SREDNJA ELEKTRO IN RAČUNALNIŠKA ŠOLA VELENJE

Trg mladosti 3, Velenje

MLADI RAZISKOVALCI ZA RAZVOJ ŠALEŠKE DOLINE

RAZISKOVALNA NALOGA

UPORABA RAČUNALNIŠKEGA VIDA ZA ZAZNAVANJE VOZNEGA PASU NA SLOVENSKIH CESTAH

Tematsko področje: Interdisciplinarno: računalništvo, matematika

Avtor:

Luka Lah, 4. letnik

Mentor:

Rok Urbanc

Velenje, 2020

Raziskovalna naloga je bila opravljena na ŠC Velenje, Elektro in računalniška šola, 2019/2020.

Mentor:

Rok Urbanc

Datum predstavitve: marec 2020

Ključna dokumentacijska informacija

ŠD ŠC Velenje, šolsko leto 2019/2020

KG prepoznavanje voznega pasu

AV LAH Luka

SA URBANC Rok

KZ 3320 Velenje, SLO, Trg mladosti 3

ZA ŠC Velenje, Elektro in računalniška šola, 2020

LI 2020

IN UPORABA RAČUNALNIŠKEGA VIDA ZA ZAZNAVANJE VOZNEGA PASU NA SLOVENSKIH CESTAH

TD Raziskovalna naloga

OP VIII, 44 str., 0 tab.,0 graf.,28 slik, 2 pril., 19 vir

IJ SL

JI sl

AI Živimo v dobi, kjer si življenja brez tehnologij ne znamo predstavljati. Večino proizvajalcev teh tehnologij pa si prizadeva, da bi vsakdanje procese še bolj poenostavili. Od pametnih inštalacij, pametnih mobilnikov, računalnikov, pametnih zvočnikov, kot je Amazon Aleksa, osebnih asistentov itd. se večina današnjih inovacij pomika v smer avtomatizacije procesov in prostoročnega delovanja. Zadnjih nekaj let smo začeli res podrobneje izkoriščati moč umetne inteligence ti. AI. S pomočjo le-te pa smo začeli uporabljati naše računalnike, da bi avtomatizacijo uvedli tudi v avtomobilsko industrijo in tako naredili vožnjo brez-ročno. Ampak ali je mogoče z današnjimi programskimi jeziki in orodji takšen sistem ustvariti doma? Zanima pa nas tudi, kakšne so ovire in kako se spopasti z njimi.

Key words documentation

ND ŠC Velenje, 2019/2020

CX lane recognition

AU LAH Luka

AA URBANC Rok

PP 3320 Velenje, SLO, Trg mladosti 3

PB ŠC Velenje, Elektro in računalniška šola, 2020

PY 2020

TI USING COMPUTER VISION TO DETECT LANES ON SLOVENIAN ROADS

DT RESEARCH WORK

NO VIII, 44 pag., 0 col.,0 gra.,28 pic., 2 anne., 19 sources

LA SL

AL sl/en

AB We live in era, where life without technologies is unimaginable. Most manufacturers want to further improve those technologies and simplify everyday tasks in the process. From smart installations, smart phones, computers, smart speakers like Amazon Alexa, to personal assistants etc. Most innovations nowadays are stepping in the direction of automatization of processes and hand-free operation. In the last few years we really started to exploit the power of artificial intelligence also known as AI. However, with its help we started using our computers to bring automatization to the automobile industry and make our driving experience hand-free. Nevertheless, is it possible to make such a driving experience with today’s tools and computer languages at home? We also want to know what the obstacles are and how to deal with them.

Seznam okrajšav in kratic

% – odstotek

angl. – polni pomen iz angleškega jezika

slo. – prevod v slovenščino

° - stopinje

dipl. – diplomirani

ERŠ – Elektro in računalniška šola

inž. – inženir

npr. – na primer

oz. – oziroma

ŠCV – Šolski center Velenje

ti. – tako imenovani

URL – enotni naslov vira (angleško Uniform Resource Locator) je naslov spletnih strani v svetovnem spletu

wiki – Wikipedia

www – angl. world wide web (svetovni splet)

fps – sličice na sekundo (angl. Frames per second)

Kazalo vsebine

Kazalo slik

Kazalo Enačb

1 Uvod

Ljudje si ne predstavljamo sveta brez naprav. Prve izvedbe le-teh so začele nastajati že v prvi polovici 20. stoletja. Najprej smo začeli razvijati naprave, kjer je bil v ospredju računalnik. Takšne naprave je bilo moč nadzirati samo preko terminala oz. delčkov. Danes pa prehajamo v čas, ko je v središče vedno bolj postavljen človek, s tem pa se eksponentno veča tudi moč umetne inteligence in uporaba le-te. Skoraj vsaka industrija že uporablja nekakšen način učenja računalnika (angl. Machine learning) ali umetne inteligence. Najpogosteje pa se uporablja za analiziranje naših navad. V velikih znamkah lahko tako predvidijo, kakšni izdelki so nam všeč in kako čim bolj prilagoditi sisteme uporabnikom. Zadnjih nekaj let pa so te tehnologije začeli izkoriščati tudi v avtomobilski industriji. Nekatere znamke, kot je Tesla, uporabljajo umetno inteligenco že več let (avtopilot se prvič pojavi pri Tesli X 2014). Tudi večina drugih znamk zdaj opaža visoko vrednost na tem področju.

Danes ima že skoraj vsak nov avtomobil pomoč pri uravnavanju v vozni pas ali pa celo avtopilota. Veliko avtomobilov omogoča tudi branje prometnih znakov. Ti pa so namenjeni, da bi vozniku olajšali branje prometnih znakov in tako zmanjšali naprezanje voznikov in s tem tudi utrujenost voznikov v avtomobilu.

1.1 Hipoteze

Pred raziskovanjem smo si zastavili naslednje hipoteze:

- Prepoznavanje voznega pasu na slovenskih cestah ne bo natančno.
- Program bo moč napisati brez poznavanja matematike ali osnov le-te.
- Procesorska moč pri izvajanju programa z računalniškim vidom ni pomembna.
- Procesiranje lahko izvedemo tudi preko oblačnih storitev.
- Računanje 4 sličic na sekundo je dovolj za varno usmerjanje avtomobila na slovenskih cestah.

1.2 Ideja za izdelavo raziskovalne naloge

Ideja za raziskovalno nalogo izhaja iz vsakdanjega življenja. Vedno več lahko na različnih medijih slišimo o tehnologijah, povezanih z AI in računalniškim vidom. Vsakdo pa je že vsaj slišal za Teslo. Tesla je blagovna znamka električnih avtomobilov, ki nudi avtopilota s stopnjo avtonomnosti 3 (avtomobil upravlja sam, vendar more biti uporabnik prisoten in mora tudi poseči za volan, če pride do napake). Prav zaradi te znamke so marsikateremu pritegnili zanimanje avtopiloti in sistemi uravnavanja v vozni pas. Kmalu za tem so ljudje na internetu začeli izdelovati lastne projekte z raznimi mikrokrmilniki Arduino ali Raspberry. Tako smo se tudi sami želeli prepričati, ali je mogoče ustvariti podoben sistem in kaj vse potrebujemo zanj. Zanimale so nas vse slabosti, prednosti, lastnosti …, ki pridejo s takšnim sistemom, ter cena projekta. Na koncu smo želeli tudi preveriti, kako zanesljiv je takšen sistem. Ali mogoče ogroža nas in druge v prometu?

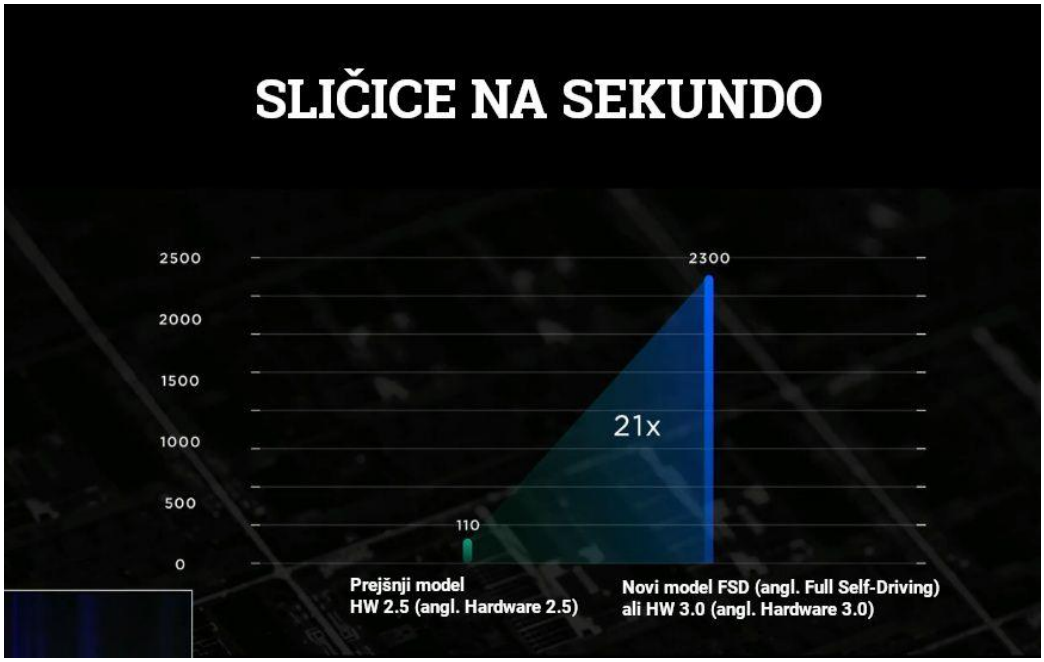
2 PREGLED STANJA TEHNIKE

Danes je že skoraj vsak avtomobilski proizvajalec naredil vsaj 1 avtomobil s stopnjo avtonomnosti 3. Če avtomobila niso dali v prodajo, pa so ga predstavili na različnih predstavah, kot je CES (angl. costumer electronics show). Na CES 2020 so takšne avtomobile predstavljali celo proizvajalci pametnih telefonov in ostale sodobne elektronike (npr. Sony, LG …). Daleč pred vsemi na trgu za porabnike in za prodajno rabo pa je Tesla.

Teslini prodajni modeli lahko nudijo 3. stopnjo avtonomnosti vse od leta 2014, ko je prvič prišel model X. Tesla ima trenutno 12 ultrasoničnih senzorjev in kar 8 kamer, tako da vozilo dobi pogled nad celih 360 stopinj. Kamere in senzorji s pomočjo računalniškega vida potem zaznavajo:

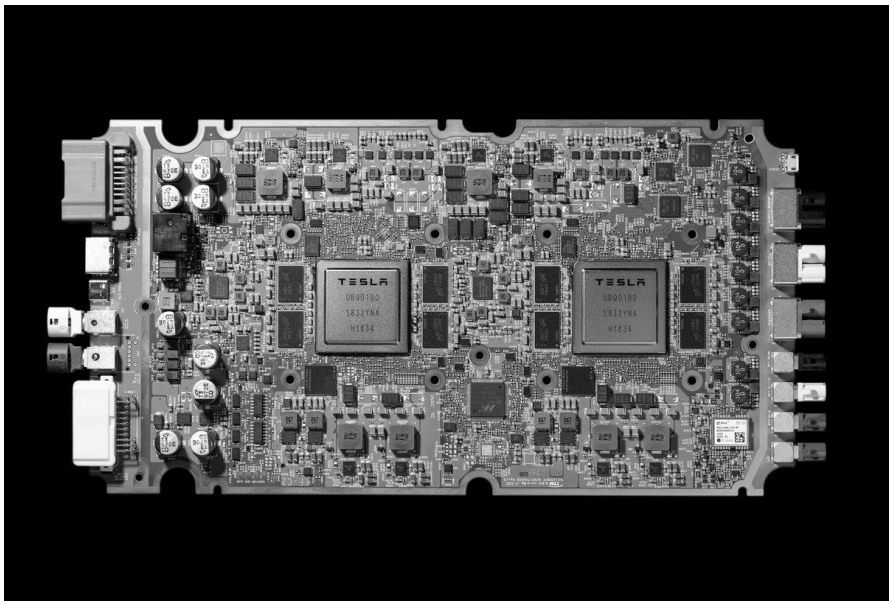
- gibanje,
- prometne pasove (vsak pas posebej – leva in desna črta prometnega pasu sta ločeni),
- potek ceste (kje poteka cesta vključno z nasprotnim pasom),
- predmete ali osebe na cestišču oziroma na poti vozila,
- predmete ali osebe ob cestišču,
- semaforje in barve na njih in
- prometne znake.

Najnovejši modeli procesorja tesle omogočajo kar do 2300 sličic na sekundo, za razliko od prejšnjih, ki so bili kar 21 x počasnejši (slika 1). Kot primerjava je lahko pametni telefon visokega cenovnega razreda, ki omogoča 960 sličic na sekundo in s tem tudi super počasne posnetke. To pomeni, da lahko procesor iz kamere v tesli izračuna še več kot 2 x več sličic na sekundo od najboljših pametnih telefonov na marketu, kar je iz varnostnega stališča odlično in daleč pred konkurenco.



