y > < width > < height > .
```

Skripta naziva *crop_images.py* prolazi kroz sve slike i kroz sve datoteke s njihovim anotacijama iz prvog skupa. Iz anotacija uzima x i y koordinate centra te širinu i visinu detektirane bespilotne letjelice. Ti se podaci jednostavnim računskim operacijama transformiraju u 2 koordinate - gornja lijeva i donja desna točka *bounding boxa*. Takav *bounding box* koji označava bespilotnu letjelicu se izrezuje iz početne slike. Slike izrezanih bespilotnih letjelica nakon rezanja se skaliraju te se ponovno spremaju.

crop_images.py

```
from PIL import Image
from os.path import isfile, join
from os import listdir

path_labels = "/home/dominik/PycharmProjects/create_dataset/
    ↪ labels"
path_images = "/home/dominik/PycharmProjects/create_dataset/
    ↪ JPEGImages"
path_cropped_img = "/home/dominik/PycharmProjects/
    ↪ create_dataset/cropped"

labels = [f[:-4] for f in listdir(path_labels) if isfile(join
    ↪ (path_labels, f))]

for label in labels:
    img = Image.open(f"{path_images}/{label}.jpg")
```

```
with open(f"{path_labels}/{label}.txt") as f:  
    contents = f.read().split()[1:]  
    contents = [float(coord) for coord in contents]  
  
if len(contents) == 0:  
    continue  
  
    x_dim = img.size[0]  
    y_dim = img.size[1]  
  
    x1 = contents[0] - contents[2] / 2  
    y1 = contents[1] - contents[3] / 2  
    x2 = contents[0] + contents[2] / 2  
    y2 = contents[1] + contents[3] / 2  
  
    x1 = int(x1 * x_dim)  
    y1 = int(y1 * y_dim)  
    x2 = int(x2 * x_dim)  
    y2 = int(y2 * y_dim)  
  
    img_res = img.crop((x1, y1, x2, y2))  
    img_res.save(f"{path_cropped_img}/{label}.jpg")  
  
    # img_res.show()
```

Drugi skup podataka korišten je iz javno dostupnog GitHub repozitorija [8] te se sastoji od slika i njihovih anotacija u formatu

```
< class_name > < x1 > < y1 > < width > < height > .
```

gdje x1 i y1 označavaju koordinate gornje-ljeve točke *bounding boxa*. Skripta naziva *create_dataset.py* koristi se za rezanje slika iz drugog skupa i njihovo spremanje u ranije spomenute klase.

create_dataset.py

```
import numpy as np
import scipy.io
from PIL import Image
import os
import re

def atoi(text):
    return int(text) if text.isdigit() else text

def natural_keys(text):
    return [atoi(c) for c in re.split(r'(\d+)', text)]

subfolders = [1, 2, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18,
    ↪ 19, 20, 26]

labels = {}
for subfolder in subfolders:
    mat = scipy.io.loadmat(f"/home/dominik/PycharmProjects/
        ↪ create_dataset/new_dataset/drive-download/{subfolder
        ↪ }/annotation.mat")
    labels[subfolder] = mat['box']

for subfolder, arr in labels.items():
    list_of_files = os.listdir(f"/home/dominik/PycharmProjects
        ↪ /create_dataset/new_dataset/drive-download/
        ↪ {subfolder}")
    list_of_files.sort(key=natural_keys)
    list_of_files.remove("annotation.mat")
```

```

for contents, pic in zip(arr, list_of_files):
    img = Image.open(f"/home/dominik/PycharmProjects/
        ↪ create_dataset/new_dataset/drive-download/
        ↪ subfolder}/{pic}"))

if not np.all(contents):
    continue

    x1 = contents[0]
    y1 = contents[1]
    x2 = contents[0] + contents[2]
    y2 = contents[1] + contents[3]

    img_res = img.crop((int(x1), int(y1), int(x2), int(y2))
        ↪ )

    img_res.save(f"/home/dominik/PycharmProjects/
        ↪ create_dataset/new_dataset/cropped/{subfolder}/{
        ↪ pic}"))

# img_res.show()

```



Slika 4.2: Primjeri originalnih (gore) i izrezanih (dolje) bespilotnih letjelica

Irezane slike dodatno su ručno presložene u 9 klasa, odnosno foldera od kojih se svaki sastoji od slika jedne vrste bespilotne letjelice. Treniranje je obavljeno pomoću konvolucijske sijamske mreže koja se sastoji od 7 konvolucijskih slojeva. Struktura mreže nalazi se u datoteci *siamese_net.py*.

