VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMOBILNÍHO A DOPRAVNÍHO INŽENÝRSTVÍ

INSTITUTE OF AUTOMOTIVE ENGINEERING

VÝPOČTOVÝ MODEL OKOLÍ AUTONOMNÍHO VOZIDLA

COMPUTATIONAL MODEL OF THE ENVIRONMENT OF AN AUTONOMOUS VEHICLE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Radek Doležel

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Pavel Kučera, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automobilního a dopravního inženýrství
Student:	**Bc. Radek Doležel**
Studijní program:	Automobilní a dopravní inženýrství
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Pavel Kučera, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Výpočtový model okolí autonomního vozidla

Stručná charakteristika problematiky úkolu:

Naprogramování výpočtového modelu okolí autonomního vozidla a objektů v něm se pohybujících. Využít je možné softwaru Simulink nebo i další programovací jazyky C, C++, atd. Model okolí by měl popisovat chování autonomního vozidla a objektů okolo se pohybujících, dále musí být model schopen předpovídat trajektorii objektů okolo vozidla.

Cíle diplomové práce:

Vytvoření virtuálního okolí vozidla.
Implementace snímačů.
Predikce trajektorie jednotlivých pohybujících se objektů.
Ověření funkčnosti.

Seznam doporučené literatury:

DIETSCHE, Karl-Heinz a Konrad REIF, 2022. Automotive Handbook. 11. Karlsruhe: Robert Bosch. ISBN 978-1-119-91190-6.

CHOLLET, François a Rudolf PECINOVSKÝ, 2019. Deep learning v jazyku Python: Knihovny Keras, Tensorflow. 1. Praha: Grada Publishing. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

FIJALKOWSKI, Bogdan Thaddeus. Automotive Mechatronics: Operational and Practical Issues. 2. vydání. Dordrecht: Springer, 2009. ISBN 978-94-007-1182-2.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

prof. Ing. Josef Štětina, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Cílem práce je vytvořit funkční výpočetní model predikce pohybu vozidel na základě rešerše senzorů a jejich umístění na vozidle, neuronových sítí pro počítačové vidění, datasetů pro učení sítí a programů pro vytvoření simulace a virtuálního okolí. V práci je popsán postup vytvoření virtuálního okolí vozidla a simulace. Dále jsou vytvořeny návrhy umístění senzorů včetně jejich parametrů. Následně je představen naprogramovaný algoritmus predikce trajektorie vozidel včetně učení a implementace neuronové sítě. Na závěr jsou prezentovány výsledky vytvořeného algoritmu.

KLÍČOVÁ SLOVA

Autonomní řízení, kamera, LiDAR, neuronová síť, počítačové vidění, simulace, predikce trajektorie

ABSTRACT

The aim of this thesis is to develop a functional computational model for vehicle motion prediction based on a search of sensors and their locations on the vehicle, neural networks for computer vision, datasets for network learning, and programs for creating simulations and virtual environments. The paper describes the process of creating the vehicle virtual environment and simulation. In addition, sensor placement designs including their parameters are developed. Subsequently, the programmed vehicle trajectory prediction algorithm including learning and neural network implementation is presented. Finally, the results of the developed algorithm are presented.

KEYWORDS

Autonomous driving, camera, LiDAR, neural network, computer vision, simulation, trajectory prediction

BIBLIOGRAFICKÁ CITACE

DOLEŽEL, R. *Výpočtový model okolí autonomního vozidla*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automobilního a dopravního inženýrství. Vedoucí diplomové práce Pavel Kučera. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149139.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Pavel Kučera, Ph.D. a s použitím informačních zdrojů uvedených v seznamu.

V Brně dne 26. května 2023 ……..……..

Bc. Radek Doležel

PODĚKOVÁNÍ

Tímto děkuji doc. Ing. Pavlu Kučerovi, Ph.D. za cenné připomínky a rady při vypracovávání diplomové práce a za vstřícný přístup při konzultacích. Dále bych chtěl poděkovat svým rodičům za podporu při studiu.

OBSAH

Úvod

V dnešní době jsou autonomní vozidla jednou z nejperspektivnějších oblastí výzkumu a vývoje v automobilovém průmyslu. Tyto vozidla jsou vybavena nejrůznějšími senzory, které slouží k tomu, aby dokázala bezpečně a efektivně komunikovat s okolním prostředím a přizpůsobovat se podmínkám na silnici. Avšak testování a ověřování funkčnosti autonomních vozidel v reálných podmínkách je velmi náročné a drahé. Proto se v poslední době stále více využívají simulace ve virtuálním prostředí, které umožňují simulovat různé situace, testovat reakce autonomních vozidel a správnost implementace snímačů v bezpečném prostředí.

Prvním cílem diplomové práce je vytvořit virtuální prostředí, které umožní simulovat reálné okolí autonomního vozidla. Pro tento cíl bude využitý program RoadRunner. Dále se bude práce zabývat implementací kamery s LiDARem, které budou sloužit ke sběru dat o okolí vozidla pro následnou predikci trajektorie pohybujících se vozidel, které se mohou v okolí vozidla vyskytovat.

Za tímto účelem je první kapitole vytvořeno stručné shrnutí používaných senzorů a možnosti jejich implementace do vozidla. Dále byly poskytnuty nezbytné podklady pro zpracování počítačového vidění a krátké představení programů pro vytvoření simulace. Ve druhé kapitole je popsáno vytváření virtuálního okolí vozidla. Ve třetí kapitole je představena implementace kamery s LiDARem na vozidlo včetně popsání parametrů senzorů.

Následujícím cílem diplomové práce je vytvořit výpočtový model okolí autonomního vozidla, který bude schopen predikovat trajektorie pohybujících se objektů v jeho okolí. Samotný algoritmus je popsán ve čtvrté kapitole včetně učení neuronové sítě a její vložení do simulace.

Konečným cílem práce je ověřit funkčnost vytvořeného výpočtového modelu, a to zejména v oblasti schopnosti detekce a predikce trajektorií objektů v okolí vozidla. Proto jsou v páté kapitole představeny výsledky získané simulací a na nich popsány vyskytující se odchylky predikce.

1 SENZORY, ZPRACOVÁNÍ OBRAZU A PROGRAMY PRO VYTVOŘENÍ SIMULACE

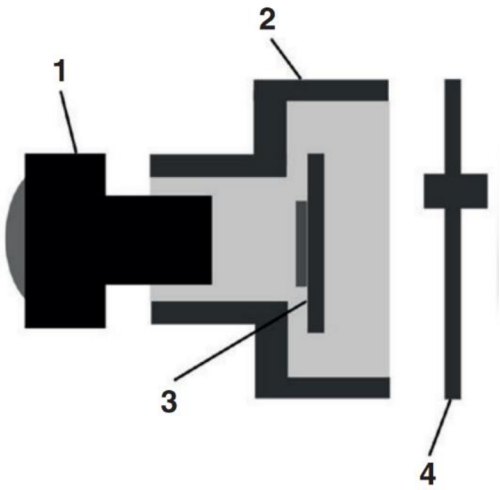
Proto aby mohla být vytvořena predikce pohybujících se objektů v okolí autonomního vozidla, je zapotřebí nejprve zjistit, zdali se v okolí daný objekt nachází. K tomu stroje využívají senzory, které nahrazují lidské smysly.