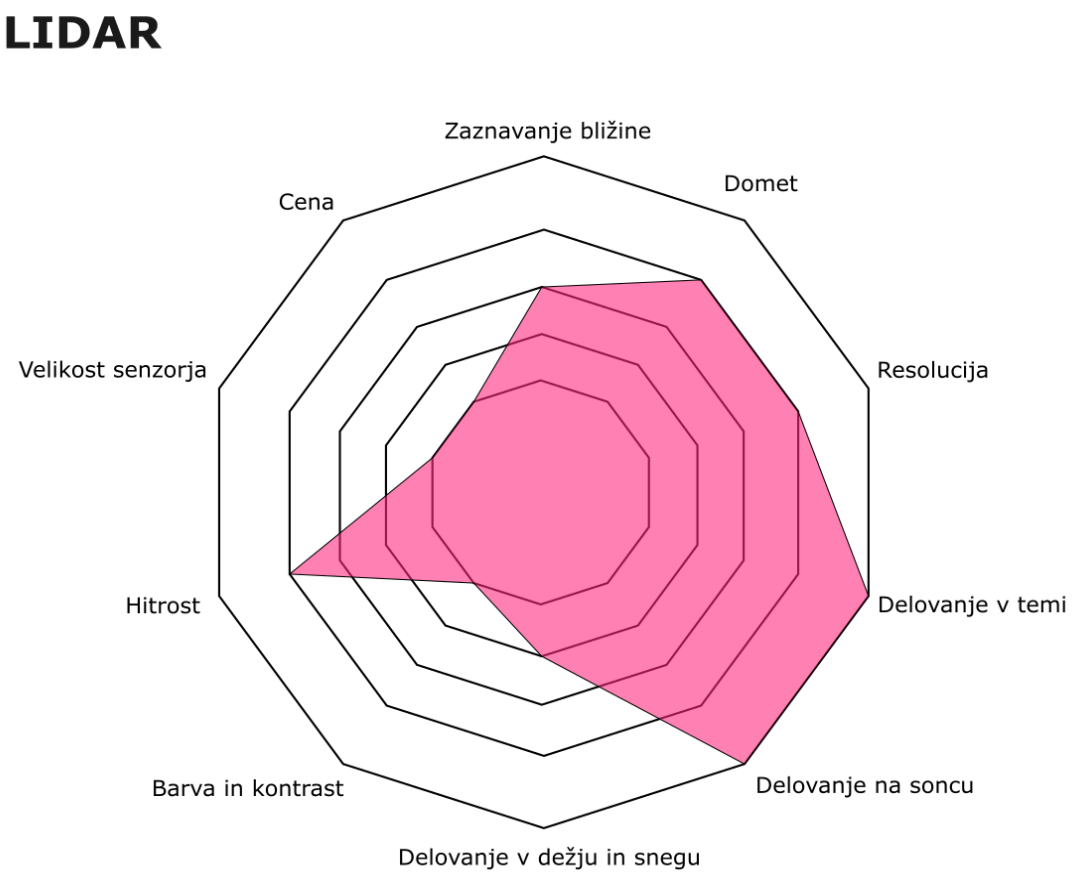
Slika 1: Primerjava nove in stare strojne opreme, lasten vir

In s pomočjo njihovega lastnega čipa (slika 2) za procesiranje zgoraj navedenih stvari tako tvorijo enega trenutno najboljših avtopilotov na svetu. Seveda pa je Tesla še daleč od samo vozečega vozila, čeprav je daleč pred konkurenco.



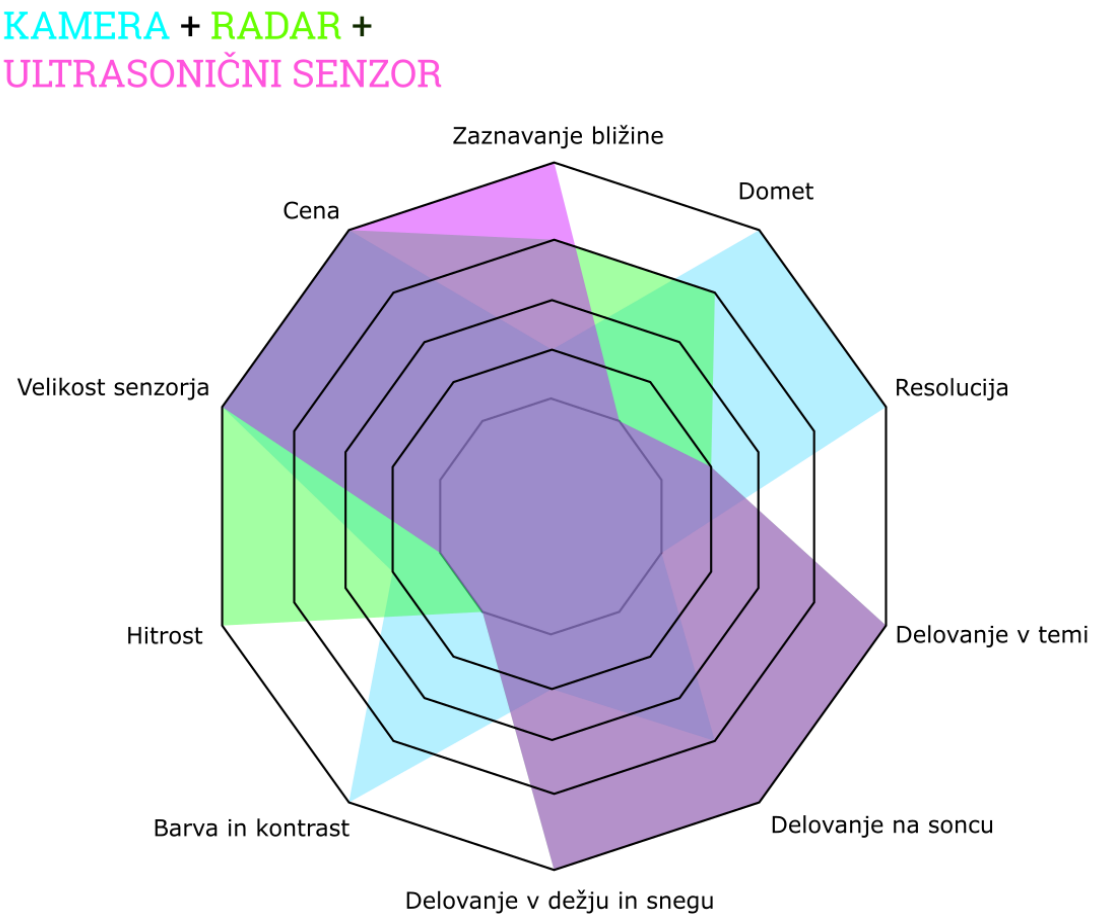
Slika 2: Teslin čip za avtopilota, vir[1]

Tesla je poseben primer tudi zaradi vizije, ki jo ima Elon Musk (CEO), ki naj bi bila zato, da bi prodajali vrhunske avtomobile z vrhunsko zmogljivostjo in avtopilotom po ceni, dostopni večini uporabnikom. Konkurenca tako uporablja radarje LIDAR, ki so zanesljivejši od samih kame,r vendar pa so cenovno težje dostopni. Pri 4 LIDAR senzorjih ima konkurenca samo za senzorje tako stroške za prodajno ceno Teslinega modela 3. LIDAR je senzor s srednjim dometom (30 m), dobrim zaznavanjem bližine (okoli 20 m), zelo dobro ločljivostjo in deluje odlično v temi in na soncu (slika 3).



Slika 3: Ocenjevanje zmožnosti LIDAR senzorja od 1 do 5, lasten vir

Kot že omenjeno, Tesla uporablja kamere. Kamere imajo slabo zaznavanje bližine, slabo odzivnost in slabo delovanje v temi. Ker pa se Elon Musk tega zaveda, Tesla nima samo kamer, temveč ima tudi radar in ultrasonične senzorje. Ti senzorji zdaj nadomestijo slabosti, ki jih imajo kamere same (slika 4).



Slika 4: Ocenjevanje zmožnosti kombinacije senzorjev od 1 do 5, lasten vir

Vedno več pa se uporablja tudi procesiranje v oblaku. Procesiranje v oblaku je vrsta procesiranja, pri katerem naprava določen račun ali postopek pošlje v oblak. Oblak nato izračuna zahtevo/račun/postopek in nato rešitve pošlje nazaj na napravo. Tako naprava ne potrebuje posebne procesorske moči, saj le-to dobi preko oblaka.

V avtomobilski industriji pa ne zasledimo veliko računanja v oblačnih storitvah. To je verjetno zaradi varnostnih razlogov. Npr. če imamo nek avtomobil, ki omogoča avtopilota, vendar pa se vsi izračuni dogajajo v oblaku, se v primeru izgube signala izgubi tudi procesorska moč in s tem neha delovati tudi avtopilot. Če potemtakem voznik v situaciji, kjer je avtomobil brez signala, ne bi odreagiral dovolj hitro, bi bili priča avtomobilski nesreči.

3 MATERIALI IN METODE DELA

Ker je projekt bolj informacijsko orientiran, smo rešitev želeli izdelati s čim manj stroški. Vse kar je potrebno za izdelavo projekta, so:

- računalnik s sistemom Windows (priporočeno Windows 10) in Python 3.8,
- stojalo za telefon (za v avtomobil),
- telefon s kamero (ločljivost 720 x 1280 px s vsaj 30 fps oziroma sličicami na sekundo),
- avtomobil z armaturno ploščo.

Vse delo smo si razdelili na dva dela, to sta terensko delo ter laboratorijsko delo. Pri terenskem delu smo s pomočjo kamere na telefonu in stojala za v avtomobil snemali vožnjo oziroma cesto (). Telefon smo pri tem poizkušali nastaviti tako, da bi bil v središče postavljen naš vozni pas, in da bi se na posnetku opazilo čim manjši del armaturne plošče. Pri tem smo zajeli več posnetkov, dolgih po nekaj minut, da bi imeli neko podlago/gradivo iz realnega sveta, na katerem bi kasneje izvajali računanje z računalniškim vidom.

Pri laboratorijskem delu pa smo se ukvarjali z izdelavo programov za prepoznavo voznega pasu (angl. lane recognition). Najprej smo želeli izvedeti, kako bo izgledal program za prepoznavo voznega pasu brez računalniškega vida. Zato smo se najprej lotili prepoznave s pomočjo filtrov in matematičnih formul. Naslednji korak pa je bil uporaba nevronske mreže računalniškega vida oziroma nenadzorovano učenje računalnika (angl. unsupervised machine learning). V tem koraku smo poizkušali naučiti program, da prepozna določene objekte na cestišču zato, da bi lahko v prihodnosti dodali še sistem avtomatskega zaviranja ali prepoznavo znakov.

Metodi smo želeli povezati tudi z ukazi za uravnavanje avtomobila, da bi tako naredili preprostega pomočnika za brez-ročno vožnjo v avtomobilu. Na koncu smo želeli tudi izvedeti, katera od metod je učinkovitejša za uporabo in kako varni sta metodi ter tudi slabosti in prednosti, ki jih ponujata.

4 ZAČETEK RAZISKOVANJA

Pred samim začetkom raziskovalne naloge smo poskrbeli za ustrezen material, ki smo ga potrebovali, in pa tudi za ustrezno programsko opremo (Python 3.8). Na računalnik z operacijskim sistemom Windows naložimo Python 3.8 in v procesu nameščanja dodamo program v pot (angl. Add to Path). To nam bo kasneje omogočalo, da s pomočjo ukaza python ali pip zaženemo program Python oziroma nameščevalec paketov (angl. package manager) znotraj Windows terminala (drugače bi se morali postaviti na lokacijo, kjer je nameščen program Python).

4.1 Python

Python je visokonivojski programski jezik. Je tudi eden najbolj priljubljenih programskih jezikov v letu 2020. Ustvaril ga je Guido van Rossum leta 1990. Python ima popolnoma dinamične podatkovne tipe, zna samodejno upravljati s pomnilnikom in podpira večino vrst programiranja oziroma programske paradigme. Je pa tudi jezik, za katerega je napisanih največ knjižnic, saj je lahko vsakdo kontributor skupnosti. Vsak uporabnik lahko tako napiše svojo knjižnico in jo objavi v skupnost. Te knjižnice lahko nato vsakdo uporablja takoj, ko jih naloži in vključi v svoj program.

Sam programski jezik pa spada v manjšo skupnost jezikov, ki ga uporabljajo tolmača. Tolmač pomeni prevajanje programskega jezika po vsaki vrstici posebej. Obraten primer je nato prevajalnik, ki omogoča prevajanje celotne kode naenkrat.

Tolmač ima naslednjo prednost: svoje napake lahko odkrijemo dosti prej, saj se program izvaja vrstico po vrstico. Ena velika slabost pa je potemtakem hitrost celotnega izvajanja programa, ki je za razliko od prevajalnika velikokrat počasnejša.

Slika 5: Python logotip, lasten vir

4.2 Računanje v oblačnih storitvah

Računanje v oblačnih storitvah ali računalništvo v oblaku je slog računalništva, pri katerem so računalniška sredstva na voljo kot storitev preko interneta. Takšen koncept se prvič pojavi v sedemdesetih letih. Danes je eden največjih ponudnikov prav Microsoft, saj ponuja večino svojih izdelkov v internetni obliki. Seveda lahko te izdelke še vedno naložimo, vendar pa to ni potrebno. S pomočjo storitev v oblaku lahko uporabljamo programe, kot so Microsoft Word, Powerpoint, OneNote, Excel … To pomeni, da računanje za dane naloge oziroma uporabo programov izvaja Microsoftov strežnik.

Takšen pristop pa dandanes uporablja vedno več ponudnikov programske opreme. Več ameriških podjetij ponuja tudi že procesorsko moč za igranje računalniških igric. To pomeni, da lahko uporabniki zgolj z internetno povezavo, telefonom in kazalnimi napravami (miško in tipkovnico) igrajo najzahtevnejše računalniške igre.

Edini problem takšnega pristopa je zakasnitev (angl. latency), kar pomeni, da mora uporabnik oz. naprava na strežnik poslati prošnjo in odgovor prejeti nazaj. Tukaj pride do časovnega zamika, ki je odvisen od kvalitete interneta. Trenutna zakasnitev podatkov, ki potujejo iz Amerike v Evropo, je nekje okoli 70 ms. Takšna zakasnitev je precej neopazna, če uporabljamo npr. Microsoft Word.