```
import torch.nn as nn

class SiameseNetwork(nn.Module):
    def __init__(self):

        super(SiameseNetwork, self).__init__()

        self.net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(32),

            nn.Conv2d(32, 64, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(64),

            nn.Conv2d(64, 128, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(128),

            nn.Conv2d(128, 256, kernel_size=1, stride=2),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(256),

            nn.Conv2d(256, 256, kernel_size=1, stride=2),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(256),

            nn.Conv2d(256, 512, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),

            nn.Conv2d(512, 1024, kernel_size=1, stride=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(1024),

        )
```

Sijamska mreža više je puta istrenirana u raznim kombinacijama skupa podataka za treniranje, veličine batcha i broja epoha. Za svaku od kombinacija nacrtana je funkcija gubitka te su na temelju nje odabrani potencijalni modeli za testiranje i kasnije korištenje. Korištena funkcija gubitka u svakom modelu je *Triplet loss* opisana u 3. poglavlju. Neki od primjera korištenih trojki: *ancor-positive-negative* vide se na slici 4.3. Za testiranje je napravljen manji skup podataka koji se sastoji od 19 slika iz skupa podataka za treniranje koje su izdvojene prije treniranja. Slike su posložene u 9 klase kao i u skupu podataka za treniranje.



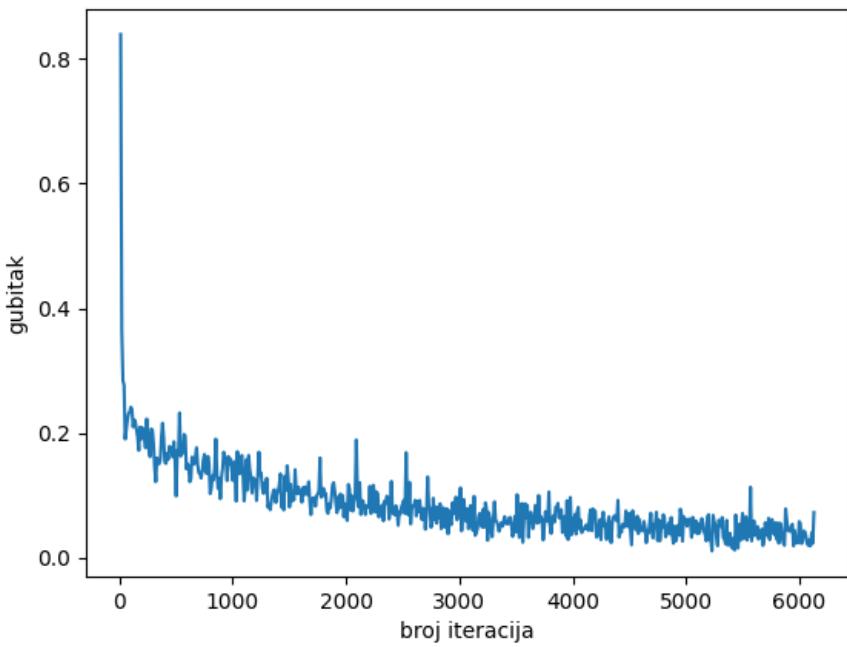
Slika 4.3: Primjeri korištenih trojki za treniranje sijamske mreže: *ancor-positive-negative*

Za odabir najbolje mreže korištena je datoteka *siamese_test.py* koja učitava skup podataka za testiranje i model. Iz skupa se uzima svaka od 19 slika kao referentni primjer te se računa sličnost s njezinim pozitivnim uzorkom. Ta se vrijednost uspoređuje s dobivenom sličnošću sa 16 preostalih negativnih uzoraka iz skupa za testiranje, što daje ukupno 304 kombinacije. Ako je sličnost s pozitivnim uzorkom veća od sličnosti s negativnim, smatra se da je taj rezultat točan. Jedna metrika za točnost modela računa se kao postotak točnosti kod svih 304 kombinacija (*examplewise correct %*), a druga izračunata metrika predstavlja postotak uspješnosti za svaki od odabranih referentnih primjera, tj. za svaku od 19 slika dobiveni se rezultat smatra točnim samo ako je svih 16 usporedbi s pozitivnim i negativnim primjerom dalo točan rezultat (*completely correct batches %*). Testiranje je provedeno za nekoliko modela po 20 puta kako bi se dobili precizniji rezultati, a najbolji rezultat dobiven je od mreže gdje je veličina batcha postavljena na 64 te je uzet model nakon 50. epohe. Rezultati navedenih metrika za pojedine modele vidljivi su u tablici 4.1.

