SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta elektrotechniky a informatiky

Evidenčné číslo: FEI-104376-92650

Neuroevolúcia autonómneho vozidla

Diplomová práca

2022 **Bc. Marko Chylík**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-104376-92650

NEUROEVOLÚCIA AUTONÓMNEHO VOZIDLA

DIPLOMOVÁ PRÁCA

Študijný program:	Robotika a kybernetika
Študijný odbor:	kybernetika
Školiace pracovisko:	Ústav robotiky a kybernetiky
Vedúci záverečnej práce/školiteľ:	prof. Ing. Ivan Sekaj, PhD.

Bratislava 2022 **Bc. Marko Chylík**

Slovenská technická univerzita v Bratislave
Ústav robotiky a kybernetiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022
Evidenčné číslo: FEI-104376-92650

STU FEI

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Marko Chylík**
ID študenta:	92650
Študijný program:	robotika a kybernetika
Študijný odbor:	kybernetika
Vedúci práce:	prof. Ing. Ivan Sekaj, PhD.
Vedúci pracoviska:	prof. Ing. Jarmila Pavlovičová, PhD.
Miesto vypracovania:	Ústav robotiky a kybernetiky

Názov práce: **Neuroevolúcia autonómneho vozidla**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom diplomovej práce je navrhnúť metodiku a vytvoriť algoritmy riadenia simulačného modelu kolesového autonómneho vozidla (AV). Vozidlo vníma svoje prostredie a pohybuje sa po určenej dráhe, pričom jeho správanie je determinované umelou neurónovou sieťou. Učenie siete je realizované evolučným algoritmom, ktorý minimalizuje zvolenú účelovú funkciu.

Úlohy:
1. Analyzujte požadovanú funkcionalitu AV.
2. Navrhnite komponenty AV pre vnímanie prostredia a pre jeho riadenie.
3. Navrhnite algoritmy riadenia AV pomocou neuro-evolúcie.
4. Získané výsledky vyhodnoťte.

Termín odovzdania diplomovej práce:	13. 05. 2022
Dátum schválenia zadania diplomovej práce:	17. 02. 2022
Zadanie diplomovej práce schválil:	prof. Ing. Jarmila Pavlovičová, PhD. – garantka študijného programu

Čestné vyhlásenie

Podpísaný Bc. Marko Chylík, čestne vyhlasujem, že som diplomovú prácu na tému **Neuroevolúcia autonómneho vozidla** vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol prof. Ing. Ivan Sekaj, PhD.

Bratislava, dňa 13.5.2022

.....

podpis autora

Poďakovanie

Ďakujem prof. Ing. Ivanovi Sekajovi, PhD. za jeho odborné rady, ktoré výraznou mierou prispeli k nájdeniu riešení pre dané úlohy.

ANOTÁCIA DIPLOMOVEJ PRÁCE

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný odbor: kybernetika

Študijný program: Robotika a kybernetika

Autor: Bc. Marko Chylík

Diplomová práca: Neuroevolúcia autonómneho vozidla

Vedúci diplomovej práce: prof. Ing. Ivan Sekaj, PhD.

Mesiac, rok odovzdania: Máj, 2022

Kľúčové slová: Autonómne vozidlo, neuroevolúcia, segmentačná neurónová sieť, CARLA, evolučné algoritmy, genetický algoritmus, viacvrstvová neurónová sieť

Cieľom práce je vytvoriť model autonómneho vozidla v simulačnom prostredí CARLA, ktoré je navrhnuté priamo na účely výskumu autonómnych vozidiel. Vozidlo bude schopné vnímať okolité prostredie a na základe podnetov z neho upravovať svoje jazdné vlastnosti tak, aby splnilo vopred preddefinovanú úlohu.

V rámci vnemu prostredia sa venujeme spracovaniu obrazu prednej kamery, ktorý použijeme na detekciu čiar pomocou hlbokej segmentačnej neurónovej siete, ktorá bude schopná robiť segmentáciu v reálnom čase.

Práve čiary budú oporným bodom pre orientáciu vozidla v priestore. Priestor budeme zároveň monitorovať radarom, ktorý bude hľadať prekážky v okolí vozidla. V neposlednom rade musíme vhodne integrovať navigačný systém, na základe ktorého bude vozidlo dostávať informácie o bodoch, ktoré vedú k žiadanému cieľu. Všetky tieto súčasti budú vstupmi neurónovej siete a ich fúzia vedie k úspešnému vyriešeniu úlohy.

Riadenie pohybu bude mať za úlohu klasická viacvrstvová neurónová sieť, trénovaná evolučnými algoritmami, ktoré optimalizujú správanie siete tak, aby čo najviac korelovala s definovanými kritériami pomocou účelovej funkcie. Tá bude zohľadňovať správanie vozidla na trénovacej dráhe, pričom minimalizáciou tejto funkcie nájdeme vhodné riadenie pomocou spomínanej MLP siete.

Takto natrénované vozidlo následne budeme testovať na rôznych testovacích scenároch, aby sme overili robustnosť navrhnutej riadiacej viacvrstvovej neurónovej siete.

MASTER THESIS ABSTRACT

Slovak University of Technology in Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Branch of Study: Cybernetics

Study Programme: Robotics and Cybernetics

Author: Bc. Marko Chylík

Master Thesis: Neuroevolution of Autonomous Car

Supervisor: prof. Ing. Ivan Sekaj, PhD.

Year, Month: 2022, May

Keywords: Autonomous Car, neuroevolution, segmentation neural network, CARLA, evolutionary algorithms, genetic algorithms, multi-layer perceptron neural network

Main goal of the thesis is to create an autonomous vehicle model in simulation environment named CARLA, which is very usable for the purposes of autonomous vehicles research. The vehicle will be able to perceive the environment and based on environment's impulses needs to edit its behavior to be able to complete the task.

In next chapter we dedicate to image processing of the front camera of the car, which will be used to segmentation of lanes by deep segmentation neural network, which needs to work in real-time to become useful for us.

Lanes will be the main entry for the vehicle's orientation in the environment. Area around the vehicle will also be monitored by radar to find possible obstacles in front of the car. We need to properly integrate the navigation system, which is crucial to get waypoints towards the goal. All of these will become entries to multi-layer perceptron network used to control the steering of the vehicle. The fusion of entries is the key part of the thesis.

The multi-layer perceptron network will be trained by evolutionary algorithms, especially genetic algorithm, which is optimizing the fitness function, which is used to evaluate the quality of driven path by the vehicle on training path. The global minimum of this function should show up the best solution.

After that, vehicle trained by neuro-evolution is going to drive on test paths, which are here to determine which of the solution has enough level of robustness.

Obsah

Zoznam použitých skratiek

GA – genetický algoritmus

NS – neurónová sieť

Actor – súčasť simulácie

SNS – segmentačná neurónová sieť

Dataset – zbierka dát potrebných na trénovanie neurónovej siete

AI – umelá inteligencia (z angl. Artificial inteligence)

Úvod

V dobe plnej rýchlych technologických objavov, ktoré nám denno-denne zjednodušujú život čelíme mnohým, ešte neprebádaným, či len slabo preskúmaným problémom, ktorých riešenie by mohlo ľudstvu priniesť veľké výhody. Jedným nich je aj riadenie vozidiel. Síce máme jasne definované pravidlá cestnej premávky, aj tak dochádza každodenne k veľkému počtu nehôd, pričom niektoré z nich končia žiaľ aj tragicky. Čo ak sa týmto úmrtiam dá zabrániť práve pomocou technológii? Vývoj technológii zvyšuje výrazne komfort ľudí, v takomto prípade dokonca aj zachraňuje ich životy. Problém autopilota v autách je už známy dlhé roky, avšak zatiaľ stále neexistuje žiadne dostatočne spoľahlivé (v tomto prípade dokonalé) riešenie. Keď sa však zamyslíme detailne nad problematikou, uvedomíme si, že je len otázkou času, kedy sa nám podarí navrhnúť taký systém, ktorého chybovosť bude veľmi blízko nule. Prečo? Počítače majú značnú výhodu oproti ľudom – ich vnemy vedia byť teoreticky neobmedzené. Môžu použiť veľké množstvo kamier a vidieť tak okolie celého vozidla, využitie ďalších pokročilých snímačov, pomocou ktorých bude riadiaci program mať dokonalý rozhľad o všetkom vo, ale aj mimo vozidla. V neposlednom rade vieme využívať prediktívne systémy a autá „budú vedieť“ už niekoľko sekúnd vopred, čo sa stane. Ak si predstavíme situáciu, kde bude väčšina áut v reálnej premávke fungovať v režime autopilota, budú môcť kolaborovať v prípade hroziacej kolízie pomocou komunikácie medzi vozidlami, čo je opäť výhoda, ktorú ľudia za volantom nemajú. Ľudstvo však v tejto otázke čakajú aj netechnické výzvy, najmä čo sa týka etiky a legislatívy pri riadení vozidiel autopilotom.

V tejto práci sa budeme venovať návrhu riadiaceho algoritmu vozidla tak, aby sa vozidlo stálo autonómnym – to znamená, že bude schopné vykonávať akcie podľa vlastného uváženia. Cieľom je, aby boli prvky autonómie viditeľnými voči klasickým riadiacim algoritmom, ktoré fungujú na princípoch akcie-reakcie. Vozidlo bude schopné využívať vlastné skúsenosti z predošlých podobných situácii a tým pádom reagovať vopred na už známe scenáre. Základným prvkom riadenia bude metóda umelej inteligencie, ktorá presne napodobňuje ľudské myslenie – neurónovú sieť. Táto neurónová sieť bude učená evolučnými algoritmami. Tieto algoritmy napodobňujú prírodnú evolúciu a v tomto prípade doslova simulujú učenie ľudského mozgu pri učení šoférovania auta v autoškole.

Vývoj takýchto algoritmov v reálnom svete by bol samozrejme priveľmi nákladný – preto na overenie algoritmov riadenia použijeme pokročilý simulátor vozidiel, ktorý disponuje reálnou fyzikou ako áut samotných, tak aj okolitého prostredia, v ktorom autá jazdia. Pre naše účely je najvhodnejším open-source simulátor CARLA, ktorá ponúka programátorské rozhranie pre realizáciu experimentov spojených s autonómnymi vozidlami, taktiež reálnu fyziku, dopravné značenia a ďalšie, pokročilé možnosti.

CARLA ponúka okrem iného výborné programátorské rozhranie medzi simulátorom a riadiacim programom v jazyku Python, ktorý použijeme na realizáciu celej práce. Python je jedným z najpopulárnejších jazykov súčasnosti, je open-source, podobne ako CARLA.

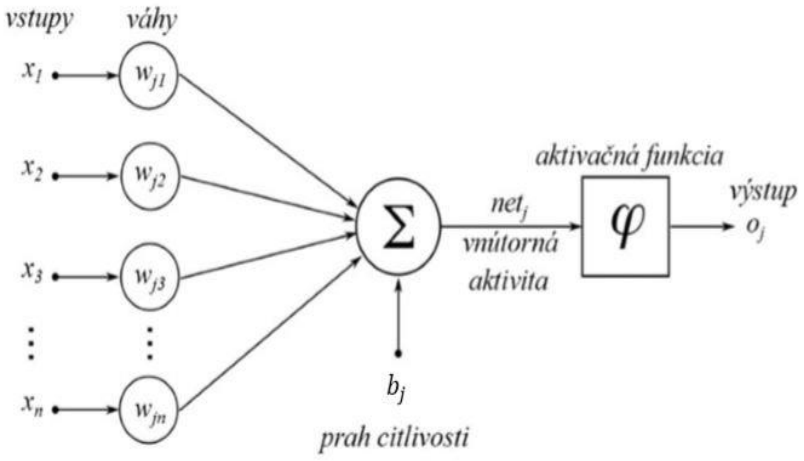
1 Umelé neurónové siete

Neurónová sieť (ďalej NS) je tvorená z veľkého počtu základných stavebných kameňov – neurónov. Už názov napovedá, že motivácia k vzniku NS je napodobnenie ľudského mozgu pomocou rôznych matematických operácii. Mozog disponuje schopnosťou dávať veci do súvislostí a dokáže tak z nich vyvodzovať určité závery – presne toto isté dokáže aj NS, avšak musíme dáta pre ňu dostatočne predpripraviť do matematickej (či počítačovej) reprezentácie. Neurónová sieť je však v podstate istá forma expertného systému, keďže ju vieme natrénovať na konkrétny typ problému – detekcia ľudí, riadenie systémov, či segmentácia obrazu, avšak nie je možné použiť rovnakú sieť na všetky tri typy spomenutých úloh – každá takáto úloha potrebuje špecifickú topológiu a častokrát aj iný spôsob trénovania siete.

1.1 Neurón

Ako sme spomínali v predošlej časti, neurón je základným stavebným kameňom NS. Výstup neurónu vyzerá nasledovne:

$$o_j = \varphi\left(\sum_{i=1}^{n} w_{ji}x_i + b_j\right) \tag{1}$$



Obrázok 1: Neurón

Zo vzťahu vyplýva, že neurón najprv vynásobí váhy synapsií so vstupmi, ktoré sú práve pomocou týchto synapsií prepojené s neurónom. Následne každý neurón má aj svoj bias, alebo inak povedané prah citlivosti. Týmto vie neurónová sieť posunúť hodnoty aktivačných funkcií. Výstup neurónu však vychádza priamo z aktivačnej funkcie.

1.2 Aktivačné funkcie

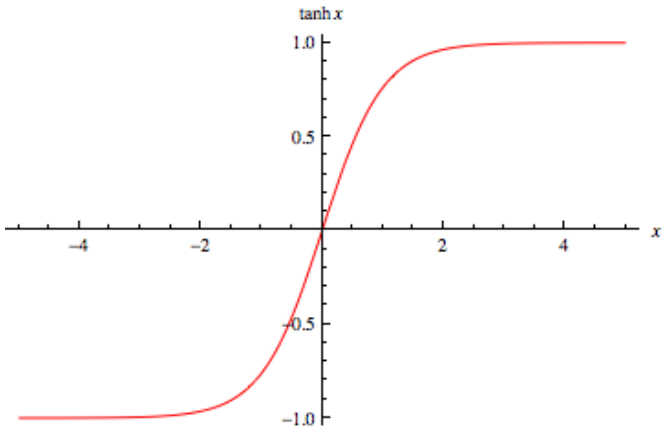
Spravidla sa ako aktivačné funkcie používajú nelineárne funkcie – vnášajú do NS práve týmito nelinearitami nové možnosti. Najpoužívanejšími sú *hyperbolický tangens*, *sigmoida*, či *ReLU*. V našej práci používame ako aktivačnú funkciu hyperbolický tangens (tanh), ReLU a špeciálnu nelinearitu typu *swisch*.

1.2.1 Hyperbolický tangens

Pri viacvrstvovej perceptrónovej sieti použitej na riadenie auta budeme ako nelineárnu aktivačnú funkciu používať hyperbolický tangens. Jej tvar je

$$\tanh(x) = \left(\frac{2}{e^{-2x}+1}\right) - 1 \qquad (2)$$

Vzťah (2) si prenesieme do dvojosového grafu tak (Obrázok 2), aby sme vedeli analyzovať definičný obor a obor hodnôt tejto funkcie:



Obrázok 2: Funkcia y = tanh(x)

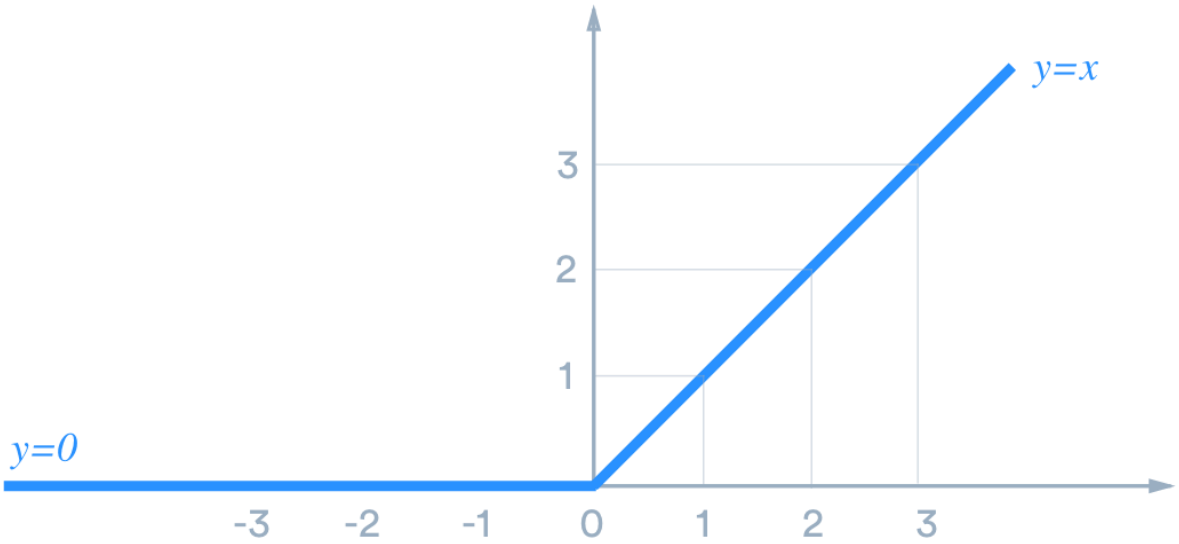
vyššie vidíme, že síce je funkcia definovaná na celej množine reálnych čísel, jej hodnoty sa spojite menia len v rozsahu (definičnom obore) <-3;3>, avšak najväčšiu zmenu vidieť medzi <-1;1>. Obor hodnôt je taktiež <-1;1>. Z analýzy teda vyplýva, že na plné využitie funkcie budeme musieť váhy nastaviť vhodne tak, aby neuróny pred aktiváciou dosahovali hodnoty v rozmedzí definičného oboru vhodného pre naše použite.

1.2.2 Rektifikovaná lineárna jednotka (ReLU)

Hlboké siete, ktoré využijeme na vnem okolia, používajú ako nelinearitu v drvivej väčšine prípadov ReLU. Obdobne, ako pri hyperbolickom tangense uvádzame tvar funkcie:

$$\mathrm{ReLU}(x) = x^{+} = \max(0, x) \qquad (3)$$

Z predpisu funkcie je teda jasné, že funkcia v prípade záporného vstupu vráti 0 a v prípade kladného vstupu ho zreplikuje. Existuje viacero modifikácii, či parametrizácii tejto funkcie, avšak v hlbokých sieťach je spravidla použitá práve základná verzia (Obrázok 3):

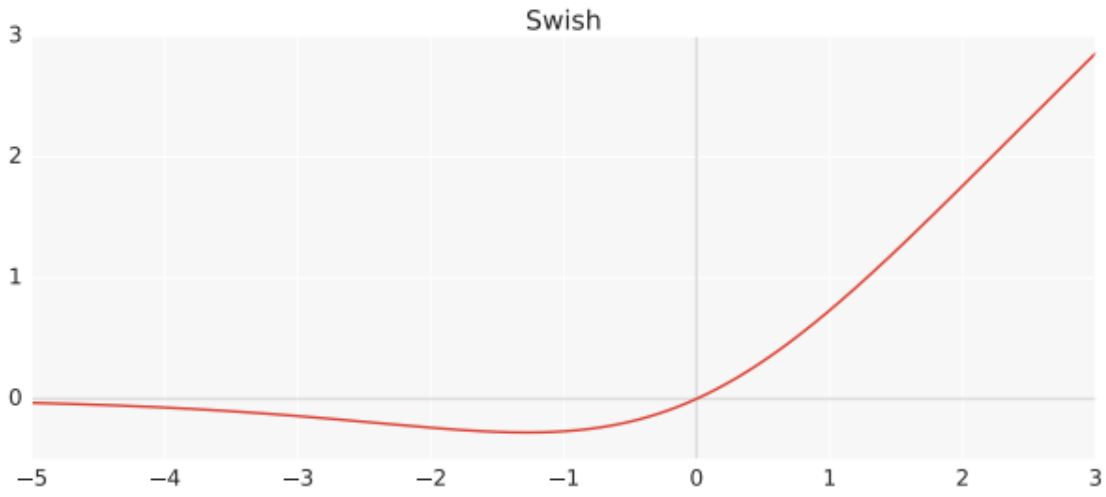


Obrázok 3: Funkcia y = ReLU(x)

1.2.3 Swish

Táto nelinearita je veľmi podobná všetkým nelinearitám, ktoré spomíname. Bola vyvinutá experimentálne spoločnosťou Google pri vývoji ich hlbokých neurónových sietí, kedy chceli spojiť vlastnosti viacerých nelinearít do jednej a tento experiment im náramne vyšiel, keďže presnosti sietí stúpli rádovo o celé percentá.

$$\mathrm{h}-\mathrm{swish}(x) = x\frac{ReLU\ 6(x+3)}{6} \tag{4}$$



Obrázok 4: Swish

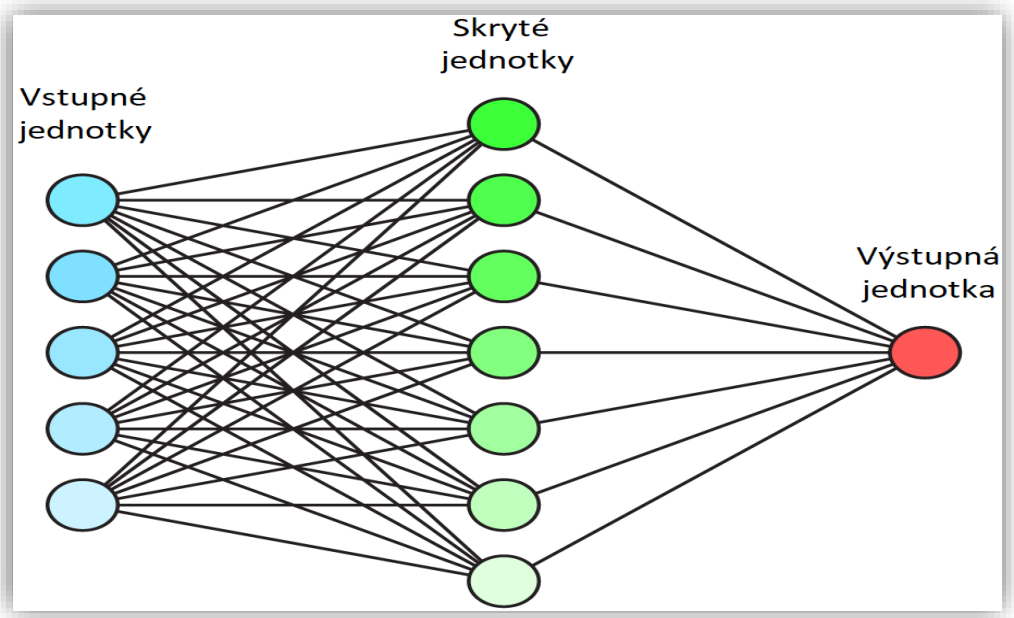
1.3 Viacvrstvové perceptrónové siete (MLP)

V predošlej kapitole sme si predstavili neurón – aby sme si vedeli predstaviť MLP sieť, predstavme si orientovaný graf, kde vrcholy sú neuróny a hrany medzi nimi sú synapsie. Neuróny sa skladajú do vrstiev, pričom spravidla každá vrstva obsahuje hneď niekoľko neurónov – prepojených s ďalšou vrstvou. Spôsob prepojení môže byť dopredný, alebo rekurentný. Pri dopredných MLP najčastejšie sa používa architektúra, kde je každý neurón aktuálnej vrstvy prepojený z každým neurónom predošlej vrstvy a taktiež nasledujúcej vrstvy – takáto architektúra v rámci MLP sa nazýva *plne-prepojená* (Obrázok 5). MLP sieť má tri základné vrstvy – *vstupnú, skrytú* a *výstupnú*.

Vstupná vrstva – počet neurónov zhodný s počtom vstupov. Spracovanie vstupov na tvar, ktorému rozumie NS, väčšinou normalizácia hodnôt podľa aktivačnej funkcie

Skrytá vrstva – minimálne jedna, ale môže ich byť aj viac. Operuje pomocou svojich neurónov ich vzťahmi pre výstup (1), napodobňuje ľudskú schopnosť hľadania súvislostí medzi vstupmi a výstupmi. Pri zložitejších úlohách bežne viac ako jedna vrstva, prípadne väčší počet neurónov

Výstupná vrstva – počet neurónov zhodný s počtom potrebných výstupov. Každý výstup je ohraničený aktivačnou funkciou a môže byť následne prerátaný na žiadanú mierku/jednotku.



Obrázok 5: Dopredná MLP sieť

1.4 Algoritmy trénovania neurónových sietí

Trénovacie algoritmy vieme rozdeliť na dve veľké skupiny. Učenie podľa predlohy (z učiteľom) a učenie bez učiteľa. Hlavný rozdiel je v potrebne dát - v prípade učenia s učiteľom musíme vopred disponovať vhodným súborom dát, podľa ktorého vieme natrénovať neurónovú sieť. Napríklad, pri klasifikačnej sieti potrebujeme dvojice klasifikovaný obrázok – očakávaný výstup. Pri učení bez učiteľa nepoznáme ideálny výsledok, avšak vieme vyjadriť kvalitu riešenia pomocou matematických vzťahov – či už použijeme učenie posilňovaním, alebo optimalizačnú metódu pri hľadaní parametrov siete, v oboch prípadoch sa opierame o kvalitu riešenia. Našim cieľom je však v oboch kateogóriách trénovacích algoritmov to isté – pomocou trénovacieho algoritmu nájsť také parametre siete, ktoré s dostatočnou kvalitou riešia úlohy, na ktoré sú nasadené.

