Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

PROJEKTNO DELO - UI

Študenti:
Viktor Vlahek
Gašper Hriberšek
Luka Knez

Datum in mesto:
09.06.2023, Maribor

Mentor: dr. Matej Črepinšek

Koordinator: dr. Martin Šavc

KAZALO

PREDSTAVITEV ČLANOV SKUPINE BLOP-AI

Viktor Vlahek je naš vodja skupine. Je študent, ki se je ukvarjal z dronsko fotografijo parkirišč in ocenjevanje pravilnosti parkiranja na podlagi zbranih podatkov.

Gašper Hriberšek je študent, ki se osredotoča na prepoznavanje prometnih znakov in pravilno reakcijo v skladu s prometnimi predpisi. Njegovo znanje nam pomaga prepoznati ali je parkiranje v skladu s pravili.

Luka Knez je študent, ki se ukvarja s zajemanjem ovir na cestišču s pomočjo iPhone Lidar senzorja. Njegova vloga je pomembna pri zagotavljanju dodatne varnosti med parkiranjem z zaznavanjem ovir in njihovim pravilnim upoštevanjem.

IDEJA PROJEKTA

Na podlagi pregledov in predlogov našega mentorja smo se kot skupina odločili, da se osredotočimo na parkiranje z uporabo umetne inteligence, s ciljem izboljšati izkušnjo parkiranja. Naš glavni cilj je izboljšati proces parkiranja z izkoristitvijo edinstvenega nabora spretnosti vsakega člana skupine. Moj poseben prispevek k projektu je razvoj algoritma, ki bo zmožen natančno prepoznati parkirne črte, parkirne prostore ter vozila, na podlagi posnetkov z dronom iz ptičje perspektive. Medtem se moji soigralci osredotočajo na detekcijo parkirnih znakov ter zagotavljanje dodatne varnosti s pomočjo senzorjev LiDAR.

IZBOLJŠANJE VARNOSTI NA CESTAH IN POMOČ VOZNIKOM Z DOMA NAREJENIM DRONOM IN LIDAR TEHNOLOGIJO IPHONE 12 PRO:

Naš univerzitetni projekt je bil posvečen izkoriščanju zanimivih tehnologij (kot sta iPhone LiDAR in Quadcopter) za ustvarjanje celovitega sistema, namenjenega izboljšanju varnosti na cestah, pomoči voznikom pri prepoznavanju ovir in prometnih znakov ter izboljšanje dostopnosti za voznike z različnimi potrebami. Še posebej smo se osredotočili na inovativno parkiranje vozila, ki javlja šoferju vozila kako dobro je avto parkiran. Tako šoferju omogoča boljši pogled na avtomobil iz ptičje perspektive in mu olajša samo parkiranje. Program prav tako javlja zvočna sporočila če se na cesti pojavi višinska ovira, kar lahko pripravi šoferja na nepredvidene ovire (robniki, kamenje …). Naš celoten izdelek je namenjen vsem voznikom, ki želijo boljši nadzor nad razmerami na cesti in zanesljiv pripomoček pri parkiranju.

"ParkGuard Pro: partner za natančno parkiranje"

V svetu, kjer se zadnje čase povečuje število oseb z raličnimi limitacijami, in posebnimi potrebami. Smo razvili aplikacijo, ki poveča obvestila med vožnjo in uporabniku sporoča boljši pretok podatkov nad parkiriščem.

Naša tehnološko napredna rešitev je rezultat večmesečnega truda. Ne samo, da preprečuje nesreče med parkiranjem s pametno zaznavo prekoračenja robnikov, temveč tudi omogoča udobno izstopanje iz vozila invalidom s prepoznavo bližnjih ovir. Vendar "ParkGurd Pro" sega dlje od parkiranja. Zaznava prometne ovire, vas opozarja na nepravilno parkiranje na mestih za invalide in vam omogoča jasen vpogled v razmere na cesti.

Skupaj smo ustvarili „ ParkGuard Pro “ ki se postavlja kot zanesljiv sopotnik na cesti, prinašavečjovarnost, udobje in neodvisnost vsem voznikom, ne glede na njihove posebne potrebe.

Testirani programi delujejo na naslednjih testiranih okoljih:

ParkGuard pro je skupek večih različnih programo, ki so preverjeni in delujejo v naslednjih okoljih:

- **ParkGuard Pro** je testiran in deluje zanesljivo na M1 virtualki z Ubuntu preko Parallels za Mac.
- Testirali smo ga tudi na virtualizaciji VMware na računalniku z i7 6700K procesorjem.
- Prav tako smo ga stestirali na procesorju Ryzen 7 4700u z nativnim sistemom Linux Ubuntu in potrdili da deluje vredu.

To so preizkušene konfiguracije nad katerimi delujejo. Naj omenimo da Yolo ni podprt na novejših m1 procesorjih in smo zaradi tega trening izvedli na sistemu z zgoraj navedenimi Intelovimi in AMDjevimi procesorji. Program ni specifičen na zgoraj navedene procesorje in zagotovo deluje tudi na ostalih sistemih kot so (i7 2700k, ryzen 7 1600x …) nemoremo pa potrditi, da deluje na kakšnih arm procesorji. Prav tako je pogoj za delovanje programa uporaba operacijskega sistema Linux z ustreznimi knjižnicami. Te knjižnice so: **Pytorch, numpy, Kafka, Redis …** .

Praktični primeri uporabe :

1

Izogibanje neprijetnim situacijam:

Voznik želi varno in natančno parkirati vozilo ter se izogniti neprijetnim situacijam, kot je prekomerno prehajanje robnikov.

Koraki izvedbe:

Med parkiranjem "SmartGuard Pro" zazna oviro na sivinski sliki in javi zvočno opozorilo, češ da se približuje robnik.

Rezultat: Z uporabo "SmartGuard Pro" rešitve voznik zagotavlja natančno in varno parkiranje. Zvočna opozorila preprečujejo nepotrebno prehajanje robnikov, kar preprečuje morebitne poškodbe pnevmatik in okvar vozila. S tem je zagotovljeno, da bo vozilo pravilno postavljeno na parkirnem mestu, brez neprijetnih in nerodnih situacij.

2
Izkrcanje invalidov:

Voznik z invalidnostjo želi udobno in varno izstopiti iz vozila na parkirnem mestu ter se prepričati, da ni ovir za izkrcanje.

Med izkrcanjem invalidne osebe "SmartGuard Pro" zazna oviro na sivinski sliki in javi zvočno opozorilo, češ da odpiranje vrat in varno izkrcanje invalidne osebe ni mogoče (morda je kakšna ovira, ni dovolj prostora?).

3
Ocena Kakovosti Parkiranja in Opozorilo o Morebitni Nevarnosti

Voznik si želi natančno oceniti, kako dobro je vozilo parkirano, ter preprečiti morebitno oviranje drugih vozil ali nevarnost pri parkiranju.

"SmartGuard Pro" analizira pozicijo in usmeritev vozila glede na parkirno mesto. Zraven javlja primere če se zgodi, da je avto parkiran preko črte in to javi.

Prav tako, če se bo zaznala morebitno nevarnost ali ovira, ki bi lahko privedla do poškodbe vozila ali oviranja drugih vozil, bo sprožila zvočno opozorilo.

Voznik bo imel popoln pregled nad kakovostjo parkiranja svojega vozila. "SmartGuard Pro" bo zagotovil, da je vozilo pravilno postavljeno na parkirno mesto, hkrati pa bo preprečil morebitno nevarnost ali oviranje drugih vozil. S tem se povečuje varnost parkiranja in zmanjšuje tveganje za morebitne trke ali neprijetnosti.

4
Zaznavanje prometnih znakov:

Voznik želi prepoznati in razumeti prometne znake na cesti, da bi bolje upošteval prometne predpise in varno nadaljeval pot.

"SmartGuard Pro" bo prepoznal prometne znake na cesti, kot so omejitve hitrosti, prepovedi ali obveznosti.
Prepoznani prometni znaki se bodo prikazali na zaslonu vašega pametnega telefona, skupaj z okvirjem kjer se znak nahaja.

Rezultat: Voznik bo imel realnočasovno informacijo o prometnih znakih na cesti, kar bo prispevalo k boljšemu spoštovanju prometnih predpisov. "SmartGuard Pro" omogoča voznikom, da varno in samozavestno nadaljujejo pot ter se izognejo morebitnim kaznim ali nevarnim situacijam, povezanim s kršenjem prometnih predpisov.

UPORBLJENA ORODJA:

Pri našem projektu smo uporabljali različna orodja in vzpostavili ustrezno okolje za delo. Za komunikacijo med člani skupine in mentorjem smo uporabljali **Microsoft Teams**, ki nam je omogočal enostavno izmenjavo sporočil, organizacijo sestankov in deljenje dokumentov.

> Za shranjevanje podatkov smo vzpostavili platforme, kot je **GitHub**, kjer smo hranili in upravljali naše projekte. Za vodenje projekta in sledenje nalog smo uporabili **Jiro**, ki nam je omogočal sledenje napredku, dodeljevanje nalog ter učinkovito organizacijo dela.

Programski jezik, ki smo ga uporabljali za implementacijo naših rešitev, je bil **Python**. Python je bil izbran zaradi svoje prilagodljivosti, enostavnosti uporabe ter obsežne podpore za različne knjižnice in orodja, ki smo jih potrebovali v projektu.

Poleg tega smo se zanašali na naslednja orodja in tehnologije: **YOLO** za detekcijo objektov, **Grafana** za vizualizacijo podatkov, **Redis** kot hitro podatkovno skladišče, **Kafka** kot platformo za pretok podatkov in sporočil, **OpenCV** za obdelavo slik ter **Blender** za ustvarjanje in manipulacijo 3D modelov. Ta orodja so nam pomagala pri razvoju in implementaciji naših rešitev v okviru projekta.