Tablica 4.1: Rezultati različitih modela testiranih na skupu podataka za testiranje

Model (veličina batcha - broj epoha)	<i>completely correct batches</i>	<i>examplewise correct</i>
model 64 - 20	63.42%	84.14%
model 64 - 50	81.06%	91.47%
model 64 - 100	61.32%	88.19%
model 64 - 150	64.21%	85.59%
model 64 - 198	70.53%	86.27%
model 128 - 20	61.58%	82.00%
model 128 - 50	43.01%	84.26%
model 128 - 100	42.64%	76.65%
model 128 - 150	62.11%	83.87%
model 128 - 198	54.21%	80.00%

Postupak treniranja za korištenu mrežu trajao je približno sat vremena, a nakon treniranja mreže pomoću ispisanih i spremlijenih rezultata nakon svaka 2 batcha iscrtana je funkcija gubitka koja se nalazi na slici 4.4.



Slika 4.4: Funkcija gubitka korištene mreže

4.4. Testiranje

Za testiranje algoritma Deep SORT korišten je GitHub repozitorij [13] u kojem se nalazi originalni kod [20] pisan u Pythonu. U njemu se dodatno nalaze i ranije spomenute datoteke za treniranje i testiranje sijamske mreže te datoteka *test_on_video.py* u kojoj se nalazi glavna funkcija (engl. *main*) koja se poziva. Algoritam je izmijenjen na način da je implementiran u ROS-u. ROS je robotski operacijski sustav otvorenog koda koji je napravljen kako bi pojednostavio rad s robotima. U glavnoj funkciji učitava se istrenirana sijamska mreža iz prošlog poglavlja i videozapis s detekcijama te se nad njima testira algoritam. Videozapis i detekcije učitavaju se preko *rospy* biblioteke. *Rospy* je ROS-ova biblioteka koja sadrži podršku za ROS čvorove za primanje (engl. *subscriber*) i pisanje (engl. *publisher*) poruka. Čvorovi predstavljaju izvršni dio koji je povezan na ROS mrežu. Čvor za primanje poruke služi za pretplaćivanje na željenu temu (engl. *topic*) i čitanje poruke s nje. Svaki put kada je neka poruka primljena poziva se *callback* funkcija koja je zadana imenom kao argument prilikom stvaranja čvora. Funkcija kao prvi argument prima poruku u obliku koji je također zadan kao argument kod stvaranja čvora. Čvorovi za pisanje poruke služe za slanje poruke na zadanu temu. Kod stvaranja čvora za pisanje kao argument se zadaje ime teme, oblik poruke te veličina reda u koji se stavljaju poruke ako ih čvor koji čita te poruke ne stigne dovoljno brzo primiti.

U glavnoj funkciji *test_on_video.py* datoteke stvorena su dva čvora za čitanje i jedan čvor za pisanje. Jedan od čvorova za čitanje učitava sliku odnosno videozapis, a drugi učitava detekcije bespilotnih letjelica s korištenog videozapisa u obliku koordinata. Detekcije se zatim provlače kroz algoritam. Čvor za pisanje koristi se za ispis praćenih objekata koji se dobivaju iz algoritma Deep SORT.

poziv *main* funkcije