1.4.1 Algoritmus spätného šírenia chyby

Veľmi efektívny algoritmus, používaný pri učení s učiteľom. V prvom rade potrebujeme rozdeliť dáta na *trénovacie, validačné* a *testovacie* dáta. Následne NS vypočíta výstup pomocou aktuálnych váh a biasov. Ten sa porovná s reálnym výstupom, ktorý je známy už pred trénovaním a vyčísli chybu. Túto chybu opätovne šíri po sieti späť pomocou rovnakých synapsii späť v sieti až pokým sa nerozšíri na všetky dostupné neuróny. Nasleduje samotný proces učenia – úprava váh a biasov. Ak by sme chceli vyjadriť matematicky výpočet nových váh, vyzeralo by to:

$$w_{ij}^{m(new)} = w_{ij}^{m(old)} + \eta \delta_j^m V_j^{m-1} \tag{5}$$

Vo vzťahu intuitívne vieme, že w prislúcha označeniu váh, pričom indexovanie i a j určuje váhu medzi konkrétnymi dvomi neurónmi (i je vstupný neurón, j výstupný). Horné indexovanie m značí poradie vrstvy. η je parameter rýchlosti učenia – klasicky s hodnotou v rádoch desatín, stotín, či tisícin. V prípade zlej hodnoty tohto parametra sieť nemusí zlepšovať svoju presnosť. Práve δ_j^m vyjadruje gradient chyby (vzťah 5) a V_j^{m-1} je očakávaný výstup j-teho neurónu (vzťah 6).

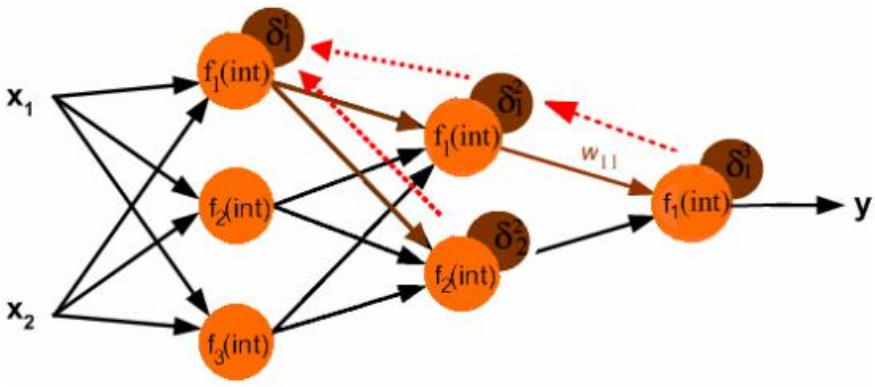
$$\delta_j^m = \mathrm{f}'(x_i^M)(\zeta_\mathrm{i} - y_i^M) \tag{6}$$

Vo vzťahu 5 vystupuje derivácia aktivačnej funkcie f (ReLU/tanh) v celkovom výstupe i-teho neurónu. ζ_i predstavuje vypočítaný výstup a y_i^M je reálny výstup – vyjadríme chybu pre poslednú vrstvu. Pre spätné šírenie chyby platí nasledujúci vzťah:

$$\delta_j^{m-1} = \mathrm{f}'(x_i^{M-1}) \sum_j w_{ji}^m + \delta_j^m \tag{7}$$

Vo vzťahu (8) možno vidieť veľkú analógiu so vzťahom (1), je to totiž len generalizovaný vzťah pre konkrétny neurón v rámci celej NS. φ je aktivačná funkcia, θ prah citlivosti neurónu a V_j^{m-1} výstup predošlých vrstiev neurónov. x_i^m sú aktuálne váhy.

$$V_i^m = f(x_i^m) = \varphi\left(\sum_{j=1}^{n} w_{ij}^m V_j^{m-1} + \theta\right) \tag{8}$$



Obrázok 6: Spätné šírenie chyby

Sieť postupne použije všetky trénovacie dáta (podľa veľkosti datasetu je niekedy potrebné aj rozdelenie trénovacích dát do menších skupín, anglicky *batch*). Vyhodnotenie všetkých trénovacích dát a vyjadrení chyby sa nazýva *epocha*. Obdobne sa vyhodnocujú aj validačné/testovacie dáta, pričom táto chyba nijako nevplýva na zmeny v parametroch siete, vyjadruje pre používateľa informáciu o priebehu učenia neurónovej siete. V prípade malej úspešnosti potrebujeme buď upraviť štruktúru NS, zmeniť optimalizačný algoritmus, prípadne zvýšiť počet epoch. Ak vieme, že trénovací dataset disponuje malým počtom dát, je možné tento dataset *augmentačnými* metódami rozšíriť, v niektorých prípadoch vieme aj svojpomocne dataset doplniť o nové dáta.

Trénovanie siete touto metódou vieme prirodzene ukončiť tromi spôsobmi:

1. Počet trénovacích epoch sa rovná maximálnemu preddefinovanému
2. Chyba sa dostala pod akceptovateľnú hranicu
3. Gradient klesol pod minimálnu hodnotu – učenie sa signifikantne spomalilo

Trénovanie vieme samozrejme ukončiť aj prerušením, či vieme definovať mnoho rozličných pokročilých podmienok okrem troch vyššie spomenutých. Po natrénovaní siete vykonávame validáciu takejto NS. V prípade, že NS bude dosahovať dostatočnej

úspešnosti na validačných dátach, mala by byť pripravená na implementáciu do ostrej prevádzky.

1.4.2 Učenie optimalizačnými metódami

V prípade, že nedisponujeme datasetom (väčšinou z dôvodu, že sa nedá vytvoriť), máme na výber viacero metód, ktoré tieto dáta nepotrebujú. Takéto typy úloh však musia mať jasne zadefinovaný problém, ktorého kvalita riešenia sa dá vypočítať nejakým matematickým vzťahom. V takýchto prípadoch vieme nasadiť na tento typ úlohy optimalizačné metódy a nájsť globálne optimá – či už minimum, alebo maximum.

My budeme ako optimalizačnú metódu používať genetické algoritmy. Ich princípu sa budeme venovať detailnejšie v 5.3. Cieľom však v takomto prípade bude, aby sme našli také váhy a biasy neurónov, ktoré zabezpečia čo najlepšie riešenie. Genetický algoritmus tak bude generovať riešenia a následne ohodnotí ich kvalitu. Podľa zadaných kritérii vyberie vždy jedince, ktoré budú vyhovovať a následne vykoná genetické operácie, ktoré zabezpečia, že sa presnosť NS bude zvyšovať z generácie na generáciu. Predpoklad takéhoto trénovania je robustnejšie riešenie, avšak taktiež je potrebný vyšší výpočtový výkon, pretože počet spustení experimentov bude omnoho vyšší ako pri klasických trénovacích algoritmoch.

Použitie genetických algoritmov na učenie neurónovej siete sa nazýva neuroevolúcia. Jej praktickej realizácii sa budeme venovať v ďalších kapitolách.

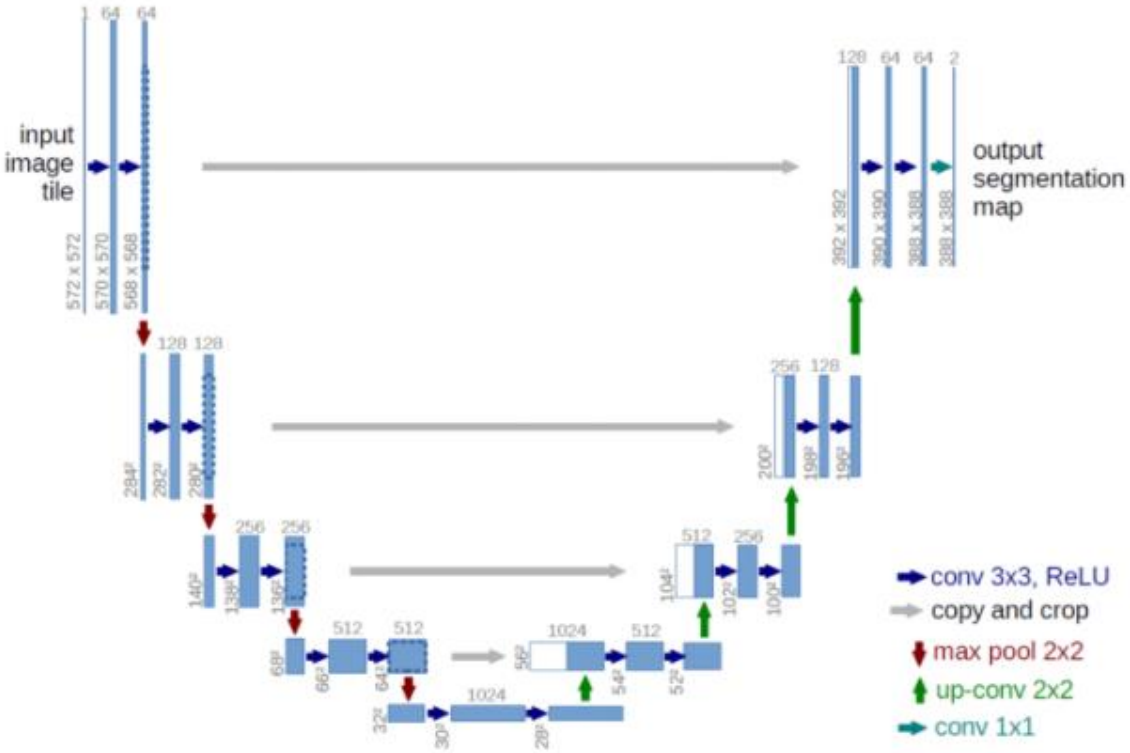
1.5 Segmentačné hlboké neurónové siete

Okrem klasickej MLP siete v práci použijeme aj segmentačné siete. Tieto siete sú všeobecne využívané na vystrihovanie určitých častí obrazu, ktoré chceme na obraze nájsť – siete hľadajú takzvanú masku. Motivácia na tvorbu takýchto sietí tkvie hlavne v medicíne, kde sú tieto siete aplikované najčastejšie na segmentovanie ochorení z rôznych röntgenových snímkov, alebo detekciu nečakaných stavov a uľahčujú lekárom život.

Implementácii segmentačných sietí v jazyku Python existuje skutočne ohromné množstvo, no my budeme trénovať v AI knižnici *PyTorch* a taktiež v jeho open-source platforme *FastAI*. Ako už názov platformy napovedá, umožňuje vytváranie rýchlych sietí (či už z pohľadu tvorby siete, alebo rýchlosti siete), čo je pre nás jedno z dvoch hlavných kritérii. Z mnohých možností sme vybrali dva konkrétne modely neurónových sietí, ktoré majú v podobných úlohách v praxi reálne uplatnenie – budeme hovoriť o ***Unet++*** a ***MobileV3Small***.

1.5.1 UNet

UNet je autoenkóder na segmentáciu obrazu, rozdelený na úrovne (stupne) kódovania, prepojenie medzi kódovacou a dekódovacou časťou je na úrovni stupňa kódovania. Ako jeden z mála modelov využíva transponovanú konvolúciu – *dekonvolúciu* pri dekódovaní:



Obrázok 7: Architektúra UNet

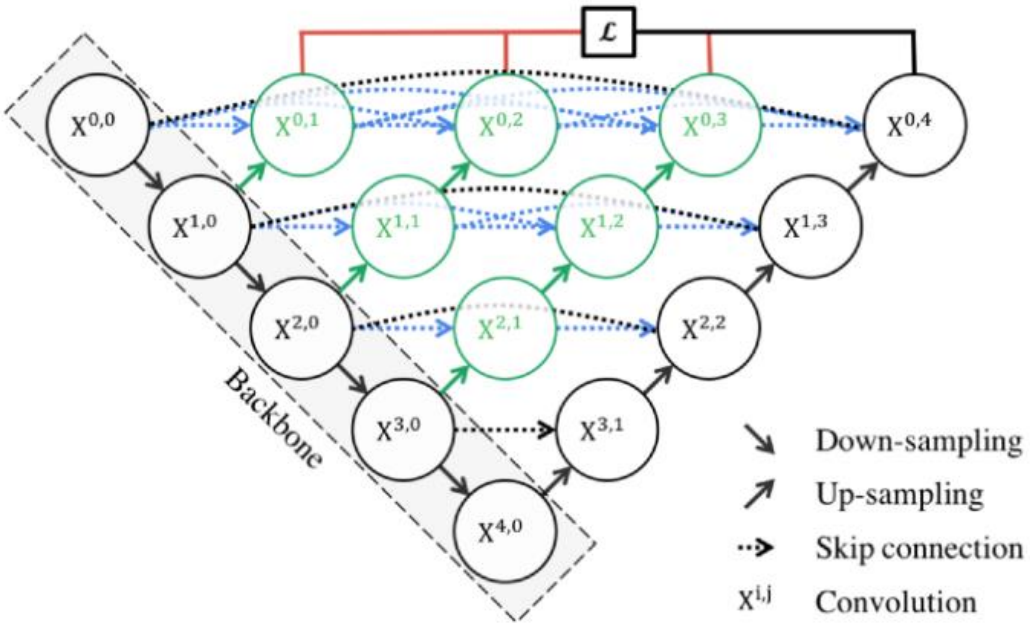
Ako naznačuje obrázok 7[1], vstupná informácia (obraz však v našom prípade bude inej veľkosti, takže veľkosti obrazov nekorešpondujú s reálnymi veľkosťami v našom prípade) najprv prechádza nižšie, pomocou zhlukovacích vrstiev - enkódovanie. Po dosiahnutí najnižšej vrstvy sa informácia šíri späť do vyšších vrstiev, kde posuny medzi jednotlivými vrstvami prebiehajú pomocou dekonvolúcie - dekódovanie. Natrénovanie siete od úplného základu trvá relatívne dlho – preto sa využívajú ako enkódery niektoré upravené modely predtrénovaných hlbokých sietí. Najlepšie výsledky dosahujú siete VGG, či ResNet.

1.5.2 UNet ++

Po určitom čase sa však zistilo, že pôvodná architektúra pre UNet, ktorá bola vyvinutá na segmentáciu pre medicínske účely môže byť vylepšená.

[1] *Understanding semantic segmentation with UNET*. 2019. Dostupné online: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Autori túto sieť nazvali UNet++, keďže majoritne vychádza z konceptu svojho predchodcu.



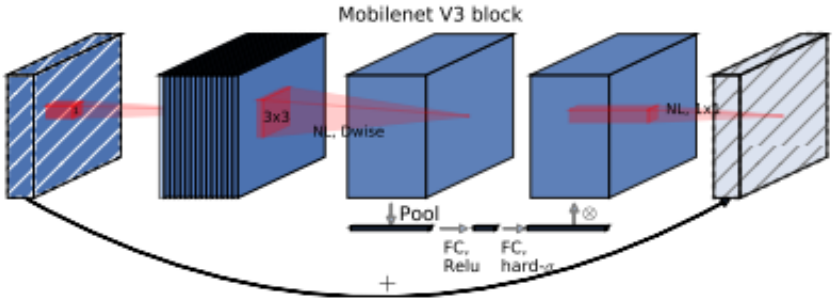
Obrázok 8: Architektúra UNet++

Oproti UNet vidíme teda na Obrázok 8[2] tri hlavné zmeny. Enkóder je zvýraznený v rámčeku a jeho vrstvy sú spojené opäť zhlukovacími vrstvami (down-sampling). Zelenou sú pridané takzvané premosťovacie cesty, ktoré obsahujú ďalšie konvolučné vrstvy – ich cieľom je redukcia nepresností, čo znamená jednoduchšie vypočítanie presnosti pre optimalizačné algoritmy, ktoré riadia učenie siete. Modrou sú naznačené prepoje medzi vrstvami nerovnakej úrovne, týmto vzniká plne prepojená sieť (vrstvy rovnakej úrovne boli prepojené už aj v rámci UNet). Týmto krokom vzniká mapa príznakov v každej dekódovacej vrstve, pričom v UNet je to len v poslednej. Inšpirácia k tomuto kroku pochádza z DenseNet a prináša zlepšenie presnosti segmentácie a šírenia gradientu medzi vrstvami. Červenou je označené šírenie výsledku segementácie z viacerých vrstiev. Toto dovoľuje natrénovanej sieti vybrať z viacerých módov – podľa počtu použitých výsledkov vrstiev. Najpresnejšie výsledky ponúka verzia *UNet++* L^4 (používa výsledky segmentácie zo všetkých dekódovacích vrstiev a celkový výsledok je ich priemer) a najrýchlejšie zase *UNet++* L^1 – tá používa len poslednú vrstvu, ktorá zvykne byť najpresnejšia.

[2] ZHOU Z., 2018. *UNet++ A Nested U-Net Architecture for Medical Image Segmentation*. arXiv: 1807.10165, 2018. Dostupné online: https://arxiv.org/pdf/1807.10165.pdf

1.5.3 MobileNetV3 Small

MobileV3Small je typologicky úplne inou sieťou, ako predošlá UNet. Bola vyvinutá najmä pre vnorené systémy, prípadne mobilné telefóny, kde sa dala použiť pri detekcii objektov v reálnom čase. Znamená to, že potrebuje veľmi malý výpočtový výkon a napriek tomu dosahuje presné výsledky vo veľkej rýchlosti. Prelomový prvok Mobile siete (už vo verzii V1) je **hĺbková konvolúcia**. Tá zmenší počet parametrov siete a urýchli tok dát. Ďalším významným prvkom je využitie SE (Squeeze and excitation) blokov, ktoré umožňujú dynamickú rekalibráciu medzi kanálmi obrazu skoro bez zvýšenia výpočtovej náročnosti.



Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Obrázok 9: Architektúra MobileNetV3Small

V architektúre sú originálne veľkosti obrázkov, ktoré nesedia presne s našim datasetom, avšak pre sieť nie je rozdielna veľkosť problémom. Stĺpec *SE* znamená, či je pri danej vrstve použitá technológia **squeeze and excitation**. *NL* je použitá nelinearita, v tomto prípade sú použité **ReLU** (RE) a **Swish** (HS). *S* indikuje veľkosť konvolučného kroku.

2 Genetický algoritmus

Genetický algoritmus (GA) je najznámejším z evolučných algoritmov, v súčasnosti zrejme ten úplne najvyužívanejší. GA je založený na báze Darwinovej evolučnej teórie a jeho cieľom je zreplikovať biologickú evolúciu a pomocou nej riešiť rôzne optimalizačné problémy. Pojem „Genetický algoritmus“ vzniká okolo 70. rokov dvadsiateho storočia, keď ho na Michigan University v USA definoval tím pod vedením Johna Hollanda. GA našiel už od svojho vzniku široké uplatnenie v mnohých sférach, keď jeho najväčším bonusom je schopnosť nájdenia globálnych extrémov rôznych funkcii. Veľmi podobný princíp, avšak s iným využitím – Genetické programovanie (GP) vzniká o desať rokov neskôr, tiež v USA. Jeho autorom je John Koza. GP bolo však na rozdiel od GA určené skôr na automatizovaný vývoj a optimalizáciu programov, prípadne strojové učenie.

2.1 Základné pojmy

Chromozóm (alebo genóm) je postupnosť istých parametrov, ktoré sú schopné vhodne zakódovať riešenie do určitého celku. Ako celok možno použiť napríklad vektory, matice, ale dokonca aj znakové reťazce, či ich kombinácie. Veľmi dôležité však je, že počas celého procesu GA je dĺžka chromozómu konštantná. Príkladom môže napr. byť:

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n) \in \beta\,;\ \beta \in Z \tag{9}$$

Predošlá rovnica naznačuje, že chromozóm α obsahuje len prvky z množiny β, ktorá je v tomto prípade definovaná len v oblasti celých čísel. Je však jasné, že množina β môže byť upravená presne na mieru riešenia konkrétneho problému.

Populácia je súbor všetkých chromozómov, ktoré majú byť súčasťou GA. Pri populácii platí podmienka, že jej veľkosť nesmie počas GA meniť. Spôsob výberu jednotlivých chromozómov v populácii však môže byť rôzny a obsah populácie v každom cykle GA sa obvykle mení podľa kritérii prispôsobených úlohe.

Generácia predstavuje jeden výpočtový cyklus GA. Počas každej generácie sa vyhodnocuje napríklad fitness funkcia, obvykle si program pamätá pre štatistické vyhodnotenie aj minimálnu hodnotu fitness funkcie v každom cykle, či napríklad aj poradové číslo cyklu, taktiež sú obvykle aplikované metódy GA na zmenu jednotlivých chromozómov a taktiež sú možné ďalšie dodatočné úpravy reťazcov.

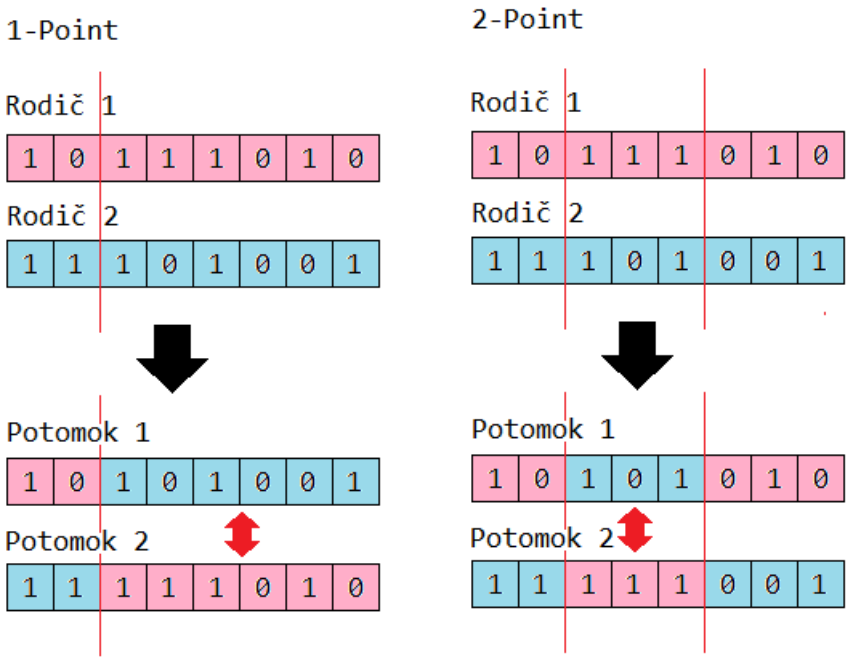
Fitness funkcia je najdôležitejšou časťou genetického algoritmu. Je to funkcia navrhnutá predom tvorcom GA. Mala by byť zostrojená tak, aby vedela na základe kritérii potrebných na optimalizáciu vyhodnotiť každé riešenie a nájsť najvhodnejšie z nich. Dobre navrhnutá fitness funkcia môže eliminovať aj nesprávne navrhnuté parametre genetických operácii.

2.2 Genetické operácie

V terminológii genetických algoritmov a všeobecne genetického programovania poznáme tri základné operácie. Sú nimi kríženie, mutácia a výber. Už podľa názvu operácii vidieť, aké funkcionality tieto operácie budú spĺňať, pričom všetky tieto operácie majú spoločnú vlastnosť – sú inšpirované evolúciou a existujú aj v prírode.

2.2.1 Kríženie

Kríženie je možné prirovnať k prirodzenému rozmnožovaniu v prírode. Zoberú sa dva náhodné chromozómy v populácii a na základe nastavenia kríženia si vymenia ich gény. Pre klasické metódy existujú napríklad jednobodové, dvojbodové a štvorbodové kríženie. Znamená to, že sa pôvodný reťazec roztrhá na určitý počet častí (pre jednobodové sú to 2 časti) a jedna z týchto častí sa vymení s inou časťou iného chromozómu. Dobre to znázorňuje aj nasledujúci obrázok:



Obrázok 10: Jednoduché kríženie

Genetické operácie ponúkajú aj vylepšené verzie kríženia, ktoré príroda nepozná – ide napríklad o kríženie viacerých rodičov ako dvoch, či napríklad výber konkrétnych génov, ktoré majú byť vymenené. Zaujímavým príkladom kríženia je medziľahlé kríženie – toto kríženie totiž kombinuje kríženie aj s mutáciou, keďže je vhodné na nájdenie riešení, ktoré sa nachádzajú medzi rodičovskými reťazcami:

$$P = R_1 + r.\alpha.(R_2 - R_1)\ ;\ 0\ \leq \alpha\ \geq 2 \tag{10}$$

P je novovzniknutý reťazec, R_1, R_2 sú rodičovské reťazce a ak je α < 1 tak potomkovia budú medzi rodičovskými reťazcami. Ak nie, tak budú v okolí. Túto špecifickú formu kríženia tu spomíname najmä kvôli tomu, že bude využívaná neskôr v práci.