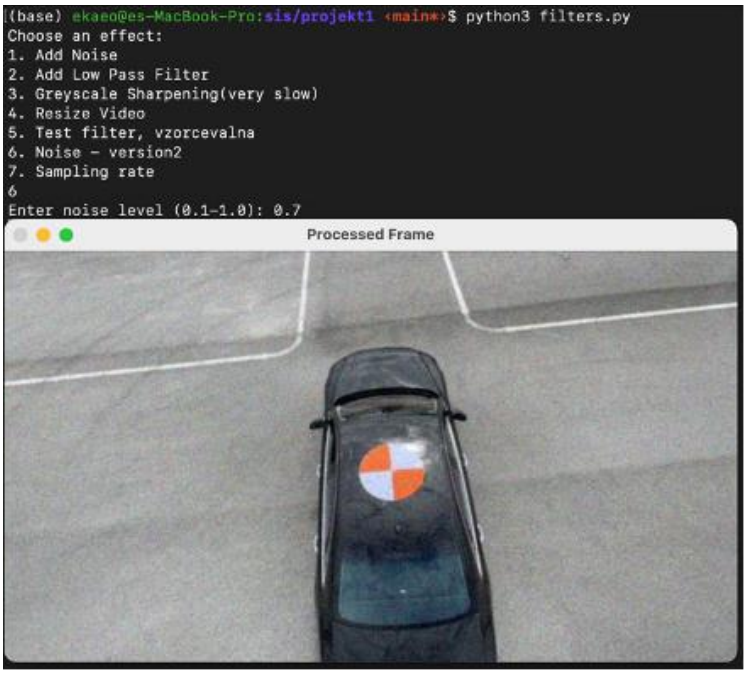
PREDMET: SIGNALI IN SLIKE

UVOD:

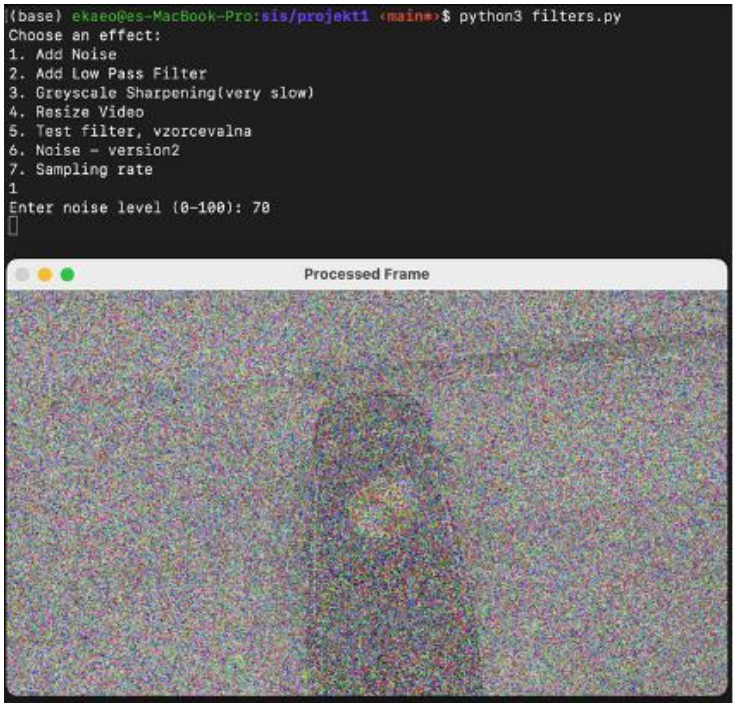
Cilj izdelave tega projekta je bil izdelan z namenom simulacije prenašanja podatkov iz senozarja LiDAR na sistem REDIS kjer ga lahko nekako prikažemo. Sestavljen je iz proizvajalca in odjemalca. Proizvajalec zajema podatke o položaju vozlišč v 3D modelu in jih pošilja odjemlaci preko Apache Kafka sporočilnega sistema. Potrošnik prejema sporočila in in, obdeluje podatke in prikazuje 2D vizualizacijo spremenjivega položaja vozlišč.

VIDEO PREDVAJALNIK S POMOČJO KAFKE IN REDISA

Ustvarili smo video predvajalnik z namenom simuliranja realnega časovnega pretakanja videa iz kamere v sistem Redis.

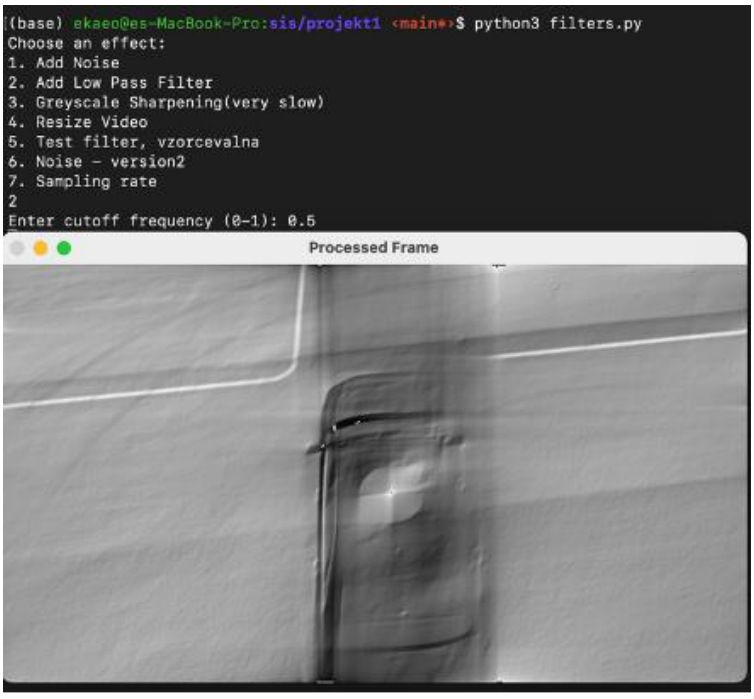


Nato smo sledili z dodajanjem šuma videoposnetkom. Pri strojnem učenju je ključno iz več razlogov. En razlog je, da smo želeli simulirati realne pogoje in preizkusiti obnašanje algoritma YOLO ob prisotnosti šuma. S tem smo pridobili vpogled v to, kako dobro se YOLO spopada s šumom v video posnetkih.
Dodajanje šuma nam tudi pomaga preprečiti prekomerno prilagajanje modela na specifične vzorce, ki se pojavljajo le v čistih posnetkih. To prispeva k boljšemu splošnemu delovanju modela na šumnih ali manj popolnih video posnetkih v resničnem svetu.



Nizkoprepustni filter smo dodali z namenom zmanjšanja šuma in izboljšanja kakovosti videoposnetkov. Njegova glavna funkcija je zmanjšanje visokih frekvenc, kar omogoča ohranjanje bolj pomembnih nizkofrekvenčnih informacij, medtem ko se odstranjujejo visokofrekvenčni šumovi in nepomembne podrobnosti

Dodali smo tudi postopek Grayscale sharpening, ki je namenjen izboljšanju ostrine in kontrasta v videoposnetkih. Ta tehnika temelji na povečanju razlike med sosednjimi slikovnimi piksli v črno-beli sliki, kar vodi do bolj izrazitih robov, bolj definiranih podrobnosti in večje vizualne jasnosti.



SIMULACIJA POŠILJANJA LIDAR PODATKOV PREKO KAFKE

TEŽAVE:

Največja težava pri izdelavi tega projekta je ta, da je iz oblaka točk, ki ga zajema LiDAR zelo težko implementirati nek video predavajalnik.
Sinhronizacija izrisa slike na strani odjemalca in toka prejetih informacij iz strani odjemalca.
Zaradi počasnosti grajenja slike, saj se slika za vsako poslano točko obnavlja je to časovno potrošno delanje. (meni se je do dokončnega rezultata slika gradila okoli 4 ure).

CILJI PROJEKTA:

Pri tej projektni nalogi je moj cilj implementirati nekaj zelo podobnega vendar ne čisto enakega klasičnemu predvajalniku. Cilj naloge je simulirati kako LiDAR v realnem času pošilja točke odjemalcu, ki na podlagi prejetih podatkov gradi višinsko sliko. To bi bilo zelo uporabno pri samem avtomobilu, kjer informacije o okolju tečejo v neko podatkovno bazo in se izvaja njihova obdelava.

LOGIKA:

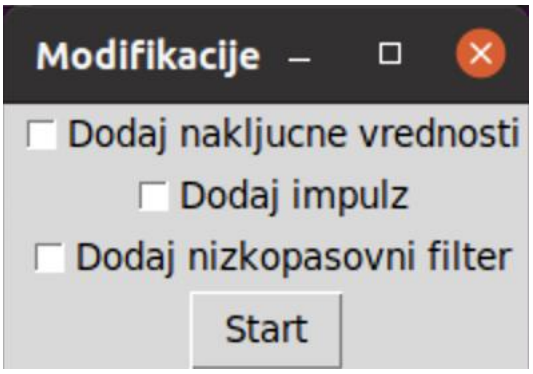
Proizvajalec :

Proizvajalec je odgovoren za zajem položaja vozlišč iz 3D modela, obdelavo podatkov ter pošiljanje sporočil preko Apache Kafka sporočilnega sistema. Osnovne funkcionalnosti proizvajalca vključujejo:

- Branje 3D modela in obdelava modela prebranega 3D modela v formatu .obj s pomočjo knjižnjice Trimesh.

- Razvrščanje podatkov na manjše paketke, ki se nato pošiljajo v Redis is katerga jih bo odjemalec prebral.

- Grafični vmesnik, ki uporabniku omogoča manipulacijo podatkov.



Slika 1: Izgled prikazuje okno modifikacij.

MANIPULACIJA 3D PODATKOV :

Cilj manipulacije podatkov je povzeti znanje iz predmeta Signalov in Slik in ga uporabiti na praktičnem primeru, kjer lahko popačimo prejete podatke, ki se pošiljajo v Redis. Tukaj so implementirane funckije za manipulacijo podatkov:

- **Generiranje naključnih vozlišč :**
  Ta možnost omogoča da proizvajalec generira naključna vozlišča namesto branja iz 3D modela. To poeni, da bodo namesto dejanskih podatkov o položaju vozlišč uporabljena naključno ustvarjena vozlišča z enakomerno porazdeljenimi koordinatami znotraj določenega območja,

- **Dodajanje impulza :**
  Ta možnost omogoča, da se vertexi 3D modela pričnejo množiti z pred ustvarjenim vektorjem, ki je sestavljen iz naključno generiranih števil znotraj območja -1 <-> 1. Vsakič ko je na vrsti nova točka se uporabi naslednji impulz iz vektorja. S tem dosežem spremenljivost gibanja vozlišč med prehajanjem skozi impulze,

- **Uporaba nizkofrekvenčnega filtra :**
  Ta možnost omogoča uporabo nizkofrekvenčnega filtra na položaje vozlišč. Ta filter omogoča zmanjšanje visokofrekvenčnega šuma ali sprememb položajev vozlišč. Filtriranje se izvede s pomočjo Butterjevga filtra, ki se uporabi na vsako vozlišče posebej,

```
v 5.783198 0.001900 1.090471
v 5.809988 0.002100 1.083357
v 5.792970 0.001600 1.073494
v 5.800084 0.002700 1.100285
v 5.809988 0.002100 1.083357
v 5.783198 0.001900 1.090471
v 5.792970 0.001600 1.073494
v 5.819851 0.000900 1.066339
v 5.802923 0.001000 1.056434
v 5.809988 0.002100 1.083357
v 5.819851 0.000900 1.066339
v 5.792970 0.001600 1.073494
v 5.802923 0.001000 1.056434
v 5.829756 0.000700 1.049411
v 5.812737 -0.000500 1.039548
v 5.819851 0.000900 1.066339
v 5.829756 0.000700 1.049411
v 5.802923 0.001000 1.056434
v 5.812737 -0.000500 1.039548
v 5.839528 0.000200 1.032434
v 5.822642 -0.000400 1.022620
v 5.829756 0.000700 1.049411
v 5.839528 0.000200 1.032434
v 5.812737 -0.000500 1.039548
v 5.822642 -0.000400 1.022620
v 5.849482 -0.000700 1.015374
v 5.832505 -0.001400 1.005602
v 5.839528 0.000200 1.032434
v 5.849482 -0.000700 1.015374
v 5.822642 -0.000400 1.022620
v 5.832505 -0.001400 1.005602
v 5.859295 -0.001000 0.998488
v 5.842368 -0.002100 0.988584
v 5.849482 -0.000700 1.015374
v 5.859295 -0.001000 0.998488
v 5.832505 -0.001400 1.005602
v 5.842368 -0.002100 0.988584
v 5.869158 -0.001400 0.981470
v 5.852182 -0.002200 0.971698
v 5.859295 -0.001000 0.998488
```