Pri uporabi hitrega in varnega reagiranja pa je takšen slog procesiranja skoraj izključen, saj hitrost internetne povezave ni zanesljiva, hkrati pa je tudi zakasnitev prevelika.

5 KAJ SPLOH JE RAČUNALNIŠKI VID?

Računalniški vid je vrsta učenja računalnika (angl. machine learning). Primer programske opreme računalniškega vida je Tensor flow. Ta ima trenutno narejeno največjo bazo prepoznavanja, namenjeno vsakdanjim uporabnikom. Z njim lahko večinoma prepoznavamo objekte, kot so avtomobili, zvezki, osebe, živali … Če bi želeli ustvariti lastno prepoznavo specifičnih objektov, bi morali računalnik »natrenirati« (angl. train). Trenutna bolj poznana je implementacija Tenserflow-u, to omogoča YOLO.

Svoj računalnik lahko naučimo prepoznavati različne objekte s treningom. Ta trening navadno vsebuje bazo slik. Računalniku tako za specifičen objekt določimo, kaj v resnici je. S pomočjo matematičnih funkcij računalnik nato naredi povzetek o objektu oziroma izračuna aritmetične vrednosti slik. To naredi s pomočjo ti. Tensorjev ali nevronske mreže.

5.1 Učenje računalnika

Učenje računalnika ali angl. machine learning je znanstvena študija o algoritmih in statističnih modelih, ki jih uporabljajo računalniki brez eksplicitnih navodil. Svoja navodila tako tvorijo s pomočjo določenih vzorcev. V grobem poznamo 2 vrsti učenja računalnika: nadzorovano in nenadzorovano (pri pomoči k takemu načinu dobi novo ime utrjeno učenje računalnika).

5.1.1 Nadzorovano učenje računalnika

Pri nadzorovanem učenju računalnika potrebujemo 2 vrsti podatkov. Prva vrsta so podatki za učenje, medtem ko so druga vrsta podatki za testiranje. Podatki za učenje so tako uporabljeni za učenje računalnika (npr. 100 meril pestiča določene rože in 100 meril lista iste rože).

Pomembno je, da imamo pri podatkih za učenje vsaj 4 lastnosti. Te lastnosti so pomembne zaradi računanja tensorjev. Tensor je v poenostavljenem pomenu razdalja med točkami. Vsak par prej omenjenih lastnosti tako predstavlja eno točko v koordinatnem sistemu. Nato se izračuna razdalja med točkami. Razdalje oziroma tensorje navadno delimo s takšnim številom, da dobimo na koncu število med 0 in 1. Razdalje si želimo omejiti z 0 in 1, saj se lahko v procesu računanja zgodi tudi več milijonov operacij. Z omejenostjo pa zmanjšamo dolžine števil in tako skrajšamo računski proces.

Zdaj se lahko zamislimo, zakaj je pomembno čim več podatkov za učenje računalnika. Več kot bo podatkov učenja, natančnejše bo učenje računalnika.

Nato pride na vrsto testiranje ustvarjenega algoritma. S pomočjo podatkov za testiranje lahko sedaj ugotovimo, kako natančen je naš algoritem. Najlažje je, da naredimo neko zaokroževanje primernih minimumov in maksimumov ter primerjamo, ali je testni podatek (tudi ta mora imeti 4 lastnosti) v izračunanih mejah. Če je, lahko tako določi, da testni podatek spada v npr. določeno vrsto neke rože. Tukaj lahko zdaj tudi vidimo pravilnost določevanja ter jo ustrezno nastavljamo.

5.1.2 Utrjeno in nenadzorovano učenje računalnika

Nenadzorovano učenje računalnika pa je kompleksnejše, saj uporablja nevronske mreže (angl. neural network). Nevronske mreže pri nenadzorovanem učenju ne dobivajo nobenega vnosa od uporabnika (torej ne ve, kakšno vrsto podatkov obdeluje). Pri takšnem učenju računalnik različne podatke razdeli v skupine.

Nevronske mreže pa večinoma delujejo na podlagi zmag in porazov. Tukaj mi algoritmu določimo, kaj je njegova naloga, npr. igranje igre Pong. Tukaj začne algoritem igrati igro, s tem pa se skozi generacije razvija. Takšnemu učenju računalnika pravimo tudi utrjeno učenje računalnika.

Če program izgubi, je to zanj poraz, zato bo poizkušal odpraviti napake, ki jih je naredil. Število točk je lahko v tem primeru nagrada. Če je v novi generaciji dosegel algoritem večje število točk, si tukaj zapomni in poizkuša razločiti, kaj je povzročilo to spremembo. Skozi generacije se nato program zelo razvidno razvije, problem pa je, da je takšen način navadno zelo časovno potraten, saj lahko takšen algoritem preide tudi skozi nekaj milijonov generacij (vsaka pa lahko traja tudi po več ur). Navadno lahko večji napredek vidimo že na več tisoči generaciji.

Torej nevronska mreža vzame več vnosov oziroma vhodnih podatkov, ki jih nato pošlje čez milijone skritih plasti (odvisno od nevronske mreže) in vrne nek izhodni podatek (število izhodnih podatkov je tudi odvisno od nevronske mreže).

6 TERENSKO DELO

Terensko delo, kot že samo ime pove, je delo, pri katerem podatke pridobivamo zunaj laboratorija. Pri terenskem delu smo potrebovali avtomobil, stojalo za telefon (za v avto) ter pametni mobilni telefon (resolucija kamere 1280 x 720 px – v našem primeru Samsung Galaxy S9).

Slika 6: Slika aplikacije opreme, lasten vir

Tukaj lahko že naslov namigne, da gre pri terenskem delu za zajemanje posnetka. Da bi zbrali čim več podatkov, smo posnetkov naredili več. Zanimalo nas je tudi, kakšne so razlike npr. med dvema avtomobiloma.

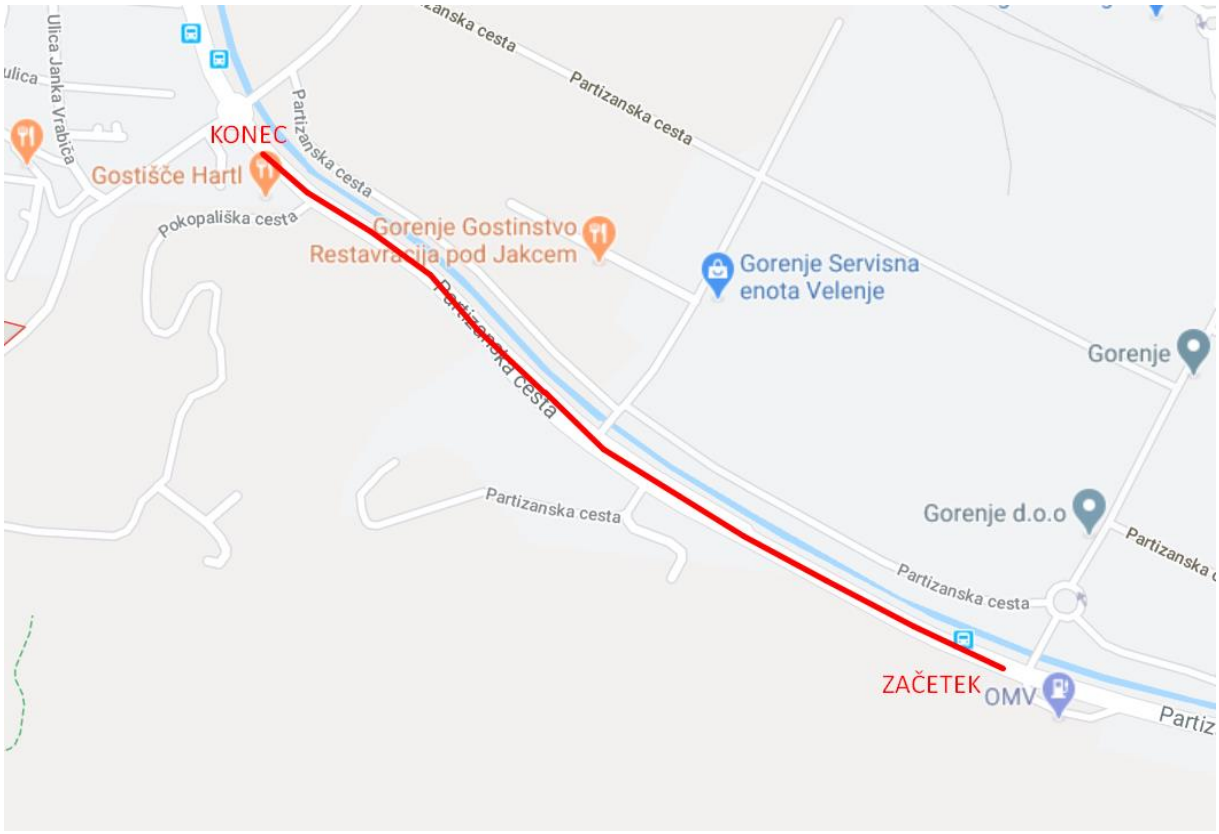
Pri vožnji smo zbiranje podatkov uporabili 2 avtomobila:

- Mazda 3 2009 (bencin)
- Toyota Verso 2012 (dizel)

Podstavek za telefon smo kupili v trgovini Big Bang:

- Cellular line BIGCRABDUALFIX

Da bi bila prepoznava voznega pasu čim natančnejša, pa smo izbrali cesto, ki ima večino časa črte voznega pasu na obeh straneh (smer Velenje-Šoštanj, slika 7):



Slika 7: Pot snemanja, lasten vir

Vsi posnetki in ustrezne rešitve problemov so v prilogi.

7 PREPOZANAVA VOZNEGA PASU S POMOČJO FILTROV IN FUNKCIJ

Najosnovnejši in hkrati najbolj matematično usmerjen način prepoznave voznega pasu je s pomočjo filtrov in funkcij. Način naj bi bil natančen, saj lahko ob ustrezni uporabi filtrov pridobimo presenetljive rezultate.

Za obdelavo podatkov smo najprej opravili terensko delo, kjer smo te podatke pridobili v obliki videoposnetka.

Celoten testni program je bil izdelan s pomočjo programskega jezika Python. V Pythonu smo uporabili naslednje knjižnice:

- Numpy (nameščen skupaj s programom Python 3.8)
- Math (nameščen skupaj s programom Python 3.8)
- Time (nameščen skupaj s programom Python 3.8)
- Matplotlib
- Scipy
- OPEN CV2

7.1 Filtri

Filtri so matematične funkcije, ki se izvajajo nad sliko. V računalništvu je slika 2 dimenzionalna tabela, ki vsebuje vrednosti barv, navadno RGB (odvisno od formata slike in modula za pretvorbo). Vse kar filtri počnejo, je spreminjanje vrednosti 2D-tabele po določenih navodilih.

Pri procesu razvijanja smo uporabili naslednje filtre in procese:

- obrezovanje slike,
- spreminjanje kontrasta,
- razločevanje bele barve (ker je večina črt na voznih pasovih belih),
- sivinski filter,
- Gaussov filter,
- poudarjanje robov,
- Canny filter,
- Hughove črte.

Pri zajemanju posnetka tako dobimo 1280 x 720 slikovnih točk. V teh slikovnih točkah nas zanima samo del ceste z voznim pasom in avtomobili. Zato tukaj uporabimo obrezovanje slike. Na koncu moramo seveda vsako obdelano sličico v posnetku sestaviti nazaj v posnetek.

7.1.1 Obrezovanje slike

Obrezovanje slike v našem primeru naredimo s pomočjo knjižnice OPEN CV2. Drugače je obrezovanje slike proces, pri katerem določimo 3 ali več slikovnih točk. Te slikovne točke bodo nato izhodišča za novo nastalo sliko. Na obrezani sliki zdaj vidimo vse slikovne točke, ki so znotraj nastalega lika, ki ga tvorijo izhodiščne slikovne točke.

7.1.2 Spreminjanje kontrasta

Kot že omenjeno, ima vsaka slikovna točka svojo vrednost. Spreminjanje kontrasta je dobesedno zmnožek koeficienta kontrasta in vrednosti slikovne točke (slika 8). Če pretvorimo vrednosti RGB (100, 0, 20) v desetiške, dobimo 6553620. To pomeni, da se kontrast spremeni na podlagi določenega koeficienta:

$$spremenjen\ kontrast = vrednost\ slikovne\ točke\ (6553620) * k$$

Slika 8: Spreminjanje kontrasta s koeficientom 2, lasten vir

Za spreminjanje kontrasta smo zopet uporabili knjižnico OPEN CV2.

7.1.3 Razločevanje bele barve

Naši možgani znajo razločiti belo barvo. Računalnik pa lahko na drugi strani ima probleme. Pri perfektnih pogojih je barva bela oziroma ima RGB vrednost (255, 255, 255), če pa k tem pogojem dodamo oblačno vreme, se te vrednosti kar hitro zmanjšajo npr. (177, 178, 176).

Dobljena vrednost računalniku ne pomeni več bele barve, temveč pove, da je dobljena barva siva.

Ta problem smo rešili s pomočjo svoje funkcije, ki po določenem pragu vsem točkam, ki ne spadajo v barvno območje, zapiše vrednost 0 (spremeni jih v črno barvo). Torej smo s pomočjo zank pogledali vsako slikovno točko in pregledali nastavljene pogoje ter spremenili vrednosti (slika 9).