2.2.2 Mutácia

Z biologického, ale aj z programátorského hľadiska ide o najdôležitejšiu operáciu. Keď v prírode zaručuje vývoj nových vlastností jedincov a taktiež vývoj nových častí prispôsobených prostrediu, v GA zaručuje prísun nových, unikátnych jedincov často kombináciou práve aktuálne nevyužívaných možností. Pri genetickom algoritme máme na výber opätovne z niekoľkých verzií mutácie. Tou základnou (rovnica 9) je vybratie náhodného génu v rámci reťazca a jeho nahradenie iným, vygenerovaným génom. Vygenerovaný gén pritom musí byť z rozsahu platných riešení definovaných vopred. Výber týchto génov záleží od nami zvolenej pravdepodobnosti vybrania génu. Štandardne sa využívajú hodnoty pravdepodobnosti od 0,1% do 10%, pričom pri základných mutáciách sú to spravidla práva vyššie hodnoty. Genetický algoritmus vieme týmto spraviť agresívnejším a má tendenciu rýchlejšie prehľadať hľadaný priestor a nájsť okolie optimálnych hodnôt.

$$\alpha_{new} = (\alpha_{min}; \alpha_{max}) \tag{11}$$

Pre subpopuláciu najlepších riešení sa zvykne používať aditívna mutácia. Táto mutácia upravuje náhodné gény (opäť sú vybrané na základe pravdepodobnosti). Cieľom tejto mutácii je v menšom okolí optimálneho riešenia nájsť práve to optimálne:

$$\alpha_{new} = \alpha_{old} + (\beta_{min}\ ; \beta_{max}) \tag{12}$$

Podľa posledného vzťahu vidíme, že pri tejto mutácii vopred vyberáme minimálnu/maximálnu veľkosť prírastku, označenú parametrami β_{min}/β_{max}. Čím menšia táto hodnota bude, tým presnejšie riešenie vieme pomocou aditívnej mutácie nájsť. V genetickom programovaní existujú aj iné formy mutácie, napríklad multiplikatívna. V práci však viac foriem mutácie využitých nie je.

2.2.3 Výber

Výber je v biologickom ponímaní prirodzená selekcia v rámci prírody – prežije iba najsilnejší. V GA je prístup v podstate analogický a cieľom je vybrať najlepších jedincov, no opätovne nám výber ponúka aj viacero možností – nemusíme garantovane vybrať iba najlepších jedincov na základe kvality riešenia, ale môžeme vybrať aj náhodných jedincov, alebo doslova simulovať prírodné výbery. Vieme ich začleniť do nasledovných kategórii:

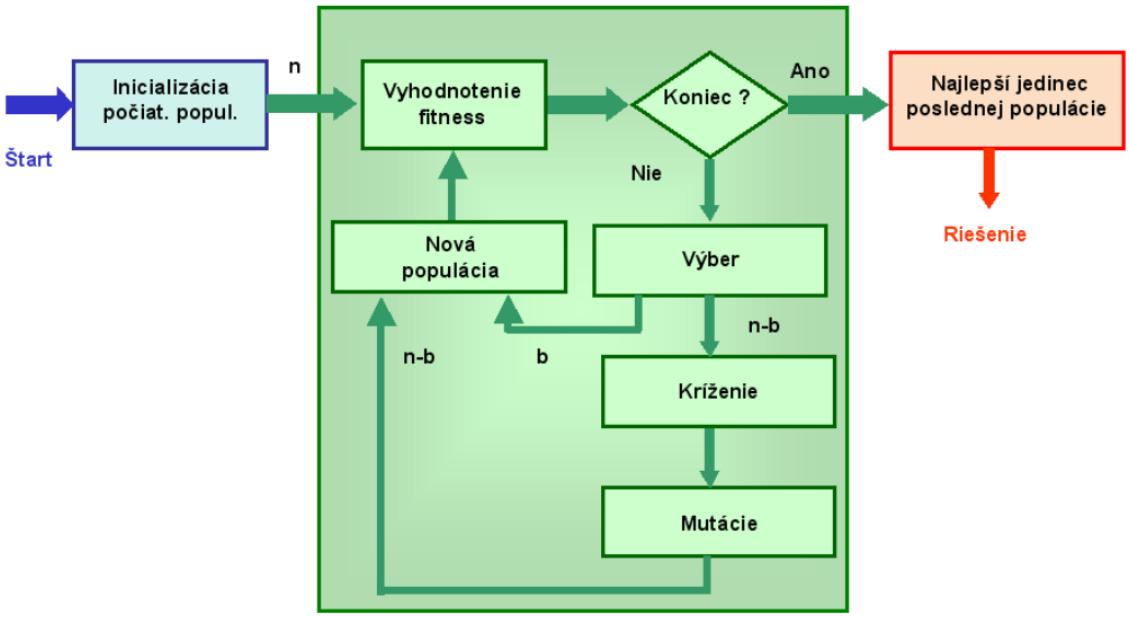
1) *Výber na základe úspešnosti* (*selbest, selsort*) – na základe fitness funkcie sa vyberie n najúspešnejších reťazcov (prípadne aj m ich kópii)
2) *Náhodný výber (selrand)* – vyberie sa n náhodných reťazcov
3) *Turnajový výber (seltourn)* – Z populácie sa vyberú dva náhodné jedince a lepší z nich sa zapíše do novej populácie. Obidva jedince sa vrátia naspäť do starej populácie a výber sa opakuje až kým nie je vybraný potrebný počet reťazcov.
4) *Výber pomocou váhovaného ruletového kolesa (selsus)* – výber, ktorého ekvivalentom je otáčanie váhovaným ruletovým kolesom. To je rozdelené na kruhové výseky, z ktorých každý je priradený jednému reťazcu a jeho veľkosť je nepriamo úmerná hodnote jeho účelovej funkcie (resp. priamo úmerná jeho úspešnosti). Úspešnejšie reťazce majú väčšiu šancu byť vybrané.
5) *Výber na základe diverzity (seldiv)* – výber chromozómov s čo najviac rozličnými vlastnosťami v porovnaní s ostatnými členmi populácie

2.3 Realizácia genetických algoritmov

Ako základ pre realizáciu genetických algoritmov použijeme *Genetic Toolbox*[3], ktorý je naprogramovaný v jazyku MATLAB. Ako sme spomínali v úvode, celá praktická realizácia diplomovej práce je v jazyku Python a preto po vzore spomínaného toolboxu preprogramujeme základné funkcie do tohto jazyka. Okrem natívnych súčastí Pythonu použijeme aj knižnicu *NumPy* – táto knižnica poskytuje veľmi podobné funkcie pre operácie s maticami ako MATLAB. Na stránke nájdeme aj návod[4] na používanie knižnice pre MATLAB používateľov.

[3] Zdroj [5]
[4] Numpy for MATLAB users: https://numpy.org/doc/stable/user/numpy-for-matlab-users.html

2.4 Priebeh genetického algoritmu



Obrázok 11: Bloková schéma GA

Pri popisovaní priebehu GA si pomôžeme obrázkom 11. Na začiatku inicializujeme populáciu – vytvoríme n reťazcov, ktoré budú ako prvé hodnotené a budú slúžiť ako základ pre ďalšie generácie GA. Následne začína samotný proces genetického algoritmu. Ten beží určitý počet generácii a tak na úvod skontroluje, či riešenie nespĺňa kritériá, ktoré očakávame (prípadne nenastal maximálny počet generácii). Ak áno, genetický algoritmus končí a spravidla sa ukladá priebeh optimalizácie a najlepší jedinec. Ak nie, v rámci GA sa uskutočnia výbery do subpopulácii – uloží sa najlepší jedinec, s ktorým sa nebude manipulovať a taktiež sa vytvoria „pracovné“ populácie – subpopulácia najlepších, ktorá prejde len ľahkými úpravami s cieľom nájsť ešte lepšie riešenie v blízkosti aktuálnych najlepších, subpopulácia pracantov, ktorí budú mutovaní a krížení s cieľom nájsť nových najlepších jedincov. Tieto subpopulácie sa na záver spoja s najlepším jedincom a vytvoria opätovne populáciu o veľkosti rovnakej, aká bola na začiatku. Nová populácia je ohodnotená fitness funkciou. Po tomto kroku sme opäť na začiatku GA a tieto kroky budeme opakovať až pokým nebudú naplnené podmienky ukončenia optimalizácie.

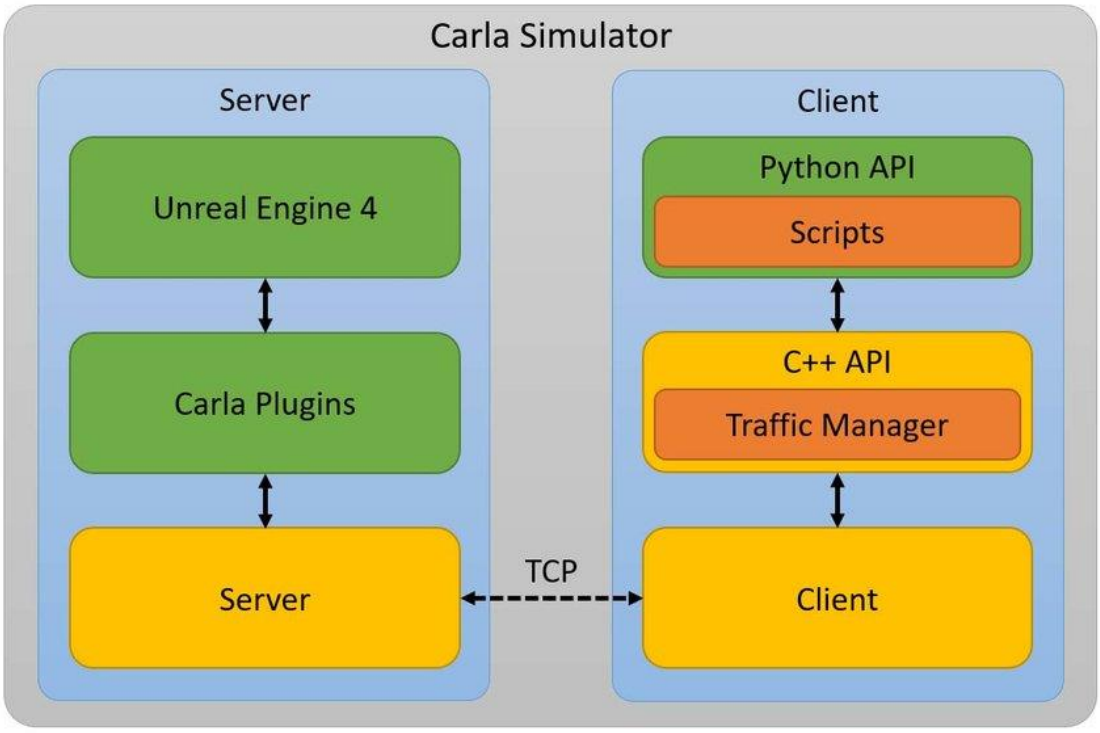
V prípade, že nájdený výsledok nám nevyhovuje, musíme analyzovať dôvody zlyhania a zmeniť niektoré parametre algoritmu tak, aby bol čo najpresnejší pre naše potreby.

3 Simulačné prostredie

Na realizovanie algoritmov riadenia autonómneho vozidla sme si museli však najskôr zvoliť vhodné simulačné prostredie. Aby sme sa čo najviac priblížili realite, potrebovali sme voľne dostupné prostredie, ktoré spĺňa nasledujúce podmienky:

- pripravené rozhranie medzi simulátorom a užívateľom (API),
- integrované snímače a kamery
- reálna fyzika vozidiel, riadenie vozidiel pomocou princípov podobných realite,
- interakcia s okolitým prostredím.

Všetky vyššie zmienené podmienky spĺňa simulačné prostredie **CARLA**. Jej základ tvorí Unreal Engine 4, ktorý zabezpečuje fyzikálne javy v simulácii podobné realite, pričom samotný simulátor je naprogramovaný v jazyku C++ a ponúka užívateľské rozhranie v jazykoch Python a C++. Keďže je CARLA open-source projekt, je dostupný aj celý jej zdrojový kód, pričom má užívateľ na výber, či si chce vyrobiť vlastnú verziu, alebo použije niektorú z stabilných verzií, ktoré sú dodávané ako aplikácie. CARLA je stále v aktívnom vývoji a nami použitá stabilná verzia pre účely diplomovej práce bola **0.9.12** (august 2021). Architektúra simulátora je na koncepte server – klient, pričom samotný simulátor pôsobí ako server a jednotlivé užívateľské skripty sa na neho pripájajú. Prepojenie medzi klientom a serverom je prostredníctvom protokolu *TCP*. Vďaka tomu sa vieme simuláciu rozdeliť na viacero počítačov – server môže bežať na separátnom počítači, pričom klienti sa budú na neho pripájať. Samozrejme, klient a server môžu bežať simultánne aj na jednom zariadení. Klient má po pripojení veľkú kontrolu nad serverom a vie meniť jeho správanie. Pri použití viacerých klientov súčasne je treba mať na pamäti ich prípadnú synchronizáciu, avšak v našej aplikácii používame pre jednoduchosť len jedno súčasné spojenie medzi klientom a serverom. Detailnejší koncept simulátora vidíme na Obrázok 12. Z neho taktiež vyplýva, že na strane klienta beží aj „Traffic manager“ – ten má za úlohu priradzovať úlohy a riadiť autá riadené autopilotom – teda tie, ktoré nebude riadiť náš program. Zabezpečuje, aby sa tieto autá správali podľa pravidiel cestnej premávky (rešpektovanie semaforov, značiek).



Obrázok 12: Koncept simulátora CARLA

3.1 Nastavenia simulácie

Na úvod si musíme uvedomiť, aké experimenty chceme vykonávať na simulátore. Pre naše účely potrebujeme, aby boli naše experimenty (ideálne) stále opakovateľné. To znamená, že pri každom spustení rovnakého scenára, by malo byť správanie okolitého sveta rovnaké. CARLA podporuje beh simulácie vo *fixnom* a *premenlivom* kroku simulácie. Dôležitým nastavením je aj, či chceme aby simulácia bežala *synchrónne* alebo *asynchrónne*.

	FIXNÝ KROK	**PREMENLIVÝ KROK**
SYNCHRÓNNY MÓD	Klient má plnú kontrolu nad simuláciou a opakovateľnosť je vysoká	Vysoké riziko nerealistických simulácii
ASYNCHRÓNNY MÓD	Server beží čo najrýchlejšie, čo môže spôsobiť vynechanie niektorých dát v prípade zložitého spracovania simulačných dát	Zložitá opakovateľnosť simulácii

Tabuľka 1: Možné konfigurácie simulácie

Podľa vyššie uvedenej tabuľky je zrejmé, že správne nastavenie simulácie pre naše účely bude kombinácia **fixného kroku simulácie (0,05s)** a **synchrónneho módu**. Farby v tabuľke vyznačujú vhodnosť jednotlivých nastavení. Zelená znamená vhodné riešenie, červená nevhodné a bielou je vyznačené riskantné nastavenie.

3.1.1 Mapy

CARLA ponúka v súčasnosti osem máp. Pri nastavovaní mapy je potrebné mať na pamäti, že všetky nasledujúce nastavenia (v ďalších kapitolách) sa resetujú. Preto, ak nechceme použiť základnú mapu, je potrebné ju zmeniť ako prvú. Základná mapa je v HD rozlíšení a môže znamenať dodatočnú záťaž pre grafickú kartu, preto sa odporúča pri experimentoch s väčším množstvom spustení používať mapy v nižšom rozlíšení. V rámci ôsmych základných máp sa nachádza viacero mestských častí. V nich vieme nájsť aj semafory, či križovatky, ktoré môžu slúžiť na pokročilé testovanie rôznych experimentov. Pri našich experimentoch budeme používať najmä mapu „*Town04*“ – táto mapa obsahuje menšie mesto a diaľnicu.

Obrázok 13: Ukážka mapy Town04

3.1.2 Počasie

Nastavenie počasia bude mať veľký vplyv na kvalitu spracovania jednotlivých snímačov a kamier, samozrejme taktiež má vplyv na fyziku auta v prípade daždivého počasia je predĺžená brzdná dráha vozidla a auto je všeobecne zložitejšie riaditeľné.

CARLA opäť poskytuje viacero možných prednastavených počasí, taktiež si vie ale užívateľ navoliť úplne vlastné počasie vyskladaním parametrov ako oblačnosť, dážď, vietor (smer a intenzita), uhol slnečného svetla, hmla či vlhkosť vozovky. CARLA podporuje aj dynamické nastavenie počasia, čo znamená, že sa počasie mení v čase a vyberá sa náhodne. Pre naše potreby však bude stačiť použiť počasie typu „**ClearNoon**“. Toto počasie nastaví nulový uhol slnečného svetla a taktiež vypne vietor, či oblačnosť (tým aj daždivosť).

3.2 Súčasti simulácie (actors)

Z hľadiska architektúry simulátora je všetko, čo hrá nejakú rolu v simulácii jej súčasťou (ďalej len ako „actor“). Podľa dokumentácie to môžu byť:

- vozidlá,
- chodci,
- senzory a kamery,
- pozorovateľ,
- dopravné značky a semafory.

CARLA má všetkých účastníkov simulácie v spoločnej knižnici (blueprints) a ak ich chce užívateľ použiť, stačí ich jedným jednoduchým príkazom pridať do sveta (vygenerovať ich inštanciu). Každý účastník vykonáva špecifickú úlohu v simulácii. Po skončení simulačnej epochy sa odporúča všetkých nepotrebných účastníkov zo sveta opätovne zmazať, tak aby nevznikli neočakávané problémy.

3.3 Vozidlá

Najdôležitejšou časťou simulácie sú samozrejme vozidlá. Užívateľ si vie vybrať presný typ vozidla, ktorý preferuje. Vozidlá okrem osobitého dizajnu majú špecifické dynamické vlastnosti, ktoré treba mať na pamäti. Na výber sú vozidlá od bicykla až po dodávky. Ak na vozidlo chceme upevniť snímače, bolo by dobré, aby bol stále používaný len jeden typ vozidla. V našej simulácii využívame model vozidla *Tesla model 3*.

Obrázok 14: Tesla model 3 v CARLA prostredí

3.3.1 Ovládanie vozidiel

Podľa nasledujúcej Tabuľka 2 vieme nastavovať riadiacu štruktúru *VehicleControl:*

Názov	Následok	Interval hodnôt	Hodnota
Throttle (plyn)	Zrýchlenie vozidla	<0,0 ; 1,0>	0,0
Steer (natáčanie)	Zmena smeru jazdy	<-1,0 ; 1,0>	0,0
Brake (brzda)	Spomalenie vozidla	<0,0 ; 1,0>	0,0
Hand-brake (ručná brzda)	Zaistenie vozidla	True/False	False
Reverse (režim cúvania)	Cúvanie	True/False	False
Manual gear shift (manuálne radenie)	Radenie rýchlostí	True/False	False
Gear (prevod)	Aktuálna rýchlosť	<1 ; 7>	-

Tabuľka 2: Riadiaca štruktúra vozidla

Síce nám riadiaca štruktúra ponúka sedem možností, my budeme v našom programe využívať len prvé tri – pridávanie plynu, zatáčanie vozidla a brzdenie. Ručná brzda bude vypnutá a taktiež nebudeme uvažovať nad cúvaním a využijeme automatické preradzovanie rýchlostí vozidla, preto vypneme manuálnu prevodovku.

3.4 Senzory a kamery

Veľmi dôležitou súčasťou projektu sú senzory a kamery. Venujeme im spoločnú kapitolu, pretože v CARLE toho majú veľa spoločného. Sú špeciálnymi typmi actorov, ktoré pre ich optimálnu funkcionalitu okrem pridania do sveta aj definovať parametre:

a) Vozidlo (alebo iný actor) na ktoré budú pripevnené
b) Pozícia senzora/kamery vzhľadom na actora
c) Definícia osobitých parametrov senzorov/kamier
d) Funkcia, ktorá spracuje dáta ktoré senzor/kamera namerali

V našom prípade budeme mať samozrejme všetky snímače a kamery upevnené na vozidle, ktoré budeme chcieť riadiť. Keďže využívame v simulácii synchrónny mód, pre jednoduchšie spracovanie dát sa odporúča namerané dáta vkladať za sebou do radu (FIFO[5]). Tým zabezpečíme, že sa nám nestratia žiadne dáta a v prípade, že niektorý snímač nenameral žiadne dáta (napríklad grafické dáta z kamier nechodia zo simulátora každý

[5] Rad, ktorý využíva princíp prvý dnu, prvý von. Všetky dáta sú postupne spracovávané

jeden krok), ušetríme výpočtový výkon - keďže ho nebudeme zbytočne spracovávať. Ak by sme takýto prístup nezvolili, funkcia na spracovanie dát by sa musela vykonať vždy. Senzory vždy v rámci dát, ktoré nasnímajú vracajú tri údaje, ktoré môžu byť použité na synchronizáciu. Je to poradové číslo kroku servera (tick), čas v sekundách od spustenia servera(timestamp) a taktiež pozíciu senzora v danom čase (transform).

3.4.1 Senzory

Dokumentácia CARLA rozdeľuje senzory na nasledujúce skupiny:

1) **Detektory**
   a) *Detektor prekročenia čiary*
   b) *Detektor najbližšej prekážky*
   c) *Detektor kolízie*
2) **Iné**
   a) *LIDAR*
   b) *RADAR*
   c) *GNSS*
   d) *IMU – akcelerometer, gyroskop a kompas*

V nasledujúcej tabuľke spomíname všetky senzory a aj ich umiestnenia na vozidle:

Názov senzoru (CARLA)	**Pozícia vzhľadom k vozidlu (x, y, z) [m]**
Detektor prekročenia čiary (LaneInvasion)	(0, 0, 0)
Detektor kolízie (CollisionDetector)	(1.5, 0, 0.7)
RADAR (Radar)	(1.5, 0, 0.5)
IMU (IMU)	(0, 0, 0)

Tabuľka 3: Integrované senzory v práci

3.4.1.1 Detektor prekročenia čiary

V projekte sa budeme snažiť, aby auto jazdilo autonómne, pričom samozrejme cieľom je, aby bolo jeho správanie čo najbližšie k ľudskému šoférovi. To znamená, že chceme, aby sa auto držalo v pruhu a nevybočovalo z neho, pokiaľ to nie je nutné. Aby sme vedeli takýto stav detegovať, použijeme tento pred-definovaný senzor. Senzor sa nachádza v ťažisku vozidla a v prípade, že ťažisko vozidla prejde čiarou, senzor si uloží dáta o akú čiaru išlo a kedy sa tak stalo.

3.4.1.2 Detektor kolízie

Azda najdôležitejší senzor – jeho aktivácia totiž znamená kolíziu a tým pádom sme sa dostali do patovej situácie (keďže nepoužívame cúvanie). Senzor je umiestnený vpredu vozidla, tak aby čo najrýchlejšie detegoval kolíziu.

3.4.1.3 RADAR

Princíp radarového senzoru je postavený na vystrelení veľkého množstva lúčov v rôznych azimutoch a výškach letu tak, aby dostatočne dôveryhodne pokryli žiadaný priestor. V tomto senzore je potrebné nastaviť štyri parametre:

- **Horizontálny rozhľad**: určuje maximálny/minimálny azimut, pod ktorým sú lúče vystreľované. V projekte je tento rozhľad nastavený na **90**, to znamená, že lúče sú vystreľované v horizonte < -45°, 45° >.
- **Vertikálny rozhľad**: určuje maximálnu/minimálnu výšku letu lúčov. Princíp je úplne rovnaký ako pri horizontálnom rozhľade. V projekte je nastavený na **25**, a tak možné letové výšky sú < -12.5°, 12.5° >.
- **Maximálny dostrel radarového lúča**: Maximálna vzdialenosť, ktorú lúče dokážu namerať (v metroch). V projekte využívame hodnotu **50**.
- **Počet bodov v jednom meraní**: Koľko bodov je schopný radar namerať pri jednom meraní. V projekte je hodnota nastavená na **1500**.

Treba poznamenať, že merania radaru prechádzajú neskôr dôslednou filtráciou tak, aby sme predišli falošným, či nepresným meraniam. Viac v neskorších kapitolách.

Obrázok 15: Vizualizácia radarových meraní

3.4.1.4 Inerciálna meracia jednotka

Tento snímač nie je v projekte integrovaný priamo, sú len využívané jeho jednotlivé súčasti, keďže samozrejme potrebujeme vedieť aktuálnu polohu vozidla a taktiež rýchlosti/zrýchlenia, ktorými vozidlo disponuje. IMU využíva samotná CARLA pri vstavaných metódach vozidiel.

3.4.2 Kamery

CARLA ponúka nasledujúce kamery, pričom obdobne, ako pri senzoroch si použité kamery v projekte prejdeme detailnejšie v nasledujúcich podkapitolách:

a) Hĺbková
b) RGB
c) Podľa optického toku
d) Segmentačná

Názov kamery (CARLA)	Pozícia vzhľadom k vozidlu (x, y, z) [m], klonenie [°]	Rozlíšenie (šírka x výška)
Predná kamera (RGB)	(2.5, 0, 1.3), -5	1024 x 512
Segmentačná (Semantic segmentation)	(2,5, 0, 0.7), 0	1024 x 512
Pozorovacia kamera (RGB)	(-2, 0, 2), -8	1024 x 512

Tabuľka 4: Nastavenie kamier

Ako vyplýva z tabuľky 4, na vozidle sa nachádzajú dve RGB kamery a jedna segmentačná. Všetky kamery v projekte využívajú na vykresľovanie ich obrazu separátne vlákno oproti samotnému hlavnému programu, pričom pri každej kamere si vie užívateľ zvoliť či chce, aby bola vykresľovaná, alebo nie. Pri kamerách taktiež vieme nastaviť, ako často chceme dostávať obraz – v našom prípade volíme hodnotu 0, ktorá znamená, že obraz z kamery je generovaný tak rýchlo, ako sa to len dá.