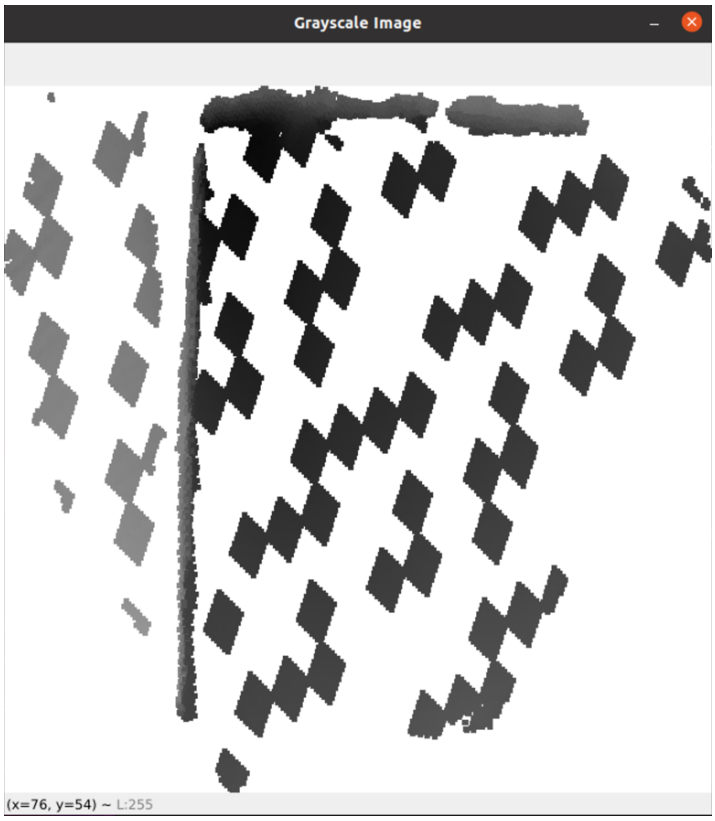
Slika 2: prikazuje lokacije vozlišč v vhodnem modelu .obj formata.

```
(.venv) parallels@ubuntu-linux-20-04-desktop:~/Desktop/Kafka$ python Pro_3D.py
ID poslanega vertexa: 0: [-0.6705267285858498, 0.27113077173856404, 0.08392678423896682]
ID poslanega vertexa: 1: [-0.21855578380414098, 0.2500272139982367, -0.9718902749600518]
ID poslanega vertexa: 2: [0.2919559314710505, -0.47571180070899044, 0.5213457892540214]
ID poslanega vertexa: 3: [0.1341640404199027, -0.5651532889009763, 0.5708568426380576]
ID poslanega vertexa: 4: [-0.6376081463397583, -0.9147372887843457, -0.844023907941341]
ID poslanega vertexa: 5: [0.584405767501768, -0.9767544560691697, 0.5301718664085617]
ID poslanega vertexa: 6: [-0.7067493463587817, 0.5825800285542164, -0.03429000253826375]
```

Slika 3: prikazuje vrednosti, ki so bile alternirane z opcijo dodajanja vrednosti.

Potrošnik:

Potrošnik je odgovoren za sprejemanje sporočil iz protokola Apache Kafka. Iz prebranih sporočil izlišči vozlišča in začne ustvarjati realno časovno grajenje sivinske slike v realnem času.



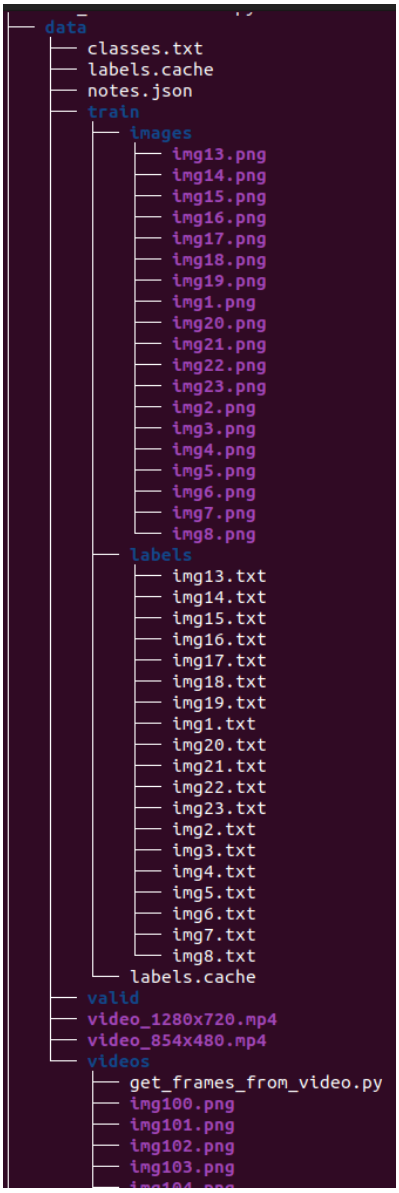
Slika 4: prikazuje grajenje sivinske slike iz strani odjemalca.

Slika 5: Izgled dokončane slike.

Zajem podatkov

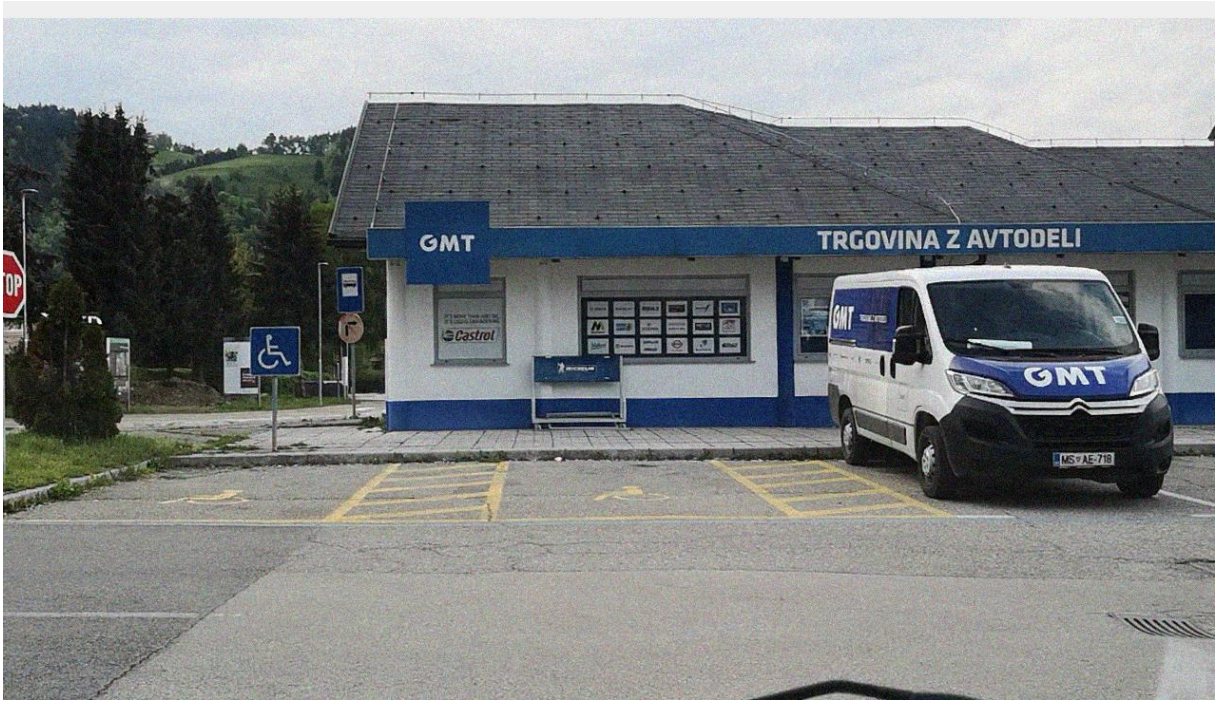
Ker sem dvakrat spremenil temo šele med samim zajemom podatkov so bili na koncu uporabni le podatki iz zadnjih dveh snemanj. Skupno sem tako zajel nekje 15 posnetkov dolgih približno 20 sekund. Zajeti so bili na raznih parkiriščih po Mariboru in Slovenj Gradcu v različnih vremenskih razmerah. Ker smo uporabljali yolo model smo za učenje potrebovali slike in ne videe. Zato sem s pomočjo programa ffmpeg iz videov izrezal določene slike (frames). Model yolo pa poleg slik za učenje potrebuje tudi oznake oz. labele. V ta namen smo uporabili program label-studio. To je program, ki se uporablja za anotacijo različnih vrst podatkov (slik, videov, zvoka, …). V našem primeru sem ga torej uporabil za anotacijo slik, pri katerih pa nam že omogoča izvoz anotacijskih podatkov v formatu yolo.
(slika prikazuje primer podatkov za treniranje mreže)

(slika prikazuje drevo direktorijev in podatkov)



Sistem za manipulacijo vhodnih podatkov

Da pa bi lahko privedli več raznolikosti v naše podatke in mrežo učili tudi na ne tako idealnih podatkih, smo izdelali sistem za alteracijo vhodnih podatkov. V mojem primeru sistem omogoča spreminjanje hitrosti predvajanja, dodajanje šuma na posamezno poslano sliko in pa rotacija (trenutno le 180 stopinj). Opcije izbiramo z nastavljanjem parametrov pred zagonom programa.
(slika prikazuje sliko nad katero smo uporabili dodajanje šuma in rotacijo)


GMT
TRGOVINA Z AVTODELI
Castrol
GMT
GMT
MS AE-718

PREMDET: SISTEMSKA PROGRAMSKA OPERMA

UVOD:

Pri predmetu Sistemska programska oprema smo imeli cilj vzpostaviti učinkovit komunikacijski sistem s pomočjo Redisa in Kafke ter nato uporabiti Grafano za merjenje performanc komunikacijskega kanala.

Naša naloga je bila izdelati robusten sistem, ki omogoča hitro in zanesljivo prenašanje podatkov med različnimi komponentami našega projekta. Za dosego tega smo izbrali Redis kot hitro podatkovno skladišče, ki nam je omogočilo učinkovito shranjevanje in dostop do podatkov. Poleg tega smo uporabili Kafko, ki je omogočala pretok podatkov in sporočil med komponentami sistema.