```
if __name__ == '__main__':
    boolean = False
    tracker = None
    cv_image = None
    detections = []

    rospy.init_node('sort', anonymous=True)
    subscriber = rospy.Subscriber("/YOLODetection/
        ↪ detected_objects", objectList, callback)
    image_sub = rospy.Subscriber("/zedm/zed_node/rgb/
        ↪ image_rect_color/compressed", CompressedImage,
        ↪ image_callback)
    pub = rospy.Publisher('/trackers', objectList, queue_size
        ↪ =10)

    r = rospy.Rate(10)
    deepsort = deepsort_rbc(wt_path='/home/dominik/
        ↪ catkin_ws_deep_sort/src/deep_sort/src/
        ↪ model_new2_200_6450.pt')

while not rospy.is_shutdown():

    if boolean and (len(detections) != 0) and cv_image is
        ↪ not None:
        detections = np.array(detections)
        out_scores = np.array(out_scores)
        tracker, detections_class = deepsort.run_deep_sort(
            ↪ cv_image, out_scores, detections)
        boolean = False
        boolean2 = True

    r.sleep()
```

Osim stvaranja čvorova u glavnoj funkciji potrebno je inicijalizirati ROS čvor koji ima svoje jedinstveno ime. Također se zadaje *rospy.Rate* koji određuje frekvenciju na kojoj će se vrtjeti *while* petlja. Detekcije se u ovom slučaju s teme čitaju frekvencijom od približno 10 Hz pa je zbog optimalnog izvršavanja algoritma tako postavljeno i u kodu. Unutar *while* petlje poziva se funkcija za izvršavanje koraka algoritma Deep SORT te se postavljaju zastavice za sinkronizaciju s *callback* funkcijama.

Prilikom čitanja s teme /YOLODetection/detected_objects poziva se funkcija *callback* te se unutar funkcije prolazi po svim primljenim detekcijama i njihovim vjerovatnostima te ih se stavlja u liste.

callback funkcija

```
def callback(data):
    global detections
    global out_scores
    global boolean

    detections = []
    out_scores = []

    for i in range(len(data.obj)):
        bb = data.obj[i]

        detections.append([bb.x, bb.y, bb.width, bb.height])
        out_scores.append(bb.confidence)

    #rospy.loginfo(detections)
    #rospy.loginfo(out_scores)
    boolean = True
```

Čitanjem s teme /zedm/zed_node/rgb/image_rect_color/compressed poziva se funkcija *image_callback* u kojoj se prolazi po svim dobivenim objektima te ih se iscrtava na slici u obliku *bounding boxova* i zapisuje na temu /trackers. Zbog vizualne usporedbe na slici se iscrtavaju i početne učitane detekcije.

image_callback funkcija

```
def image_callback(data):
    global tracker
    global detections_class
    global cv_image
    global boolean2

    bridge = CvBridge()
    cv_image = bridge.compressed_imgmsg_to_cv2(data,
        ↳ desired_encoding='bgr8')

    if tracker is not None and boolean2:
        lista_trackers = objectList()
        for track in tracker.tracks:
            if not track.is_confirmed() or track.
                ↳ time_since_update > 1:
                    continue

            bbox = track.to_tlwh()
            tmp_obj = object()
            tmp_obj.x = bbox[0]
            tmp_obj.y = bbox[1]
            tmp_obj.width = bbox[2]
            tmp_obj.height = bbox[3]
            lista_trackers.obj.append(tmp_obj)

            # Draw bbox from tracker.
            bbox = track.to_tlbr()
            cv2.rectangle(cv_image, (int(bbox[0]), int(bbox[1]))
                ↳, (int(bbox[2]), int(bbox[3])), (255, 255,
                ↳ 255), 2)
            cv2.putText(cv_image, str(track.track_id), (int(bbox
                ↳ [0]), int(bbox[1])), 0, 5e-3 * 200, (0, 255,
```