1.1 SENZORY

Jedním z nejvýznamnějších senzorů je kamera, která umožňuje stroji vidět okolí. Pomocí kamery je možné zjistit, jaké objekty se v okolí nachází. Pro získání vzdálenosti detekovaného objektu je možné využít více snímačů. Pro velké vzdálenosti se využívá radar nebo LiDAR. Na krátkou vzdálenost je možné použít stereo kamery a ultrazvukové snímače. V této práci je kvůli potřebě sledování okolí a nutnosti zjištění vzdálenosti využita kombinace kamery, pomocí které se získá informace o objektech v okolí, a LiDARu, pomocí kterého bude vyhledána jejich vzdálenost.

1.1.1 KAMERA

Kamera je sestavená z objektivu, který se nachází na pozici 1 v *Obr. 1* a optického senzoru na pozici 3. Obě zmíněné součásti jsou uloženy v pouzdře (pozice 2) se zadním krytem (pozice 4), pomocí kterého je nejčastěji zrealizován výstup informací. Objektiv je soustava čoček, která mění vstupní obraz podle požadavku na rozměry obrazového snímače v závislosti na zorném úhlu kamery. Optický senzor je dnes nejčastěji vyráběn technologií CMOS. Rozlišení optického senzoru je dáno počtem fotodiod. V optickém senzoru jsou fotodiody uspořádány do pravoúhlé mřížky. [1] Kamera pouze vytváří digitální obraz, který je možné při dnešních výpočetních výkonech zpracovat neuronovou sítí, aby bylo zjištěno, zda se v okolí nalézá vyhledávaný objekt.



Obr. 1 Jednoduché schéma kamery [1]

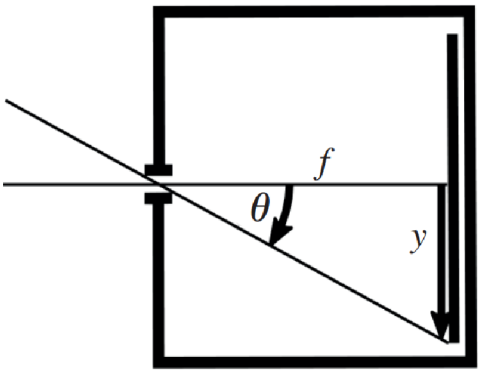
ZORNÉ POLE

Zorné pole je maximální úhel, pod kterým lze daný objekt zachytit a reprodukovat. Objektivy jsou navrženy tak, aby byly schopny zachytit a reprodukovat ostré obrazy do tohoto maximálního úhlu. Při překročení tohoto úhlu dochází k výraznému snížení ostrosti a jasu, a obraz již není možné reprodukovat. V případě fotoaparátu jsou zachycovány pouze oblasti, které spadají do tohoto maximálního úhlu a zároveň do velikosti snímače. [1]

Velikost zorného pole je závislá na velikosti optického senzoru a ohniskové vzdálenosti. Obrazové snímače jsou obdélníkové a jejich velikost se liší v horizontálním a vertikálním směru, což má vliv na zorné pole v jednotlivých směrech. Poloviční úhel zorného pole je možné získat pomocí rovnice (1), protože reálné objektivy s čočkami jsou navrženy tak, aby poskytovaly stejnou konverzi zorného pole na rozměry obrazu jako model kamery s dírkovou komorou. [1]

$$\theta = tan^{-1}\left(\frac{y}{f}\right) \tag{1}$$

Kde θ znázorňuje poloviční úhel zorného pole, y je poloviční výška nebo šířka obrazového senzoru a f je ohnisková vzdálenost. Všechny uvedené veličiny jsou znázorněny na *Obr. 2*.



Obr. 2 Model kamery s dírkovou komorou [1]

1.1.2 LIDAR

LiDAR (Light Detection and Ranging) je zařízení sloužící k měření vzdálenosti pomocí laserových paprsků v infračervené oblasti. Paprsek vyslaný vysílacím modulem se odrazí od objektu a přijímacím modulem je detekován. Polovina doby letu paprsku udává vzdálenost mezi senzorem a objektem, při známé rychlosti světla. LiDAR se používá nejen k měření vzdáleností, ale také k vytváření obrazu okolí. Pro získání 3D obrazu je třeba mít dostatečné rozlišení v požadovaném zorném poli, čehož lze docílit zvýšením počtu vysílacích nebo přijímacích prvků, nebo pomocí skeneru pro vychýlení světelných paprsků. Rozlišení je nižší než u kamery. [1]

TŘÍDY LASERŮ

K členění laserů dle bezpečnosti slouží norma ČSN EN 60825-1 [2]. Norma klasifikuje lasery do 6 tříd. Klasifikace laserů pomáhá uživatelům posoudit míru nebezpečí a určit nutná

opatření. Uvádí pouze riziko pro kůži a oči. Různé zařízení v rámci jedné třídy mohou mít však odlišné nebezpečné zóny. Použití ochranných opatření, včetně krytů, může výrazně snížit míru rizika. [2]

Z důvodu umístění laseru do venkovních prostor, kde je v kontaktu s lidmi, je vhodné se krátce zabývat bezpečností záření laseru. Pro tuto aplikaci jsou níže zmíněny pouze dvě nejbezpečnější třídy. Pro ostatní třídy je třeba využívat ochranné pomůcky pro ozáření delší než 0,25 s. [2]

TŘÍDA 1

Tato třída zahrnuje laserová zařízení, která jsou bezpečná během používání, včetně dlouhodobého přímého sledování, i v případě sledování pomocí optických pomůcek. Třída zahrnuje i lasery s vysokým výkonem, které jsou zcela zakrytovány, aby se předešlo přístupu k potenciálně nebezpečnému záření. [2]

TŘÍDA 1M

Tato třída zahrnuje bezpečná laserová zařízení, u kterých nenastane riziko při přímém sledování nechráněnýma očima. Pokud se sledování provádí pomocí optických pomůcek, jako jsou dalekohledy s průměrem větším, než je stanovená podmínka, může dojít k překročení maximální přípustné expozice a následnému poškození oka. [2]

1.1.3 GPS

GPS je síť družic kroužících kolem Země na šesti samostatných oběžných drahách. Tyto družice vysílají specializované signály na frekvenci 1,57542 GHz 50krát za sekundu, které nesou údaje o identifikaci, čase a poloze. Každá družice obsahuje dvoje atomové hodiny z cesia a rubidia zajišťující vysoce přesné měření času. [1]