Za merjenje učinkovitosti komunikacijskega kanala smo vzpostavili Grafano, ki nam je omogočila nadzor in vizualizacijo podatkov o hitrosti, zakasnitvi in obremenitvi sistema. S tem smo zagotovili, da je komunikacijski kanal deloval optimalno in da smo lahko pravočasno zaznali morebitne težave ali izboljšave.

Metrike so prilagodljive in delujejo tudi za zaznavo ljudi v slikah in prepoznavo znakov. Primer na slikah je prikazan za robnike.

KOMUNIKACIJSKI SISTEM:

Proizvajalec:

Proizvajalec je odgovoren za zajemanje video posnetkov in pošiljanje posameznih okvirjev ter metapodatkov prek komunikacijskega kanala Redis in Kafka. Uporablja knjižnico OpenCV za zajem okvirjev, **Redis** za shranjevanje in dostop do podatkov ter Kafka za pošiljanje sporočil o novih okvirjih.

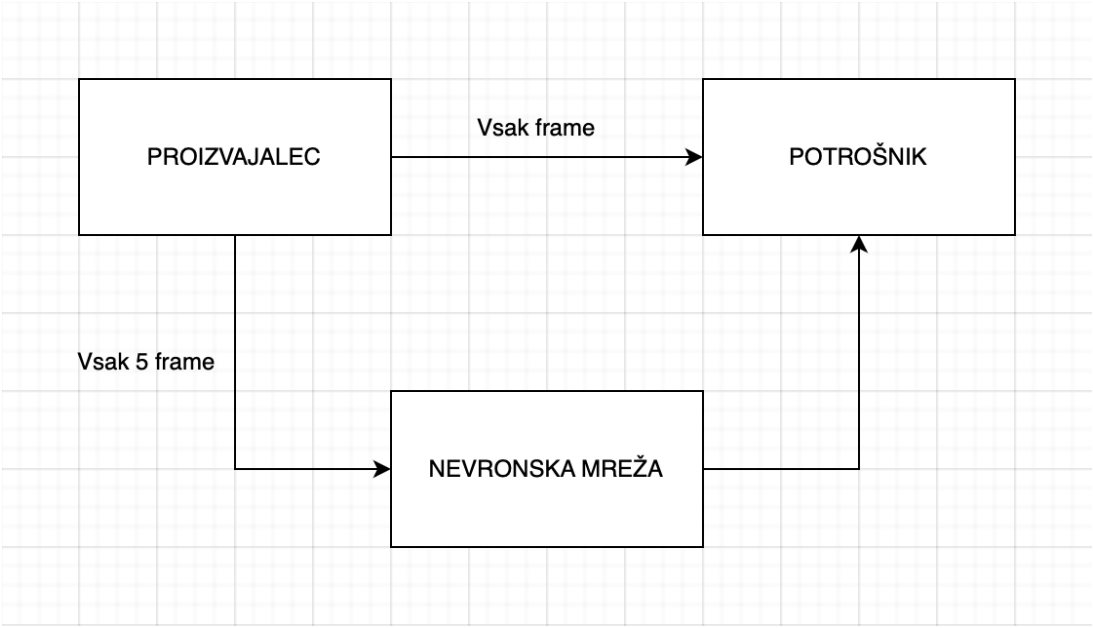
Odjemalec:

Odjemalec sprejema sporočila o novih okvirjih prek komunikacijskega kanala Kafka in jih obdeluje. Uporablja knjižnico OpenCV za prikaz okvirjev, Redis za dostop do zadnjega okvirja in Kafka za prejemanje sporočil. Poleg tega z detekcijo objektov izrisuje robnike na okvirjih.

Nevronska mreža:

Nevronska mreža uporablja nevronsko mrežo YOLO za detekcijo robnikov na okvirjih. Sprejema okvirje prek komunikacijskega kanala Kafka in uporablja knjižnico OpenCV za prikaz okvirjev. Predelane podatke o robnikih nato pošilja prek Kafka sporočil za nadaljnjo obdelavo.

Frame Skipping je implementiran kot del funkcionalnosti programa Producer zaradi počasnosti nevronske mreže, ki je zaostajala zaradi preobremenitve. Ta funkcionalnost omogoča preskakovanje okvirjev pri zajemanju videoposnetkov, kar zmanjšuje obremenitev sistema. Za preskakovanje okvirjev se uporablja preprost algoritem, ki določa, katere okvirje je treba preskočiti glede na število zajetih okvirjev in določeno število okvirjev, ki jih želimo preskočiti. S tem se doseže boljša učinkovitost sistema pri zajemanju in obdelavi videoposnetkov.



Slika 6: Prikazuje delovanje komunikacijskega kanala.

MERJENJE UČINKOVITOSTI KOMUNIKACIJSKEGA KANALA:

Ustvarili smo naslednje metrike:

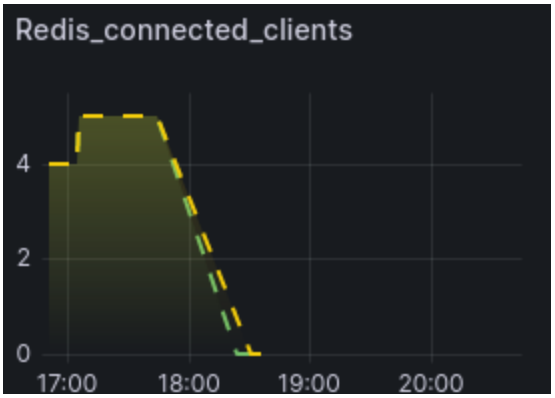
- **kafka_consumed_messages:** Število prejetih sporočil iz sistema Kafka po temah.
- **kafka_topics:** Število tem v sistemu Kafka.
- **kafka_produced_messages:** Število poslanih sporočil v sistem Kafka po temah.
- **redis_memory_usage:** Poraba pomnilnika v sistemu Redis.
- **redis_connected_clients:** Število povezanih odjemalcev v sistemu Redis.
- **redis_ops_per_sec:** Število operacij na sekundo v sistemu Redis.
- **inference_delay:** Zamik obdelave v milisekundah za nevronska omrežja.

UPORABA TIPOV METRIK:

Uporabili smo različne tipe metrik, ki so primerni za prikazovanje statističnih podatkov: **Counter:** Uporabljen za štetje števila prejetih in poslanih sporočil v sistemu Kafka.

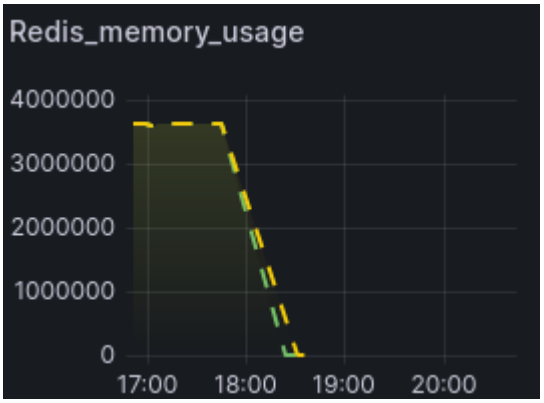
Gauge: Uporabljen za merjenje spremenljivih vrednosti, kot so poraba pomnilnika v sistemu Redis in število povezanih odjemalcev.

Histogram: Uporabljen za beleženje in analizo zamika obdelave nevronske mreže v milisekundah. Zaslonski posnetki diagnostičnih podatkov iz Grafane:



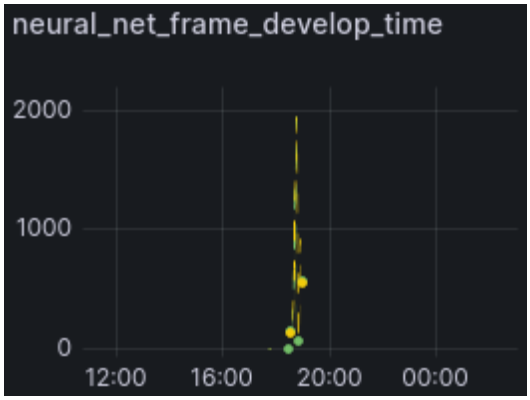
Slika 7: Prikazuje število povezanih odjemalcov na sistem REDIS skozi čas.

Slika 8: Prikazuje št.zahtev do podatkov iz redisa.



Slika 9: Prikazuje kako pomnilniško zahteven je REDIS skozi čas.

Slika 10: Prikazuje število robnikov, ki jih nevronska mreža zazna na cesti.



Slika 11: Prikazuje čas, ki ga nevornska mreža potrebuje za obdelavo podatkov.

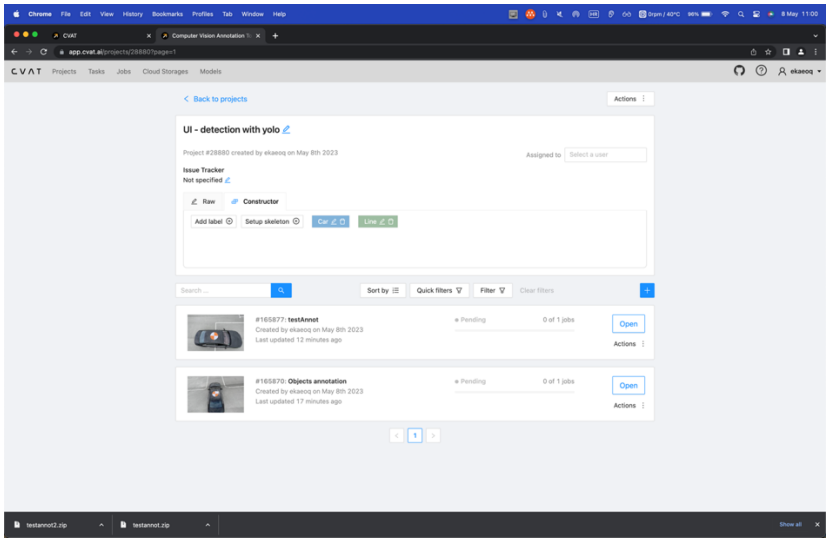
Priloženi so zaslonski posnetki, ki prikazujejo diagnostične podatke iz Grafane, vključno z izbranimi metrikami in njihovim spremljanjem v realnem času.

S temi modifikacijami smo omogočili boljši vpogled v delovanje sistema ter učinkovitejše spremljanje ključnih metrik. Grafana nam omogoča vizualizacijo in analizo podatkov na intuitiven način, kar nam pomaga pri zaznavanju morebitnih težav in optimizaciji delovanja sistema.

PREMDET: UMETNA INTELIGENCA

UVOD:

Pri predmetu umetna inteligenca je vsak član skupine je v svoji nalogi ustvaril lastno bazo trening podatkov ter natreniral svoj **Yolo_v8 model**. S tem smo pridobili raznolike modele za detekcijo objektov, ki so bili prilagojeni specifičnim potrebam vsakega člana. Vsi člani so za označevanje objektov v slikah uporabljali orodje **CVAT**.