Res je da po takšnem postopku dobimo sive barve, vendar pa na sliki navadno nastopa zelo mali odsek takšnih barv razen, če je le ta preobsijana (angl. overexposed). Čeprav zdaj dobimo odsek točk, ki ne spadajo pod črto voznega pasu, pa tukaj še nismo zaključili procesa filtriranja in bomo takšne točke najverjetneje odstranili s pomočjo drugega filtra.

Slika 9: Razločevanje bele barve, lasten vir

7.1.4 Sivinski filter

Vsaka slika je glede na barvni model sestavljena iz več kanalov. V našem primeru ima slika 3 kanale, ki so RGB oziroma Red, Green Blue. Vsak od teh kanalov je sivinski, skupaj pa tvorijo barvni model oziroma barve. Te kanale si lahko podrobneje ogledamo na histogramu (slika 10), ki prikazuje vrednosti sivinskega kanala slike v obliki stolpičnega diagrama.

Slika 10: Histogram rdeče barve, lasten vir

Pri pretvorbi zdaj odstranimo 2 kanala. Navadno ohranimo histogram rdečega (odvisno od nastavitev) kanala. Če bi učili nevronsko strukturo s vhodnimi podatki, bi pri uporabi barvne slike kot vhodnega podatka potrebovali 300 vnosov (10 x 10 x 3 zaradi 3 kanalov). Pri sivinski sliki kot vhodni podatek pa bi potrebovali le 100 vnosov (zaradi 1 kanala). Razlika 200 vnosov pa je neverjetno visoko število, saj lahko nevronska struktura za vsak vnos opravi več milijonov ali milijard operacij skozi skrite plasti.

Slika 11: Sivinski filter, lasten vir

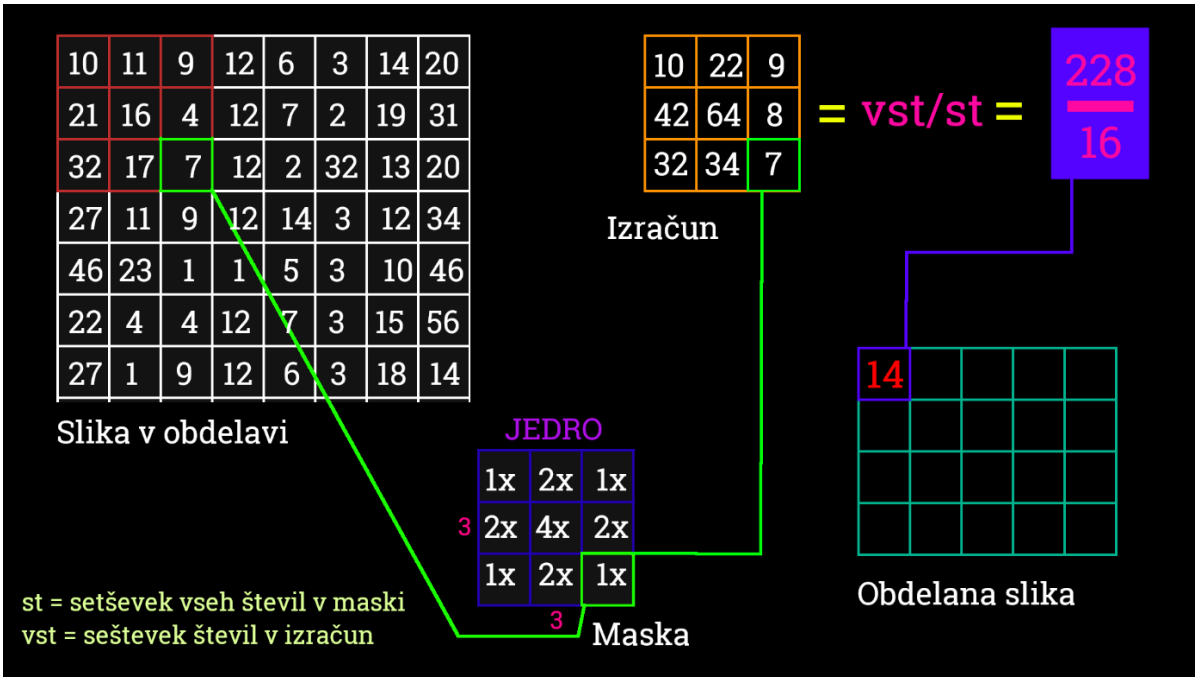
7.1.5 Gaussov filter

Gaussov filter s pomočjo maske izračuna nove vrednosti za celotno območje v velikosti jedra tega filtra. Če je standardni odklon manjši od 1 slikovne točke, glajenje oziroma takšen filter praktično nima učinka.

Pri Gaussovem filtru jedro določimo sami. Določiti moramo x (širino) in y (višino) jedra. Potem pa določimo masko. Maske pri Gaussovem filtru delujejo tako, da bolj kot se

oddaljujemo od središča jedra, manj teže oziroma vrednosti nam število predstavlja. Posledično tako takšen filter omogoča odstranjevanje šuma ter zamegljenje slike.

Z jedrom se torej postavimo na originalno sliko ali sliko v obdelavi (pred tem določimo velikost jedra, npr. 3 x 3). Vse vrednosti nato pošljemo skozi masko filtra ter dobimo izračun (slika 12). Vse vrednosti izračuna seštejemo in jih delimo s seštevkom vrednosti v maski. Tako dobimo novo vrednost, ki jo zapišemo v obdelano fotografijo.



Slika 12: Gaussov filter izračun, lasten vir

Matematično pa stvar izračunamo malo drugače. Masko dobimo iz Gaussovega jedra

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{7.1}$$

Enačba 1: Maska Gaussovega jedra, vir[15]

(i, j)-ti element maske, velikosti (2k+1)x(2k+1) slikovnih točk, dobimo kot

$$H(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{(i-k-1)^2 + (j-k-1)^2}{2\sigma^2}} \tag{7.2}$$

Enačba 2: element maske (i, j), vir[15]

Enostaven primer je prikaz matrike maske, ki tvori zmnožek s količnikom (1 / seštevkom vseh števil v jedru)

$$H = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{in} \quad H = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{7.3}$$

Enačba 3: Matrike mask, vir[15]

7.1.6 Poudarjanje robov

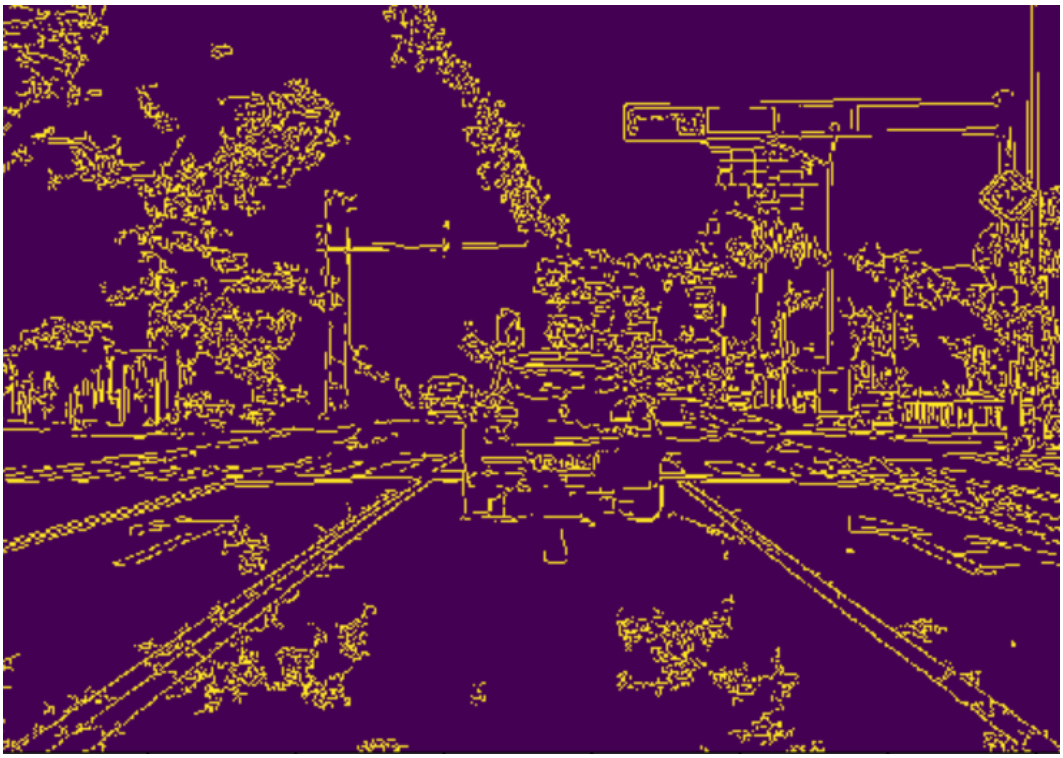
Poudarjanje robov je pomembno za uporabo Canny filtra. Za poudarjanje robov smo uporabili modul knjižnice OPEN CV2. Modul se imenuje threshold in s pomočjo praga izvede izbrano funkcijo.

Sami smo prag določili med 100 in 255, pri katerem sta vrednosti 0 - črna in 255 - bela. To pomeni, da smo obravnavali vsa bela območja (ker obdelamo že obdelano sliko s prej navedenimi filtri). Sedaj območje s pomočjo funkcije THERSH_TOZERO spremenimo vsa neizbrana območja v črno barvo (slika 13) oziroma nastavimo njihovo vrednost na 0.

Slika 13: Treshhold filter za poudarjanje robov, lasten vir

7.1.7 Canny filter

Canny filter je eden izmed detektorjev robov. Predpostavka temu filtru je sivinska slika, kar pomeni, da bi pri poizkusu z barvno sliko dobili napako pri računanju (vsaj v programskem jeziku). Ena izmed pomembnih predpostavk je tudi odstranjevanje šuma, za kar pri našem algoritmu poskrbi cel kup navedenih filtrov in funkcij (največji prispevek ima tukaj Gaussov filter). Če šuma ne bi odpravili, bi bili zaznani robovi čez večino površine slike (slika 14).



Slika 14: Canny filter brez odstranjenega šuma, lasten vir

Seveda pa preden izračunamo robove slike, omejimo območje računanja v trikotno obliko (obliko voznega pasu), da odstranimo, kolikor je le mogoče šuma.

Vsaka slikovna točka ima 2 kriterija. Prvi je smer roba v radianih, medtem ko je druga intenzivnost slikovne točke, ki zavzame vrednost med 0 in 255.

Ko je slika zglajena, izhajata $I_x \; in \; I_y$ z spoštovanjem do (angl. with respect to) x in y. I izračunamo tako, da ga združimo z jedri $K_x \; in \; K_y$ (angl. kernel) Sobelovega filtra (za prepoznavanje robov):

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (7.4)$$

Enačba 4: Matrike jeder sobelovega filtra, vir[14]

Intenzivnost G in smer slikovne točke $\emptyset$ se nato izračunata po naslednjem postopku:

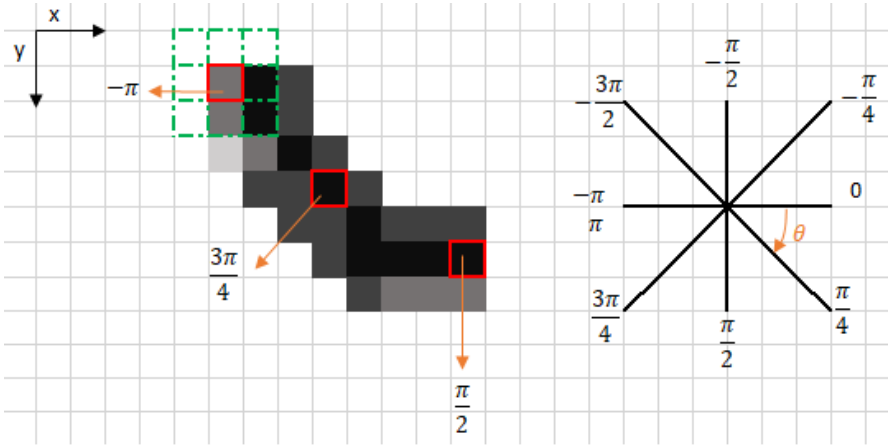
$$|G| = \sqrt{I_x^2 + I_y^2} \tag{7.5}$$

Enačba 5: intenzivnost robov, vir[14]

$$\emptyset(x,y) = \tan^{-1}\left(\frac{I_y}{I_x}\right) \tag{7.6}$$

Enačba 6: Smer slikovne točke, vir[14]

Idealno želimo imeti čim tanjše robove, zato pri Canny filtru uporabimo ti. Non-Maximum Supperssion. Postopek je enostaven: algoritem potuje skozi celotno matriko intenzivnosti robov, pri katerem najde najintenzivnejše robove (robove z najvišjo vrednostjo). Če je takšnih slikovnih točk robov več v vrsti, pa uporabi sredinski člen. Za določevanje takšnih robov pa potrebujemo smer roba. Ta nam sicer pove, kam potuje rob in kaj je samo debelina roba (slika 15).