K výpočtu polohy přijímače se používá trilaterace. Tato metoda vypočítá polohu přijímače pomocí signálů z nejméně tří družic, která umožňuje určit zeměpisnou délku a šířku. K vypočítání nadmořské výšky jsou zapotřebí signály ze čtyř družic. Přesnost systému GPS závisí na úhlu družic vůči přijímači, s většími úhly se zvyšuje přesnost. V hustě osídlených městských oblastech s kovovými budovami může být určování polohy nepřesné z důvodu odrazu signálu. GPS také umožňuje zjišťovat směr jízdy pomocí analýzy rozdílů v přijímací frekvenci v důsledku Dopplerova jevu. [1]

1.1.4 INERCIÁLNÍ MĚŘÍCÍ JEDNOTKA

Inerciální měřící jednotka, zkráceně IMU, je kombinace akcelerometru, gyroskopu a někdy také magnetometru. Akcelerometr je senzor využívaný k měření setrvačného zrychlení. Existují různé typy, jako například mechanické, křemenné a MEMS akcelerometry. Akcelerometr MEMS je hmota zavěšená na pružině podél měřené osy. Při působení lineárního zrychlení na inertní hmotu se vychyluje jedním směrem a výchylka je úměrná zrychlení. Velikost zrychlení je zjištěná pomocí vzorce (2). IMU obvykle měří zrychlení ve 3 osách. [1]

$$F = m \cdot a = c \cdot x \tag{2}$$

Kde F je síla, m je hmotnost hmoty, a získávané zrychlení, c je tuhost pružiny a x je stlačení pružiny.

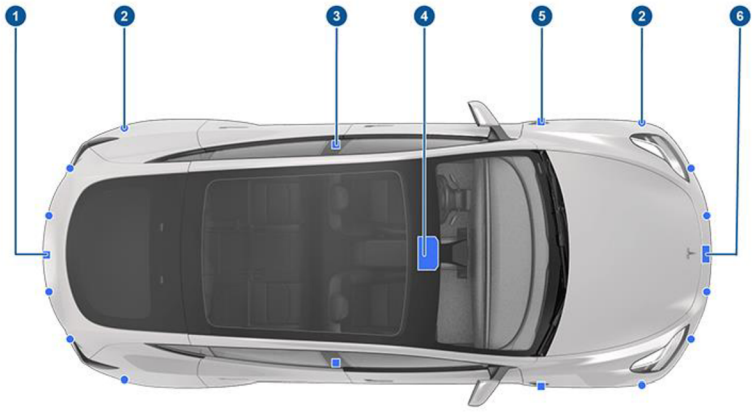
Úhlová rychlost je měřena pomocí gyroskopu. Existují dva základní typy gyroskopů: mechanické a elektronické. Mechanický gyroskop má rotační těleso uložené v kloubovém závěsu, které se otáčí v reakci na vstupní síly. Elektronický gyroskop využívá technologii MEMS a je tvořen křemíkovým čipem, který obsahuje mikro mechanické struktury pro měření úhlové rychlosti rotace. Elektronické gyroskopy jsou menší a lehčí než mechanické gyroskopy. [1]

1.2 UMÍSTĚNÍ SENZORŮ NA VOZIDLO

Pro správné fungování algoritmu je nezbytné mít senzory na vozidle rozmístěny v optimálních pozicích. Bude tak zajištěno ideální zachycení dat z okolí vozidla. Správně umístěné senzory společně s vhodně zvolenými parametry jsou nezbytné pro kvalitní výstupy algoritmu. Nevhodné umístění senzorů může vést k nespolehlivým výstupům.

1.2.1 UMÍSTĚNÍ V KAROSERII

Pro příklad umístění senzorů do karoserie byla vybrána Tesla model Y, která využívá 8 kamer pro detekci okolí. [3] Dění před vozidlem sledují 3 kamery, zabudované nad vnitřním zpětným zrcátkem na pozici 4. Boční kamery jsou dvojího typu a jsou umístěny symetricky. První pár bočních kamer sleduje strany směrem dopředu a je umístěn na pozici 3 v B sloupcích. Druhé boční kamery imitují pohled do zrcátka a jsou umístěny na pozici 5 před předními dveřmi pod zpětným zrcátkem. Poslední kamera směřuje dozadu a je umístěna nad SPZ na pozici 1. Tesla ovšem využívá pro detekci okolí pouze kamery a případně čelní radar.



Obr. 3 Umístění snímačů Tesla Model Y [3]

1.2.2 UMÍSTĚNÍ NA STŘECHU

Další možností je umístění většiny senzorů na střechu. Umístění senzorů na střechu zvolilo více vývojových týmů, zobrazených v *Tab. 1*. Tabulka zároveň obsahuje umístění a počet

LiDARů. [4] Toto umístění slouží například pro testování a ověřování algoritmů v reálnem provozu. Výhodou tohoto umístění je implementace na vozidla bez větších zásahů do karoserie. Senzory umístěny na střechu mají větší dohled a lepší výhled do okolí.

Tab. 1 Umístění LiDARů různých vývojových týmů [4]

Tým	Typ LiDARu	Počet	Rozmístění
Ford	Velodyne-16	4	2 na každé straně střechy
Cruise	Velodyne-16	5	2 na každé straně s 1 uprostřed střechy
Waymo	Velodyne-64	1	1 v centru střechy
Uber	Velodyne-64	1	1 v centru střechy
Baidu	Velodyne-16/64	2/1	1 Velodyne-64 na vrchu a 1 Velodyne-16 na každé straně střechy
Apple	Velodyne-16	12	6 vpředu a 6 vzadu střechy
UM perl lab	Velodyne-16/64	4/1	2 Velodyne-16 na každé straně a 1 Velodyne-64 na vrchu střechy
Stanford Drivng Team	Velodyne-64	1	1 v centru střechy

Obr. 4 Umístění senzorů na střechu [51]

1.3 NEURONOVÉ SÍTĚ

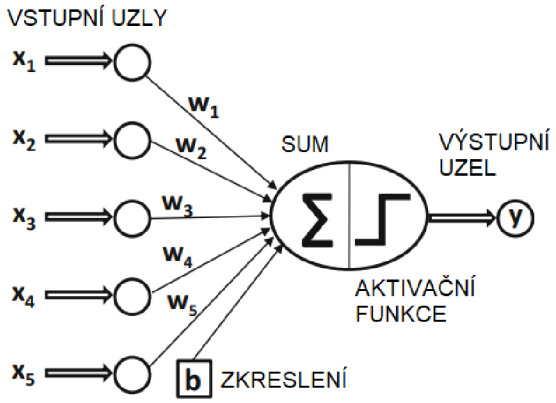
Pro získání informace, zda se v okolí vozidla nalézá hledaný objekt, je nutné zpracovat obrazová data z kamer. Při dnešních výpočtových kapacitách je vhodné použít hlubokou neuronovou síť pro počítačové vidění. V následujících kapitolách je proto provedená stručná rešerše neuronových sítí.