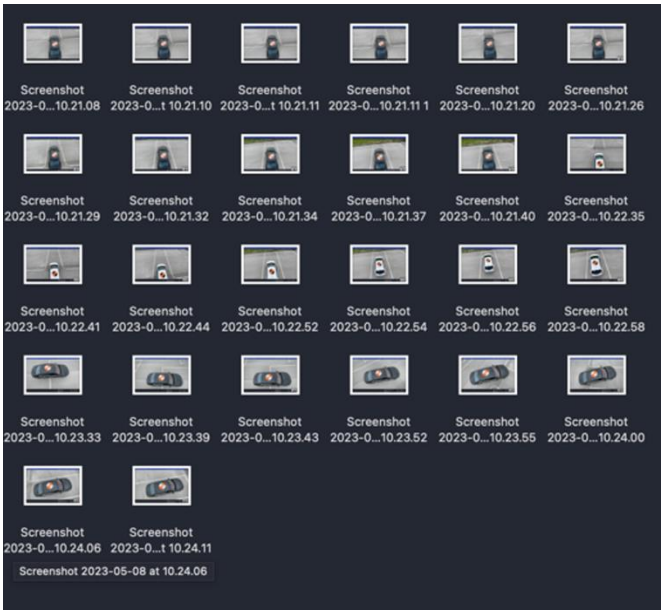


Slika 12: Prikazuje označevanje in pripravo trenirnih podatkov.

Vsi smo trenirali YOLO model z našimi podatki, ki smo jih zajeli v okviru meritev pri predmetu Signali in slike. Takšen pristop nam je omogočil izkušnjo v celotnem procesu ustvarjanja in treniranja modelov ter prilagoditev njihovim specifičnim zahtevam.

UČINKOVITOST PARKIRANJA:

Želeli smo prepoznati kvaliteto parkiranost avtomobila iz ptičje perspektive zato smo ustvarili bazo podatkov avtomobila iz pogleda ptičje perspektive:



Slika 13: Prikazuje nabor trenirnih podatkov za učinkovitost parkiranja.

Za izvedbo in preverjanje učinkovitosti naše prilagojene nevronske mreže smo kupili igrače avtomobile in zgradili testno parkirišče. To parkirišče smo uporabili kot osnovni vir vhodnih podatkov za usposabljanje in preizkušanje našega modela.

Sliki 14: Prikazujeta poligon parkirišča.

S tem pristopom smo si zagotovili popoln nadzor nad okoljem in objekti, s katerimi je bila nevronska mreža soočena. Testno parkirišče je bilo zasnovano tako, da je simuliralo realistične situacije in različne izzive, s katerimi se srečujejo vozniki pri parkiranju.

Igrače avtomobile smo postavili na parkirišče na različne načine, tako da so nekateri avtomobili bili pravilno parkirani znotraj črt, medtem ko so drugi presegali ali bili naslonjeni čez zarisane parkirne črte. Za vsak avtomobil na parkirišču smo pridobili ustrezne oznake o pravilnosti parkiranja.

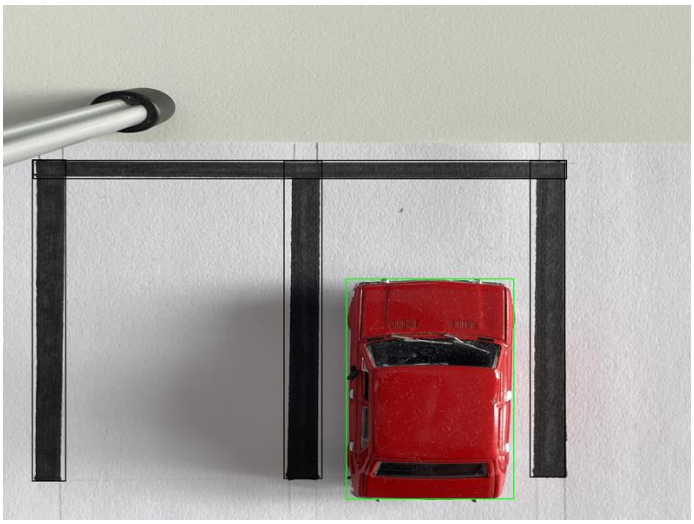
Te igrače avtomobilov in testno parkirišče smo uporabili za ustvarjanje obsežnega nabora vhodnih podatkov za usposabljanje in testiranje naše nevronske mreže. To je omogočilo modelu, da se je naučil prepoznati in razlikovati med pravilno in nepravilno parkiranimi avtomobili ter pravilno uporabljati funkcionalnost vizualne povratne informacije.

Slika 15: Prikazuje kako natreniran YOLO zaznava črto in avtomobil.

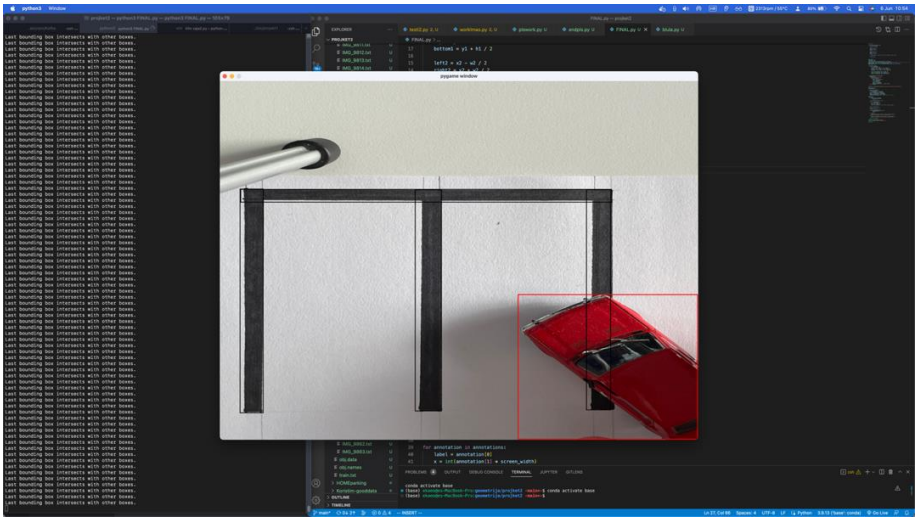
S tem pristopom smo ustvarili realistično okolje za našo nevronsko mrežo, kar je omogočilo boljše prilagajanje modela naše specifične uporabe in povečalo zanesljivost in natančnost ocenjevanja pravilnosti parkiranja.

Končni izdelek našega eksperimenta je bil prilagojen model nevronske mreže, ki je bil sposoben zanesljivo ocenjevati, kako dobro je avto parkiran. Eden izmed ključnih dodatkov v naš model je bila funkcionalnost, ki je omogočala vizualno povratno informacijo glede pravilnosti parkiranja.

Pri tem smo implementirali algoritem, ki je omogočal obarvanje okvirja okoli avtomobila na podlagi tega, kako se avto nahaja glede na zarisano parkirno črto. Če je bil avto pravilno parkiran znotraj črte, se je okvir obarval zeleno. V primeru, da je avto presegel ali bil naslonjen čez zarisano črto, pa se je okvir obarval rdeče.



Slika 16: Prikazuje, da je avto parkiran pravilno (okvir obarvan rdeče).



Slika 17: Prikazuje nepravilno parkiran avtomobil (okvir obarvan rdeče).

Ta vizualna povratna informacija je omogočila enostavno in hitro preverjanje pravilnosti parkiranja, kar je bilo izredno koristno za uporabo na majhnem parkirišču. Naša prilagojena nevronska mreža je tako zagotavljala ne le zaznavo objektov, temveč tudi oceno pravilnosti parkiranja, kar je olajšalo spremljanje in upravljanje parkirnih mest.

Ta funkcionalnost je bila dosežena z uporabo učnih podatkov, ki so vključevali primerne označbe za parkirna mesta ter informacije o tem, ali je avto pravilno parkiran ali ne. S tem smo

omogočili modelu, da se je naučil razlikovati med pravilno in nepravilno parkiranimi avtomobili ter uporabil ta znanje za ustrezno obarvanje okvirja okoli avtomobila.

Skupaj z zmožnostjo zaznavanja in klasifikacije objektov je ta funkcionalnost vizualne povratne informacije o pravilnosti parkiranja predstavljala pomemben korak naprej pri uporabi nevronske mreže za reševanje specifičnih problemov v parkiranju vozil.

Slika 18: Prikazuje simulacijo učinkovitosti parkiranja.

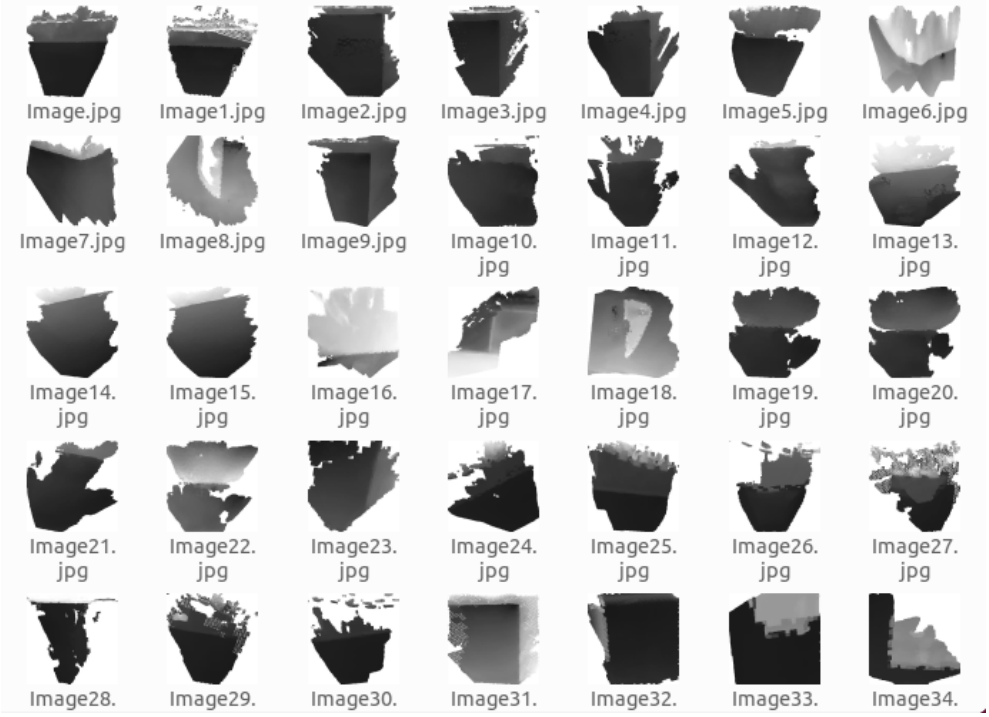
ZAZNAVA ROBNIKOV NA PARKIRIŠČU:

Cilj tega projekta je bil razvoj nevronske mreže, ki je sposobna prepoznati robnike iz globinske slike in ob tem ustrezno reagirati. Naš cilj je bil natrenirati lasten YOLOv8 model za detekcijo robnikov in zagotoviti visoko natančnost ter hitrost pri prepoznavanju teh objektov. Podobno kot pri projektnem delu iz Signalov in slik, smo se tudi tukaj osredotočili na vlogo proizvajalca in odjemalca. Proizvajalec zajema podatke o globinski sliki in jih preko Apache Kafka komunikacijskega sistema pošilja odjemalcu. Nevronska mreža zaznava objekte na prejetih sličicah in pošilja koordinate okvirja odjemalcu, ki nato prejema te podatke in prikazuje video z označenimi zaznanimi robniki.