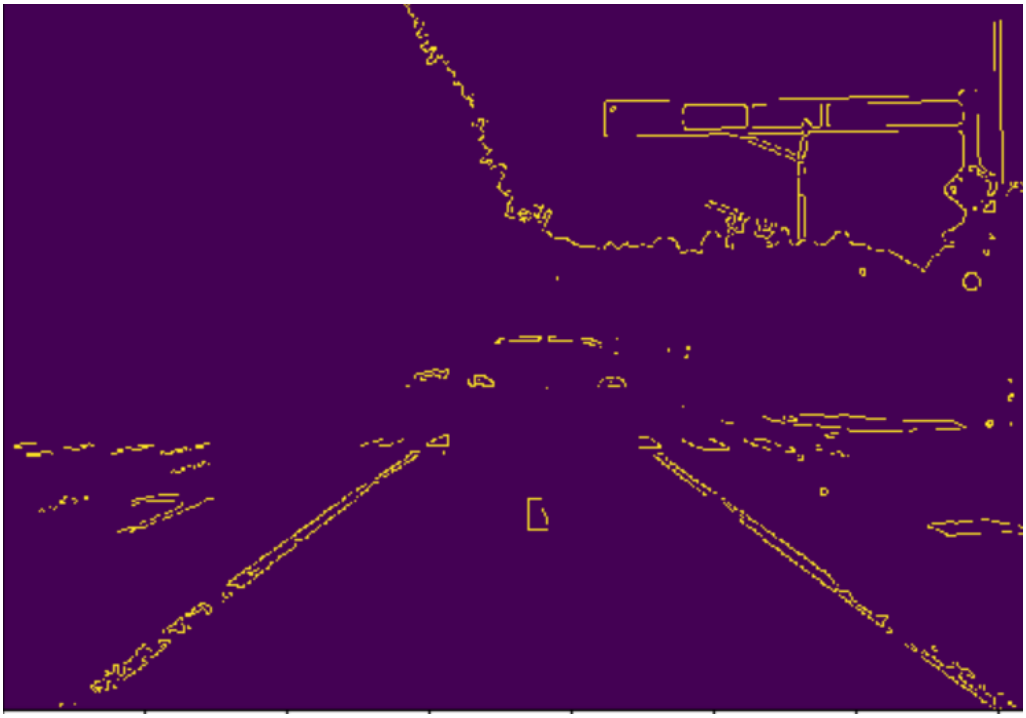
Slika 15: Preverjanje intenzivnosti robov po smeri, vir[2]

Če povzamemo, so glede na vnose za Non-max suppression koraki naslednji:

- Ustvarimo matriko, ki jo inicializiramo na 0, velikost pa nastavimo na enako velikost originalnega gradienta intenzivnostne matrike.
- Iskanje smeri roba glede na smerno vrednost iz smerne matrike.
- Ugotavljanje, če ima slikovna točka, obrnjena v enako smer, večjo intenzivnost od trenutne slikovne točke.

Po vseh zgoraj zaključenih postopkih nam algoritem vrne izračunane robove (slika 16). Ko robove izračuna, pa uporabi novo metodo ti. Double Threshold. Ta metoda ima cilj prepoznati

3 vrste slikovnih točk, močne, šibke in nepomembne. Zaradi tega postopka so uporabljeni vsi zgoraj navedeni filtri, zato da vozni pas zazna kot močne točke.



Slika 16: Canny filter, lasten vir

7.1.8 Houghova transformacija črt

Houghova transformacija črt nam omogoča, da iz robov izračunamo točke. Iz točk pa nam omogoča računanje ravnih premic, na katerih ležijo te točke. Če vse točke ne ležijo na isti premici, to filter tudi ustrezno poveže. Za vsako točko vzame potem le tej pripadajočo premico, in tako dobimo sklop daljic. Pri Houghovi transformaciji črt uporabljamo polarni koordinatni sistem s spremenljivkami **r** in $\boldsymbol{\emptyset}$. Pri takšni transformaciji bomo izpostavili črte v polarnem sistemu. Tako se enačba glasi:

$$\mathbf{y} = \left(-\frac{\cos\theta}{\sin\theta}\right)\mathbf{x} + \left(\frac{\mathbf{r}}{\sin\theta}\right) \tag{7.7}$$

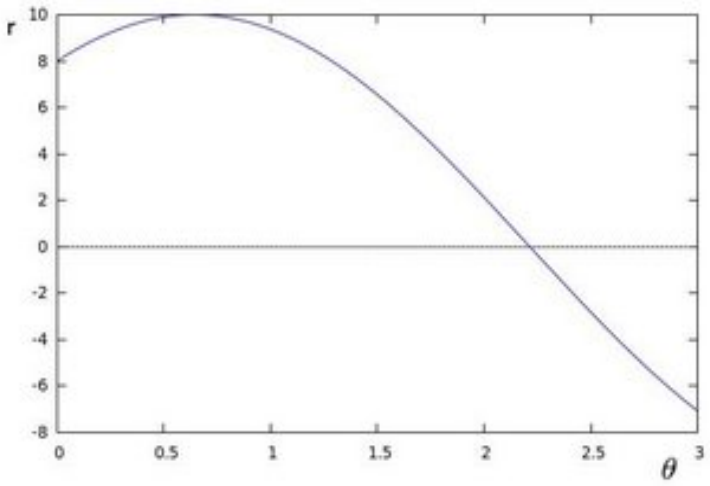
Enačba 7: Hughova transformacija (polarni koordinatni sistem), vir[12]

V osnovi lahko vsaki točki $\mathbf{x_0}$ **in** $\mathbf{y_0}$ definiramo družino črt, ki potujejo skozi njih:

$$r_\theta = \mathbf{x_0} * \cos\theta + \mathbf{y_0} * \sin\theta \tag{7.8}$$

Enačba 8: Družina črt, ki poteka skozi točki, vir[12]

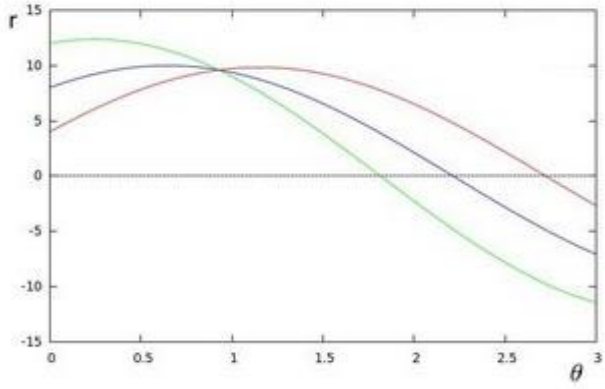
To pomeni, da vsak par r_0 in θ predstavlja vsako črto, ki poteka mimo $\mathbf{x_0}$ in $\mathbf{y_0}$. Če za dana $\mathbf{x_0}$ $\mathbf{in}$ $\mathbf{y_0}$ najdemo družino črt, ki poteka skozi to točko, dobimo sinusno obliko. Če za primer vzamemo $\mathbf{x_0} = \mathbf{8}$ in $\mathbf{y_0} = \mathbf{6}$, dobimo naslednje (pri $\theta - r$):



Slika 17: Sinusna oblika pri odkrivanju družine črt, vir[3]

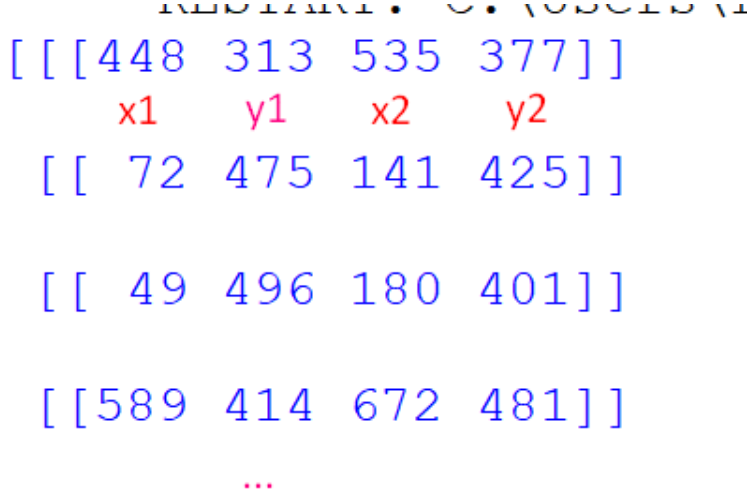
Tukaj upoštevamo samo točke, ki ustrezajo pogojem **r>0** in **0** < θ < **2**π (slika 17).

Če se sinusne oblike oziroma krivulje različnih točk sekajo pri $\theta - r$ pomeni, da ležijo na enaki premici (slika 18).



Slika 18: Sekajoče se krivulje točk, ki ležijo na isti premici, vir[4]

Houghova transformacija črt se šteje pod verjetnostno, kadar pridobivamo naključne vzorce robov (v našem primeru). Iz takšne transformacije dobimo 2 krajišči daljice (slika 19).



Slika 19: Izračunane točke verjetnostna H.T. (Houghova transformacija), lasten vir

Postopek za verjetnostno transformacijo (P.H.T. – angl. probability hough transformation) je podoben preprostemu (S.H.T. – angl. simple hough transformation).

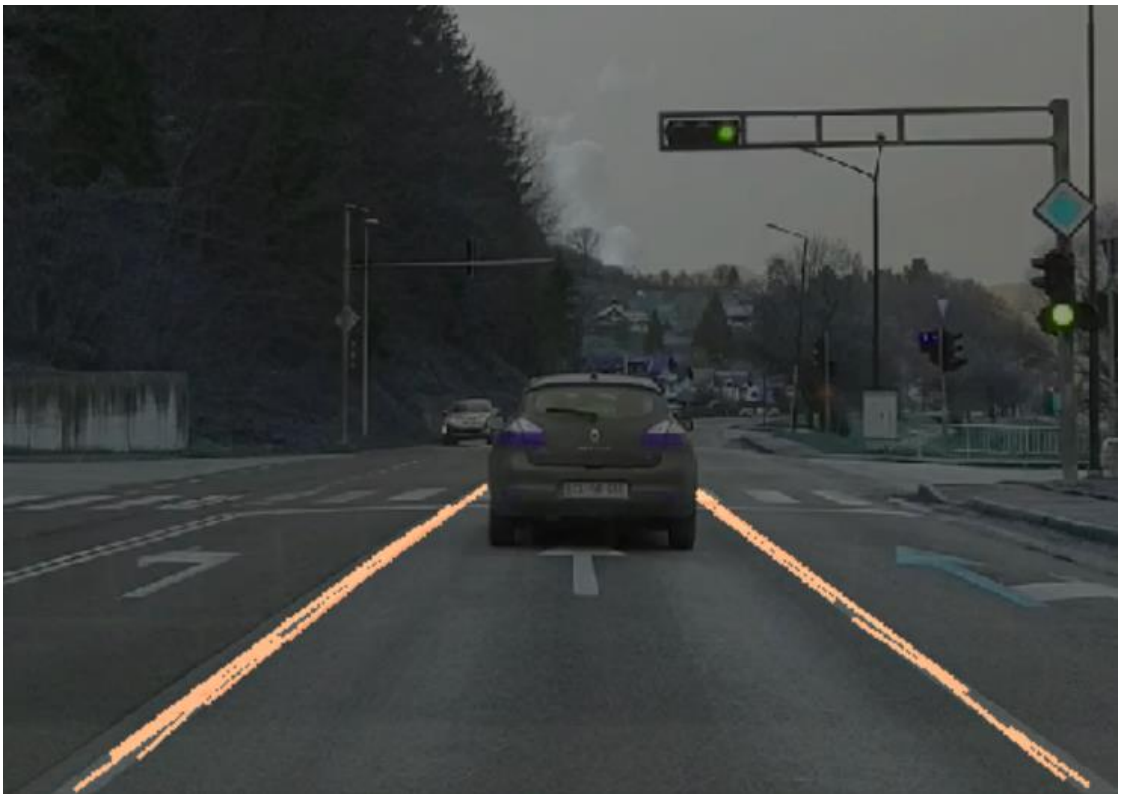
Razlika je v tem, da namesto da uporabimo vse M točke robov, uporabimo le podmnožico m. Ker je **m < M**, se zahtevnost faze zmanjša z **O (M.Nθ)** na **O (m.Nθ)**. V osnovi to deluje, zato ker bo naključna podmnožica množice **M** natančno predstavljala vse lastnosti in šum glede na območje, zasedeno na sliki (velikost **M** je enaka velikosti slike). Izbira manjše vrednosti za **m** bo zagotovila hitrejši algoritem, vendar pa ne sme biti tako majhna, da podrobnosti ni več mogoče zaznati.

7.2 Izdelava maske ter obdelava videoposnetka

Po končani obdelavi, kjer imamo sedaj zaznan vozni pas v obliki daljic, ki smo jih izračunali v postopku Houghove transformacije, pa je čas, da rezultate smiselno povežemo v obliko, ki nam je najbližja. Najlažje razumljiv je tako rezultat na sliki. Zato smo zaznane črte voznega pasu dodali neobdelani sliki in jim določili živo barvo. Tako lahko naš rezultat razumejo tudi ljudje, ki nimajo naprednega znanja matematike …

Za prekrivanje smo ustvarili lastno masko, ki za parametre dobi verjetnostne Houghove črte (mejne točke daljic) in neobdelano sliko. Tukaj zdaj ustvarimo novo plast sliki in ji določimo barvo na prosojno. Na to prazno plast nato narišemo daljice z živo barvo s pomočjo modula lines (knjižnica OPEN CV2).

V naslednjem koraku neobdelano sliko in masko združimo s pomočjo modula AddWeight (knjižnica OPEN CV2 - slika 20).



Slika 20: Neobdelana slika združena z masko, lasten vir

Vsako neobdelano sliko smo pridobili iz zanke, ki iz videa pridobi vse sličice v sekundi. Zanka torej bere skozi vse sličice v videu. Za vsako sličico smo nato dodali vso obdelavo. Vsako neobdelano sliko, ki je bila združena z masko, pa smo dodali v tabelo, imenovano IMG_ARRAY. Če je proces filtriranja slike, javi napako, pa se v tabelo doda samo neobdelana slika.