```

    ↪ 0), 2)
boolean2 = False

for det in detections_class:
    bbox = det.to_tlbr()
    cv2.rectangle(cv_image, (int(bbox[0]), int(bbox
        ↪ [1])), (int(bbox[2]), int(bbox[3]))
        ↪ , (255,255,0), 2)

    pub.publish(lista_trackers)

cv2.imshow("sort", cv_image)
cv2.waitKey(1)

```

4.5. Rezultati i usporedba s algoritmom SORT

Algoritam Deep SORT testiran je na dva različita videozapisa. Oba videozapisa provučena su i kroz algoritam SORT kako bi se napravila usporedba. Prvi stupac tablica 4.2 i 4.3 čine mjere, a u ostalim stupcima nalazi se dobiveni rezultati validacija za svaki od algoritama.

Vrednovanje modela:

- **Different tracks count :**
broj objekata s različitim identitetom koji su praćeni kroz videozapis
- **Longest track length :**
broj koji predstavlja u koliko je slika tijekom videozapisa najpraćeniji objekt bio prepoznat
- **Average track length :**
broj koji predstavlja koliki je prosječan broj slika u kojima se objekt s određenim identitetom prepoznaže
- **Max track id :**
najveći id praćenog objekta u videozapisu. Id u algoritmu kreće od 0 i povećava se svaki put kada se objekt ne uspije povezati s detekcijom
- **Reidentified tracks count :**
broj koji predstavlja koliko puta je objekt s određenim identitetom bio pronađen nakon što je ranije "nestao" iz slike

U oba videozapisa prati se po jedna bespilotna letjelica koja bi u idealnoj situaciji trebala imati isti identitet kroz cijeli videozapis. Tablica 4.2 prikazuje dobivene rezultate za prvi videozapis koji prima detekcije frekvencijom od približno 25 Hz. Rezultati su očekivani, ali slabije izraženi nego u drugom videozapisu. Za algoritam Deep SORT vidljiv je pad broja različitih identiteta praćene bespilotne letjelice od približno 34%. Najduže praćeni objekt je praćen kroz gotovo duplo više slika nego u algoritmu SORT te je prosječna duljina praćenja određenog objekta povećana za otprilike 60%. Jedna od ključnih razlika između dvaju algoritama je ta da se u Deep SORT-u određeni objekt može ponovno identificirati nakon što bude izgubljen u slici. To ponovno identificiranje događa se na temelju značajki izgleda uz pomoć sijamske mreže.

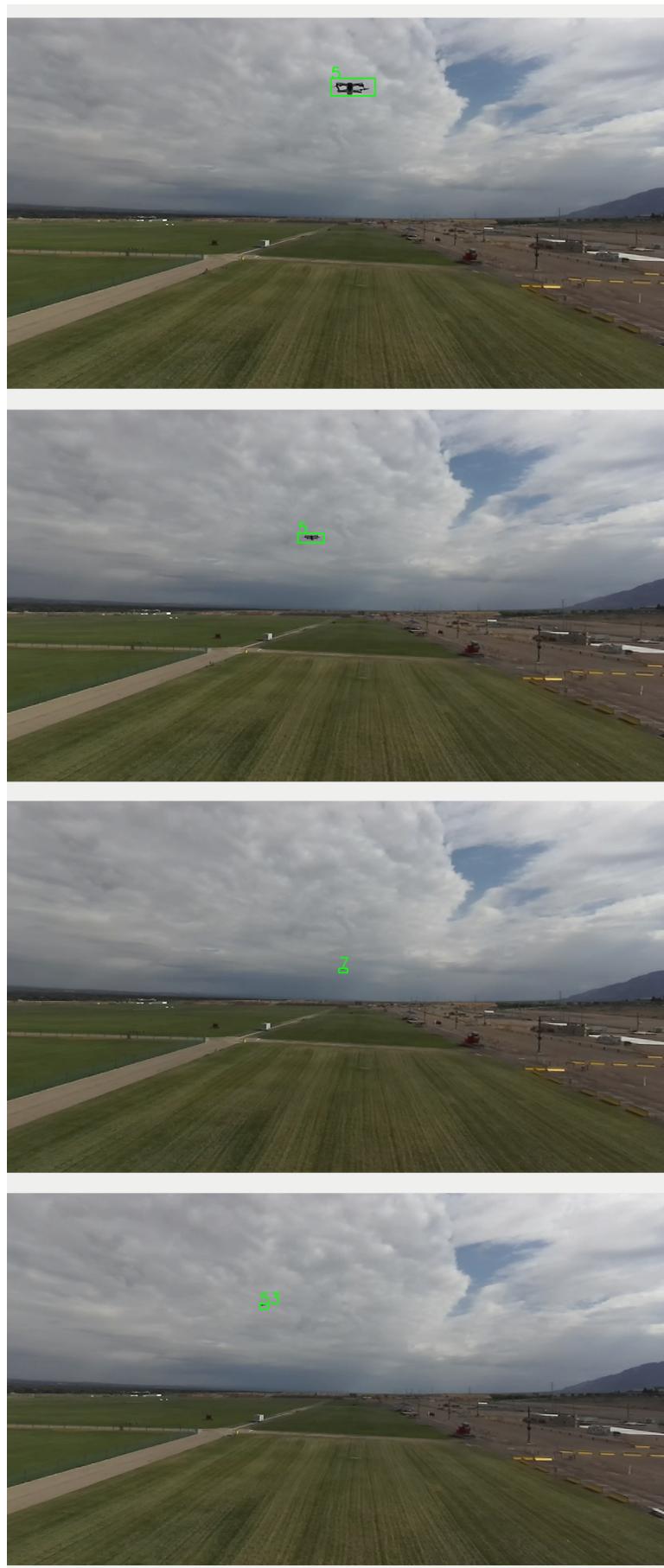
Tablica 4.2: Metrike testiranih algoritama za 1. videozapis

Mjera	SORT	Deep SORT
<i>Different tracks count</i>	52	34
<i>Longest tracking length</i>	40	78
<i>Average tracking length</i>	7.76	12.41
<i>Max track id</i>	86	68
<i>Reidentified tracks count</i>	0	19

Tablica 4.3 prikazuje rezultate dobivene testiranjem na drugom videozapisu čije su razlike između dvaju algoritama izraženije. U njemu se detekcije primaju frekvencijom od približno 10 Hz. Broj različitih identiteta u algoritmu Deep SORT pao je za 53% u odnosu na algoritam SORT. Najduže praćeni objekt praćen je u čak 73% slika više nego u algoritmu SORT, dok je prosječna duljina praćenog objekta gotovo 3 puta veća u algoritmu Deep SORT. Također je u algoritmu Deep SORT došlo do 37 reidentifikacija ranije izgubljenog objekta.