3.4.2.1 RGB kamera

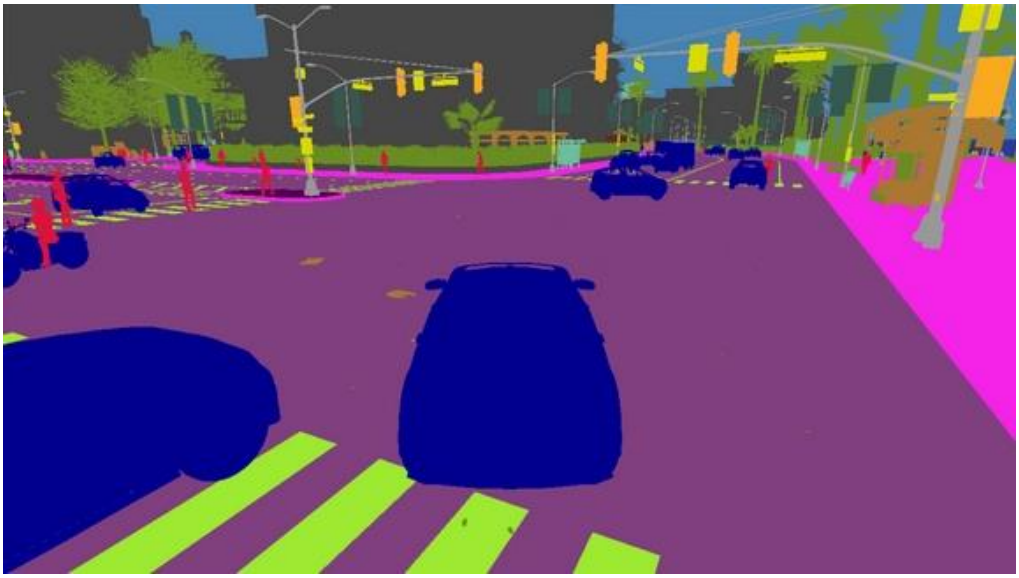
Už sme spomínali, že v projekte sú dve RGB kamery. Jedna z nich je nazvaná ako pozorovacia, keďže pomocou tohto pohľadu budeme môcť pozorovať konkrétne vozidlo pri pohľade zozadu vozidla. Táto kamera využíva špeciálny typ úchopu na vozidlo – je na ramene. Toto rameno zabezpečuje zmenšenie vibrácii a otrasov a zaručuje obraz konštantnej kvality. RGB kamera inak neponúka nejaké špecifické nastavenia, okrem pár

obrazových filtrov, ktorými pridáva do kamerového obrazu nejaké špeciálne prvky – v našom prípade však obe RGB kamery pracujú v ich základných režimoch.

Obrázok 16: Obraz z prednej RGB kamery

3.4.2.2 Segmentačná kamera

Segmentačná kamera je veľmi špecifickou kamerou, keďže v reálnom svete je potrebné takýto obraz najprv získať rozsegmentovaním klasického obrazu pomocou metód spracovania obrazu, prípadne hlbokých neurónových sietí.



Obrázok 17: Obraz zo segmentačnej kamery

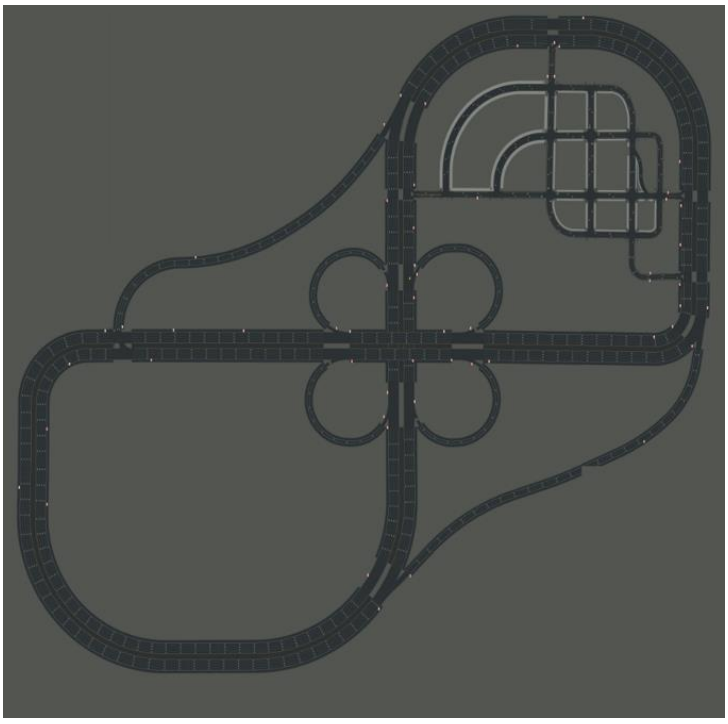
Pomocou takto segmentovaného obrazu (vyššie) vieme jednoducho rozlišovať jednotlivé elementy v rámci simulácie – čiary, semafory, vozidlá, chodcov, či aj samotnú cestu.

3.5 Navigácia

Navigácia vozidla bude pre nás veľmi dôležitou – ak chceme aby sa autonómne pohybovalo, potrebuje mať navigačné údaje na to, kde sa vlastne nachádza cieľ, kam chceme aby sa dostalo. Navigáciu vieme v CARLE rozdeliť na dva veľké svety – globálnu a lokálnu. CARLA ponúka množstvo možností, ako tieto navigačné systémy využiť, pričom my sme si vybrali agentový systém. V nasledujúcich kapitolách vysvetlíme princíp, na ktorom funguje.

3.5.1 Globálna navigácia

Globálna navigácia vychádza zo znalosti mapy. Všetky mapy v CARLE musia byť postavené vo formáte *OpenDRIVE*, konkrétne by mali obsahovať všetko podľa štandardu 1.4[6]. Užívateľ si navrhne, že chce ísť z bodu A do bodu B. Tieto body vloží do navigácie, tá pozrie na mapu a analogicky ako navigácia v reálnom svete vyberie cestu. Rozdielom oproti reálnej navigácii je, že globálna navigácia rieši aj značenie a taktiež teda zabezpečuje, že vozidlo sa pred križovatkou dostane do správneho pruhu. To v praxi znamená, že cestu rozseká na toľko navigačných bodov, koľkokrát bude musieť auto nečakane meniť smer (mimo držania sa v pruhu). V prípade, že cesta vedie cez križovatky, globálna navigácia vytvorí pred križovatkou bod a zároveň vytvorí bod za križovatkou. Týmto štýlom tak povie lokálnej navigácii, kam treba na križovatke odbočiť.



Obrázok 18: Mapa Town04 vo formáte OpenDrive

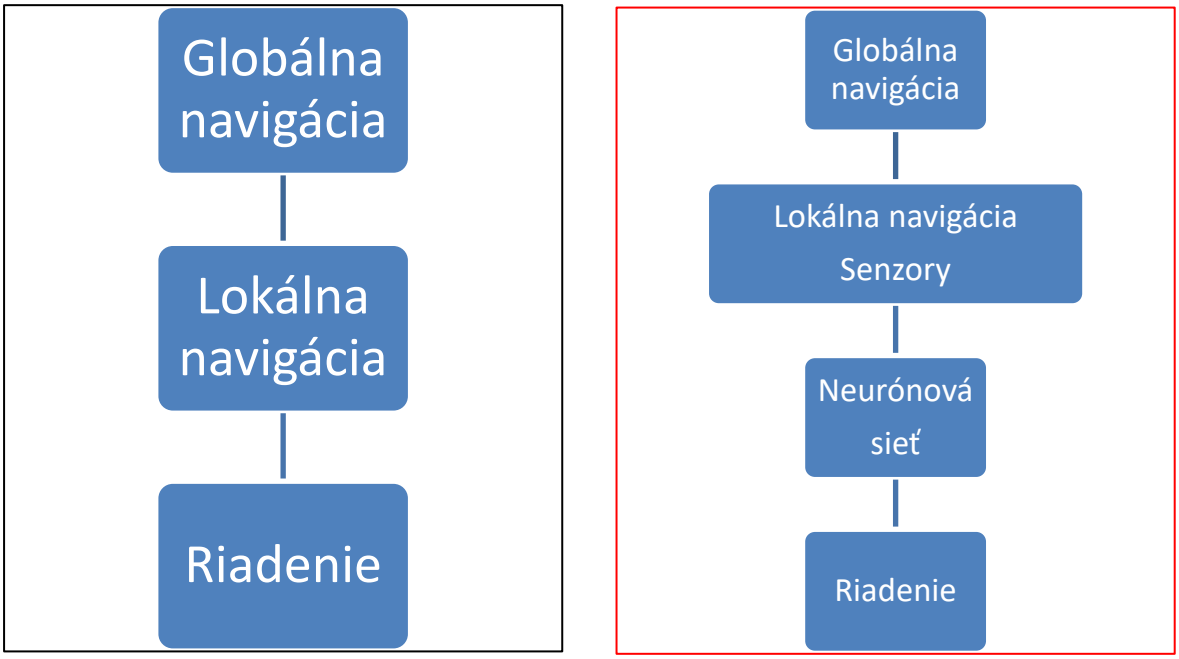
[6] OpenDrive, štandard 1.4 - https://www.asam.net/standards/detail/opendrive/

3.5.2 Lokálna navigácia

Lokálna navigácia má za úlohu detailne zabezpečovať navigovanie vozidla medzi jednotlivými bodmi, ktoré pôvodne určila globálna navigácia. V praxi to znamená, že lokálna navigácia generuje pri každom dopyte bod vzdialený x metrov od vozidla (základne 2 metre, avšak je možné túto hodnotu zmeniť), v smere najbližšieho navigačného bodu od globálnej navigácie. Týmto je zabezpečené, že vozidlo je dôkladne navigované a vždy má navigačný bod.

3.5.3 Riadenie

A na záver je potrebné, aby tieto navigačné údaje boli samozrejme aj nejako realizované a pretavené do reálnej kontroly vozidla – v tomto prípade je najjednoduchšie získavať riadenie pomocou PID regulátora. Poznáme pozíciu vozidla, taktiež poznáme žiadanú novú pozíciu vozidla a tak vieme vypočítať regulačnú odchýlku, či už medzi žiadanou polohou, alebo aj rýchlosťou. Dve takéto regulačné slučky nájdeme v CARLE a sú aktívne využívané pri vozidlách riadených autopilotom. V našej aplikácii bude cieľom nájsť vhodné riadenie pre auto inou metódou (porovnanie na obrázku 19).



Obrázok 19: Porovnanie riadenia pomocou autopilota (čierna) a nášho riadenia (červená)

Nášmu riadeniu vozidla sa detailnejšie budeme samozrejme venovať v ďalších kapitolách, porovnanie v Obrázok 19 nám bude slúžiť na popísanie súvislostí medzi týmito riadeniami, keďže využijeme ich podobnosti – v CARLA sú to agenti.

3.5.4 Agenti

Agenti majú v sebe integrovaný celý systém navigácie a zároveň aj systém riadenia. Agent sa veľmi jednoducho aplikuje na vytvorené vozidlo a pomocou pár príkazmi sme následne schopný toto vozidlo na základe agentových rozhodnutí aj ovládať. Nás však nebude zaujímať integrovaný autopilot – z agenta vieme jednoducho dostať navigačné údaje, ktoré budú veľmi potrebné pre naše riadenie, pričom jeho integrácia na vozidlo je omnoho jednoduchšia ako integrácia navigačných systémov osobitne. Využijeme riadenie rýchlosti pomocou agenta. CARLA ponúka dva typy agentov – **základný** a **behaviorálny**. Rozdiel medzi nimi je v spôsobe rozhodovania v rámci krízových situácii, kým behaviorálny agent ponúka užívateľovi si zvoliť mód podľa ktorého sa bude správať (agresívny, konzervatívny, ...), základný ponúka len bezpečné presúvanie sa po mape s využitím navigačného systému a riadenia.

3.5.4.1 Základný agent

Základný agent ponúka niekoľko voliteľných možností, v projekte je nastavený podľa nasledujúcej tabuľky:

Názov	*Nastavenie*
Požadovaná rýchlosť [km/h]	50
Ignorovanie semaforov [bool]	Áno
Ignorovanie stop značení [bool]	Áno
Ignorovanie okoloidúcich vozidiel [bool]	Áno
Ideálna vzdialenosť od okoloidúceho vozidla [m]	5
Maximálne zatočenie	0.8
Maximálne brzdenie	0.5

Tabuľka 5: Nastavenie agenta

3.6 Nastavenie simulácie

Nastavenie	**Hodnota**
Typ kroku	*Fixný*
Mód	*Synchrónny*
Perióda vzorkovania	*0,05s*
Počasie	*ClearNoon*
Model vozidla	*Tesla Model 3*
Mapa	*Town04*

Tabuľka 6: Súhrn nastavení simulácie

4 Vnem prostredia

V minulej kapitole sme sa venovali simulačnému prostrediu, pričom sme si v krátkosti predstavili aj základné elementy vnemu tohto prostredia – budú nimi kamery a senzory. Aby sme dokázali zostrojiť autonómne vozidlo, musíme sa zamyslieť, podľa čoho sa pri jeho riadení orientujú ľudia – sú to pravidlá cestnej premávky a vnem prostredia. Ako sme spomínali, o dodržanie pravidiel sa bude starať integrovaný agent, ktorý nám bude dávať informácie o bodoch, na ktoré je potrebné sa dostať. Pri realizácii tejto cesty však potrebujeme na bezpečnú prepravu ďalšie informácie:

- Detekcia čiar, ktoré ohraničujú aktuálny pruh
- Detekcia prekážok, ktoré môžu spôsobiť kolíziu
- Spracovanie aktuálnych údajov o jazde vozidla (rýchlosť, smer)

4.1 Úvod do detekcie čiar

Momentálne každé reálne vozidlo, ktoré disponuje autonómnym riadením potrebuje k svojej optimálnej činnosti viditeľné čiary. Technologickí giganti už dlhé roky používajú čiary ako ohraničenie zóny, v ktorej sa môže vozidlo pohybovať, pričom na zmenu pruhu musí mať vozidlo dôvod (analógia lokálnej navigácie v CARLE). Práve preto sa majoritná časť tejto kapitoly bude venovať detekcii čiar. Práve tento vnem prostredia totiž bude tvoriť nosnú časť informácie o aktuálnom stave prostredia.

4.1.1 Možné realizácie

Čiary sa dajú v CARLE detegovať viacerými možnosťami. V predchádzajúcej kapitole sme si ukázali segmentačnú kameru (3.4.2.2). Táto kamera nám dáva možnosť odfiltrovať ostatné elementy a tým pádom vieme teoreticky získať dokonalý obraz len čisto s elementami čiar. Tento obraz by sa dalo pomocou metód spracovania obrazu spracovať na použiteľný tvar a naozaj veľmi presne detegovať čiary. Problém je, že takéto riešenie je v realite ťažšie **realizovateľné** – najprv je totiž na klasický obraz potrebné nasadenie hlbokej siete na segmentáciu obrazu a až následne vieme zo segmentovaného obrazu extrahovať čiary. Táto možnosť by bola v reálnom svete výpočtovo omnoho náročnejšia ako ďalšie riešenie – ním je natrénovanie hlbokej neurónovej siete, ktorá bude pre nás segmentovať zo vstupného obrazu len čiary, ktoré ohraničujú aktuálny pruh. Na zisk základného obrazu použijeme prednú RGB kameru.

4.2 Dataset a augmentácia

Cieľom segmentačných sietí je schopnosť extrahovať zo vstupného obrazu tú časť, ktorá má pre nás zmysel. Na natrénovanie takýchto sietí je priamo potrebný dataset, ktorý obsahuje vstupný obraz a zároveň žiadaný výstup. Dataset si vieme buď vytvoriť, alebo použiť už existujúci. V našom prípade použijeme existujúci, ktorý vyhovuje všetkým našim podmienkam a vyzerá takto[7]:

Obrázok 20: Ukážka z trénovacieho datasetu

Trénovací dataset takýchto párov (ako na Obrázok 20) obsahuje cez tritisíc dvesto. Tento počet obrázkov by mal byť dostatočný na natrénovanie spoľahlivej hlbokej neurónovej siete. V prípade potreby vieme existujúce obrazy doplniť, napríklad

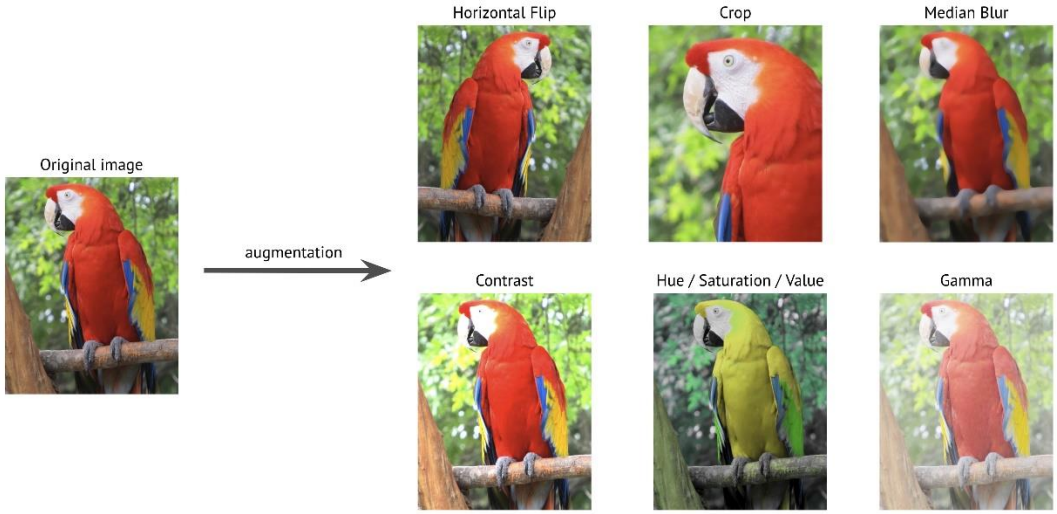
[7] Celý dataset nájdeme na [9]THEERS, M. 2021. *Lane Detection for Carla Driving Simulator: DATASET.* Dostupné online: *https://www.kaggle.com/datasets/thomasfermi/lane-detection-for-carla-driving-simulator*

nasnímaním nového obrazu, pričom necháme neurónovú sieť rozhodnúť o výsledku, následne výsledok vyhodnotíme ručne, prípadne upravíme. Ďalším spôsobom rozšírenia datasetu je *augmentácia*. Dataset taktiež obsahuje sto validačných obrázkov, ktoré do trénovacieho procesu vstupujú nezávisle a pomocou nich vieme adekvátne určiť kvalitu aktuálne natrénovanej siete.

4.2.1 Augmentácia

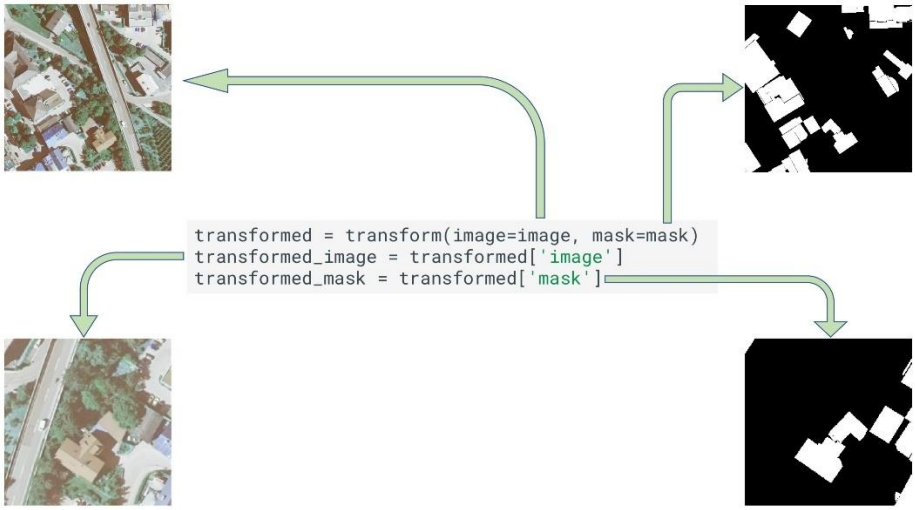
Na augmentáciu sme použili ďalšiu voľne dostupnú knižnicu v Pythone, *Albumentations* [11]. Augmentácia je proces vytvárania nových trénovácích obrázkov z už existujúcich. Na vznik novej vzorky zmeníme originálny obrázok niektorou z augmentačných operácii:

- Otáčanie obrázku (Flip)
- Orezanie obrázku (Crop)
- Rozmazanie obrázku (Blur)
- Zmena farebného nastavenia (Contrast, HSV model, Gamma)

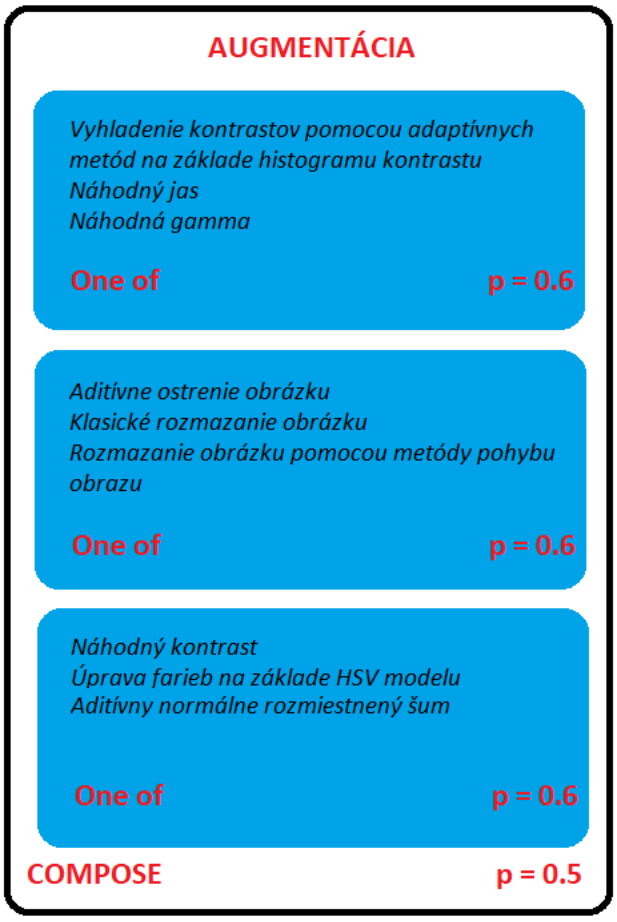


Obrázok 21: Ukážka vplyvu augmentácie

Cieľom našej augmentácie je teda rozšíriť dataset o nové, nepoznané obrázky, ktoré by mali priniesť do natrénovanej štruktúry siete vyššiu robustnosť. Pri segmentačných sieťach však nemôžeme zabúdať pri augmentácii aj na aplikovanie rovnakej augmentačnej operácie na masku. V prípade rozsiahlejšej segmentácie (detekcia viacerých elementov na obrázku) je samozrejme potrebné aplikovať augmentáciu z obrázka na všetky masky súčasne.



Obrázok 22: Ukážka augmentácie obrázku a masky



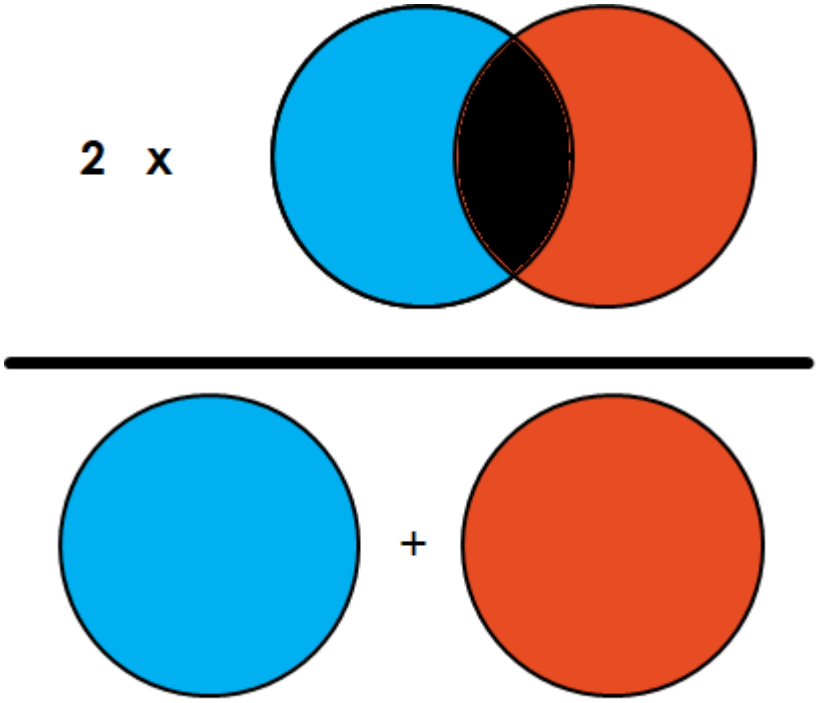
Obrázok 23: Použitá augmentácia

V projekte používame augmentáciu, ktorá umožňuje spojiť viacero augmentačných operácii (COMPOSE). Celková pravdepodobnosť augmentácie je 50%, čo znamená, že približne každý druhý obrázok by mal byť nejakým spôsobom upravený. Tieto operácie sú zosúladené v troch augmentačných ostrovoch, pričom každý ostrov obsahuje tri operácie. Ostrovy fungujú v režime ONE OF, čo znamená, že sa náhodne vyberie jedna z operácii ostrova (pravdepodobnosť výberu je rovnaká pre každú operáciu) a tá sa aplikuje na obrázok aj na masku. Pravdepodobnosť, že ostrov bude vybratý, je pri každom 60%. Môže nastať stav, kedy bude pôvodný obrázok augmentovaný tri krát, ale taktiež nemusí byť augmentovaný vôbec. Ako aj vyplýva z Obrázok 23, v rámci augmentačných metód nevyužívame otáčanie ani orezávanie obrázkov, keďže presnosti siete boli pri ich využití omnoho horšie. Využívame viacero variácii rozmazania, ostrenia obrázku a taktiež zmeny vo farebných škálach.