Na koncu pa s pomočjo knjižnice SCIPY vse sličice v tabeli sestavimo v video (format avi), da si lahko ogledamo obdelani posnetek vožnje (vsi posnetki in gradivo so v prilogi).

```
1.947526216506958
1
```

Slika 21: Čas obdelave 1 sličice v sekundah, lasten vir

8 PREPOZANAVA OBJEKTOV S POMOČJO RAČUNALNIŠKEGA VIDA

Prepoznavanje objektov s pomočjo računalniškega vida je sploh mogoče na podlagi računalniškega učenja. Če predhodno programa ne bi naučili, kateri skupek točk predstavlja kateri objekt, bi lahko zaznavali samo objekte, ne pa tudi vrste le-teh (npr. avtomobil, luč na semaforju …). V avtomobilski industriji je prepoznavanje vrste objekta za avtopilota zelo pomembno. Pomembno je zaradi načina odločanja algoritma.

Na primer, če avtomobil zazna objekt pred seboj, je edina logična odločitev zaviranje, medtem ko bi lahko pri prepoznavi vrste ugotovil, da je pred avtomobilom samo škatla kartona, ki ne bo poškodovala avtomobila ali povzročila kakršnekoli druge škode.

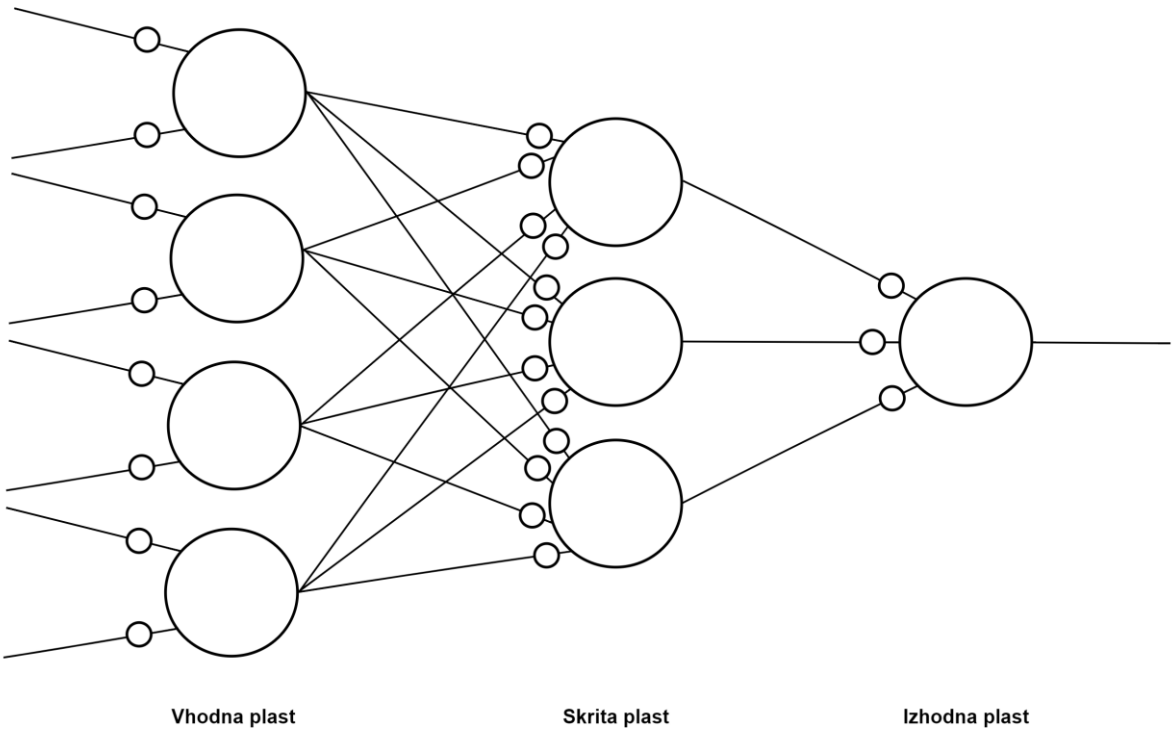
Zato je poleg prepoznave voznega pasu izjemno pomembna tudi prepoznava drugih objektov okoli avtomobila.

8.1 Nevronska mreža

Nevronska mreža je naprava ali algoritem za obdelavo informacij, ki deluje po vzoru človeških možganov. Sestavljena je iz množice umetnih nevronov. Nevronska mreža spada v skupino nenadzorovanega računalniškega učenja.

Pomen nevronskih mrež je samodejno učenje programa. Seveda znamo tudi ljudje podati navodila računalniku, a tega ne znamo opraviti s takšno natančnostjo in hitrostjo, kot lahko to stori nevronska mreža.

Nevroni so osnovni gradniki nevronskih mrež in so funkcije, ki imajo različno obtežene vhode (vsak podatek ima drugačno pomembnost). Vsi nevroni so med seboj tudi povezani. Vsak nevron ima samo 1 izhod. Množice nevronov delimo v sloje. Nevronska mreža lahko ima tako enega (enostavna) ali več (kompliciranа) slojev (slika 22).



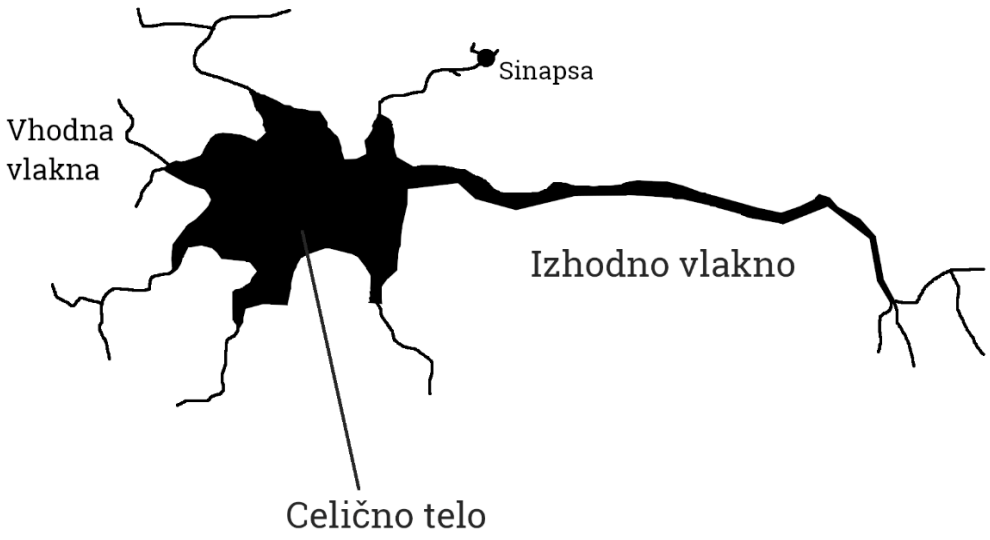
Slika 22: Topologija nevronske mreže, lasten vir

Kot že omenjeno, se pri takšnem računalniškem učenju algoritem uči sam. V postopku učenja se nato uteži vhodov posameznih nevronov, povezave med nevroni in prag, pri katerem nevron na izhodu odda signal, ustrezno oblikujejo.

8.1.1 Organski nevron

Nevron v človeških možganih sestavljajo trije glavni deli (slika 23):

- množica vstopajočih vlaken oz. dendritov,
- Celično telo oz. soma,
- eno izstopajoče vlakno oz. akson.



Slika 23: Človeški nevron, lasten vir

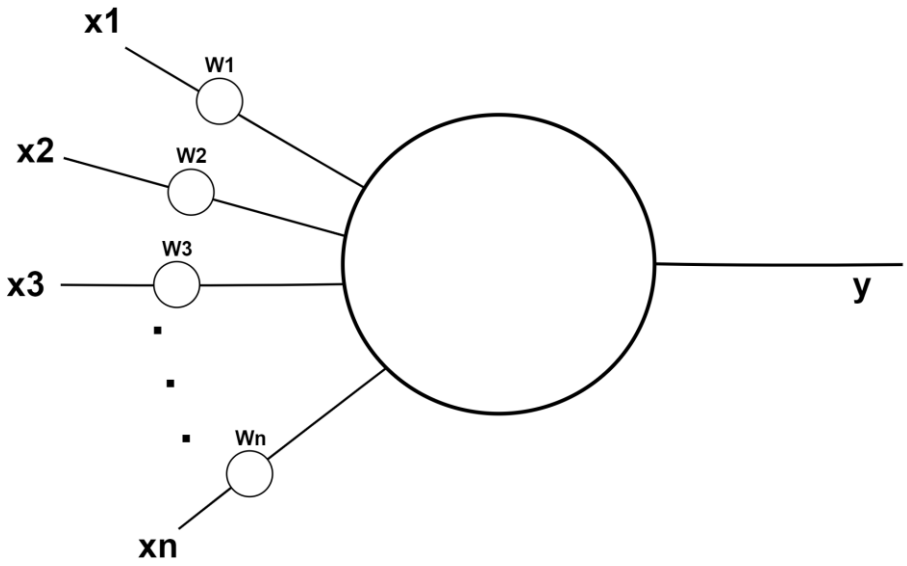
Človeški možgani vsebujejo približno 10^{11} nevronov. Teoretično lahko ima nevron tudi do 10000 ali več vhodnih povezav oziroma dendritov.

Sinapsa je lokacija na vhodni povezavi, kjer se ta nevtron stika z izhodno povezavo prejšnjega nevtrona. Nevroni generirajo električne impulze, ki se prenašajo vzdolž izhodnega vlakna do vhodnega vlakna drugega nevtrona. Od sinaps je potem odvisno, ali gre za vzbujanje ali zaviranje tega nevrona (+ ali - sinaps). Obnašanje sinaps se lahko med delovanjem tudi spreminja.

Nevron vsako sekundo generira od nekaj pa do 100 impulzov na sekundo (pri napravah to merimo z Hz).

8.1.2 Umetni nevron

Pri umetnem nevronu sedaj poizkušamo imitirati obnašanje organskih nevronov (slika 24). Vhodna vlakna tako predstavimo z vhodi x_i, izhodno vlakno pa z izhodom y. Učinkovitost sinaps oziroma utež (angl. weight) modeliramo z realnim številom w_i.



Slika 24: Umetni nevron, lasten vir

Vhod x_i predstavlja prisotnost ter frekvenco verige električnih impulzov v vlaknu. Tukaj vzbujanje ali zaviranje elektrona nadomestimo z binarnim številom 0 ali 1. Prav tako modeliramo realne vrednosti med (npr. med 0 in 1) kot modeliranje frekvence električnih impulzov.

Obtežen vhod v celično jedro predstavlja seštevek električnih impulzov iz vseh vhodnih vlaken (dendritov). Predstavljamo si ga lahko z naslednjo formulo:

$$s = \sum_i w_i x_i \tag{8.9}$$

Enačba 9: Obtežen vhod nevrona, vir[12]

Izhod y je pri tem monotono naraščajoča funkcija obteženega vhoda s:

$$y = f(x) \tag{8.10}$$

Enačba 10: Monotono naraščujoča funkcija obteženega vhoda, vir[12]

Funkcija f je prenosna oziroma aktivacijska funkcija nevrona. Najbolj znane funkcije za nevrone so pragovna (McCulloch-Pittsov model), linearna in sigmoidna.

8.1.3 Razdelitev nevronskih mrež

Nevronske mreže lahko delimo glede na tip podatkov na vhodnem in izhodnem vlaknu. Tukaj ločimo binarne in zvezne nevronske mreže.

Delimo pa jih lahko tudi glede na to, ali prejšnji izhodi nevronske mreže vplivajo na trenutni izhod: dinamične (izhodi nevronov so rekurzivno povezani nazaj na vhod) in statične nevronske mreže.

Pleada različnih vrst nevronskih mrež: binarni in zvezni perceptroni, Hopfieldove nevronske mreže, nevronske mreže z radialnimi baznimi funkcijami, Elmanove nevronske mreže, samoorganizirajoče se nevronske mreže itd.

8.2 YOLO

Zaznavo objektov ter vrste le-teh smo upravljali v Pythonu, in sicer zopet s pomočjo knjižnice OPEN CV2 in s pomočjo nevronske mreže YOLO (slika 25). Sistem računalniškega učenja in hkrati algoritem nevronske mreže je torej YOLO, medtem ko je ogrodje za ta sistem OPEN CV2. Za YOLO obstajata še ogrodji Darkflow in Darknet.

YOLO je že naučena nevronska mreža, ki lahko zazna 80 različnih vrst objektov (avtomobili, vlaki, nahrbtniki, vilice, semaforji, osebe …). Seveda pa lahko nevronsko mrežo še nadaljnje naučimo prepoznave drugih vrst objektov. Proces nadaljnjega učenja bi nato izgledal tako, da bi algoritmu posredovali čim več slik neke vrste objekta in mu povedali, za kateri objekt gre. Nevronska mreža si nato poizkusi zapomniti oblike dane vrste objekta. Več kot bo slik za učenje, natančnejša bo prepoznava.

Pri programiranju navadno vrste objektov nastavimo kot razrede. Pri YOLO so ti razredi zapisani v datoteki coco.names, vendar pa so v angleščini. Za prepoznavo smo napisali svoje objekte v slovenščini v datoteko coco1.names. Za delovanje YOLO algoritma potrebujemo tudi uteži za algoritem, ki jih najdemo v datoteki yolo.weights in pa lastnosti algoritma, ki jih najdemo v datoteki yolo.cfg.

V Python sedaj vključimo OPEN CV in zgoraj navedene datoteke. S pomočjo datoteke cfg in weights lahko izračunamo, kateri objekti so na sliki. To deluje tako, da za vsako vrsto objekta algoritem zapiše samozavest s številom od 0 do 1. Tukaj se zgodi celotna prepoznava vrst objektov.