Težave:

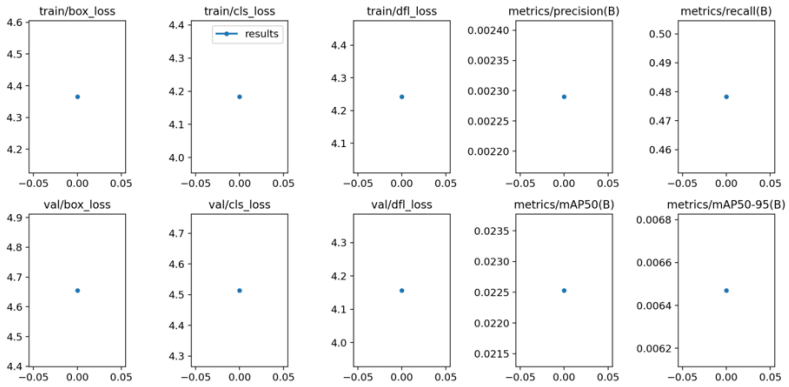
Ena od glavnih težav, s katero smo se srečali pri tem projektu, je bila težava pri implementaciji videa saj nam LiDAR zaradi omejitev iOSa pošlje le končni 3D model.
Sinhronizacija izrisa slike na strani odjemalca in tokov prejetih informacij: Druga pomembna težava je bila zagotavljanje usklajenega izrisa slike na strani odjemalca z obdelavo prejetih informacij. Zaradi asinhronosti in različnih hitrosti prenosa podatkov se je pojavila težava pri zagotavljanju, da se slika pravilno posodablja ob prejemu novih informacij. To je zahtevalo dodatne prilagoditve v logiki in algoritmu obdelave podatkov.

Zbrali smo nabor 3D modelov raznolikih robnikov na cestiščih in parkiriščih, ki smo jih nato pretvorili v globinske slike in označili robnike v OpenCV. Za trening je bilo uporabljenih 45 slik, za validacijo pa 15 slik, ki niso bile drugačne od tistih za trening.



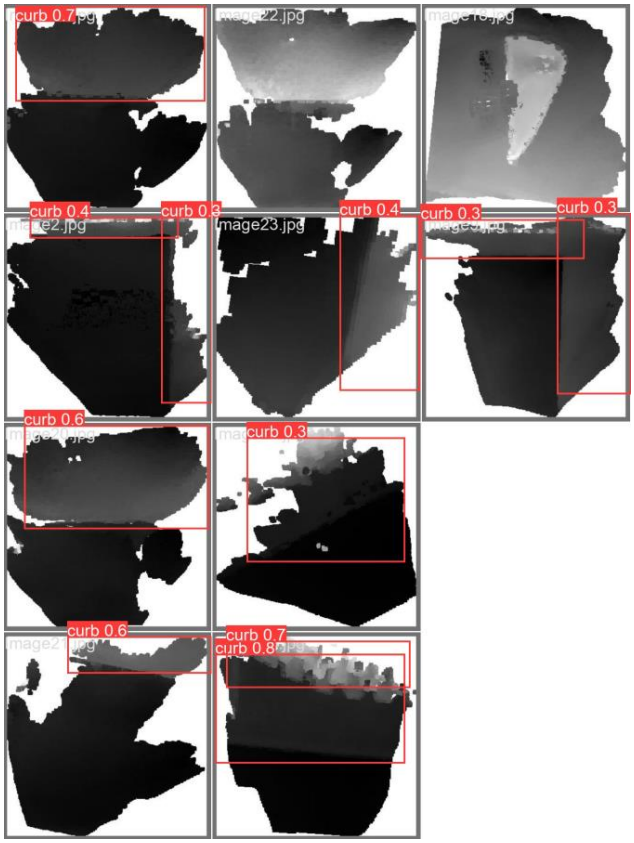
Slika 19: Prikazuje nabor trenirne množice podatkov za zaznavo robnikov.

Pri treniranju našega modela smo sprva uporabili le 1 epoch, vendar smo ugotovili, da je to premalo za dosego zadovoljivih rezultatov. Naš model ni zaznal nobenega robnika, kar je bilo očitno iz prikaza rezultatov treniranja pri 1 epochu.



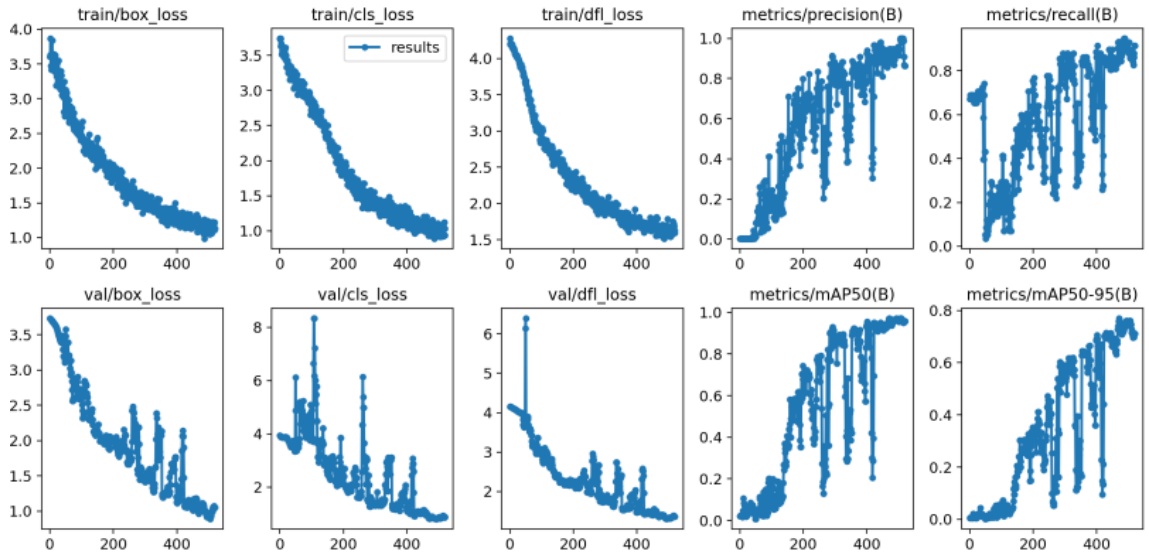
Slika 20: Prikazuje analizo treniranje Yola pri 1 epoch.

Nato smo se odločili povečati število epochov na 20. Kljub temu smo ugotovili, da je bila stopnja prepričanja našega modela še vedno zelo majhna. Prikaz rezultatov pri 20 epochih nam je to jasno pokazal.



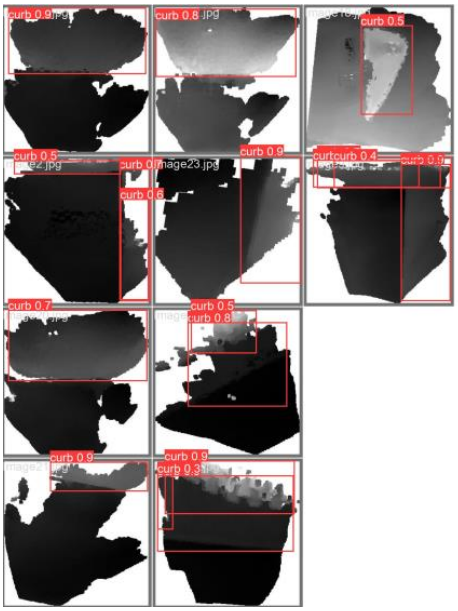
Slika21: Prikazuje samozavestnost Yola pr 20 epoch.

V tretjem poskusu smo se odločili uporabiti 300 epochov in ta pristop se je izkazal za zelo učinkovitega. Naš model je zaznal vse robnike z visoko stopnjo prepričanja. Prikaz prepričanosti nevronske mreže pri 300 epochih (slika 4) nam kaže, da se je samozavestnost modela izboljšala v primerjavi s 20 epochi (slika 2).



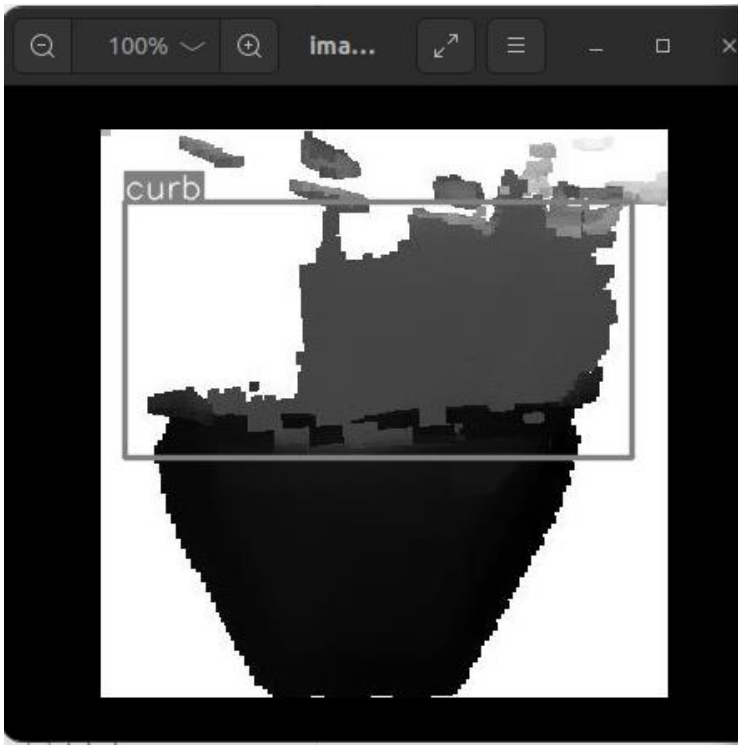
Slika 22: Prikazuje analizo treninga pri 300 epoch.

Poleg tega smo na vhodni sliki postavili ustrezne oznake (slika 5), ki smo jih uporabili za oceno uspešnosti našega modela. Prikaz prepričanosti nevronske mreže pri 300 epochih (slika 6) nam ponovno prikaže, kako dobro se model odziva in zaznava robnike.