1.3.1 NEURON

Základním prvkem neuronové sítě je neuron, který je vyobrazen na *Obr. 5*. Znázorněný neuron má 5 vstupních uzlů a jeden výstupní. Výpočet výstupní hodnoty neuronu je dán vztahem (3). [5]

$$f(y) = \Phi \cdot \left(b + \left(\sum_{i=1}^{n} x_i \cdot w_i \right) \right) \tag{3}$$

Kde Φ je aktivační funkce, b je zkreslení, x_i je hodnota vstupního uzlu, w_i je váha pro daný uzel a n je počet vstupních uzlů.



Obr. 5 Neuron [5]

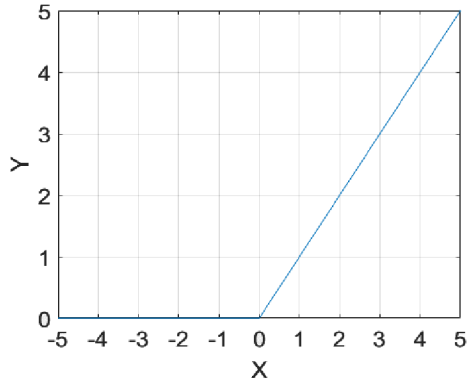
1.3.2 AKTIVAČNÍ FUNKCE

Aktivační funkce neuronu je matematická funkce, umožňující v neuronových sítích modelování nelineárních vztahů mezi vstupy a výstupy neuronu. Aktivační funkce přijímá jako vstup sumu vážených vstupních hodnot do neuronu a zkreslení. Následně generuje výstup neuronu. V počítačovém vidění jsou nejčastěji používány aktivační funkce ReLU a Sigmoid. [6]

RELU (RECTIFIED LINEAR UNIT)

Aktivační funkce ReLU se nejčastěji používá ve skrytých vrstvách. Využívá se k vynulování záporných hodnot, aby negativně neovlivňovali následující neuron. V případě kladné vstupní hodnoty je výstupní hodnota stejná. Funkce je dána vztahem (4). [6]

$$f(x) = \max(x, 0) \tag{4}$$

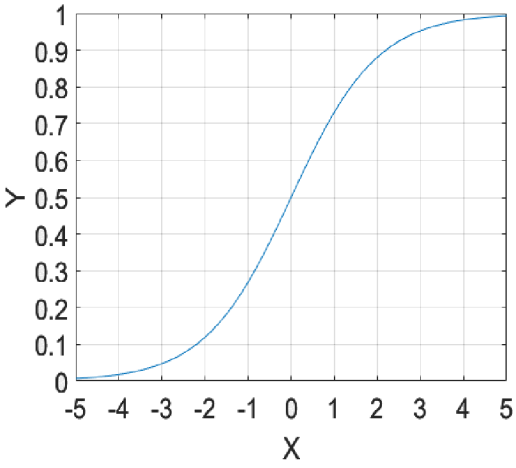


Obr. 6 Aktivační funkce ReLU [6]

SIGMOID

Tato aktivační funkce se nejčastěji používá v poslední vrstvě, pro získání pravděpodobnosti predikce. Funkce má vztah (5). [6]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$



Obr. 7 Aktivační funkce Sigmoid [6]

1.3.3 VRSTVY POUŽÍVANÉ V POČÍTAČOVÉM VIDĚNÍ

Vrstvy v neuronové síti jsou skupiny neuronů, které jsou organizovány do hierarchické struktury. Každá vrstva obsahuje soubor neuronů, které zpracovávají vstupní signály a generují výstupní signály pro další vrstvu. Pro počítačové vidění se nejčastěji používají konvoluční vrstvy a MaxPooling2D. [6]

KONVOLUČNÍ VRSTVA

Konvoluční vrstva je základní vrstva konvolučních neuronových sítí, která se používá k extrakci rysů nebo příznaků ze vstupních dat. Konvoluční vrstva se skládá z několika filtrů,

které se aplikují na vstupní data pomocí operace konvoluce. Tyto filtry detekují určité rysy v datech a vytvářejí tzv. mapy příznaků. [6]

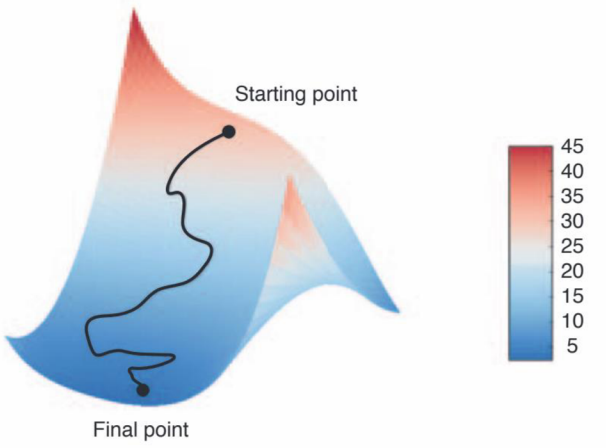
MaxPooling

MaxPooling je operace, která se používá v konvolučních neuronových sítích k redukci rozměrů mapy příznaků. Tato operace spočívá v rozdělení mapy příznaků na obdélníkové oblasti a následném výběru největší hodnoty v každé oblasti. Výsledkem je nová mapa příznaků s menšími rozměry. [6]

1.3.4 Učení sítě založené na gradientním sestupu

Učení sítě založené na gradientním sestupu je metoda učení neuronových sítí, která se snaží najít optimální parametry sítě, neboli váhy a zkreslení, které minimalizují ztrátovou funkci na trénovacích datech. Algoritmus gradientního sestupu začíná náhodným inicializováním vah sítě. Poté se provádí iterace, během kterých se vypočítá gradient ztrátové funkce vzhledem k jednotlivým vahám sítě. Gradient udává směr a velikost největšího růstu ztrátové funkce. Jeho opačný směr tedy ukazuje, jaké změny vahami sítě povedou ke snížení ztrátové hodnoty. [6]

Učení pomocí gradientu se skládá ze dvou kroků. Prvním je dopředný průchod sítí, kdy neuronová síť přijímá vstupní data a produkuje výstup. To znamená, že se vypočítají výstupy všech neuronů v síti, včetně výstupu výstupní vrstvy. Druhým krokem je zpětný průchod sítí. Chyba, která vznikne porovnáním výstupu sítě s očekávaným výstupem v trénovacích datech, je propagována zpět skrze síť, aby se určily příspěvky jednotlivých neuronů k celkové chybě sítě. Tento proces se nazývá zpětná propagace chyby. [6]



Obr. 8 Gradientní sestup pro dva parametry. [6]