Tablica 4.3: Metrike testiranih algoritama za 2. videozapis

Mjera	SORT	Deep SORT
<i>Different tracks count</i>	38	18
<i>Longest tracking length</i>	208	360
<i>Average tracking length</i>	14.05	41.78
<i>Max track id</i>	60	40
<i>Reidentified tracks count</i>	0	37



Slika 4.5: Rezultati algoritma SORT na videozapisu



Slika 4.6: Rezultati algoritma Deep SORT na videozapisu

5. Zaključak

U ovom diplomskom radu prikazani su problemi detekcije i praćenja objekata unutar područja umjetne inteligencije. Predstavljen je detaljan pregled nekih od najpoznatijih metoda za njihovo rješavanje te su prikazani njihovi aktualni izazovi. Za izvedbu ovog rada odabran je algoritam Deep SORT. Algoritam koristi konvolucijsku neuronsku mrežu koja je također predstavljena u radu. Algoritam Deep SORT je objašnjen i implementiran.

Za treniranje i testiranje korišteni su javno dostupni skupovi podataka. Cilj je bio pratiti bespilotnu letjelicu unutar videozapisa te ju identificirati kao jedinstveni objekt s određenim identitetom kroz što veći dio videozapisa. Algoritam je uspoređen s originalnom verzijom algoritma SORT koji je također implementiran i testiran na istim podacima. Napredak algoritma Deep SORT u odnosu na SORT je vidljiv, ali ne preizražen na korištenim testnim podacima. Razlog tome je taj što je algoritam Deep SORT predviđen za situacije u kojima se prati više objekata te dolazi do okluzija što u korištenim videozapisima nije slučaj jer se prati po jedna bespilotna letjelica bez okluzija.

U eventualnom dalnjem radu preporučuje se napraviti kvalitetniji skup podataka za treniranje s više različitih klasa i više slika unutar svake.

LITERATURA

- [1] . URL <https://viso.ai/deep-learning/object-detection/>.
- [2] . URL <https://www.v7labs.com/blog/object-tracking-guide>.
- [3] . URL <https://nanonets.com/blog/object-tracking-deepsort/>.
- [4] . URL <https://viso.ai/deep-learning/object-detection/>.
- [5] Shivaank Agarwal, Ravindra Gudi, i Paresh Saxena. Application of computer vision techniques for segregation of plasticwaste based on resin identification code, 11 2020.
- [6] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, i Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. URL <http://arxiv.org/abs/1602.00763>.
- [7] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, i Ropak Shah. Signature verification using a "siamese" time delay neural network. U J. Cowan, G. Tesauro, i J. Alspector, urednici, *Advances in Neural Information Processing Systems*, svezak 6. Morgan-Kaufmann, 1993. URL <https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf>.
- [8] Yueru Chen. URL <https://github.com/chelicynly/A-Deep-Learning-Approach-to-Drone-Monitoring>.
- [9] N. Dalal i B. Triggs. Histograms of oriented gradients for human detection. U *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, svezak 1, stranice 886–893 vol. 1, 2005. doi: 10.1109/CVPR.2005.177.