4.3 Metrika presnosti siete

V prípade segmentačných sietí nám klasické metrické metódy na vyjadrenie chyby neurónovej siete nebudú stačiť. Ako metriku kvality preto použijeme *Dice multi*. Je to metóda na určenie presnosti segmentačnej siete, ktorá hodnotí prekrytie masky (teda pre nás ideálnej segmentácie) a zároveň aj reálnej segmentácie. Vieme ju vyjadriť ako:

$$Dice = 2 * \frac{S_{prekrytia}}{S_{Maska} + S_{SNN}} \tag{13}$$



Obrázok 24: Dice multi

Na Obrázok 24 vidíme vizuálne znázornenie vzorca 13. Modrou je celková očakávaná plocha segmentácie, oranžovou skutočná segmentovaná a čiernou ich prekrytie.

4.4 Implementácia

Ako teoretický základ pre praktickú realizáciu použijeme časť 1.5, ktorá sa venovala segmentačným neurónovým sieťam. Aby sme implementované siete vedeli presne porovnať, na metriku ich presností použijeme *DiceMulti*, spomenutú v predošlej kapitole. Proces ich implementácie v skratke popíšeme v nasledujúcich podkapitolách.

4.4.1 Unet++ s využitím predtrénovanej ResNet (PyTorch)

Začneme s načítaním samotnej štruktúry Unet++ (Obrázok 8). Tá je súčasťou segmentačných modelov v knižnici *PyTorch* – pomocou tejto knižnice bude prebiehať aj samotné učenie siete. Dôležitým nastavením je naviazanie PyTorch ku grafickej karte, keďže výpočty na nej budú prebiehať omnoho rýchlejšie ako na CPU. Ako enkóder použijeme *ResNet34* s predtrénovanými váhami zo siete *ImageNet*. Následne nachystáme separátne datasety, najprv trénovací a potom aj validačný, niektoré obrázky prejdú augmentáciou (podľa Obrázok 23). Tieto datasety bude sieť v procese trénovania postupne načítavať pomocou *DataLoader* (jeho parametrom je veľkosť dávky [angl. **Batch size**]). Nastavíme trénovacie parametre a definujeme trénovaciu a validačnú epochu. Konkrétne parametre trénovania uvádzame v záverečnej tabuľke:

Keďže používame predtrénované váhy siete, nepotrebujeme používať vysoké počty trénovacích epoch, keďže už po malom počte je sieť blízko optimálnych výsledkov. Po skončení trénovania si uložíme výsledky trénovania a samozrejme si uložíme aj parametre natrénovaného modelu, tak aby sme ho vedeli použiť neskôr.

4.4.2 MobileNetV3Small (FastAI)

Ako sme spomínali v predošlých kapitolách, FastAI je open-source nadstavba pre PyTorch. To znamená, že postup bude veľmi podobný – najprv ako výpočtové zariadenie aktivujeme GPU pre rýchlosť výpočtov.

Aby sme vedeli sieť natrénovať, potrebujeme dataset načítať spôsobom, ktorému rozumie FastAI. Ten využíva *DataBlock*. Pri segmentačných sieťach je potrebné nadefinovať vstupný obrázok a následne aj definovať masku, pričom pri maske je potrebné čiary definovať ako samostatné objekty – budeme hľadať 3 objekty, pozadie (**back**), ľavú (**left**) a pravú (**right**) čiaru. *DataBlock* ďalej potrebuje potrebuje funkciu, ktorá bude z adresárov načítavať obrázky (vstupné aj masky). Taktiež funkciu, ktorá dočasne premenuje obrázky s maskou tak, aby ich algoritmus vedel spracovať. Potom už definujeme len pravidlo, podľa ktorého určíme, že obrázok patrí k trénovaciemu alebo validačnému datasetu a definujeme augmentáciu.

Z *DataBlocks* si vytvoríme *DataLoader* (implementácia skoro zhodná s PyTorch), pričom definujeme cestu k súboru, kde sa nachádza celý dataset a opäť veľkosť dávky.

Načítame predtrénovaný model **MobileV3Small**. Upravíme jeho základné nastavenia tak, že budeme hľadať len 3 triedy a zmenšíme počet filtrov na 8.

Na trénovanie modelu použijeme súčasť *Learner*. ten potrebuje na svoju funkcionalitu všetky predošlé spomenuté súčasti, plus nadefinujeme **DiceMulti** metriku. Následne v rámci objektu už len spustíme trénovanie siete. Po dotrénovaní uložíme model.

Z dokumentácie fastAI vyplýva, že pri dotrénovaní tejto siete je potrebné použiť menší krok učenia – preto sme ponechali odporúčaný 0,001. Taktiež nie je potrebný vyšší počet trénovacích epoch, ako tomu bolo aj pri predošlom modeli.

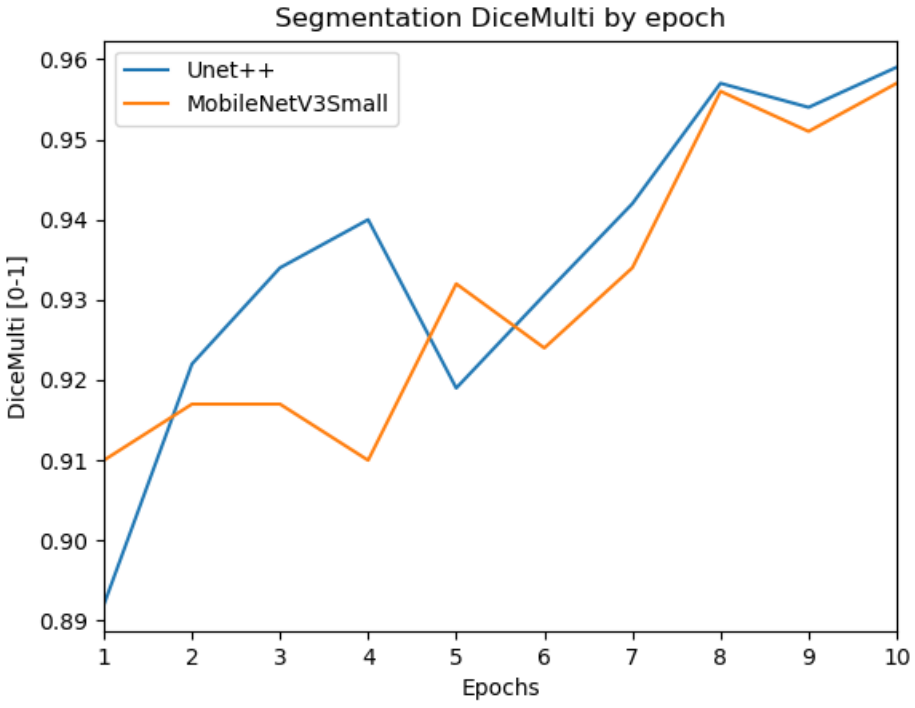
4.4.3 Parametre učenia

Model	**Unet++ 4**	**MobileV3Small**
Metrika	Dice multi	Dice multi
Optimalizačný algoritmus	ADAM	ADAM
Krok učenia	0,01	0,001
Počet trénovacích epoch	10	10
Veľkosť dávky (batch size)	2	2
Validácia každých x epoch	1	1

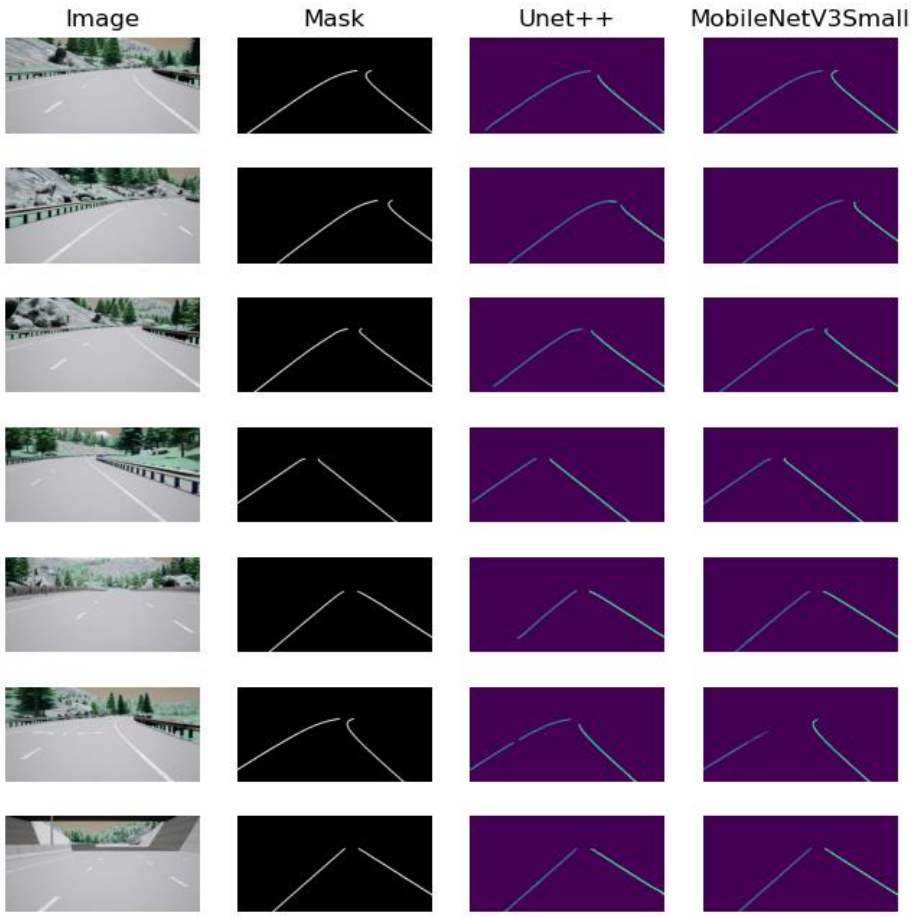
Tabuľka 7: Parametre učenia siete MobileV3Small

4.5 Výsledky

V rámci výsledkov nás budú zaujímať dve relevantné porovnania vlastností. Prvou, kľúčovou je presnosť segmentácie na základe metriky a druhou, taktiež veľmi dôležitou je rýchlosť segmentácie.



Obrázok 25: Vývoj trénovania sietí z hľadiska presnosti



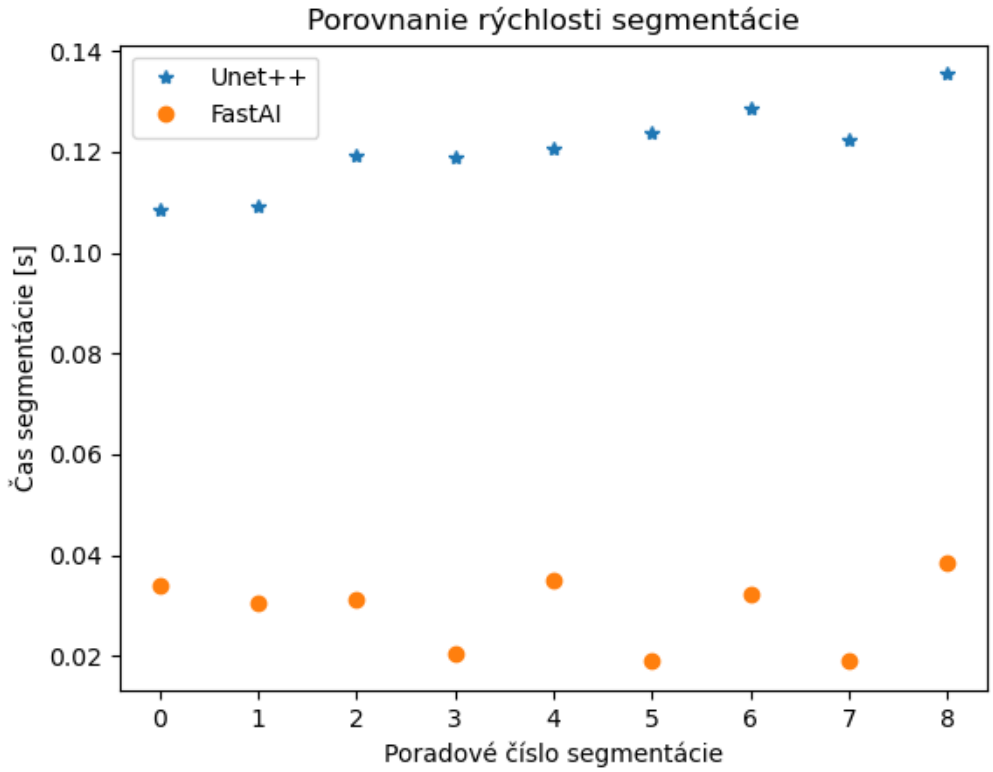
Obrázok 26: Porovnanie na validačných dátach

Na Obrázok 25 vidíme priebeh trénovania sietí. Z neho vyplýva, že **Unet++** dosahuje vyššiu presnosť ako MobileNetV3Small, avšak rozdiel je veľmi malý. Obrázok 26 potvrdzuje tieto výsledky aj na validačných dátach, keďže vidíme priame porovnanie segmentácii s maskou (a reálnym vstupným obrazom). Je dôležité povedať, že segmentácie sú veľmi presné v oboch prípadoch a obe siete sú tak vyhovujúce. Preto prejdeme k druhému kritériu – porovnania času trvania segmentácie u oboch sietí.

Na Obrázok 27 vidíme časy desiatich náhodných segmentácii z validačného datasetu. Už z neho je zrejmé, že **FastAI MobileNetV3Small** je omnoho rýchlejšia. Na potvrdenie hypotézy sme však zobrali celý dataset a zmerali časy. Následne sme urobili aritmetický priemer a výsledky sú v tabuľke nižšie.

Názov siete	Priemerný čas segmentácie [s]	Najrýchlejší čas segmentácie [s]	Najpomalší čas segmentácie [s]
Unet++	*0,119*	*0,095*	*0,181*
MoblileNetV3Small	***0,033***	***0,017***	***0,062***

Tabuľka 8: Časy segmentácie na celom datasete



Obrázok 27: Porovnanie časov segmentácie

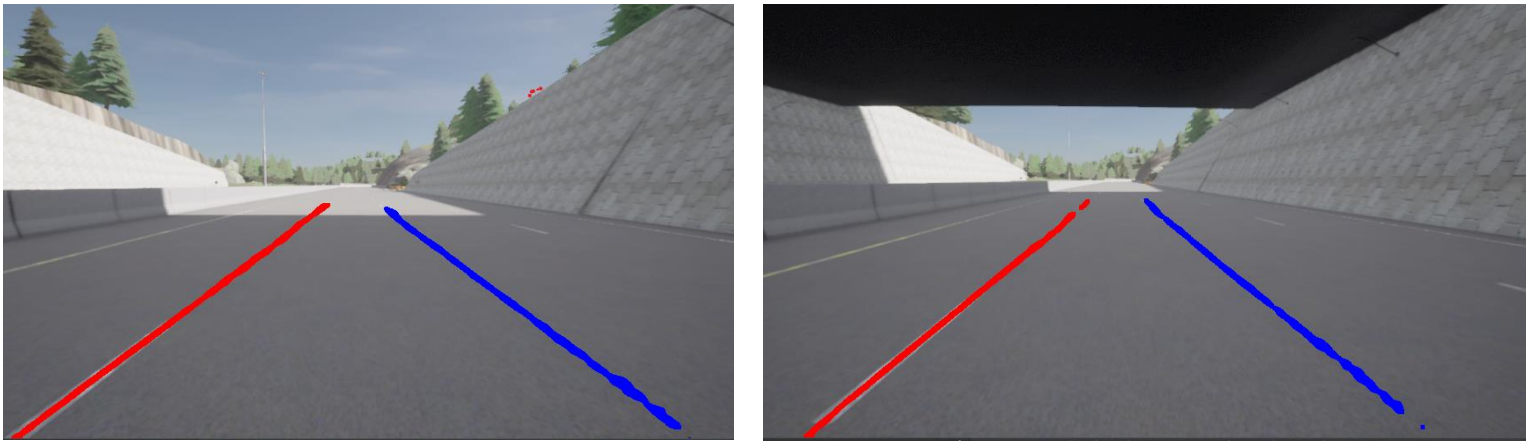
Z hľadiska rýchlosti segmentácie tak celkom jasne víťazí **MobileNetV3Small**. Keď to zoberieme celkovo, tak rozdiel v presnosti segmentácie je veľmi malý, avšak rozdiel v rýchlosti je signifikantný. Keďže segmentácia má prebiehať v reálnom čase (čo najrýchlejšie), je pre nás prvý spôsob realizácie príliš pomalý, keďže by umožňoval menej ako 10 segmentácii za sekundu. Ako sme spomínali v nastaveniach simulátora (3.1), perióda vzorkovania bude 0,05s – s určitosťou potrebujeme, aby bol čas segmentácie menší ako táto perióda vzorkovania – na detekciu čiar použijeme **MobileNetV3Small.**

4.6 Integrácia čiar

Keď už dokážeme detegovať čiary, potrebujeme túto informáciu interpretovať tak, aby bola využiteľná pre naše účely. V tejto chvíli totiž len vieme, ktoré pixely na obrázku prislúchajú čiaram. Následne je potrebné takýto detektor integrovať k vozidlu.

4.6.1 Spracovanie výstupu a filtrácia

Zo segmentačnej siete získame pravdepodobnostnú mapu, ktorá ukazuje, s akou šancou daný pixel predstavuje čiaru. Musíme si zvoliť vhodnú hraničnú hodnotu, aby sme vedeli odfiltrovať čo najviac falošných detekcií a zároveň neprišli o reálne merania. Experimentálne sme zistili, že prahová hodnota v tomto prípade je **0,3** – vykreslíme len tie pixely, ktoré túto prahovú hodnotu prekročili. Na ďalšie filtrovanie výstupu použijeme operáciu **binárne otvorenie**. Je to **erózia** nasledovaná **dilatáciou**. Cieľom takejto filtrácie je vyhladiť čiary a odfiltrovať prípadné falošné detekcie. Tie vznikajú hlavne pri detekcií v inom prostredí – napríklad na ceste pod tieňom:



Obrázok 28: Porovnanie vplyvu filtra (vľavo bez, vpravo s filtrom)

Po aplikovaní filtra na obrázku vyššie vidíme, že neprichádza ku falošným detekciám, avšak môže to viesť aj k zmierneniu presnosti samotnej detekcie – vidíme, že čiary po filtrácii nie sú až tak hrubé. Dôležitý je však v tomto prípade tvar, keďže čiaru budeme neskôr interpretovať aproximáciou, vynechanie pár bodov neurobí s finálnym výsledkom v podstate nič.

4.6.2 Inverzné mapovanie

Ako prvé potrebujeme teda výstupný obraz (o veľkosti **u** x **v** pixelov) z detektora premeniť na trojrozmerné prostredie – v súradniciach **X**, **Y** a **Z**. Proces mapovania hovorí:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{14}$$

My však v tomto prípade potrebujeme riešiť inverznú úlohu, keďže práve teraz súradnicami u a v disponujeme. Preto bude inverzný vzťah vyzerať:

$$r(\lambda) = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \lambda K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \lambda \epsilon R_{>0} \tag{15}$$

λ je vo vzťahu vyššie označené ako reálne číslo. Je to tak preto, lebo jeho konkrétnu hodnotu nepoznáme. Aby sme si však uľahčili riešenie, vieme povedať, že body $r(\lambda)$ bude ležať na ceste, keďže to je výsledok segmentácie čiar. Bod r teda bude ležať na rovine, charakterizovanej normovaným vektorom n a bodom na rovine r_0:

$$r(\lambda) \leftrightarrow n^T(r - r_0) = 0 \tag{16}$$

Normovaný vektor má jednoduchý tvar $n = (0, 1, 0)^T$, pričom sme v rovine cesty. Os kamery však nie je súbežná s osou cesty a musíme uvažovať aj s natočením - pridáme ešte rotačnú maticu. $n_C = R_C(0, 1, 0)^T$ Je potrebné si uvedomiť, že kamera je teda v rámci roviny na pozícii (0, 0, 0) a zároveň, že kamera je v nejakej výške oproti ceste. Túto výšku označme **h**. Pre jednoduchosť vyberieme, že $r_0 = hn_C$ a tak prepíšeme (16):

$$\begin{gathered} 0 = n_C^T(r - r_0) \\ 0 = n_C^T r - n_C^T r_0 \\ n_C^T r = h \end{gathered} \tag{17}$$

Za r dosadíme bod v pôvodných súradniciach:

$$\begin{gathered} h = n_C^T \lambda K^{-1}(u, v, 1)^T \\ \lambda = \frac{h}{n_C^T K^{-1}(u, v, 1)^T} \end{gathered} \tag{18}$$

Z tohto tvaru sme už schopní získať celý vzťah dosadením do pôvodnej rovnice:

$$\begin{gathered} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{h}{n_C^T K^{-1}(u, v, 1)^T} K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \\ \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{h}{R_C(0, 1, 0)^T (u, v, 1)^T} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \end{gathered} \tag{19}$$

Zo vzťahu vyššie už samozrejme všetky hodnoty poznáme – rotačná matica a aj výška kamery závisí na našom nastavení (nastavenia kamery sú v Tabuľka 4).

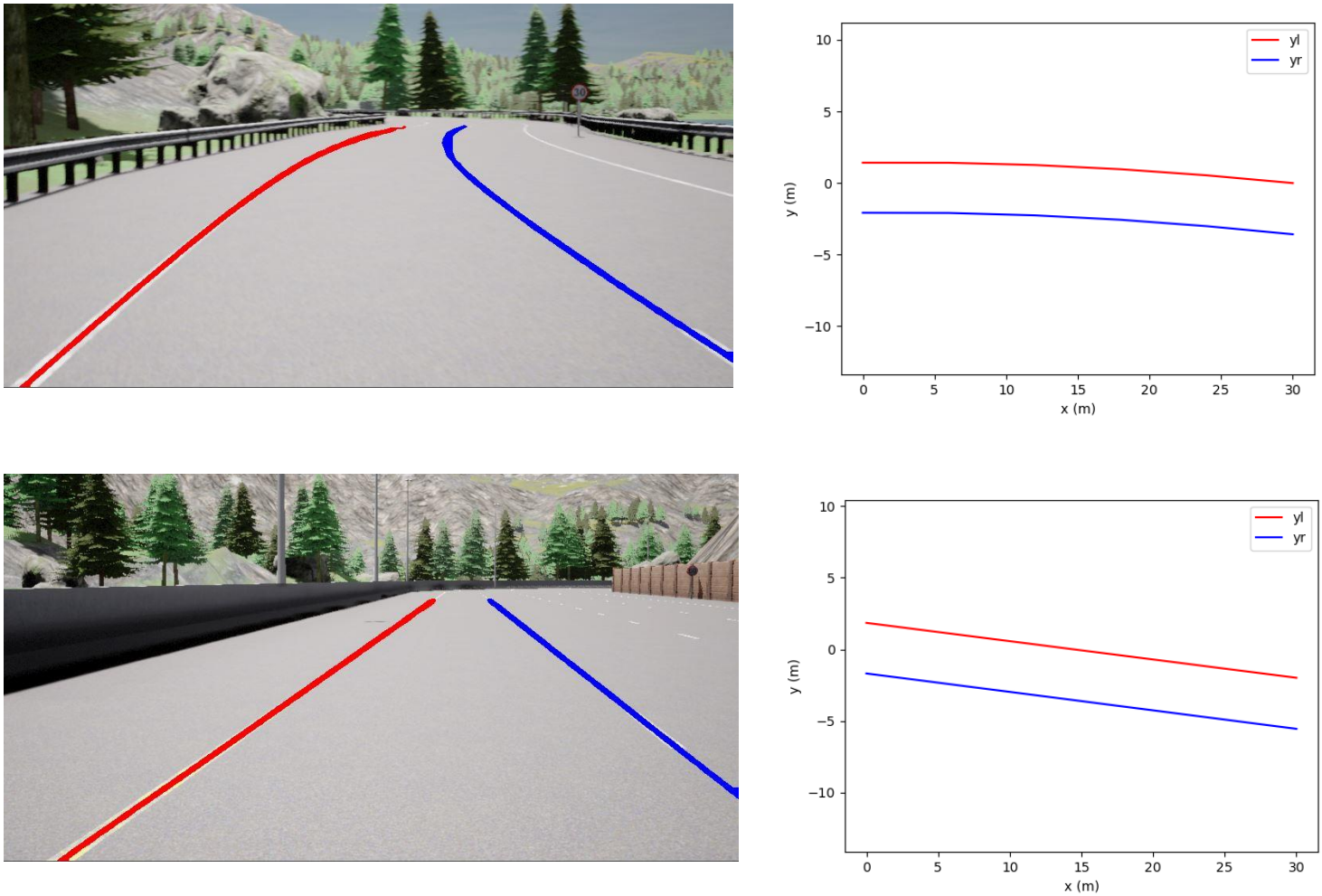
4.6.3 Polynomiálna aproximácia

Na interpretáciu čiar v numerickom tvare použijeme **polynomiálnu aproximáciu** kubického tvaru, keďže taký je približný tvar očakávaných čiar:

$$y_{LINE}(x_{LINE}) = a_3 x_{LINE}^3 + a_2 x_{LINE}^2 + a_1 x_{LINE} + a_0 \tag{20}$$

Týmto získame predpis, pričom vieme tak vyjadriť ku každému x vyjadriť prisluchajúcu súradnicu y. Tento postup je samozrejme potrebný vykonať pre obe čiary samostatne, pričom je potrebné najprv spraviť priestor inverzne zmapovať. Pri mapovaní treba vhodne interpretovať aj pravdepodobnosť, že pixel z pôvodnej pravdepodobnostnej mapy prislúcha niektorej čiare. Konkrétny algoritmus nájdeme v programovej dokumentácii.