1.3.5 Techniky zpracování obrazu

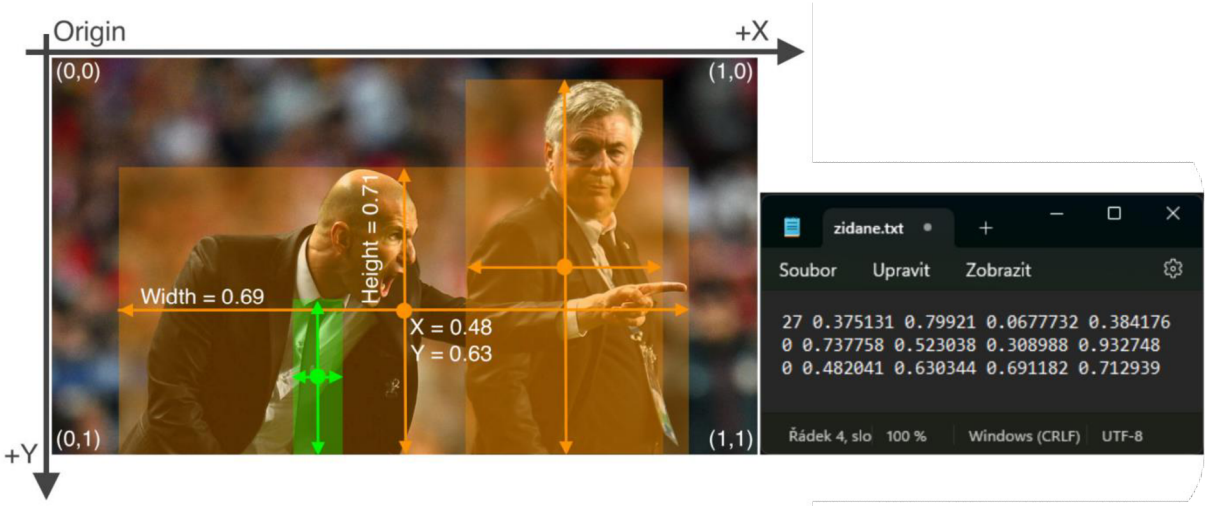
Techniky zpracování obrazu umožňují strojům získávat, analyzovat a interpretovat obrazová data. Tyto techniky zahrnují klasifikaci, detekci objektů a segmentaci obrazu. Díky těmto technikám je možné automatizovat řadu úloh.

KLASIFIKACE

Klasifikace obrazu je proces, při kterém se přiřazují kategorie nebo třídy k vstupnímu obrazu, který zůstane nezměněný. Tato technika má za úkol rozpoznat, o jaký objekt nebo scénu se jedná na základě vizuálních znaků v obraze. Pro trénování klasifikátorů jsou používány trénovací datasety, které obsahují obrazy s anotacemi kategorií nebo tříd. Klasifikace muže být například využita k rozpoznávání tváří pro bezpečnostní účely. [7].

DETEKCE OBJEKTŮ

Díky detekci objektů je vytvářen ohraničující rámeček kolem sledovaných objektů, aby bylo možné identifikovat, kde se na obrázku nachází. Cíle detekce jsou identifikovat, jaké všechny objekty jsou na snímku přítomné a kde se nacházejí, a odfiltrovat sledovaný objekt od ostatních objektů. Výstupem je matice počtu detekcí krát 5 sloupců. V prvním sloupci je zapsáno, o jakou třídu se jedná a v dalších 4 jsou zapsány souřadnice ohraničujících rámečků. Obdobným způsobem jsou zapsány anotované obrázky pro trénování. [7]



Obr. 9 Znázornění výstupních hodnot z YOLOv8 [39]

SEGMENTACE

Segmentace snímku umožňuje získat ze snímku oblast zájmu, která je vymezena objektem na úrovni jeho pixelů. Proces segmentace spočívá v rozdělení obrazu na různé oblasti, nazývané obrazové objekty. Je prováděna na základě vlastností obrazu, jako je například podobnost nebo diskontinuita. Cílem segmentace je zjednodušit obraz pro lepší následnou analýzu. [7]

1.3.6 NEURONOVÉ SÍTĚ PRO DETEKCI OBJEKTŮ

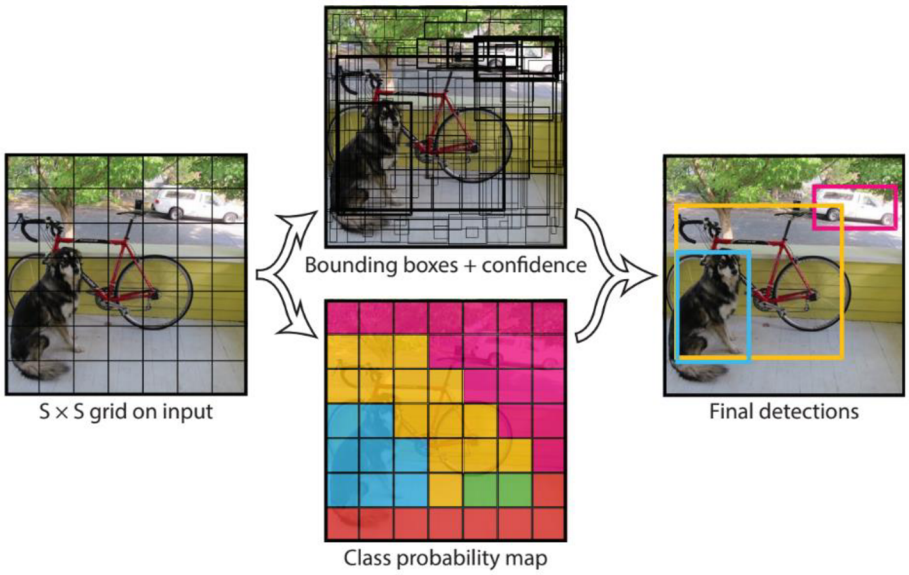
V této kapitole jsou představeny dvě neuronové sítě, které mohou být vhodné pro vytvářenou simulaci. FASTER R-CNN je prezentována, protože dosahuje vysoké přesnosti. YOLO je níže analyzováno, protože nabízí vhodnou alternativu mezi rychlostí a přesností.