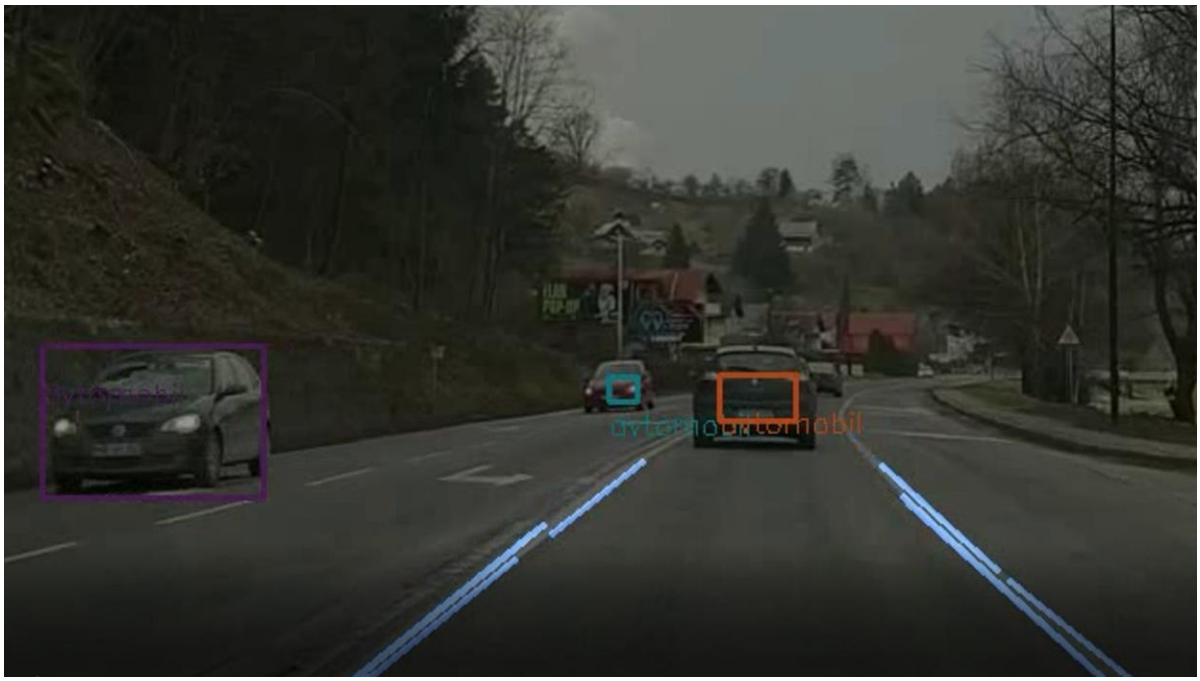
```python
import cv2
import numpy as np
import matplotlib.pylab as plt
from scipy.ndimage import rotate

vidcap = cv2.VideoCapture('input1.avi')
success,image = vidcap.read()
count = 0
alpha = 2
beta = -140
img_array = []
net = cv2.dnn.readNet("yolov3-tiny.weights", "yolov3-tiny.cfg")
razredi = []
razredi1 = []

with open("coco.names", "r") as datoteka:
    razredi = [line.strip() for line in datoteka.readlines()]
with open("coco1.names", "r") as datoteka:
    razredi1 = [line.strip() for line in datoteka.readlines()]

imena_plasti = net.getLayerNames()
izhodne_plasti = [imena_plasti[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 190, size=(len(razredi), 3))

while success:
    visina, sirina, kanali = image.shape

    #izluščevanje podrobnosti iz slike
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(izhodne_plasti)

    #Ustvarjanje maske
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Prepoznan objekt
                center_x = int(detection[0] * sirina)
                center_y = int(detection[1] * visina)
                w = int(detection[2] * sirina)
                h = int(detection[3] * visina)
                # Koordinate štirikotnika
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    font = cv2.FONT_HERSHEY_SIMPLEX
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(razredi1[class_ids[i]])
            color = colors[i]
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, label, (x, y + 30), font, 0.5, color, 1)

    img_array.append(image)
    success,image = vidcap.read()
    count += 1

out = cv2.VideoWriter('final1.avi', cv2.VideoWriter_fourcc(*'DIVX'), 15, (720, 500))
```

Slika 25: Algoritem za prepoznavanje vrste objektov, lasten vir

Glede na samozavest lahko določimo zaznane objekte. V našem primeru smo naredili novo masko, kjer pri vsaki vrsti objekta s samozavestjo, večjo od 0.5, naredi nov kvadrat okoli zaznane vrste in iz izbranega indeksa razredov zapiše na sliko zraven kvadrata tudi ime.

Za barve smo naredili novo tabelo z vsemi barvami do vrednosti 180, saj smo ugotovili, da se nasičena modra barva vidi zelo slabo.

Za vsako vrsto objekta nato določimo drugačno barvo. V naš program sedaj vstavimo filtriran videoposnetek s prepoznavo voznega pasu, ga razbijemo na sličice, sličice obdelamo s prepoznavo vrste objektov in posnetek ponovno sestavimo (slika 26).



Slika 26: Prepoznava voznega pasu in vrste objektov, lasten vir

9 SISTEM ZA URAVNAVANJE V VOZNI PAS

Sistem za uravnavanje v vozni pas je algoritem ali program, ki s pomočjo matematičnih funkcij uravnava avtomobil med 2 prepoznani premici. Če smo se predhodno poglobili v zaznavo voznega pasu, pa se zdaj spustimo k uporabi tega sistema. K nadzorovanemu učenju računalnika smo želeli implementirati pomoč pri uravnavanju v vozni pas.

To lahko storimo s pomočjo matematike. Pri nadzorovanem sistemu lahko računamo koeficiente premice, saj v procesu dobimo točke na premici za vsako stran voznega pasu. Formula za izračun kota med premicama se glasi:

$$\tan\alpha = \frac{\mathbf{k2} - \mathbf{k1}}{\mathbf{1} + \mathbf{k1} * \mathbf{k2}} \tag{9.11}$$

Enačba 11: Račun kota med premicami

Glede na pravokotnico smo nato izračunali približni osnovi kot med vsako premico (stran voznega pasu). Približna osnovna kota smo zdaj uporabili kot privzeta kota.

Tukaj si lahko dodamo spremenljivko, imenovano KOT_VOLANA in s prireditvenim stavkom nastavimo vrednost na 0 (privzeta vrednost), saj je privzeto volan naravnan naravnost. Večina družinskih avtomobilov ima rotacijo volana med 972 in 1152°, medtem ko je rotacija koles okoli 270°.

Če rotacijo volana delimo s rotacijo kolesa, dobimo koeficient, ki nam pove, za koliko stopinj se obrne kolo pri premiku volana za 1°.

$$\frac{\mathbf{972}}{\mathbf{270}} = \mathbf{3.6} \tag{9.12}$$

Enačba 12: Računanje koeficienta za premik koles

kar pomeni, da pri rotaciji volana za 1°, premaknemo kolesa avtomobila za 3.6°.

Če bi se eden izmed kotov povečal, drugi pa pomanjšal, mora program večji kot zmanjšati na privzeto vrednost. To storimo tako, da najprej izračunamo spremembo kota, kar storimo tako, da od privzetega kota odštejemo novi kot. Nato moramo na podlagi spremembe zavrteti naš volan. Naša kolesa se morajo obrniti pod kotom, ki je enak spremembi.

Ker pa se kolesa ne vrtijo enako kot volan, moramo za vrtenje volana upoštevati prej izračunano razmerje 3.6. Če moramo tako kolesa zavrteti za npr. 20°, je posledično rotacija volana enaka zmnožku želenega kota koles in razmerja med volanom in kolesi. Zdaj vemo, da se v takšnem primeru volan obrnemo za 72°.

Pri vožnji v ovinek se kolesa obrnejo. Ker pa snemamo s 30 sličicami na sekundo in obdelamo le 1 sličico na 4 sekunde, vidimo, da programski jezik Python ni primeren za prepoznavo voznega pasu preko filtrov in funkcij, saj je prepočasen. Veliko hitreje bi takšno nalogo opravil programski jezik C, vendar pa bi tukaj morali najverjetneje napisati svoje filtre in funkcije za celotno obdelavo.

To pomeni, da bi računalnik spremenil pozicijo volana in koles le 1 x na 2 sekundi, kar se mogoče zdi dokaj hitro. Verjetno bi sistem tudi deloval do določene hitrosti, potem pa bi bili premiki preveliki in bi posledično prišlo do prekomernega krmiljenja, saj bi se začelo dogajati, da bi bila naslednja slika obdelana šele po tem, ko bi že opravili zavoj.

10 HITROST RAČUNANJA IN VARNOST

Hitrost računanja vsekakor vpliva tudi na varnost. Teoretično hitreje kot računamo, bolj je sistem varen. Zato je tudi toliko navdušenja za novi sistem Tesle, ki lahko računa kar 2400 sličic na sekundo. Za obdelavo 1 sličice vsake 10 cm naše vožnje bomo sedaj izračunali, koliko sličic na sekundo potrebujemo obdelati pri najvišji hitrosti v Sloveniji, zato da si bomo lažje predstavljali števila in varnost, ki je povezana z njimi.

Najvišja dovoljena hitrosti v Sloveniji je 130 km/h. Tukaj lahko uporabimo fizično formulo za hitrost $v = s/t$, da izračunamo čas.

Tako vstavimo:

$$\mathbf{130(km/h)} = \frac{\mathbf{0.1}(\boldsymbol{m})}{\boldsymbol{t}} \tag{10.13}$$

Enačba 13: Enačba za hitrost

ter množimo s časom in delimo hitrost. Preden izračunamo, moramo vse enote postaviti na enako osnovo. To storimo tako, da km/h delimo s 3.6 ter tako dobimo m/s.

$$\frac{\mathbf{0.1(m)}}{\mathbf{36}(^{\boldsymbol{m}}/_{\boldsymbol{s}})} = \mathbf{t} \tag{10.14}$$

Enačba 14: Izpostavljen čas pri enačbi za hitrost

Nato dobimo čas:

$$\mathbf{t} \cong \mathbf{0.0027} \tag{10.15}$$

Enačba 15: Izračunan čas

Ker je čas manjši od 1, pomeni, da moramo za računanje sličic na sekundo uporabiti naslednjo formulo:

$$\frac{\mathbf{1}}{\boldsymbol{t}} = \boldsymbol{fps} \tag{10.16}$$

Enačba 16: Računanje sličic na sekundo

Se pravi, da za računanje 1 sličice vsake 10 cm naše vožnje potrebujemo sistem, ki omogoča 371 sličic na sekundo.

1 posneta in obdelana sličica na vsake 10 cm se sploh ne zdi razdalja, ki bi zagotavljala optimalno varnost. Vendar pa je takšna razdalja že kar dobra in omogoča kar natančno delovanje sistema za uravnavanje v vozni pas.

Tesla tako pri obdelavi 2400 sličic na sekundo obdela 1 posneto sličico vsake n centimetrov. N izračunamo tako, da najprej dobimo čas s pomočjo sličic na 1 sekundo.

$$\frac{\mathbf{1}}{\boldsymbol{fps}} = \boldsymbol{t} \tag{10.17}$$

Enačba 17: Računanje časa iz sličic na sekundo

Kar pomeni, da je čas $4.16 * 10^{-4}$.

Ker nas tokrat zanima razdalja, uporabimo formulo:

$$\boldsymbol{s} = \boldsymbol{v} * \boldsymbol{t} \tag{10.18}$$

Enačba 18: Enačba za razdaljo

in tako dobimo razdaljo približno 1.5cm. Se pravi, da lahko sistem Tesle pri najvišji dovoljeni hitrosti v Sloveniji posname in obdela sličico na vsake 1.5cm. Tesla svoj trenutni sistem uvršča v zelo varen, vsaj s stališča hitrosti obdelave sličic.

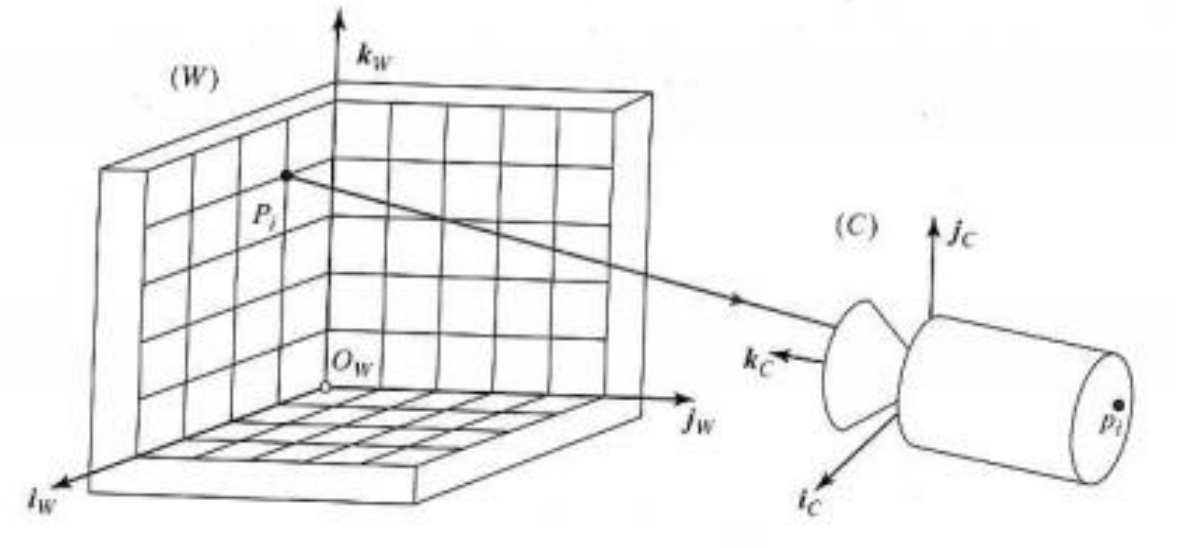
Seveda pa je tukaj še nevronska mreža, ki se razvija vsak dan s pomočjo analize vožnje uporabnikov tega avtomobila, ki še ni dovolj varna oziroma napredna,, da bi lahko bil voznik odsoten.