Slika 23: Prikazuje samozavestnost Yola pri 300 epochih

PROGRAMA ZA ANALIZO USPEŠNOSTI DETEKCIJE NA NATRENIRANEM MODELU:

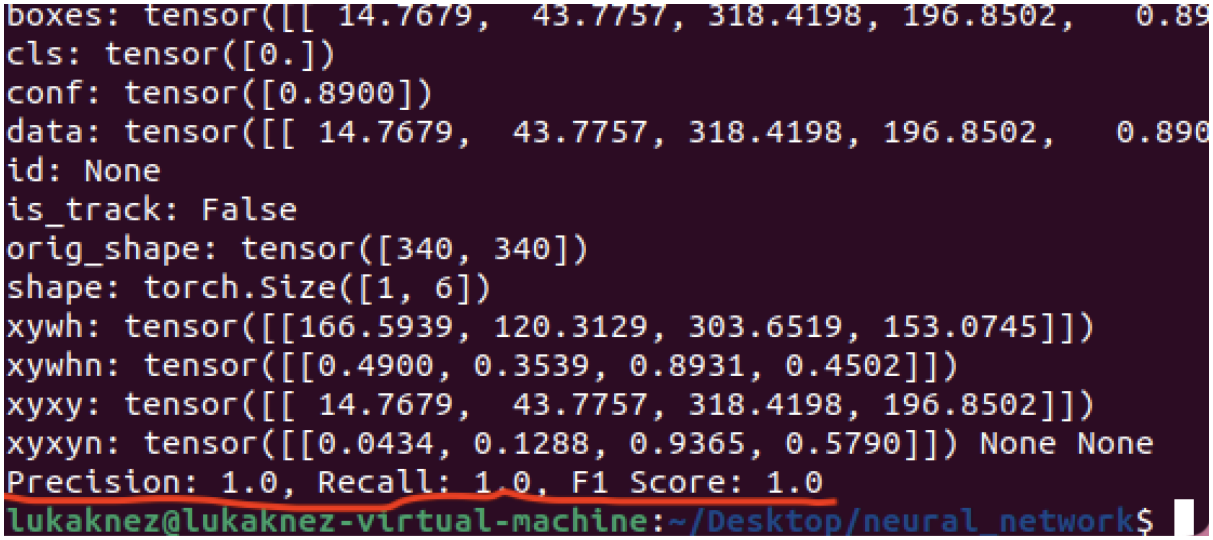


Slika 24: Prikazuje zaznavo robnika z uporbo "Annotator".

Za dosego tega cilja smo uporabili uteži iz natreniranega YOLO modela in uporabili knjižnice, kot sta PIL in ultralytics. Cilj tega programa je bilo tesitranje uspešnosti in označevanje lokacije robnika na vhodni sivinski sliki.

Uvedli smo naslednje metrike za oceno zanesljivosti modela:

Preciznost (precision):
Preciznost meri razmerje pravilno zaznanih pozitivnih primerov (pravilno zaznanih robnikov) glede na skupno število pozitivnih primerov (zaznanih robnikov).
Višja vrednost preciznosti kaže na manjšo stopnjo lažno pozitivnih zaznav (napačno zaznanih robnikov).
Odzivnost (recall):
Odzivnost, znana tudi kot občutljivost ali stopnja zaznavanja, meri razmerje pravilno zaznanih pozitivnih primerov (pravilno zaznanih robnikov) glede na skupno število pravih pozitivnih primerov (prisotnost robnikov v podatkovnem naboru).
Višja vrednost odzivnosti kaže na manjšo stopnjo lažno negativnih zaznav (nezaznanih robnikov).
F1 ocena:
F1 ocena je harmonično povprečje preciznosti in odzivnosti ter zagotavlja uravnoteženo merjenje uspešnosti.
F1 ocena je uporabna, ko želimo doseči ravnotežje med preciznostjo in odzivnostjo.



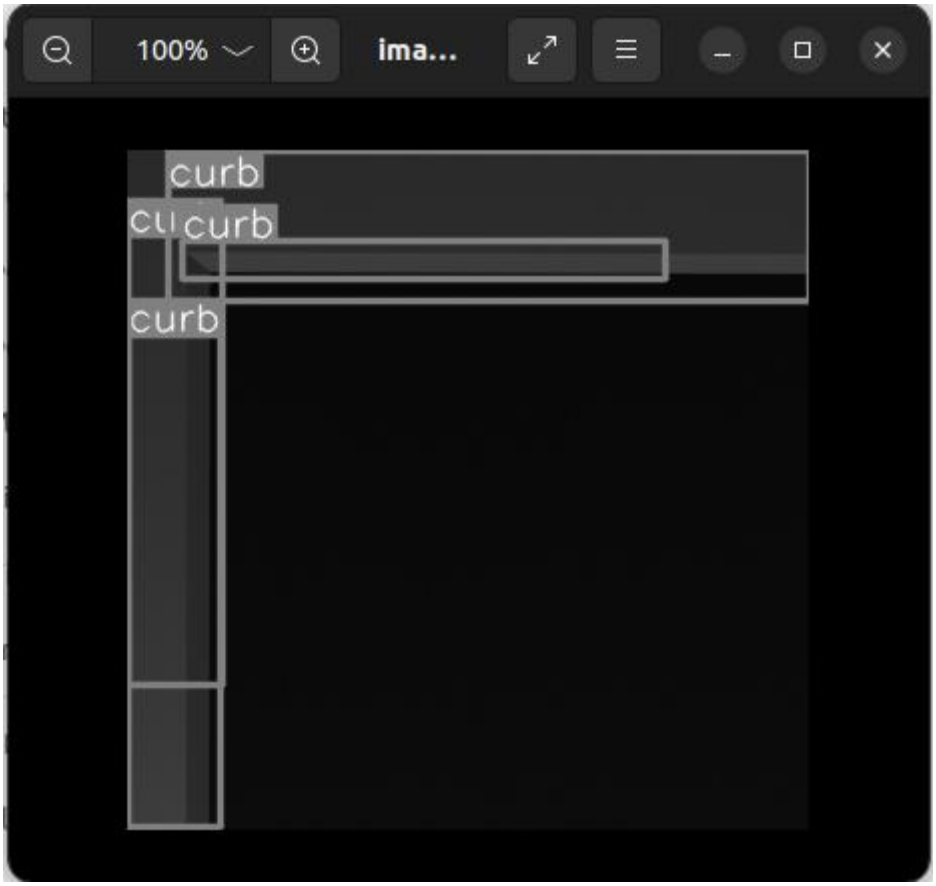
Slika 25: Prikazuje kako natančen je bil Yolo proti našim oznakam:

REALNO ČASOVNO ZAZNAVANJE ROBNIKOV Z SISTEMOM APACHE KAFKA:

Zaradi omejitev LiDAR tehnologije na napravah, kot je iPhone, ki omogoča samo končno sliko, smo v našem projektu ustvarili simulacijo v programu Blender. V tej simulaciji robniki simulirajo gibanje po tekočem traku, medtem ko naša nevronska mreža, ki temelji na modelu YOLOv8, poskuša zaznati te robnike.

Simulacija poteka na naslednji način:

Ustvarili smo 3D model tekočega traku s premikajočimi se robniki v programu Blender.
V simulaciji zajemamo slike iz različnih perspektiv, ki predstavljajo slikanje z LiDARjem.
Slike nato predložimo nevronski mreži, ki smo jo natrenirali na podlagi modela YOLOv8 za zaznavanje robnikov.Nevronska mreža analizira vsako posamezno sliko in poskuša zaznati prisotnost in položaj robnikov.
Če mreža zazna robnik, se o tem pošlje obvestilo preko Apache Kafka sporočilnega sistema.S tem ustvarimo tekočo simulacijo prenosa podatkov, ki bi jih sicer zajel LiDAR.



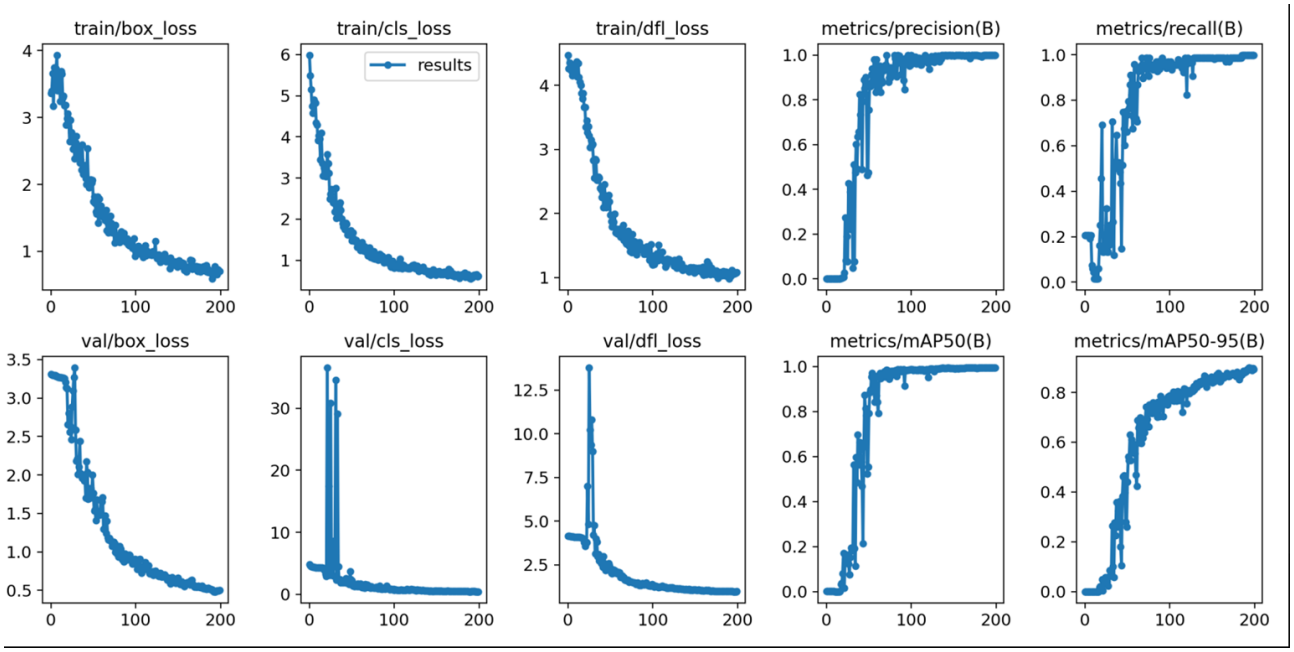
Slika 26: Prikazuje detekcijo nevronske mreže preko Kafke

PREPOZNAVA PARKIRNIH ZNAKOV IN CESTNIH OZNAK:

Naša množica podatkov za detekcijo je vključevala 3 znakov ki so bili naslednjih tipov: parkirni prostor za invalide, parkirni prostor za ljudi z majhnimi otroci in parkirni prostor za invalide. Kljub standardizaciji znakov podjetja včasih ne upoštevajo teh standardov, zato obstaja veliko različnih variacij znakov.