Je v ňom popísaný celý proces zisku polynomiálnej aproximácie tvaru čiar zo vstupného obrázku kamery. Na ukážku výsledkov uvádzame nasledovné porovnanie:

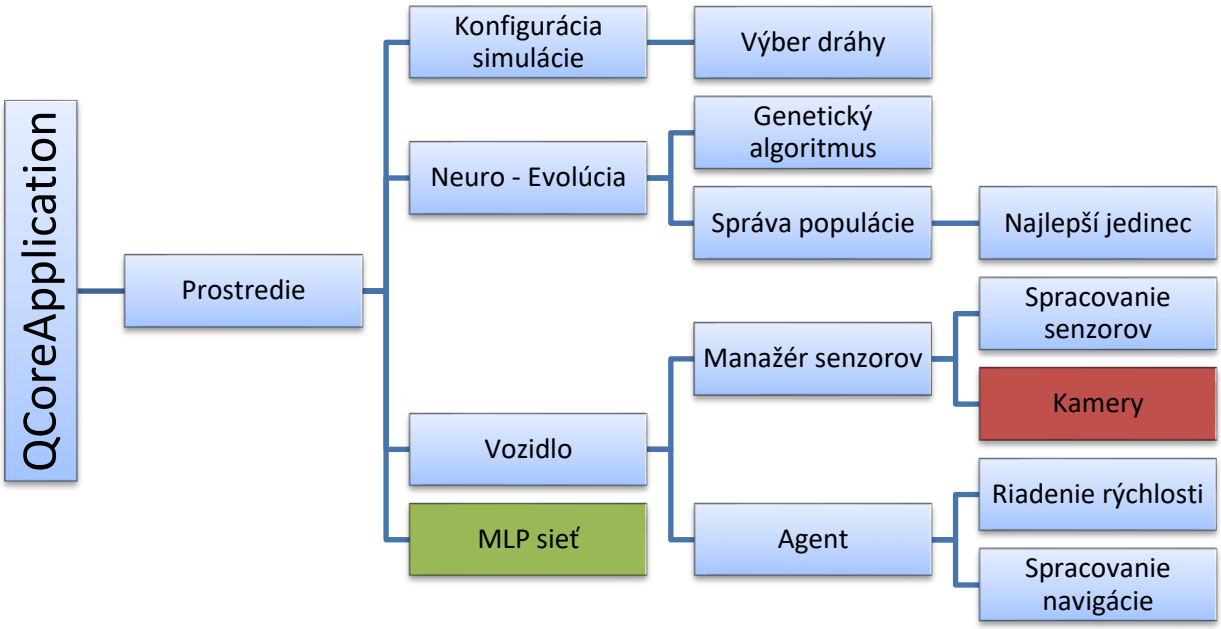


Obrázok 29: Ukážky polynomiálnych aproximácii

Ako vidíme na obrázkoch vyššie, polynomiálne aproximácie nám dávajú presnú a relevantnú informáciu o lokalite čiar v metroch. Takáto informácia bude ľahko použiteľná pri aplikácii. Výpočtový čas aproximácie je približne 10ms, čo spoločne s priemerným časom detekcie čiar stále vyhovuje nastaveniam simulácie (menej ako 50ms).

5 Implementácia

Základným elementom celého programu bude súčasť knižnice Qt, konkrétne *QCoreApplication*. Úlohou tohto elementu je udržať aplikáciu nažive a spracovávať viacero vláken, ktoré sú využívané počas behu programu. Kompletnú architektúru vidíme na nasledujúcom obrázku:



Obrázok 30: Architektúra

V nasledujúcich podkapitolách tak postupne prejdeme celou hierarchiou architektúry.

5.1 Prostredie (CarlaEnvironment)

Aby sme vedeli vytvoriť akési premostenie medzi simulátorom a našou aplikáciou budeme potrebovať vhodné rozhranie, ktoré nazveme prostredie (*CarlaEnvironment*). Cez prostredie vieme nastaviť, ako sa program bude správať a v neposlednom rade spúšťa ostatné súčasti, ktoré sú v hierarchii pod ním.

Podľa nastavenia simulátora (Tabuľka 6) vieme, že jeden krok simulácie odsimuluje 0,05 sekundy a taktiež simulátor beží v synchrónnom móde. Tieto, a mnohé ďalšie nastavenia prebiehajú v konfigurátore simulácie, ktorý je volaný bezprostredne po spustení ktoréhokoľvek scenára (*CarlaConfig*). Na vybudenie ďalšieho kroku tak potrebujeme

signál pre simulátor – **tick**. Tento signál môže vyslať v našom programe len prostredie (podľa dokumentácie sa odporúča vysielať signál len z jedného miesta).

V prostredí taktiež uchovávame informácie o vozidlách, nachádzajúcich sa v simulačnom prostredí, pričom sa tu taktiež vyberá, kedy a kde bude vozidlo vložené do simulácie (aj so všetkými súčasťami vozidla v rámci jeho hierarchie).

V prípade trénovacieho scenára prostredie spustí aj blok neuro-evolúcie (*NeuroEvolution*). Ten následne zabezpečuje trénovanie UNS.

5.2 Konfigurácia simulácie (CarlaConfig)

Na konfiguráciu simulácie a aj ďalších súčastí použijeme konfiguračný súbor s prílohou *.ini* (vzorová ukážka takéhoto súboru je v Príloha C:). Úlohou konfigurátora je rozdeliť tento súbor podľa hlavičiek sekcií a načítať ich. Následne vo vhodných tvaroch doručiť konfiguračné dáta pre časti, kde budú žiadané. Konfigurátor bezprostredne po svojej inicializácii aplikuje nastavenia pre CARLA server podľa sekcie CARLA zo súboru.

5.2.1 Výber dráhy

Podúlohou pre konfigurátor bude výber dráhy. Tú v konfiguračnom súbore nevidíme, avšak informácia o tom, ktorú dráhu má konfigurátor vybrať dostáva priamo z nadradeného prostredia. V aktuálnej situácii máme 3 možné dráhy – všetky na mape *Town04*, súradnice v zátvorkách sú v osiach X a Y, keďže Z je zanedbateľné:

Názov dráhy	Trénovacia	Testovacia 1	Testovacia 2
Štart [X,Y]	[-9.6, -143.7]	[-67.2, 33.9]	[303.2, -172.4]
Medzicieľ 1 [X,Y]	[-8.6, 107.5]	[98.6, 34.7]	[258.9, -186]
Medzicieľ 2 [X,Y]	[-121.8, 395.2]	[272.5, 37.5]	[242, -249.7]
Medzicieľ 3 [X,Y]	[-363.4, 406]	[409.1, -37.2]	[200, -229.6]
Medzicieľ 4 [X,Y]	[-463.6, 333.6]	[410.7, -228.7]	----------
Cieľ [X, Y]	[-490, -174]	[211.2, -392.1]	[220.5, -169.5]

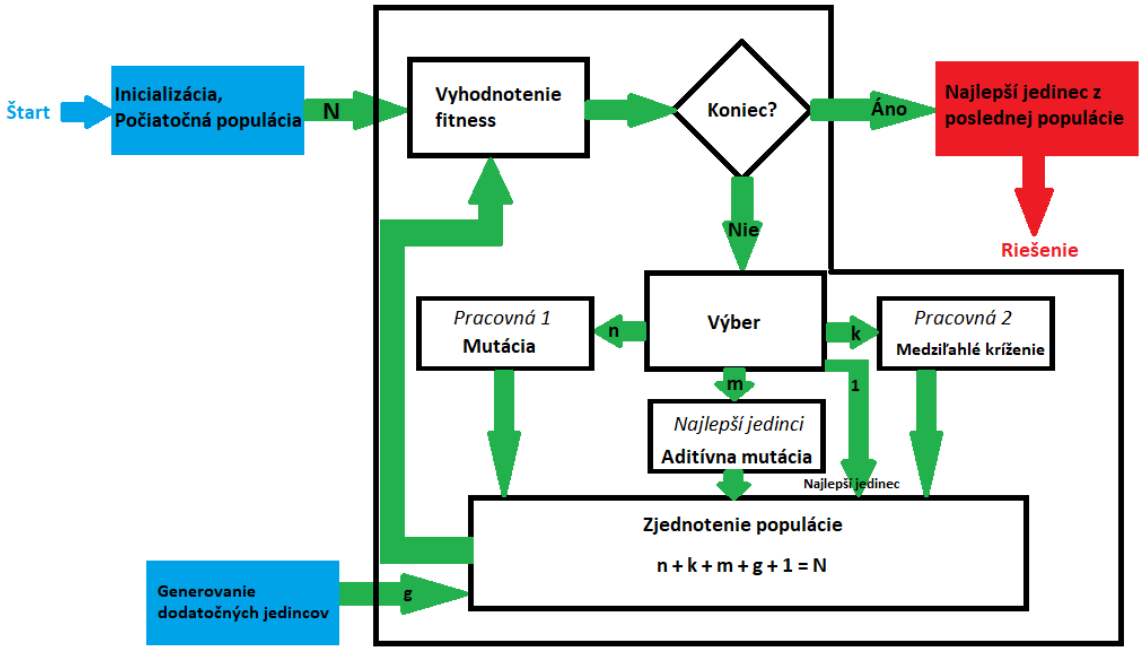
Tabuľka 9: Navrhnuté dráhy

5.3 Neuro – evolúcia (NeuroEvolution)

Ako sme už mnohokrát spomínali, neuro-evolúcia bude použitá na učenie riadiacej UNS pre natáčanie kolies. Základnou úlohou tejto súčasti je prevádzať genetické operácie medzi cyklami genetického algoritmu a vyhodnocovať žiadanú fitness funkciu. Taktiež sa stará o priebežné ukladanie priebehu genetického algoritmu ako aj o uloženie najlepšieho jedinca a ďalších dát, potrebných k zopakovaniu jazdnej dráhy tohto jedinca. V prípade testovacích dráh táto súčasť nevykonáva optimalizačnú funkciu, len poskytne najlepšieho jedinca pre UNS.

5.3.1 Genetický algoritmus

Teoretické pozadie genetického algoritmu sme spomínali v 2. kapitole. Na základe týchto znalostí sme navrhli genetický algoritmus nasledovne:



Obrázok 31: Bloková schéma použitého genetického algoritmu

Podľa Obrázok 31 implementujeme genetický algoritmus. Vidíme, že z celej populácie vyberieme jedincov do troch ostrovov – *najlepšej* a *dvoch pracovných*.

V projekte sme porovnali viacero parametrizácii genetického algoritmu, avšak ako ideálny pomer rýchlosti optimalizácie a zároveň času optimalizácie (zväčšením celkovej populácie zvýšime celkovú výpočtovú náročnosť) sa ukázalo použitie celkovej populácie o veľkosti 20 (N) jedincov, v tabuľke vidíme preto relatívnu početnosť (určenú v percentách / podľa jasného vzťahu) a taktiež reálny počet:

Názov ostrova	**Spôsob**	**Početnosť**	**Počet**	**Operácia**
Najlepší jedinec	selsort	1	1	----------
Najlepší jedinci	selsort	15% z populácie	3 (m)	Aditívna mutácia
Pracovná 1	selrand	35% z populácie	7 (n)	Medziľahlé kríženie
Pracovná 2	seltourn	35% z populácie	7 (k)	Klasická mutácia
Generovaná	genrpop	N – n – m – k – 1	2	----------

Tabuľka 10: Rozdelenia do skupín

Na ušetrenie času optimalizácie je vždy najlepší jedinec z predošlej populácie automaticky ohodnotený a nevykonáva sa ňom trénovací scenár. Čas prejdenia celej trénovacej dráhy sa pohybuje v rozmedzí 1 – 3 min (podľa výkonnosti počítača).

Pri genetických operáciách sme použili overené hodnoty, ktoré výborne vplývajú na neuroevolúciu. *Aditívna mutácia* má mieru mutácie 1%, pričom maximálne povolené zmeny sú v 10% rozsahu jednotlivých génov. *Klasická mutácia* má mieru mutácie 15% a *medziľahlé kríženie* nepoužíva susedné reťazce a parameter alfa má hodnotu 0,75.

5.3.2 Účelová funkcia

Neuro – evolúcia taktiež ohodnocuje kvalitu jednotlivých riešení – v terminológii genetických algoritmov sa takáto funkcia nazýva *fitness* funkcia. Cieľom celého genetického algoritmu je nájsť jej globálne minimum. V našom prípade využívame funkciu zloženú z dvoch pokutových elementov a dvoch odmien. Vyzerá nasledovne:

$$fit = 5 * crossings + 5000 * bad - (3 * range + 2500 * goals) \quad (21)$$

, pričom $crossings$ je počet prejdení cez čiaru do iného pruhu – výnimkou je ak navigácia vyžaduje zmenu pruhu, vtedy takéto prejdenie nie je určené ako nesprávne, bad je kolízia, prípade iná situácia z ktorej sa auto nevie dostať – za určitých okolností je možné, že vozidlo aplikuje maximálne (minimálne) natáčanie a bude sa točiť dokola, tento stav dokážeme detegovať pomocou nastavenia minimálnej prejdenej vzdialenosti za posledných 10 simulačných krokov. Vozidlo dostáva taktiež odmeny - $range$ je prejdená vzdialenosť a $goals$ počet dosiahnutých (čiastkových) cieľov.

5.3.3 Správa populácie

Druhou úlohou je správa populácie. Neuro-evolučná časť na začiatku vypočíta koľko členov bude prislúchať jednotlivým skupinám podľa veľkosti celkovej populácie. Taktiež v procese GA spája jedincov po vykonaní genetických operácii späť do hlavnej populácie a následne spracúva dáta o ich fitness funkcii.

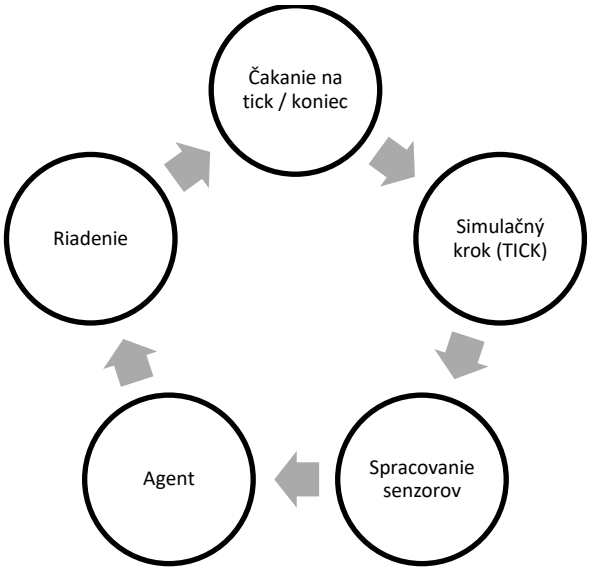
Po skončení priebehu GA ukladá najlepšieho jedinca a vývoj fitness funkcie na disk, pričom sa zároveň vykreslí priebeh genetického algoritmu. Čiastkové ukladanie najlepšieho jedinca prebieha pre istotu každý cyklus, aby sme zabránili strate dát, keďže program je výpočtovo veľmi náročný.

Pri testovacích scenároch a ukážke výsledku optimalizácie si neurónová sieť pri svojej inicializácii vyžiada najlepšieho jedinca z populácie. V takomto prípade neprichádza k ďalšej optimalizácii.

5.4 Vozidlo (Vehicle)

Poslednou, zároveň najväčšou súčasťou hierarchie z Obrázok 30 je vozidlo. Cieľom je, aby vozidlo nieslo so sebou všetky potrebné súčasti na jeho riadnu funkcionalitu.

Základným elementom je samozrejme model vozidla v prostredí CARLA, pripomenieme, že ako model použijeme Tesla Model 3.

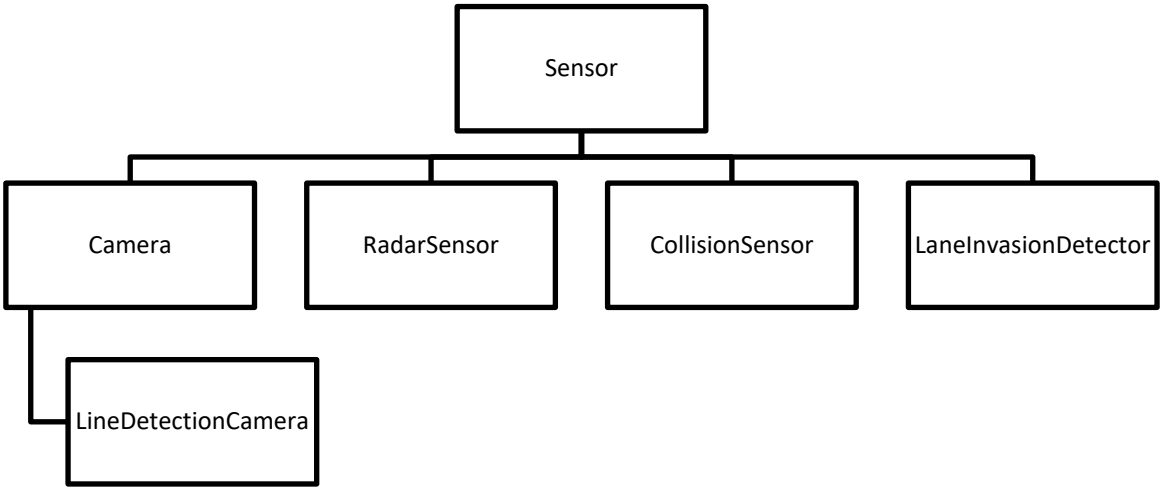


Obrázok 32: Cyklus chodu vozidla

Vozidlo v projekte vždy funguje na rovnakom princípe, ktorý sa dá zjednodušiť na päť krokov v cykle (Obrázok 32). Vozidlo vždy čaká na signál od prostredia, že sú nové dáta – tick. Po ňom sa spracujú senzorické dáta nadobudnuté simuláciou počas jedného simulačného kroku. Na základe týchto dát agent navrhne riadenie rýchlosti a zároveň pomocou jeho plánovača získame aktuálne navigačné dáta. Navrhneme riadenie pomocou UNS a aplikujeme ho. Overí sa či vozidlo nespĺňa kritériá na ukončenie simulácie.

5.4.1 Manažér senzorov

Manažér senzorov je z hľadiska vnemu prostredia kľúčovým prvkom pre vozidlo. Riadi všetky senzory a kamery v projekte a taktiež rozhoduje o ich použití. O tom, ktorý senzor je aktívny, rozhoduje zoznam senzorov v konfiguračnom súbore v sekcii *[Sensors]* (Príloha C:. Manažér senzorov si aktívne senzory zaradí do listu, pričom pre kamery vytvorí ešte dodatočný zoznam. Manažér vystupuje aj ako brána medzi jednotlivými senzormi a vozidlom, keďže priamo cez manažéra sú možné prístupy k dátam. Architektúra jednotlivých senzorov je zobrazená na nasledujúcom obrázku:



Obrázok 33: Senzory

Každý senzor je založený na princípoch definovaných v triede *Sensor*. Táto trieda zabezpečuje rovnaké spracovanie pre každý senzor – nové senzorické dáta sa uložia do radu – v ňom počkajú, pokým ich program vyzve na spracovanie týchto dát. Týmto spôsobom zabezpečíme, že žiadne dáta senzoru nebudú vynechané. V niektorých prípadoch sa v simulačnom kroku neobdržia nové dáta pre senzor, v tomto prípade sa nič nespracúva a použijú sa posledné spracované dáta.

5.4.1.1 Radar (RadarSensor)

Základným elementom pre detekciu prekážok bude **radar** (3.4.1.3)Radar vráti niekoľko stoviek nameraných bodov, ktoré potrebujeme filtrovať. Cieľom je, aby detegoval prekážky, ktoré sú reálnymi objektami a môžu spôsobiť kolíziu. Potrebujeme zabrániť falošným detekciám kvôli lúčom smerujúcim na zem, ktoré výrazne znepresňujú merania. Preto budeme využívať len merania, ktoré nemajú zápornú výšku letu. Nameraných bodov bude však aj po tomto kroku stále veľmi veľa. Aby sme vedeli interpretovať radarové meranie ako potenciálny vstup do riadiacej MLP siete, rozdelíme si horizontálnu plochu na tri časti:

- *Naľavo od vozidla (-45°; -25°)*
- *Priamo pred vozidlom (-10°; 10°)*
- *Napravo od vozidla (25°, 45°)*

Merania roztriedime podľa azimutu lúču a na záver urobíme aritmetický priemer všetkých meraní, ktorý nám dá relevantnú informáciu o prekážkach v danom smere. Tým, že vieme navoliť maximálnu nameranteľnú vzdialenosť pre každé meranie – 50 metrov, vieme povedať, či sa v danom smere nachádza prekážka.

5.4.1.2 Detektor kolízii (CollisionSensor)

Na detekciu kolízii použijeme snímač opísaný v 3.4.1.2. Snímač funguje v dvoch režimoch – základne je v **bezkolíznom**, v prípade kolíznej udalosti sa prestaví do **kolízneho** režimu. Kolízna udalosť sa môže počas jazdy vozidla stať len raz, keďže prechod späť do bezkolízneho režimu možný nie je. Režim senzoru je využitý ostatnými súčasťami, keďže prípadná kolízia znamená pre nás neriešiteľný stav. Manažér senzorov odosiela signál priamo do prostredia - prostredie v takomto prípade následne zabezpečí ukončenie jazdy vozidla.

5.4.1.3 Detektor prekročenia čiary (LineInvasionDetector)

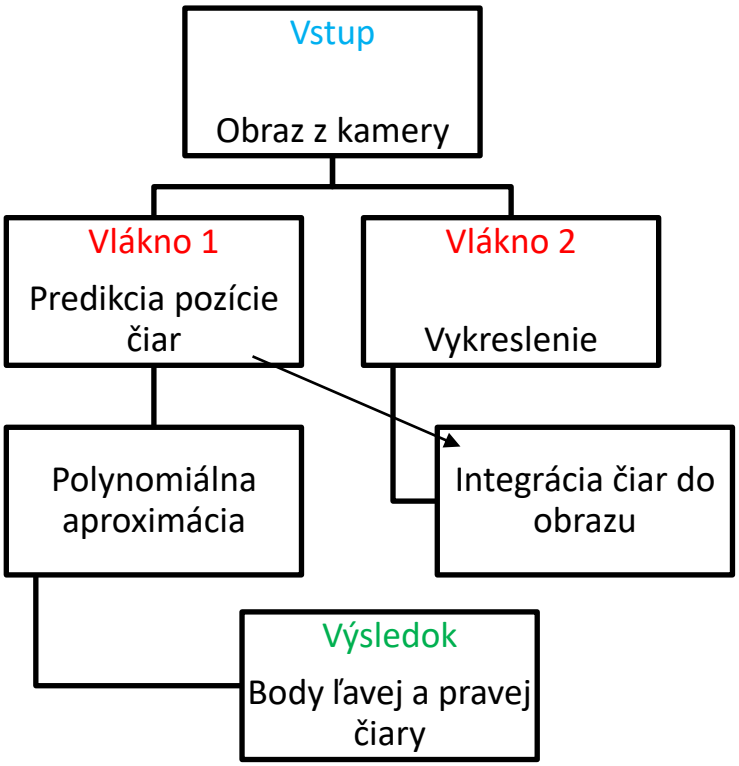
O dáta sa postará integrovaný snímač z CARLA podľa 3.4.1.1. V prípade prekročenia čiary nám snímač dávkuje viacero dát. Pre nás bude dôležitý údaj v ktorom simulačnom kroku prišlo k prekročeniu čiary a o ktorú čiaru išlo. Po skontrolovaní, či dáta korelujú s realitou príde k pričítaniu počítadla, ktoré indikuje celkový počet prekročení čiar počas simulačnej jazdy vozidla. Na konci jazdy vozidla je údaj o počte prekročení čiary zdieľaný cez manažéra senzorov.

5.4.2 Kamery (Camera)

Síce kamera je taktiež derivátom základného objektu Sensor (Obrázok 33), disponuje ďalšími možnosťami, ktoré klasické senzory nemajú. Najdôležitejším prídavkom je, že návratové dáta z tohto snímača sú v podobe obrazu – kamery sú taktiež na rozdiel od ostatných snímačov spracúvané pomocou grafickej karty a nie pomocou procesora. Frekvencia toku dát je taktiež nižšia, ako tomu je v prípade senzorov – nie každý simulačný krok sú generované obrazové dáta. Nastavenia kamier sú uvedené v Tabuľka 4. Každá kamera má možnosť vykresliť jej obrazové dáta pre užívateľa. V architektúre sú preto zvýraznené červenou, znamená to, že využívajú separátne vlákna na vykreslenie. Funkcia v rámci vlákna beží v nekonečnom cykle, pričom je možné ju prerušiť signálom zvonka – signál vysiela manažéra senzorov. Ten následne musí počkať na potvrdenie, že vlákno je skutočne ukončené a až následne môže dôjsť k ukončeniu simulácie.