- [10] Rohith Gandhi. URL <https://towardsdatascience.com/siamese-network-triplet-loss-b4ca82c1aec8>.
- [11] David Held, Sebastian Thrun, i Silvio Savarese. Learning to track at 100 FPS with deep regression networks. *CoRR*, abs/1604.01802, 2016. URL <http://arxiv.org/abs/1604.01802>.

- [12] Sean Benhur J. URL <https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942>.
- [13] Shishira R Maiya. github-repository. URL https://github.com/abhyantrika/nanonets_object_tracking.
- [14] Michael Motro i Joydeep Ghosh. Measurement-wise occlusion in multi-object tracking. U *2018 21st International Conference on Information Fusion (FUSION)*, stranice 2384–2391, 2018. doi: 10.23919/ICIF.2018.8455339.
- [15] Hyeonseob Nam i Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. U *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 4293–4302, 2016. doi: 10.1109/CVPR.2016.465.
- [16] Kedar Potdar, Chinmay Pai, i Sukrut Akolkar. A convolutional neural network based live object recognition system as blind aid. 11 2018. doi: 10.13140/RG.2.2.34494.54085.
- [17] P. Viola i M. Jones. Rapid object detection using a boosted cascade of simple features. U *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, svezak 1, stranice I–I, 2001. doi: 10.1109/CVPR.2001.990517.
- [18] D. Vrbanic. Algoritam deep sort; diplomski projekt. 2022.
- [19] Dominik Vrbanić. URL https://github.com/dvrbanic/create_dataset.
- [20] Nicolai Wojke, Alex Bewley, i Dietrich Paulus. github-repository. URL https://github.com/nwojke/deep_sort.
- [21] Nicolai Wojke, Alex Bewley, i Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. URL <http://arxiv.org/abs/1703.07402>.

Praćenje višestrukog broja objekata jedne klase uz tehniku re-identifikacije

Sažetak

U ovome diplomskom radu nabrojane su i ukratko objašnjene najsuvremenije tehnologije za praćenje više pokretnih objekata u slici. Također su predstavljeni neki od izazova u tom području. Odabran je jedan od algoritama, Deep SORT, koji koristi neuronske mreže kako bi pratio objekte u stvarnom vremenu. Algoritam je implementiran i testiran na problemu praćenja bespilotnih letjelica.

Ključne riječi: praćenje objekata, neuronske mreže, sijamske neuronske mreže, stvarno vrijeme, bespilotna letjelica, Deep SORT

Tracking multiple objects of one class with re-identification technique

Abstract

In this diploma thesis, the most modern technologies for tracking several moving objects in the image are listed and briefly explained. Some of the challenges in this area are also presented. One of the algorithms, Deep SORT, was chosen, which uses neural networks to track objects in real time. The algorithm was implemented and tested on the problem of unmanned aerial vehicles tracking.

Keywords: object tracking, neural networks, siamese neural networks, real time, unmanned aerial vehicle, Deep SORT