Podatkovna množica slik vsebuje videe raznih parkirišč resolucije [720p, 30fps]. Zajeti so bili v okolici Maribora in Slovenj Gradca. Iz teh videov smo nato izluščili najbolj čiste slike in jih uporabili za našo trenirno bazo.

Podobno kot pri zaznavi robnikov smo model preizkusili z 20 epochi, vendar je bil rezultat precej nenatančen. Zato smo poskusili z 200 epochi, kar nam je dalo bolj zanesljive rezultate.



Slika 27: Prikazuje analizo rezultatov treninga pri 200 epoch.

Nevronska mreža je bila naučena, da detektira prometne znake na slikah in deluje na principu segmentacije. Uporabili smo knjižnico OpenCV, da smo lahko mreži podali sliko za sliko iz video posnetka in detektirali prometne znake na vsaki posamezni sliki.

Ko mreža detektira prometni znak, sledi odziv na detekcijo. Če avto ni parkiran pravilno (na primer, na mestu za invalide, čeprav voznik ni invalid), voznika opozorimo. To storimo tako, da preverjamo, ali se avto premika ali stoji. Sledimo spremembam pozicij segmenta, ki ga vrne mreža. Če so se koordinate segmenta spremenile za več kot določen prednastavljen prag, sklepamo, da se vozilo premika. Če je vozilo parkirano in je bil prometni znak detektiran, preverimo, ali voznik ni invalid, in ga nato opozorimo s sporočilom.

Na ta način naš sistem za detekcijo prometnih znakov in odziv na detekcijo omogoča nadzor in opozarjanje voznikov glede pravilnega parkiranja na parkiriščih.

Slika 28: Prikazuje detekcijo znakov. Iz slike lahko razberemo tudi da je bila zajeta ko se je avto premikal na kar nakazuje rdeče zapisan true v zgornjem levem kotu.

PREMDET: RAČUNALNIŠKA GEOMETRIJA

UVOD

Cilj predmeta Računalniška geometrija je razviti orodja za reševanje izzivov v drugih predmetih. Vključujejo redčenje vertexov v 3D modelih, implementacijo QuickHulla za oceno učinkovitosti parkiranja ter opazovanje obnašanja vertexov v 3D modelih. Ta znanja so uporabna v računalniški grafiki, animaciji, simulaciji in drugih področjih.

OCENA PARKIRANJA S POMOČJO VIKTOR ALGORITMA

Po najkrajši možni poti, naš program uporablja knjižnico Pygame za ustvarjanje grafičnih vmesnikov. Naloži slike, jih prikaže na zaslonu in nariše omejevalne okvire okoli predmetov na podlagi opomb, prebranih iz datoteke. Preveri tudi presečišča med zadnjimi omejevalnimi polji in ostalimi ter vizualno pokaže, ali obstajajo presečišča. Na koncu shrani posnetke zaslona, preden se zaprejo.

```
    for box in bounding_boxes:
        x, y, width, height = box
        left = int(x - width / 2)
        top = int(y - height / 2)
        pygame.draw.rect(screen, (0, 0, 0), (left, top, width, height), 2)


    last_box = bounding_boxes[-1]
    intersects = False

    for box in bounding_boxes[:-1]:
        if check_intersection(last_box, box):
            intersects = True
            break

    if intersects:
        pygame.draw.rect(screen, (255, 0, 0), (left, top, width, height), 2)
        print("Last bounding box intersects with other boxes.")
    else:
        pygame.draw.rect(screen, (0, 255, 0), (left, top, width, height), 2)
        print("Last bounding box does not intersect with other boxes.")
```

V tem izrezku kode program ponavlja vsako omejevalno polje na seznamu bounding_boxes. Za vsako polje ekstrahira koordinate in dimenzije (x, y, širina, višina). Nato izračuna leve in zgornje koordinate polj tako, da od vrednosti x oziroma y odšteje polovico širine in višine.

Nato na zaslonu nariše pravokotnike z uporabo pygame.draw.rect(), pri čemer poda vrednosti levih, zgornjih, širin in višin, pridobljenih prej. Pravokotniki so narisani s črno barvo (0, 0, 0) in debelino obrob 2 slikovnih pik.

Ko nariše vse omejevalne okvire, dodeli zadnje omejevalne okvire na seznamu bounding_boxes spremenljivki last_boxes. Nato inicializira logično spremenljivko intersects kot False.

Zanka ponavlja vse omejevalne okvire razen zadnjega (bounding_boxes[:-1]). Za vsako polje pokliče funkcijo check_intersection(), da ugotovi, ali obstaja presečišče med zadnjimi polji in trenutnimi polji. Če je najdeno presečišče, je spremenljivka intersects nastavljena na True in zanka je prekinjena.

Na podlagi vrednosti presečišč programa nariše pravokotnike okoli zadnjih polj z uporabo rdeče barve (255, 0, 0), če obstajajo presečišča, ali zelene barve (0, 255, 0), če ne obstajajo presečišča. Poleg tega natisne ustrezna sporočila, ki označujejo, ali se zadnja omejevalna polja sekajo z drugimi polji ali ne.

Vsi podatki pa se preberu iz file-a, z uporabo tega skripta:

```
def read_annotations_from_file(filename):
    annotations = []
    with open(filename, 'r') as file:
        lines = file.readlines()
        for line in lines:
            values = line.strip().split()
            label = int(values[0])
            x = float(values[1])
            y = float(values[2])
            width = float(values[3])
            height = float(values[4])
            annotations.append([label, x, y, width, height])
    return annotations
```

ZMANJŠEVANJE VOZLIŠČ V 3D MODELIH S POMOČJO DELAUNAY TRIANGULACIJE:

V okviru projekta na področju računalniške geometrije sem se osredotočil na razvoj funkcije za zmanjševanje vozlišč v 3D modelih s pomočjo Delaunay triangulacije. Ta funkcija predstavlja moje lastno delo in je rezultat mojega raziskovanja in implementacije algoritma za zmanjšanje vozlišč v modelih.

Cilj funkcije decimateVertexes je bil optimizirati geometrijsko strukturo 3D modelov, pri čemer ohranja kakovost in natančnost geometrije. Funkcija uporablja Delaunay triangulacijo, ki je postopek za izračun triangulacije iz točk v prostoru, pri čemer zagotavlja določene pogoje glede kakovosti triangulacije.

Pri implementaciji funkcije sem najprej izvedel Delaunay triangulacijo za določene modele, nato pa sem izračunal povprečno razdaljo med vozlišči in njihovimi sosednjimi vozlišči znotraj triangulacije. Na podlagi določenega praga (threshold) sem nato odločil, katera vozlišča ohraniti in katera odstraniti.

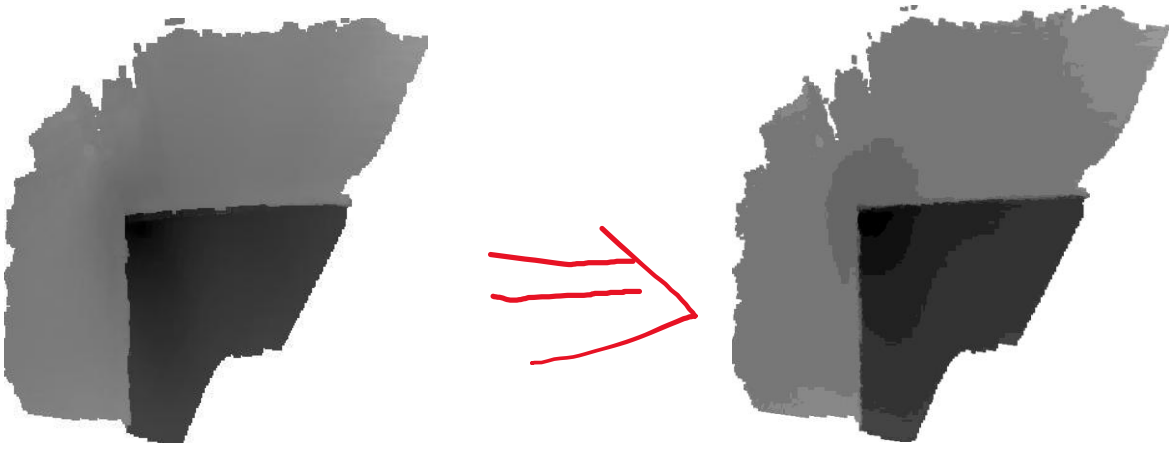
Moje delo je vključevalo tudi skrbno preverjanje pravilnosti implementacije, odpravljanje morebitnih napak ter optimizacijo algoritma za doseganje boljše učinkovitosti pri obdelavi 3D modelov. S tem sem zagotovil, da je funkcija zanesljiva, hitra in uporabna pri različnih scenarijih uporabe v računalniški geometriji.

Skupaj s postopkom Delaunay triangulacije, ki je ključnega pomena za zmanjševanje vozlišč v 3D modelih, je funkcija decimateVertexes moje lastno delo, ki predstavlja prispevek k področju računalniške geometrije in optimizaciji obdelave 3D modelov.

Vidimo lahko da funkcija odstrani veliko večino vertexov z zelo majhno pokvaritvijo slike:

```
Povprecna razdalja (min, max, mean): 0.0 1.0 0.019990612473416935
Stevilo vertexov pred decimacijo: 169887 Stevilo vertexov po decimaciji: 28849
```

OUTPUT:



Slika 29 prikazuje sliko pred (desno) in po decimaciji vertexov (levo)

ALGORITEM IZBOČENE LUPINE V 3D QUICKHULL 3D

Algoritem izbočene lupine v 3D (Quickhull 3D)

Kot prvo projektno nalogo smo morali implementirati quickhull v 3 dimenzionalnem prostoru. V našem projektu tega na koncu nismo vključili vendar bomo delo vseeno predstavili. Za generacijo točk smo napisali funkcijo, kateri podamo število točk, ki točke generira s pythonovim modulom random. Ob generaciji pa skrbimo da ne pride podvajanj tock.

Točke uporabljajo Gaussovo porazdelitev.

Po generaciji točk se prične sam algoritem. Najprej iz točk poiščemo ekstremne točke iz vseh treh osi in iz štirih izmed njih sestavimo začetni tetraeder.

Ideja je nato da preverjamo katere točke se nahajajo zunaj in katere znotraj geometrijskega telesa, ter to telo postopoma povezujemo s točkami izven njega. Torej postopno širimo telo, dokler ga ne sestavljajo le najbolj zunanje točke. Tu se nam je pri implementaciji zataknilo in v končni nalogi ne pridemo dejansko do želenega končnega rezultata.

Za izris uporabljamo modul matplotlib, in pa ker program vsebuje veliko seznamov, smo jih poskusili kar v čim večji meri nadomestiti s polji iz modula numpy.
(Na sliki je prikazan začetni tetraeder)