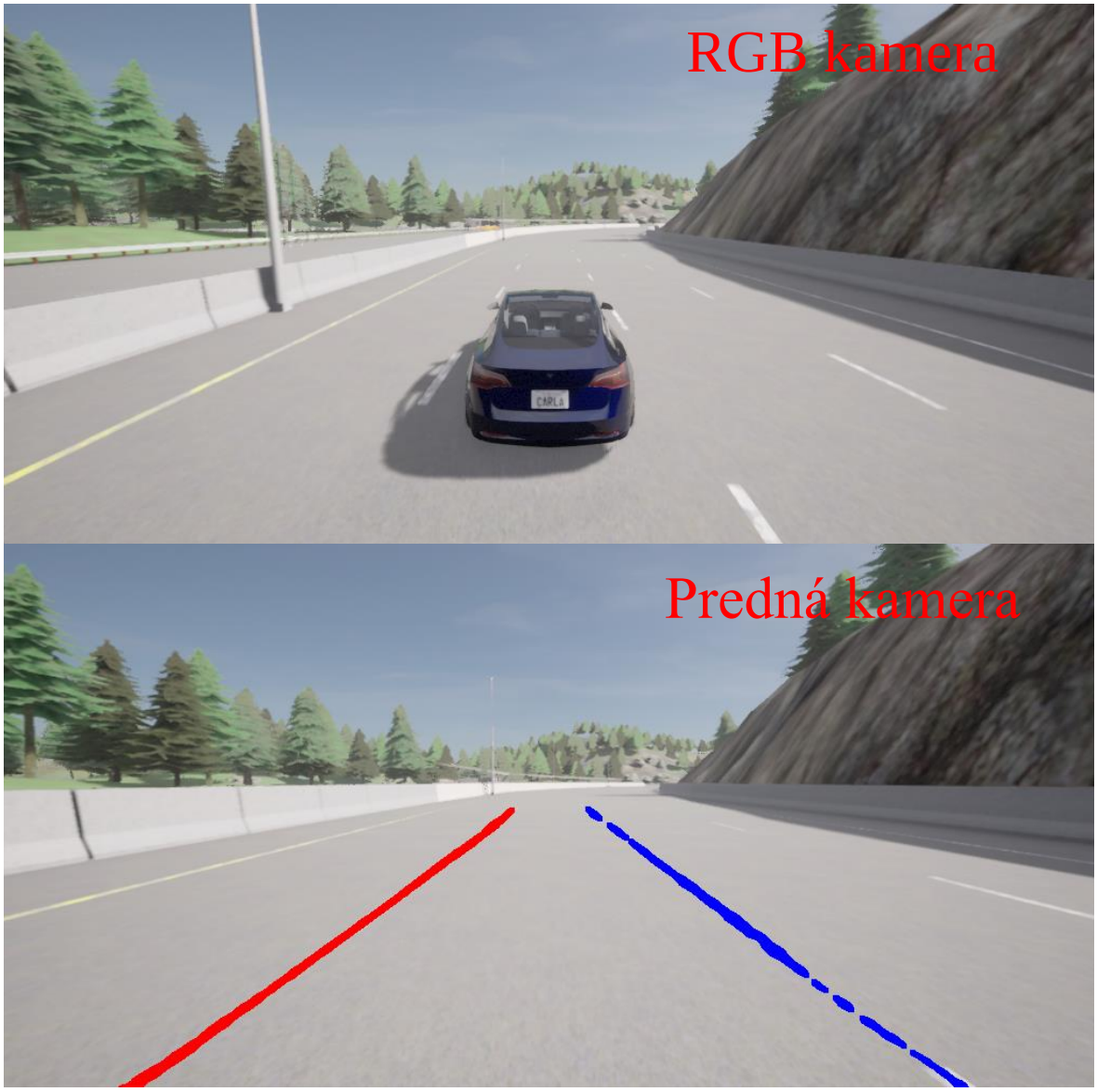
5.4.2.1 RGB predná kamera na detekciu čiar (LineDetectorCamera)

Špeciálne vlastnosti ponúka aj kamera na detekciu čiar. Je umiestnená v prednej časti vozidla. Táto kamera využíva integrovaný detektor čiar, pomocou segmentačnej neurónovej siete *MobileNetV3 Small*, ktorému sme sa venovali v predošlej kapitole (4).



Obrázok 34: Detekcia čiar pomocou obrazu z prednej kamery

5.4.2.2 Porovnanie kamier



Obrázok 35: Porovnanie použitých kamier

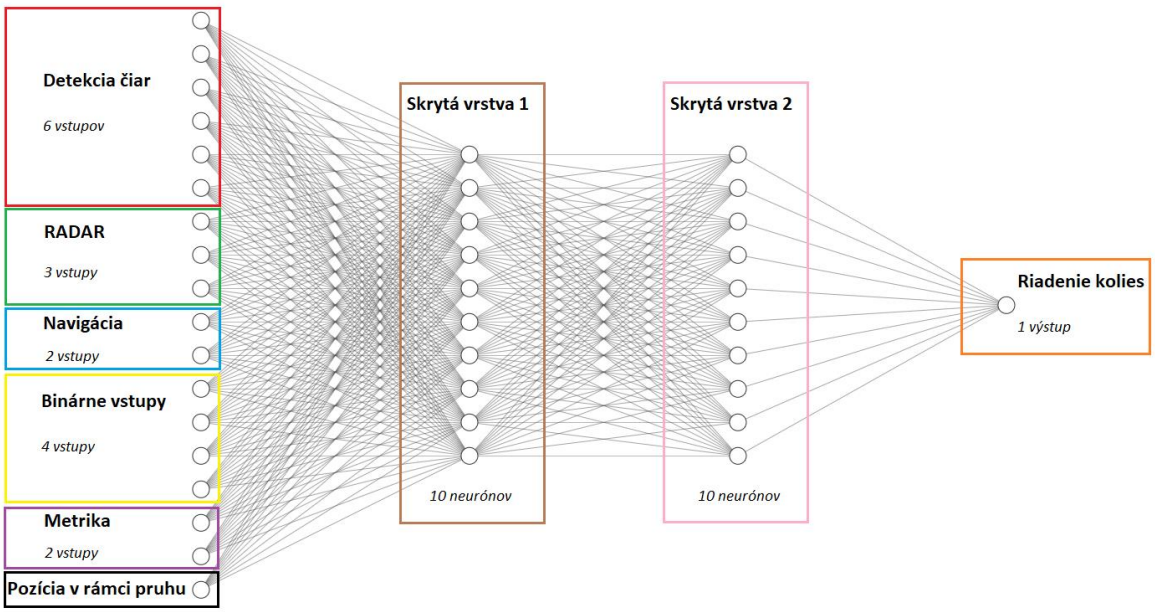
Obrázok vyššie ukazuje rozdiel medzi umiestnením a taktiež úlohou kamier. Predná kamera pri svojom vykresľovaní automaticky aj integruje detegované čiary do svojho obrazu – kompletná sekvencia je na Obrázok 34.

5.4.3 Agent

Agent plní pri vozidle dve úlohy – riadi jeho rýchlosť pomocou PID regulátora a dodáva navigačné údaje, teda konkrétne body, cez ktoré musí vozidlo prejsť, aby dosiahlo žiadaný cieľ. Jeho konkrétne nastavenia sme spomínali v 3.5.4.1.

5.5 Riadiaca MLP sieť

Väčšinu predošlých krokov sme urobili, aby sme dokázali použiť na riadenie natáčania kolies MLP sieť. Po viacerých experimentoch so štruktúrou siete sa nám ako najlepšia overila štruktúra s dvomi skrytými vrstvami po 10 neurónov.



Obrázok 36: Štruktúra MLP siete

Počet celkových vstupov je 18, vieme ich rozdeliť do šiestich kategórii, ako sú aj na obrázku vyššie. Všetky vstupy sú navrhnuté tak, aby mohli nadobudnúť hodnoty < -1 ; 1 >. Tento koncept je veľmi dôležitý a tak sú všetky vstupy na záver kontrolované, aby skutočne boli v tomto rozsahu.

5.5.1 Detekcia čiar

Ako sme spomínali na Obrázok 34, pomocou detektora čiar získame 6 bodov. Sú to súradnice v osi y, ktoré sú dané polynomiálnou aproximáciou pre hodnoty $x = [0, 10, 20]$. Keďže v tomto nevieme presne odhadovať maximálnu hodnotu, použijeme klasickú vektorovú normalizáciu.

5.5.2 RADAR

Tri radarové merania, získané podľa 5.5.2, normalizujeme pomocou vzťahu:

$$radar_{Norm_k} = 1 - (\frac{radar_k}{radar_{MAX}}) \tag{22}$$

Pričom $radar_{MAX}$ má hodnotu 50 a predstavuje teoretickú maximálnu nameranteľnú hodnotu – tento parameter je voliteľný. Normalizácia takto znamená, že čím bližšie sa prekážka nachádza, tým vyššia hodnota normalizácie bude. Tieto vstupy nadobúdajú len hodnoty < 0 ; 1 >.

5.5.3 Navigácia

$$nav = [\frac{x_v - x_{wp}}{2}, \frac{y_v - y_{wp}}{2}] \tag{23}$$

Podľa vzťahu vyššie normalizujeme navigačné dáta. Navigácia by nám mala dávať bod v tvare $[x_{wp}, y_{wp}]$ a $[x_v, y_v]$ získame z IMU. Očakávame, že navigačný bod je maximálne 2 metre od auta, preto delíme touto konštantou obe chyby.

5.5.4 Binárne vstupy

Binárne vstupy môžu nadobúdať len hodnoty -1, 0 alebo 1. Sú štyri:

- Podľa pozície čiary. Ak je auto bližšie k ľavej, výstup je 1, ak k pravej, výstup je -1. (ak je presne v strede, tak 0)
- Podľa najbližšej prekážky. Ak je vľavo, -1, ak vpravo, 1 a ak pred vozidlom, tak 0.
- Podľa rýchlosti v smere k cieľu. Ak auto mieri na cieľ, výstup je 0, inak opačne podľa smeru ktorým by mal ísť
- Rozdiel medzi odporúčaným natáčaním a našim aktuálnym. Ak je odporúčané viac, výstup je -1, ak menej, tak 1.

5.5.5 Metrika

Metrika je aktuálne natáčanie a natáčanie kolies 10 krokov späť. Normalizované podľa povoleného rozsahu natáčania (s_{Max}) – v našom prípade je táto hodnota 0,8.

$$met_k = [\frac{s_{k-1}}{s_{Max}}, \frac{s_{k-10}}{s_{Max}}] \tag{24}$$

5.5.6 Pozícia v rámci pruhu

Vieme ju vyjadriť ako vzdialenosť od stredu pruhu, keďže vieme, že ľavá čiara ($left$) bude mať vzhľadom na vozidlo zápornú hodnotu a pravá ($right$) zase kladnú. Následne ju normujeme podľa šírky (w) tohto pruhu:

$$\begin{aligned} w &= |left| + |right| \\ pos &= \frac{left + right}{w} \end{aligned} \tag{25}$$

Vo väčšine času bude tento vstup nadobúdať nízke hodnoty, ak však vozidlo bude prekračovať čiary, tak vstup sa bude blížiť rýchlo k maximálnym povoleným hodnotám.

5.5.7 Riadenie kolies (výstup)

Výstupom z UNS je riadenie kolies. Tento výstup je prírastkový – znamená, že NS určí zmenu natáčania voči poslednému kroku, nie výsledné natáčanie. Robí tak podľa:

$$\boldsymbol{s_k = s_{k-1} + (s_{NN} * limit)}$$
$$, \; s_k \in < -0{,}8 \; ; 0{,}8 > \tag{26}$$

$\boldsymbol{s_k}$ je natáčanie kolies v aktuálnom kroku, $\boldsymbol{s_{k-1}}$ je natáčanie kolies v minulom kroku a $\boldsymbol{s_{NN}}$ je spomínaná zmena natáčania určená neurónovou sieťou.

Ako sme spomínali normalizáciu pre vstupy, niečo obdobné potrebujeme spraviť aj s výstupom – podľa 1.2.1 poznáme obor hodnôt aktivačnej funkcie **tanh**. Obor hodnôt je pre nás príliš vysoký a je potrebné ho limitovať – na to použijeme *dynamické obmedzenie*.

$$\boldsymbol{x = \frac{v\,[km/h]}{10}}$$
$$\boldsymbol{limit} = \begin{cases} 0{,}1; & \boldsymbol{x} < 1 \\ \dfrac{0{,}1}{x}; & \boldsymbol{x} \geq 1 \end{cases} \tag{27}$$

Podľa vzťahu (27) vieme vyjadriť toto dynamické obmedzenie, $\boldsymbol{v}$ je rýchlosť vozidla. Je inšpirované taktiež realitou, keďže vo vozidle pri vyššej rýchlosti nie je na zmenu smeru potrebné robiť rovnaké zmeny natáčania ako pri pomalšej jazde.

Je dôležité podotknúť, že simulátor beží s periódou vzorkovania 0,05 sekundy. To znamená, že za sekundu je vozidlo schopné urobiť 20 zmien. Ak by sme pri vyššej rýchlosti povolili rovnaké zmeny ako pri stojacom vozidle, vozidlo nie je schopné nájsť zovšeobecňovaciu schopnosť.

6 Neuroevolúcia

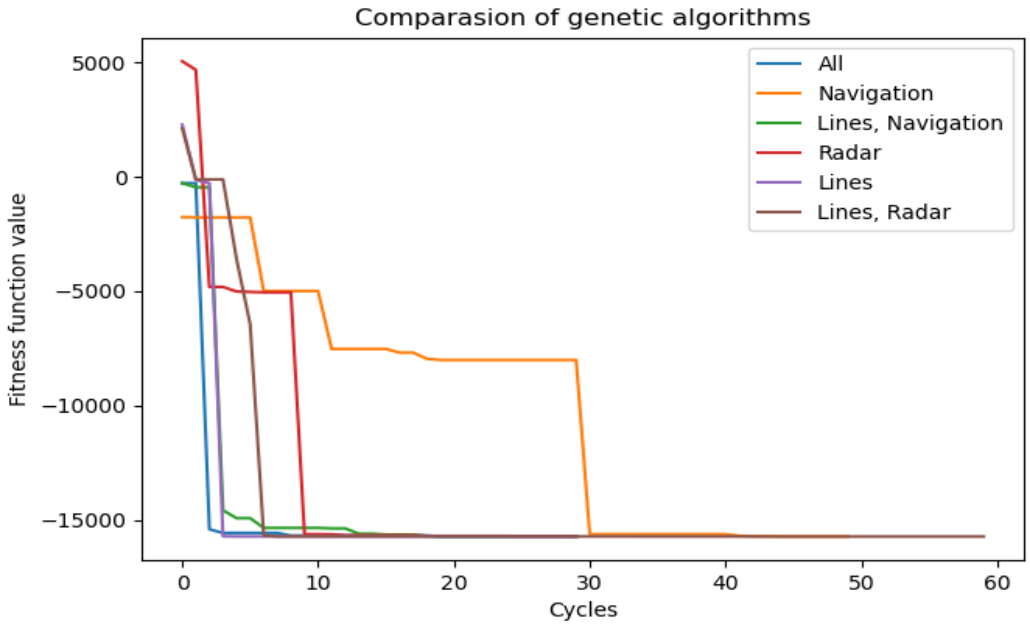
Podľa predošlej kapitoly vieme, že sme navrhli niekoľko možných vstupov pre MLP sieť. Čo však ešte nevieme povedať, je aký majú jednotlivé vstupy vplyv na celkový výsledok – vykonáme preto niekoľko experimentov, ktoré nám napovedia viac. Postupne budeme vypínať jeden, alebo viacero z hlavných vstupov: **detekcia čiar**, **radar** a **navigácia**.

6.1 Trénovanie

Na trénovanie použijeme trénovaciu dráhu (Tabuľka 9), genetický algoritmus (Obrázok 31) bude optimalizovať fitness funkciu (21).

Minimálna fitness	***Počet cyklov***	***Detekcia čiar***	***Radar***	***Navigácia***
-15721	30	áno	nie	nie
-15720	50	nie	áno	nie
-15723	50	nie	nie	áno
-15708	25	áno	nie	áno
-15722	30	áno	áno	áno

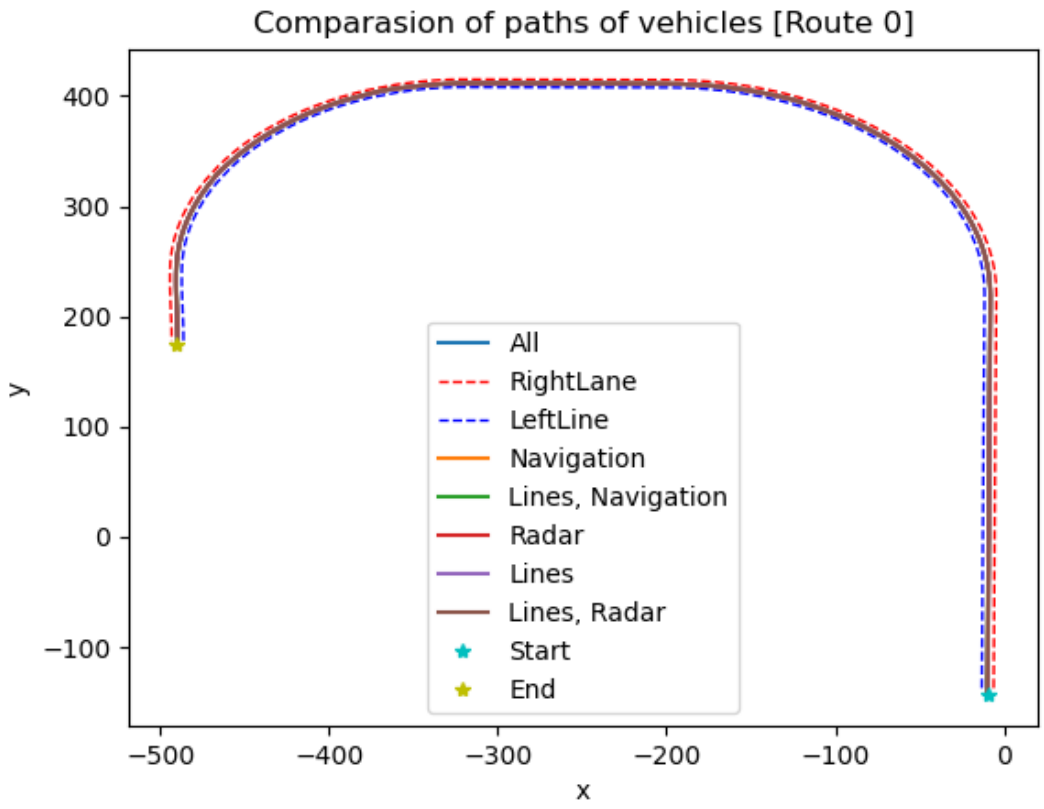
Tabuľka 11: Experimenty



Obrázok 37: Porovnanie optimalizácie genetických algoritmov

Keďže čiastkových cieľov má trénovacia dráha päť, očakávame, že výsledná fitness funkcia jedinca, ktorý zabezpečí prejazd celou dráhou bude nižšia ako -15000. Fitness funkcia bude minimalizovaná podľa ostatných členov – ideálne bez zbytočných prechodov cez čiaru pruhu a taktiež čo najpriamočiarejším pohybom smerom k cieľom.

V Tabuľka 11 vidíme porovnanie experimentov a aj použité vstupy. Ako indikuje Obrázok 37, všetky experimenty boli úspešné – genetický algoritmus dokázal natrénovať MLP sieť. Overiť to môžeme pomocou záznamu dráhy vozidla na Obrázok 38:

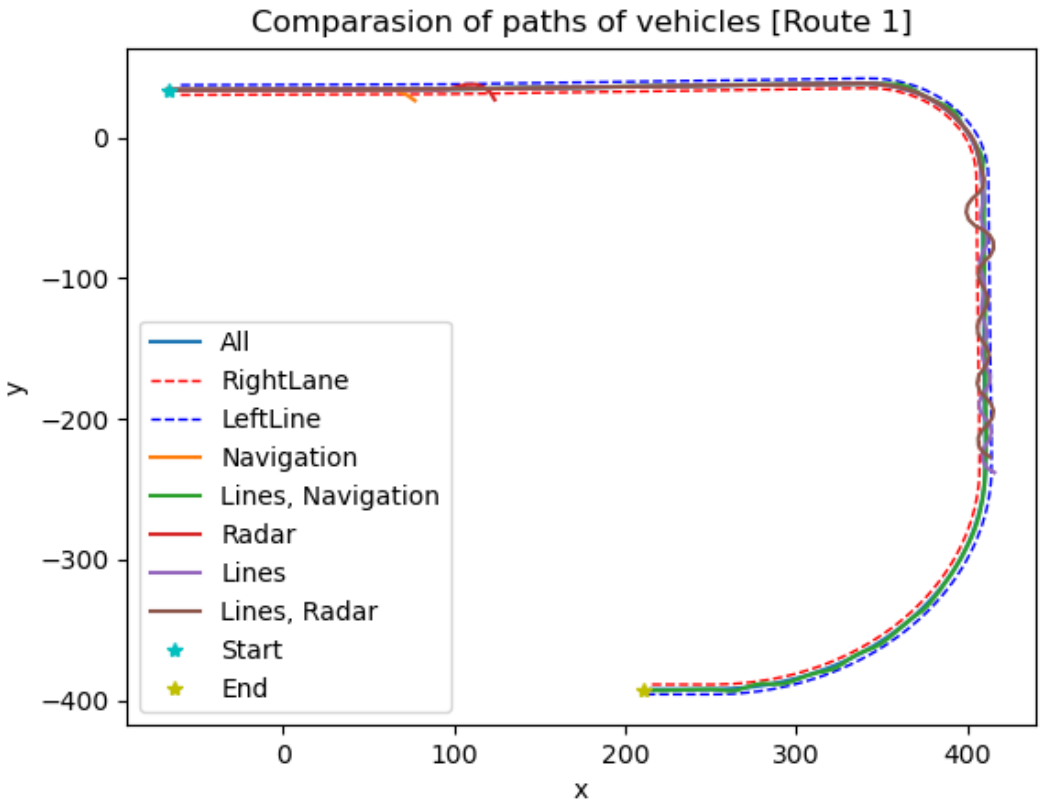


Obrázok 38: Porovnanie jázd vozidla na trénovacej dráhe

Očakávané výsledky potvrdzuje aj obrázok vyššie. Trénovacia dráha však nie je v konečnom dôsledku relevantným porovnaním pre tieto experimenty, keďže pri nej nevieme jasne určiť najlepšiu konfiguráciu – hodnoty fitness funkcií sú veľmi podobné, preto musíme pristúpiť k ďalším porovnaniam. Chceme zistiť, ktorý vstup je dôležitým pri zisku zovšeobecňovacej schopnosti NS, teda ktoré riešenie disponuje najvyššou robustnosťou. Na to využijeme testovacie dráhy – prvá sa nachádza taktiež na diaľničnom úseku mapy, tá druhá je v mestskej časti. Presné body trasy sme spomínali v Tabuľka 9.

6.2 Prvá testovacia dráha

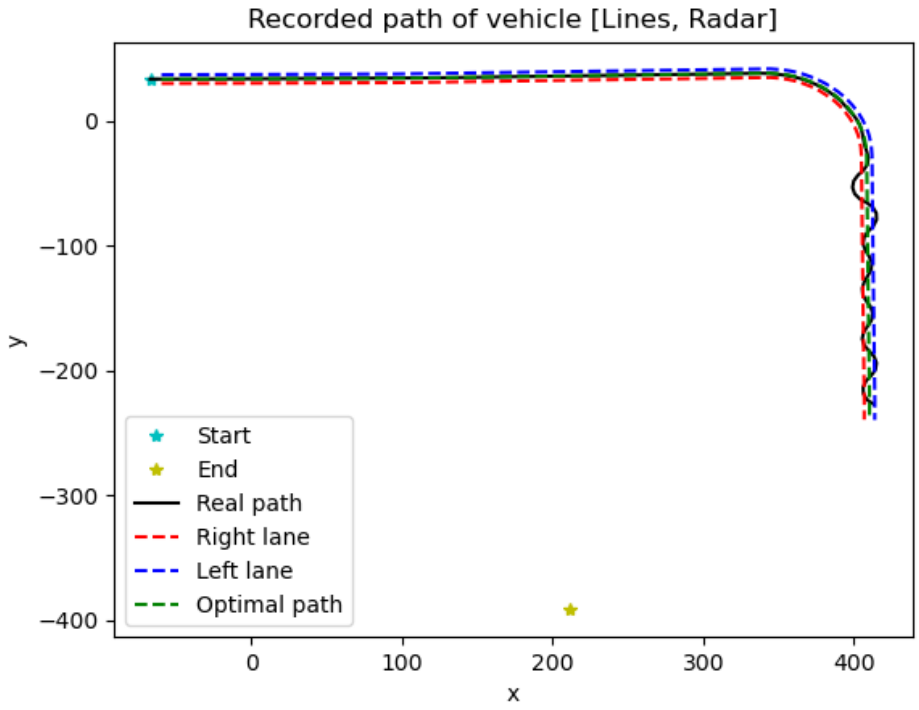
Prvá, konzervatívnejšia testovacia dráha sa nachádza v opačnom smere na diaľničnom úseku. Táto dráha overí, či vozidlo, ktoré doteraz zabáčalo len doprava, dokáže zabáčať aj doprava. Okolie by malo byť podobné, ako tomu bolo pri trénovacej dráhe.