FASTER R-CNN

FASTER Region-based Convolutional Neural Network je jeden z nejpřesnějších algoritmů pro detekci objektů v obraze. FASTER R-CNN je složená z dvojice hlavních neuronových sítí. Prvním z nich je plně konvoluční neuronová síť, která navrhuje oblasti (RPN), zatímco druhá je detektor objektů Fast R-CNN, který tyto navržené oblasti využívá. Tyto dva moduly dohromady tvoří jednu integrovanou síť pro detekci objektů. Tato síť naučená na datasetu Pascal VOC 2007+2012 dosahuje přesnosti mAP 73.2 % při rychlosti 7 FPS. [8]

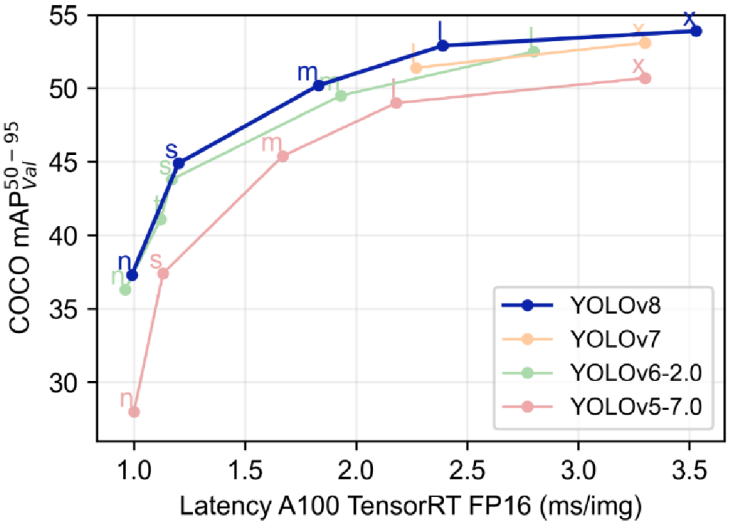
YOLO

You Only Look Once je na rozdíl od FASTER R-CNN pouze jedna neuronová síť. YOLO využívá rysy získané v celém obrazu pro predikování všech ohraničujících rámečků všech tříd současně čehož je dosaženo nejprve rozdělením vstupního snímku na čtvercové buňky o rozměru S. Pokud střed ohraničujícího rámečku spadá do dané buňky, je odpovědná za detekci daného objektu. Poté každá buňka vyhodnotí predikci a důvěryhodnost ohraničujících rámečků, za které je zodpovědná, a vytvoří skóre pravděpodobnosti tříd. Z tohoto YOLO vytvoří konečnou predikci ohraničujících rámečků a obvykle odstraní duplicitní detekce metodou NMS. Tato logika neuronové sítě umožnuje end-to-end tréning s detekcí v reálném čase a zachování vysoké přesnosti predikce. [9]



Obr. 10 Znázornění funkčnosti YOLO [9]

V letech 2020 až 2023 po přechodu na frameworku Pytorch [10] bylo vydáno mnoho nových verzí YOLO. Jedná se o verze 5, 6, 7, 8 a jejich modifikace. Každá verze má několik modelů, některé obsahují i malé modely pro telefony. [11] Jejich srovnání lze vidět na *Obr. 11*.



Obr. 11 Srovnání modelů jednotlivých verzí YOLO [11]

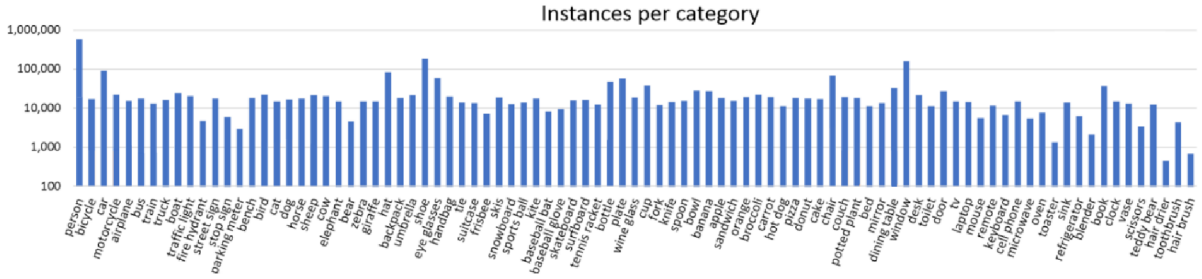
1.4 DATASET PRO DETEKCI OBJEKTŮ

Dataset pro detekci objektů je sada dat, která obsahuje obrazová data s ručně označenými anotacemi o umístění a typu objektů na obrázku. Tyto datasety se používají pro trénování a testování modelů neuronových sítí, které mají za úkol detekovat objekty na obrázcích a přiřadit jim kategorie. [7]

Dataset se nejčastěji vybírá podle dané aplikace. Pro simulaci vytvářenou v této práci, je třeba vybrat dataset, který obsahuje dostatek instancí vozidel. Množství obrázků na jednu třídu by mělo přesahovat 1500. Zároveň by na těchto obrázcích mělo být více než 10 000 instancí dané třídy. Dalším faktorem je rozmanitost, která musí být vhodná pro aplikaci neuronové sítě. Samozřejmostí je úplné a přesné označení všech instancí. [12]

1.4.1 MS COCO

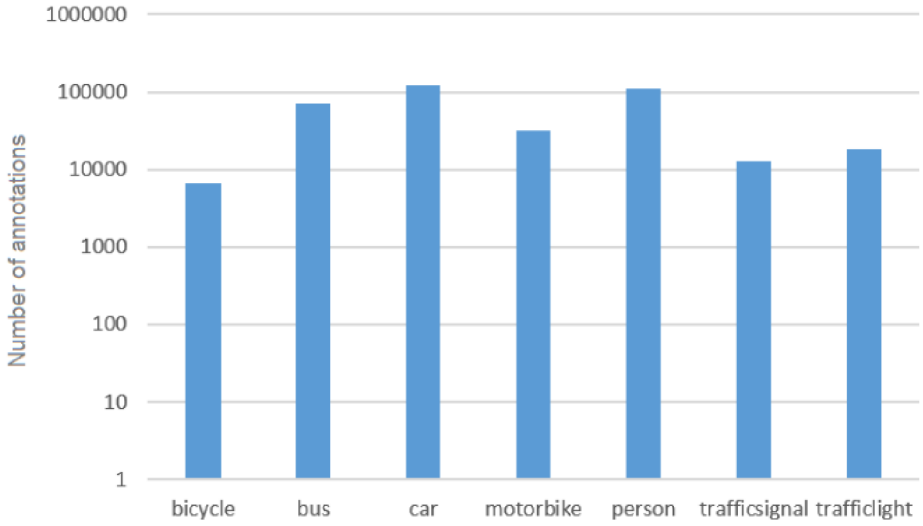
Dataset Microsoft Common Objects in Context je velmi rozsáhlý dataset pro rozpoznávání objektů a sémantické segmentace v obrazech. Obsahuje 91 různých tříd v 328 tisících obrázků s 2,5 miliony označených instancí. Obrázky jsou různorodé, pocházejí z různých zdrojů a zahrnují běžné objekty, jako jsou lidé, vozidla, zvířata nebo jídlo. [13] Tento dataset se v dnešní době nejčastěji používá pro porovnávání výkonnosti neuronových sítí pro počítačové vidění.