11 GEOMETRIJSKA KALIBRACIJA KAMER

Vsaka kamera ima svoja odstopanja. Glede na to, da mora naš program računati pot in uravnavanje avtopilota, pa morajo biti »oči« oziroma kamera vsakega avtomobila enaka. Ne samo enaka, mora imeti tudi isti pogled oziroma sliko. Tako moramo navadne kamere pri računalniškem vidu kalibrirati, saj lahko drugače pride do zalo nevarnih napak.

Kamero se kalibrira s pomočjo realnega koordinatnega sistema s pobarvanimi območji (pike, črte …). Pri enakih kamerah pride do odstopanj, saj ne more biti vsaka kamera ista. Zato slikamo omenjeni realni koordinatni sistem, da dobimo neko predpostavko, kaj kamera vidi (vedeti moramo tudi položaj kamere v tem koordinatnem sistemu - slika 27). Na podlagi slike, ki jo kamera posname, lahko zdaj primerjamo to s kamero, na kateri je bil narejen program. Po navadi zdaj 2 posneti sliki primerja program ali programska oprema. Na večini kamer pa zdaj najdemo odstopanja nekaj desetin milimetra.

Zdaj, ko vemo, kakšna so odstopanja, lahko programsko kamero nastavimo, zato da bo posnetek isti predpostavljeni kameri (kameri, na kateri je bil narejen program). Seveda programska oprema to stori s pomočjo matematike. Temu postopku pravimo kalibracija. Navadno je ta postopek natančen na tisočinko milimetra ali še bolj.



Slika 27: Geometrijsko kalibriranje kamere, vir[5]

12 ZAKLJUČEK IN UGOTOVITVE

Ugotovili smo, da je sistem za uravnavanje vozila v vozni pas in prepoznavo vrst objektov možno narediti doma. Za izdelavo takšnega sistema potrebujemo znanje iz programiranja in matematike. Pri procesu je vključenih nekaj zahtevnejših matematičnih formul, ki niso podane v sklopu srednješolskih znanj. Seveda pa potrebujemo tudi omenjeno opremo.

Videli pa smo, da bi ob uporabi kamer potrebovali namenske kamere ali stojala, saj je pogled kamere zelo težko nastaviti na roko. Potrebovali pa bi tudi sistem za dušenje tresljajev, saj jih ima naš posnetek ogromno, s tem pa pridobi ogromno nezaželenega šuma, ki ga kasneje odpravijo filtri (tako podaljšamo čas obdelave).

Izvedeli smo tudi, da kljub izdelanemu sistemu, le-tega verjetno nikoli ne bomo mogli uporabiti v avtomobilu, saj je hitrost računanja prepočasna. Hitrost bi lahko povečali z uporabo programskega jezika C, vendar pa bi pri tem delo od nas zahtevalo še obširnejše znanje matematike in programiranja. Za hitrejše delovanje pa bi potrebovali tudi eno namensko strojno in programsko opremo.

Presenetilo nas je, da potrebujemo za uravnavanje v vozni pas 2 premici, medtem pa je naš algoritem filtriranja ustvarjal po najintenzivnejših robovih okoli 10 premic. Iz 10 premic pa je težje določiti 1, ki bo natančna in primerna za uravnavanje avtomobila (zato smo morali napisati svojo funkcijo, ki izračuna približni 2 premici).

Presenetljivo enostavno je bilo tudi vzpostavljanje nevronske mreže za prepoznavo vrst objektov. Tukaj smo morali določiti, katero nevronsko mrežo bomo uporabljali (YOLO), določiti uteži in prag (samozavest), ob katerem bomo prepoznali vrsto objekta. Zanimiva je bila tudi hitrost računanja takšne nevronske mreže. V teoriji naj bi omogočala nekje 222 obdelanih sličic na sekundo

Ugotovili smo, da današnji proizvajalci avtomobilov z avtopiloti (Tesla) poskrbijo za varnost na cesti s tehničnega stališča. Najrazvitejši sistemi avtopilotov za market potrošnikov pri najvišji hitrosti v Sloveniji obdela 1 sličico vsake 1.5 cm, medtem ko bi naš program najverjetneje nekje okoli 1 na vsake 70 m (pri odstranjevanju manj pomembnih filtrov tudi še vedno okoli 15 m).

Izvedeli smo, da se v avtomobilski industriji uporablja tudi kalibracija kamer, saj imajo kamere enakega proizvajalca lahko drugačen pogled. Odstopanja popravimo s pomočjo realnega koordinatnega sistema z narisanimi območji (pikami). Na 2 kamerah nato preverimo razlike zajete slike območij z realnega koordinatnega sistema (pri zajetih slikah vemo tudi pozicijo kamere v tem realnem koordinatnem istemu).

Tako lahko prvo hipotezo, da bo zaznavanje voznega pasu na slovenskih cestah natančno, delno potrdimo. Razlog odločitve leži v tem, da je sistem dokaj natančen. Problem pa so robovi, ki jih filter ne odstrani. Če iz takšnega sistema izračunamo natančnost, dobimo okoli

80 %. 80 % se morda zdi veliko, vendar pa je iz varnostnega stališča neustrezno (moralo bi biti okoli 97 % ali več).

Drugo hipotezo, da je takšen sistem moč izdelati brez osnov ali poznavanja matematike, lahko ovržemo. Pri pisanju raziskovalne naloge smo uporabili več kot 30 matematičnih formul. Večino teh pa sploh ne spada v sklop srednješolskega znanja matematike.

Tretjo hipotezo, da je procesorska moč pri uporabi računalniškega vida zelo pomembna, potrdimo, saj je le-ta zelo pomembna. V raziskovalni nalogi tega nismo mogli dokazati, saj smo celotno obdelavo obravnavali na enem računalniku. V teoriji pa je procesorska moč eden najpomembnejših gradnikov sistema računalniškega vida. To velja še bolj za obdelavo slik v realnem času.

Četrto hipotezo, da lahko računanje opravimo preko oblačnih storitev, lahko delno potrdimo. Naše oblačne storitev imajo običajno zakasnitev dovolj nizko, da jih uporabimo za računanje v sistemu za prepoznavo voznega pasu. Lahko pa pride do izpada omrežja (zmanjka omrežja), s tem pa bi izgubili vso procesorsko moč, naš avtomobil pa bi se najverjetneje zaletel.

Zadnjo hipotezo, računanje 4 sličic na sekundo je dovolj za varno usmerjanje avtomobila na slovenskih cestah, lahko zavrnemo. Za varno vožnjo moramo obdelati čim več slik v 1 sekundi, kolikor je le možno. Sistem 4 sličic na sekundo pa ni varen, čeprav je varnost relativna stvar. Pri 4 sličicah na sekundo bomo pri najvišji hitrosti v Sloveniji obdelali samo 1 sliko na približno 30 prevoženih metrov.

13 POVZETEK

Torej smo pri raziskovanju odkrili, kako delujejo sistemi za prepoznavo voznega pasu in katere filtre ter funkcije uporabljajo, kako ti filtri delujejo, kaj je nevronska mreža in kako z nevronsko mrežo YOLO in ogrodjem OPEN CV2 prepoznati vrsto objekta, kako s pomočjo kota med premicami ustvariti program za uravnavanje avtomobila v vozni pas, da je današnja tehnična (s stališča strojne opreme) varnost pri večini proizvajalcih visoka, ter da je potrebno kamere v avtomobilski industriji kalibrirati zaradi odstopanj (vsaka kamera ima neka odstopanja).

Ob celotnem procesu pa je nastalo kar nekaj naprednih programov. Nekateri izmed programov so algoritem za prepoznavo voznega pasu, algoritem za uravnavanje avtomobila v vozni pas, algoritem za prepoznavo vrst objektov … Kot rezultat pa je nastalo tudi več posnetkov, kjer so zgoraj navedeni algoritmi obdelali originalne posnetke vožnje (programi na spletišču Github – glej prilogo).

14 ZAHVALA

Raziskovalna naloga ne bi bila v takšni obliki, če nam pri nastajanju le-te ne bi pomagalo veliko ljudi. Zahvala je torej namenjena naslednjim:

- mentorju Roku Urbancu, za pomoč, voljo, vztrajnost, njegov prosti čas in spodbudo;
- staršem;
- Lidiji Šuster, prof., za lektoriranje;
- učiteljem ERŠ in ravnatelju Simonu Konečniku, univ. dipl. inž., za vso podporo in spodbudo;
- recenzentu raziskovalne naloge;
- komisiji Mladih raziskovalcev in koordinatorici gibanja Mladi raziskovalci Karmen Hudournik;
- vsem neomenjenim, ki so kakorkoli pomagali pri izdelavi naloge.

15 PRILOGE

Med priloge sodijo slike, urejevalni dokumenti slik, algoritmi, posnetki, obdelani posnetki, datoteke za sistem YOLO in drugo, kar bi utegnilo povzročiti nepreglednost raziskovalne naloge, če bi to umestili med ostalo besedilo:

A) Povezava do spletnega prostora: https://1drv.ms/u/s!AifiL-DLbMcEjxPk897cJUJUfxlM?e=JJBsHv (13. 02. 2020).

B) Povezava do sistema za upravljanje z izvorno kodo Github: https://github.com/Lahlukap669/Lane-assist-object-recognition (13. 02. 2020).

16 VIRI IN LITERATURA

Vsi viri, ki niso navedeni, so bili ustvarjeni s strani avtorja raziskovalne naloge. Za izdelavo slik smo uporabili odprtokodna programa Gimp in Inkscape ter brezplačni program Pencil.

16.1 Viri slik

Vir [1]: Sean Hollister. Čip avtopilota (Tesla). https://www.theverge.com/2019/4/22/18511594/tesla-new-self-driving-chip-is-here-and-this-is-your-best-look-yet (28. 1. 2020).

Vir [2]: Sofiane Sahir. Prepoznava robov s filtrom Canny. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123 (28. 1. 2020).

Vir[3,4]: Open CV. Houghove črte. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html (28. 1. 2020).

Vir[5]: izr. prof. dr. Božidar Potočnik, I. (2018/19). Uvod v računalniški vid in razpoznavanje vzorcev. 3. letnik RIT – UNI, Fakulteta za elektrotehniko, računalništvo in informatiko, str. 32.

16.2 Viri vsebine in enačb

1. Taz Tally. Razumevanje grayscale. https://www.linkedin.com/learning/advanced-photoshop-color-correction/understanding-grayscale-values-and-color (15. 2. 2020).
2. Sofiane Sahir. Canny filter. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123 (15. 2. 2020).
3. https://www.youtube.com/watch?v=6g4O5UOH304 (15. 2. 2020).
4. Open cv. Moduli. https://docs.opencv.org/2.4/modules/ (15. 2. 2020).
5. Jay Rambhia. Vrejetnostna Hough transformacija. https://jayrambhia.com/blog/probabilistic-hough-transform (15. 2. 2020).
6. Ian Macdonald. Verjetnostna Hough transformacija. https://pdfs.semanticscholar.org/58ce/e69ed2033f559f5ba579b48ac4359bcf524c.pdf (15. 2. 2020).
7. GeeksforGeeks. Grayscale slike. https://www.geeksforgeeks.org/python-grayscaling-of-images-using-opencv/ (15. 2. 2020).

8. Computerphile. Canny filter robov. https://www.youtube.com/watch?v=sRFM5IEqR2w (15. 2. 2020).
9. Computerphile. Kako delujejo filtri in zamegljenja? https://www.youtube.com/watch?v=C_zFhWdM4ic (15. 2. 2020).
10. Open cv. Dokumentacija. https://docs.opencv.org/3.4/ (15. 2. 2020).
11. Techtutorialsx. Python OpenCV: pretvarjanje slike v črno in belo. https://techtutorialsx.com/2019/04/13/python-opencv-converting-image-to-black-and-white/ (15. 2. 2020).
12. izr. prof. dr. Božidar Potočnik, I. (2018/19). Uvod v računalniški vid in razpoznavanje vzorcev. 3. letnik RIT – UNI, Fakulteta za elektrotehniko, računalništvo in informatiko.
13. Sergio Canu. YOLO prepoznava vrst objektov z uporabo OpenCV. https://pysource.com/2019/06/27/yolo-object-detection-using-opencv-with-python/ (15. 2. 2020).
14. Canny prepoznava robov. Ruye Wang. http://fourier.eng.hmc.edu/e161/lectures/canny/node1.html (15. 2. 2020).
15. Gaussov filter. ScienceDirect. https://www.sciencedirect.com/topics/engineering/gaussian-filter (15. 2. 2020).

17 AVTOR RAZISKOVALNE NALOGE

Luka Lah je dijak 4. letnika Elektro in računalniške šole (ERŠ) v Velenju. Za raziskovalno nalogo se je odločil, ker ga zanima programiranje, elektrotehnika ter nove tehnologije. Zanima ga tudi programiranje s programskimi jeziki, kot sta Python, Javascript ter drugi. Sam ima že tudi kar veliko izkušenj z delom s Pythonom, saj se je usposabljal na Malti (leta 2019), Arduinoti, Javascriptom, PHP-jem … Zelo je aktiven tudi na športnem področju, v atletiki je član v državni reprezentanci. V prihodnosti se želi ukvarjati s programiranjem, športom, avtomatizacijo procesov ter ostalimi športnimi in elektro ter računalniških stvarmi (slika 28).

Slika 28: Mladi raziskovalec Luka Lah, lasten vir