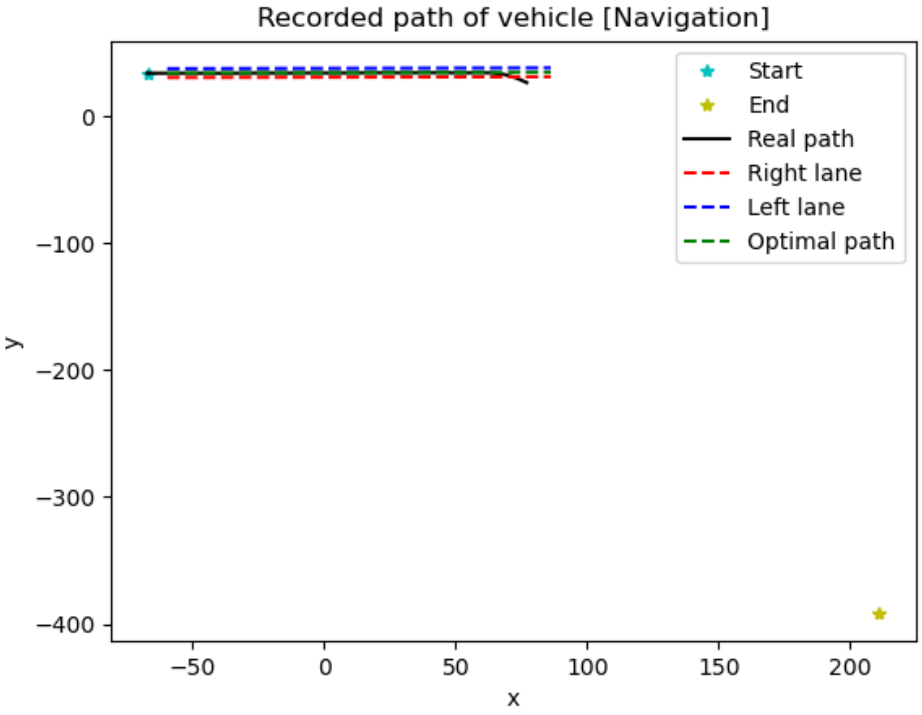
Obrázok 39: Porovnanie jázd vozidla na testovacej dráhe 1

Keď sa však pozrieme na Obrázok 39, kde sme použili základnú testovaciu dráhu, vidíme, že väčšina z navrhnutých riešení nedisponuje dostatočnou robustnosťou.

Pozrieme sa detailne aj na dôvody prečo tomu tak je. Keďže vozidlo v trénovacej dráhe zabáčalo iba doprava, avšak pri tejto dráhe vozidlo odbáča zase vždy doľava. To znamená, že ho vystavujeme neznámej situácii, pri ktorej je bez použitia navigácie stratené. Detailný pohľad na takúto situáciu nám ponúka Obrázok 40. Vozidlo na ňom síce zvládne prechod prvou ľavotočivou zákrutou, bezprostredne z jej východu však začne prechádzať zo strany na stranu a nedokáže sa stabilizovať. Do kontrastu s tým dáme Obrázok 41 – ten zlyhá bezprostredne po štarte. Je to známkou nedostatočných informácii, ktorými vozidlo disponuje o okolitom prostredí.

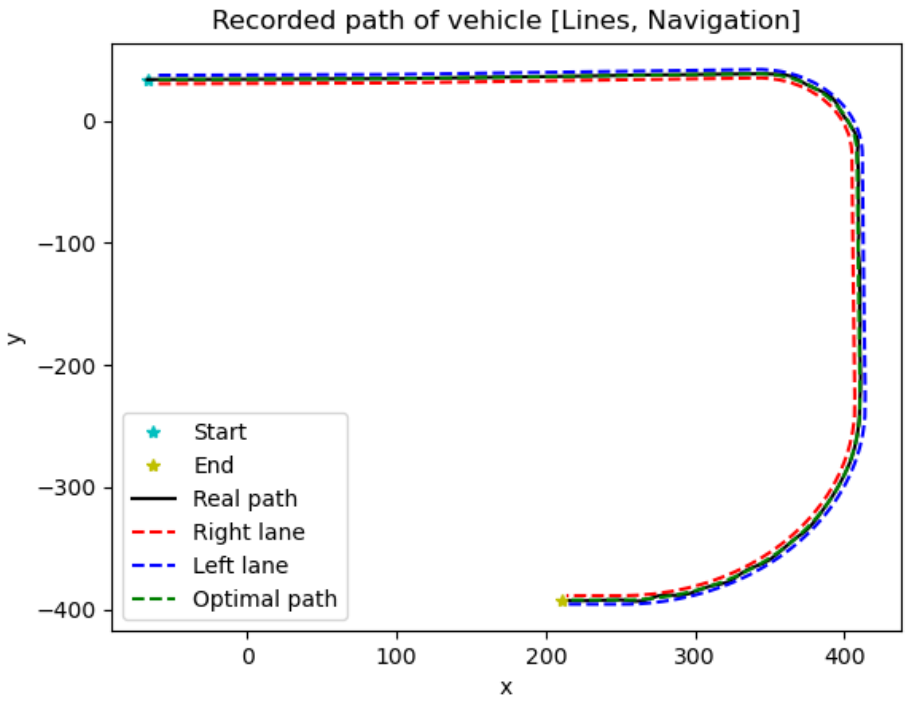


Obrázok 40: Prejazd vozidla bez použitia navigácie testovacou dráhou



Obrázok 41: Prejazd vozidla iba s použitím navigácie testovacou dráhou

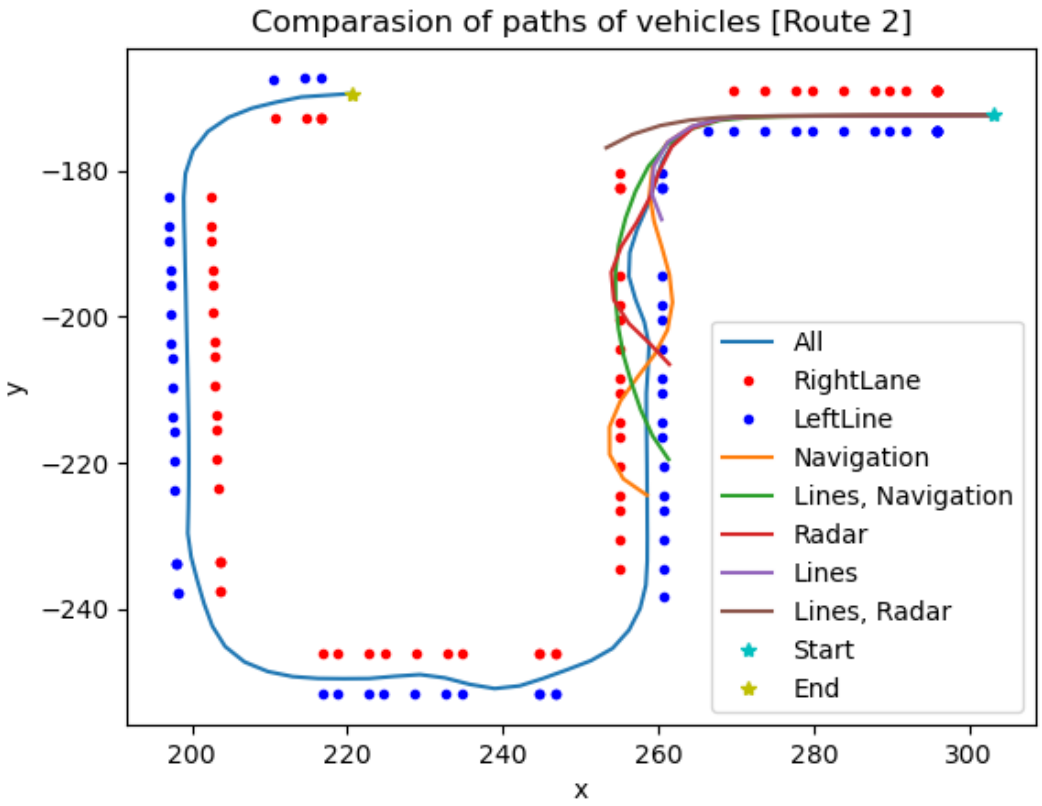
Z predošlých zistení predpokladajme, že využitie fúzie **detekcie čiar** a **navigácie** zabezpečí žiadanú robustnosť pre túto dráhu. Tieto dva hlavné vstupy sú tak minimálne potrebné na autonómne riadenie natáčania vozidla bez okolitých objektov. Takýto prejazd vidíme na Obrázok 42. Zaradenie radaru však nezhorší kvalitu riadenia pri prvej testovacej trase a zároveň je priam potrebný pre ďalšie testovacie scenáre na detekciu potenciálnych nečakaných kolízii s inými, doteraz neznámymi objektami.



Obrázok 42: Prejazd vozidla testovacou dráhou s fúziou detekcie čiar a navigácie

6.3 Druhá testovacia dráha

Druhá dráha je v mestskej časti. Je omnoho zložitejšia v porovnaní s prvou a trénovacou. V križovatkách tu nie je možné detegovať čiary a taktiež sú tu omnoho ostrejšie zákruty (do oboch smerov). Pri tomto experimente je potrebné zmeniť aj žiadanú rýchlosť vozidla – bezpečný prejazd takouto dráhou nie je možný pri rýchlosti 50 km/h a tak ju znížime na 35km/h.

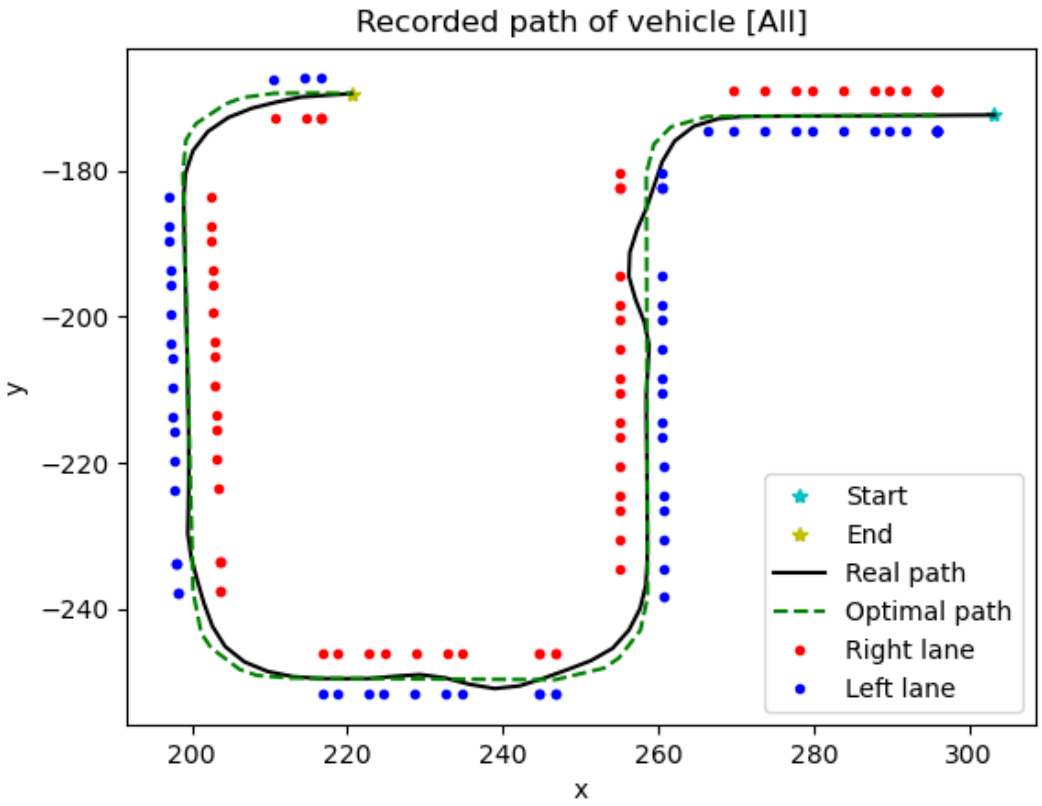


Obrázok 43: Porovnanie jázd vozidla na testovacej dráhe 2

V prípade zložitejšej dráhy prichádzame k výsledku, že iba **fúzia všetkých vstupov** vedie k dostatočne robustnému riadeniu natáčania kolies pomocou MLP siete, ktoré je teoreticky aplikovateľné na ktorúkoľvek dráhu v rámci mapy. Z Obrázok 43 vidíme aj presné miesta, kde boli čiary nedetekovateľné (popri prejazdu vozidiel nie sú viditeľné body indikujúce čiary) – v tomto prípade si muselo vozidlo vystačiť s ostatnými vstupmi, keďže tieto boli vypnuté. Ich vplyv pri tomto experimente je dobre viditeľný na Obrázok 44 – okamžite po ich opätovnej aktivácii totiž vozidlo začalo korigovať svoju pozíciu tak, aby skutočne čiarou neprešlo.

Taktiež sa nám podarila overiť hypotéza, že riadenie **bez radaru** pri prostredí s externými prekážkami (semafory, objekty vedľa cesty, chodci, ...) je nemožné.

Funkcionalitu **navigačného** systému vidíme pri riešeniach, ktoré ho nevyužívajú. Na rozdiel od predošlých experimentov sa totiž tento krát nestačí vozidlu držať v pruhu, ale bude nutné ho niekoľko krát prekročiť. Bez toho, aby navigačný systém explicitne povedal, kam má vozidlo smerovať, najmä pri križovatkách, sa stane neriaditeľným – na Obrázok 43 sú tieto riešenia zobrazené červenou, fialovou a hnedou.



Obrázok 44: Prejazd testovacou dráhou 2 s použitím fúzie všetkých vstupov

Detailný pohľad na prejazd vozidla druhou testovacou dráhou s použitím všetkých vstupov (Obrázok 44) nám ukazuje prvky autonómie vozidla, čo je výborné zistenie. Prvým z nich je, že vozidlo zabáča skôr, ako dostáva „pokyn“ na zabočenie od navigačného systému, čo znamená, že našlo vhodnú kombináciu váh, ktorými zabezpečilo fúziu vstupov tak, že dokáže vhodne predikovať nasledujúce natáčania kolies.

Záver

Prácu sme už v predošlých kapitolách rozdelili na štyri časti – *teoretický úvod do neuroevolúcie, nájdenie vhodného simulátora, vnem prostredia* a *návrh robustného riadenia natáčania kolies pomocou MLP siete*.

Teoretickým úvodom boli pre nás prvé dve kapitoly, v ktorých sme sa venovali neurónovým sieťam a evolučným algoritmom. O tieto teoretické poznatky sme sa opreli pri skoro všetkých ďalších častiach práce.

Vhodným simulátorom pre naše použitie bol open-source simulátor **CARLA 0.9.12**. Disponoval reálnou fyzikou, dobrým prístupom k riadeniu vozidla a taktiež ďalším častiam simulácie. Jeho nastaveniam a možnostiam, ktoré ponúka sme sa venovali v kapitole 3.

Po výbere simulátora a preskúmania možností, ktoré simulátor ponúka sme potrebovali nájsť vhodné vnímanie prostredia, na základe ktorého sa bude môcť autonómne vozidlo neskôr pohybovať. Tejto problematike sme sa venovali v kapitole 4 a ako hlavný vnem sme vybrali detekciu čiar. Tú sme realizovali pomocou dvoch rozdielnych hlbokých segmentačných neurónových sietí, pričom sme pri ich záverečnom porovnaní zistili, že vhodnejšou pre našu aplikáciu bude **MobileNetV3 Small**, ktorá veľmi rýchlo a presne dokázala segmentovať ako ľavú tak aj pravú čiaru. Testovanie segmentačnej siete prebiehalo na validačnom datasete a zároveň aj v real-time, kde sme overovali, či sieť dokáže aj pri zhoršených podmienkach rýchlo a presne segmentovať. Výsledky boli spoľahlivé ako v mestskej, tak diaľničnej časti mapy. Následne sme tieto čiary interpretovali pomocou kubickej polynomiálnej aproximácie, pomocou ktorej sme tak nakoniec vedeli určiť približný tvar detegovanej čiary a taktiež neskôr vhodne interpretovať tento vnem pre riadiacu MLP neurónovú sieť.

Posledným, no najdôležitejším cieľom bolo navrhnúť riadenie natáčania kolies a v prípade úspechu na trénovacej dráhe overiť robustnosť takéhoto riadiaceho prístupu. Aby sme zistili, ktorý vstup má aký vplyv na celkový výsledok, vytvorili sme viacero riadiacich neurónových sietí v závislosti od použitých vstupov. Overili sme, že v diaľničnom úseku pri trénovacej dráhe nie je potrebné použitie navigácie – tým sme overili, že vozidlo sa naučilo jazdiť na základe čiar a ostatných minoritných vstupov.

Testovacie dráhy mali za úlohu odhaliť slabiny navrhnutých riadení a oddeliť nerobustné riešenia od tých robustných. Prvá, jednoduchšia dráha vylúčila kandidátov, ktorí nevyužívali kombináciu detekcie čiar a navigácie. Dôvod nefunkčnosti týchto

riadiacich modelov sme opísali v 6.2. Keďže aj touto testovacou dráhou dokázalo bezproblémovo prejsť viacero riešení, pristúpili sme k zložitejšej alternatíve testovacej dráhy. Tá bola v mestskej časti, kde sme narazili na viacero situácii, s ktorými sa vozidlo pri trénovaní nestretlo – či už to boli chýbajúce čiary, alebo prudšie zákruty. V takomto prípade nám vyšiel očakávaný výsledok – len **fúzia všetkých vstupov** zabezpečí dostatočne robustné riadenie natáčania kolies na ktoromkoľvek mieste v mape.

V celkovom merítku je práve fúzia vstupov to, čo je dôležité aj pre ďalší vývoj autonómneho riadenia vozidiel. V reálnom systéme sa totiž častokrát stretneme s tým, že niektoré vstupy budú buď nedostupné, či nespoľahlivé. Takýmto prípadom pre našu detekciu čiar môže byť napríklad zmena počasia – sneh, či množstvo vody na vozovke a čiary budú len ťažko vnímateľné.

Možnosťou zlepšenia robustnosti je trénovanie na viacerých scenároch súčasne. Tento prístup sa používa celkom bežne pri návrhu robustných riadení. V našom prípade je veľký úspech, že sa nám podarilo navrhnúť taký autonómny systém, ktorý na základe jednej trénovacej dráhy našiel dobré riešenie na vyriešenie diametrálnej a zložitejšej dráhy.

Je dôležité povedať, že existuje ešte množstvo ďalších vstupov pre NS, ktoré by mohli priniesť zlepšenie kvality riadenia – využitie kamier na iných častiach vozidla, či 360° kamery, lidarov, alebo hĺbkové kamery, ktoré ponúkajú metódy, ktorými vieme presne určiť vzdialenosti k jednotlivým objektom podľa hĺbky obrazu. V prípade použitia okoloidúcej premávky by vozidlo mohlo využiť tieto autá taktiež na orientáciu – ako sme hovorili, spôsobov, ktoré môžu pomôcť je viacero.

K plnej autonómii vozidla je však určite potrebné riadiť aj jeho rýchlosť. V tomto prípade sme počítali len s nejakou pevnou rýchlosťou, ktorou vozidlo pôjde – ak by sme rýchlosť pri druhom testovacom experimente neznížili, auto by nebolo schopné touto dráhou prejsť. Pri plnej autonómii by však auto v prípade ostrej zákruty, či hroziacej kolízie znížilo rýchlosť samo.

Literatúra a zdroje

[1] KVASNIČKA, V. – BEŇUŠKOVÁ, Ľ. – POSPÍCHAL, J. – FARKAŠ, I. – TIŇO, P. – KRÁĽ, A. 1997. *Úvod do teórie neurónových sietí*. Bratislava, IRIS, 1997. 285 s. ISBN 80-887-77830-1

[2] ERB, R.J. *Introduction to Backpropagation Neural Network Computation*. Pharm Res 10, 165–170 (1993)

[3] MACH, M. 2009. *Evolučné algoritmy: prvky a princípy.* Košice: TUKE, 2009. 250s. ISBN 978-80-8086-123-0.

[4] SEKAJ, I. 2005. *Evolučné výpočty a ich využitie v praxi*. Bratislava, IRIS, 2005. ISBN: 80-89018-87-4.

[5] SEKAJ, I. 2021. *TOOLBOX v4 - GENETICKÉ ALGORITMY*, používateľká príručka

[6] SHER, GENE I. 2013. *Handbook of Neuroevolution Through Erlan*g, New York: Springer-Verlag 2013. ISBN 978-1-4614-4463-3. Chapter 4

[7] THEERS, M. 2021. *Algorithms for automated driving. Chapter 1: LANE DETECTION*. Dostupné online: https://thomasfermi.github.io/Algorithms-for-Automated-Driving/LaneDetection/LaneDetectionOverview.html

[8] FastAI: Vision – Image Segmentation, 2021. Dostupné online: https://docs.fast.ai/tutorial.vision.html#Segmentation

[9] THEERS, M. 2021. *Lane Detection for Carla Driving Simulator: DATASET*. Dostupné online: *https://www.kaggle.com/datasets/thomasfermi/lane-detection-for-carla-driving-simulator*

[10] VASILEV I., 2019. *Advanced Deep Learning with Python: Design and implement advanced next-generation AI solutions using TensorFlow and PyTorch*, 2019. ISBN: 978-1-78995-617-7

[11] Albumentations.ai: Mask augmentation for segmentation, 2022. Dostupné online: https://albumentations.ai/docs/getting_started/mask_augmentation/

[12] KAJAN S., 2021. *Detekcia objektov hlbokým učením*, prednáška č.8 z predmetu Hlboké neurónové siete

[13] *Understanding semantic segmentation with UNET*. 2019. Dostupné online: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

[14] RONNEBERGER O., FISCHER P., BROX T., 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv: 1505.04597, 2015. Dostupné online: https://arxiv.org/pdf/1505.04597.pdf

[15] ZHOU Z., 2018. *UNet++ A Nested U-Net Architecture for Medical Image Segmentation*. arXiv: 1807.10165, 2018. Dostupné online: https://arxiv.org/pdf/1807.10165.pdf

[16] LAMBA H., 2019. *U-Nets with ResNet Encoders and cross connections.* TowardsDataScience 2019. Dostupné online: https://towardsdatascience.com/u-nets-with-resnet-encoders-and-cross-connections-d8ba94125a2c

[17] JAIN V., 2019. *Everything you need to know about MobileNetV3*, TowardsDataScience 2019. Dostupné online: https://towardsdatascience.com/everything-you-need-to-know-about-mobilenetv3-and-its-comparison-with-previous-versions-a5d5e5a6eeaa

[18] HOWARD A., SANDLER M., CHU G., CHEN L.C., CHEN B., TAN M., WANG W., ZHU Y., PANG R., VASUDEVAN V., LE Q., ADAM H., 2019. *Searching for MobileNetV3*, arXiv: 1905.02244, 2019. Dostupné online: https://arxiv.org/pdf/1905.02244.pdf

[19] VAN GANSBEKE W., DE BRABANDERE B., NEVEN D., PROESMANS M., VAN GOOL L., 2019. *End-to-end Lane Detection through Differentiable Least-Squares Fitting*, arXiv: 1902.00293, 2019. Dostupné online: https://arxiv.org/pdf/1902.00293.pdf

Použité súčasti

[1] Python 3.8, https://www.python.org/downloads/release/python-380/

[2] PyCharm, https://www.jetbrains.com/pycharm/

[3] Conda environment, https://docs.conda.io/en/latest/

[4] CARLA 0.9.12, https://carla.org/2021/08/02/release-0.9.12/

[5] NumPy, https://numpy.org/

[6] Albumentations, https://albumentations.ai/

[7] PIL, https://pillow.readthedocs.io/en/stable/

[8] PyTorch, https://pytorch.org/

[9] TorchVision for PyTorch, https://pytorch.org/vision/stable/index.html

[10] FastAI for PyTorch, https://docs.fast.ai/

[11] OpenCV, https://opencv.org/

[12] MatPlotLib, https://matplotlib.org/

[13] PyQt5, https://pypi.org/project/PyQt5/

[14] TensorFlow 2.7.0, https://www.tensorflow.org/

Prílohy

Príloha A: Dokumentácia k programovej realizácii

Príloha B: Spustiteľný program v jazyku Python

Príloha C: Konfiguračný súbor Config.ini

```
[CARLA]
map = Town04
weather = carla.WeatherParameters.ClearNoon
ts = 0.05
sync = True

[Camera]
width = 1024
height = 512

[Sensors]
radarsensor = True
linedetectorcamera = True
defaultcamera = True
segmentationcamera = False
collisionsensor = True
obstacledetector = False
lidarsensor = False
laneinvasiondetector = True

[NnInputs]
linedetect = True
radar = True
agent = True
metrics = True
binaryknowledge = True
navigation = True

[NE]
ninput = 18
nhiddenlayers = 2
nhidden = 10
noutput = 1
popsize = 20
numcycles = 5
best = 0.15
work1 = 0.35
work2 = 0.35
base = results/
rev = 2
```

Príloha D: Videoukážky spomínaných experimentov