Obr. 12 Třídy a počty jejich instancí obsažené v MS COCO [13]

1.4.2 URBAN OBJECT DETECTION DATASET

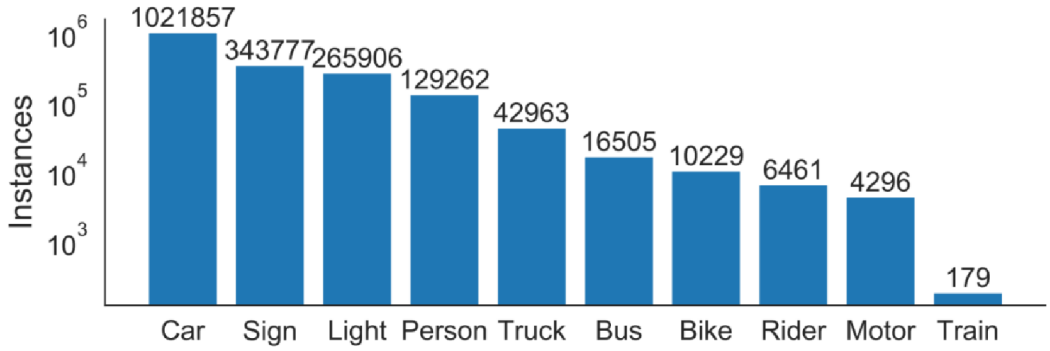
Tento dataset pochází z existujících datových sad, ze kterých byly vybrány obrázky pro detekci městských objektů. Je rozdělený do dvou hlavních skupin. První hlavní skupinu lze vidět na *Obr. 13* a druhá se skládá z dopravního značení. Snímky nejčastěji pochází z Evropského prostředí. Informace o anotaci zahrnují třídu objektu a souřadnice, které vymezují ohraničení rámečku. [14]



Obr. 13 Počet instancí jednotlivých tříd v Urban Object Detection dataset [14]

1.4.3 BDD-100K

Cílem tohoto datasetu bylo vytvořit rozsáhlý open-source soubor s videi jízdy. Dataset obsahuje 100 000 videi, které trvají okolo 40 s. Videa jsou získaná z prostředí New Yorku, oblasti Sanfranciského zálivu a dalších. Videa jsou navíc z různých denních dob a obsahují i extrémní povětrnostní podmínky jako je sníh a déšť. Videa byla rozdělena do tří skupin: trénovací s 70 000 videi, ověřovací s 10 000 videi a testovací s 20 000 videi. Snímky pro obrazové úlohy jsou vytvořeny v 10 s z každého videa a jsou opatřeny anotacemi. [15] Počty jednotlivých instancí tříd lze vidět na *Obr. 14*.



Obr. 14 Počet instancí jednotlivých tříd v datasetu BDD-100K [15]

1.5 PROGRAM PRO VYTVOŘENÍ SIMULACE

Pro simulaci, vytvářenou v této práci, je nutné použít program, který umožňuje simulovat jízdu vozidel a generovat data z implementovaných senzorů ve virtuálním 3D prostředí a zároveň podporuje jeho vložení.

1.5.1 SIMULINK A DOPLŇKY

Simulink je softwarový program, využívaný k modelování, analýze a simulaci dynamických systémů. Simulink nabízí vizuální rozhraní, které umožňuje sestavování modelů. Díky knihovně standardních komponent poskytuje snadnou a rychlou tvorbu blokových diagramů. Simulink je ideální nástroj pro výuku simulace reálných provozních problémů, protože umožňuje změnu simulačních algoritmů a parametrů během simulace s intuitivními výsledky. Zejména je užitečný pro studium chování systému. [16]

AUTOMATED DRIVING TOOLBOX

Automated Driving Toolbox disponuje souborem algoritmů a nástrojů, které slouží k návrhu, simulaci a testování systémů pro pokročilé asistenční systémy řidiče (ADAS) a autonomní řízení. Tento software umožňuje navrhovat a ověřovat systémy, jako jsou detekce a rozpoznávání objektů pomocí LiDARu, fúze senzorů, plánování trasy a řídicí jednotky vozidla. K dispozici jsou také vizuální a grafické nástroje, které zahrnují zobrazování videa, LiDARu, map a rozsahové zobrazení z ptačí perspektivy pro pokrytí senzorů, detekci a sledování objektů. [17]

Tento doplněk nabízí možnost společné simulace, která umožňuje modelování algoritmů v prostředí Simulink a vizualizaci jejich výkonu ve virtuálním simulačním prostředí Unreal Engine (UE). V rámci tohoto simulačního prostředí je možné konfigurovat předpřipravené scény nebo vytvářet vlastní a v nich umisťovat a pohybovat s vozidly. Je také možné nastavovat a umísťovat kamerové, lidarové a radarové senzory na vozidla a simulovat jejich výstupy. [18]

Nastavení parametrů senzorů v simulaci obecně zahrnuje polohu, orientaci a počet senzorů v simulaci. Pro kamerové senzory je například možné nastavit rozlišení kamery, ohniskovou vzdálenost, optický střed a distorzi. Pro lidarové senzory lze nastavit zorné pole, rozlišení ve stupních a detekční vzdálenost. U radarových senzorů je možné nastavit parametry přesnosti, frekvence, citlivost a úhel záběru. [19]

1.5.2 PROGRAM PRO VYTVOŘENÍ VIRTUÁLNÍHO OKOLÍ VOZIDLA ROADRUNNER

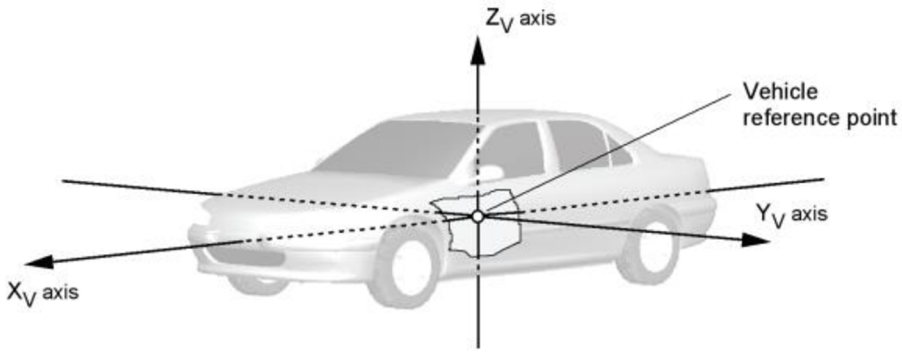
V Automated Driving Toolbox lze simulovat ve 3D prostředí v UE. Do UE lze také nahrát i vytvořené virtuální okolí, na kterém lze simulovat reálnou dopravní situaci. Pro vytváření dopravního virtuálního okolí byl vybrán program RoadRunner od společnosti MathWorks.

RoadRunner slouží pro návrh a vytváření 3D scény interaktivním způsobem. Zároveň obsahuje velké množství předchystaných dopravních značek, domů, stromů atd., nebo je možné vytvořit vlastní modely. Prostředí může být vymodelováno s vysokou podobností k realitě, z důvodu možnosti vytvoření virtuálního okolí na modelu reliéfu země s leteckým snímkem dané oblasti. Virtuální okolí z RoadRunneru se využívají pro simulaci vyvíjených

jízdních systémů. Vytvořená scéna lze exportovat do programů CARLA, Vires VTD, NVIDIA DRIVE Sim, Baidu Apollo, Cognata, Unity a Unreal Engine. [20]

1.6 SOUŘADNÉ SYSTÉMY

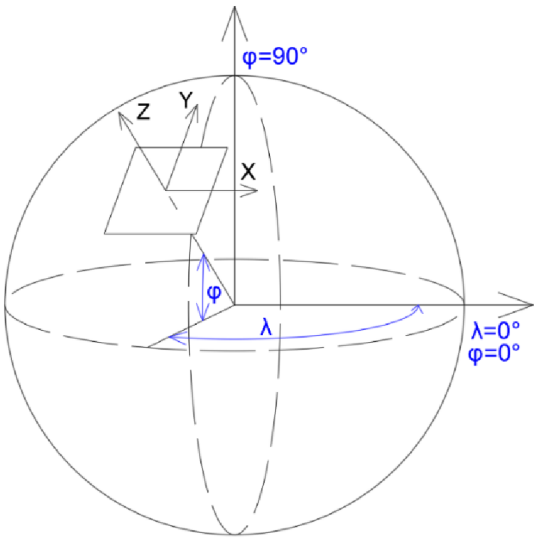
Bez souřadného systému by bylo obtížné popsat polohu objektů v prostoru a provádět výpočty vzdálenosti, rychlosti nebo zrychlení. Souřadný systém v dynamice vozidel popisuje norma SAE J670. [21] Tato norma uvádí dva pravotočivé souřadné systémy. Oba mají osu X ve směru jízdy. Rozdíl nastává v osách Y a Z, kde první systém definuje osu Z nahoru a osu Y doleva ve směru jízdy viz *Obr. 15*. Systém na obrázku bude v práci pro vozidlo využíván. Druhý systém má osy Y a Z obráceně.



Obr. 15 Souřadný systém vozidla [21]

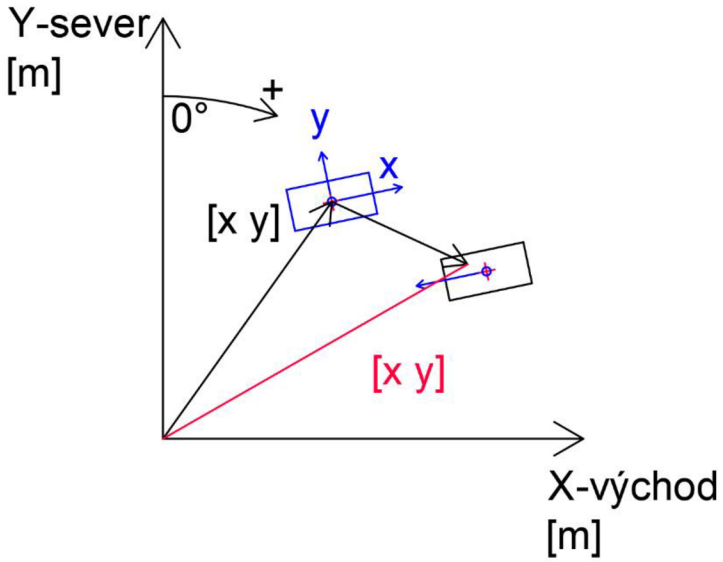
1.6.1 LOKÁLNÍ ABSOLUTNÍ KARTÉZSKÝ SOUŘADNICOVÝ SYSTÉM

Světový souřadnicový systém má význam pro plánování cest na globální úrovni, lokalizaci, mapování a simulaci jízdních scénářů. V této práci je použit kartézský souřadný systém s osou Z od země nahoru a nulový bod reprezentuje zeměpisné souřadnice. Jak je znázorněno na *Obr. 16*. Tato soustava souřadnic je využívána modulem Automated driving toolbox pro řízení vozidel ve scéně. [22]



Obr. 16 Znázornění kartézského systému

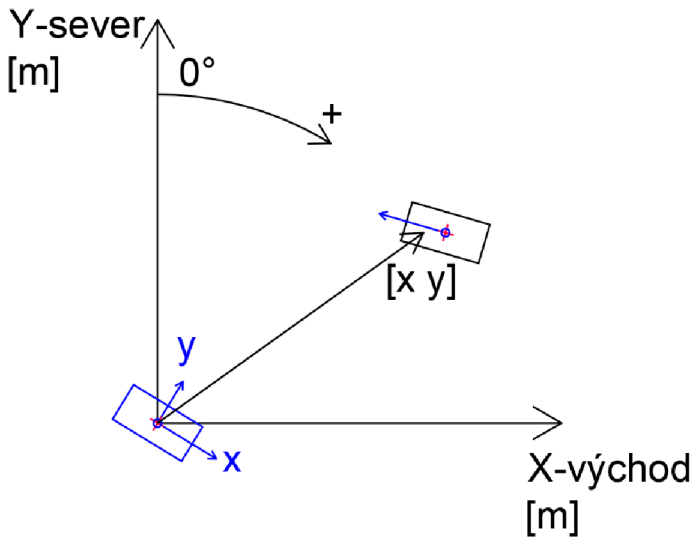
Na *Obr. 17* je vyobrazen princip získávání souřadnic detekovaných vozidel. V tomto systému je možné pracovat pouze při neustále známé pozici vlastního vozidla na mapě s definovaným nulovým bodem. Zároveň se při zvětšující vzdáleností od nulového bodu zvětšuje chyba při zpětné projekci na Zemi. Je proto použit i druhý systém, který je popsán v následující kapitole.



Obr. 17 Lokální absolutní kartézská soustava souřadnic

1.6.2 LOKÁLNÍ RELATIVNÍ KARTÉZSKÝ SOUŘADNICOVÝ SYSTÉM

Pro eliminaci chyby způsobené 2D projekcí je použitý tento systém, který používá vlastní vozidlo jako nulový bod. Další výhodou je, že tento systém dokáže pracovat bez známé polohy na mapě. Je vhodné znát natočení vlastního vozidla pro zachování os, není ovšem to nutné.



Obr. 18 Lokální relativní kartézská soustava souřadnic

2 VYTVOŘENÍ VIRTUÁLNÍHO OKOLÍ

Pro vytvoření věrohodného simulačního modelu je třeba simulovat reálné prostředí. Pro tyto účely je využitý výše zmíněný program RoadRunner, který nabízí řadu užitečných funkcí, ulehčující práci při vytváření virtuálního okolí.

2.1 VYTVOŘENÍ VIRTUÁLNÍHO OKOLÍ POMOCÍ ROADRUNNER

Při začátku vytváření nového virtuálního okolí je potřeba použít nástroj World Setting Tool. Pomocí nástroje se nastaví zeměpisná šířka a výška, která slouží pro určení nulového bodu (na *Obr. 16*) v kartézském souřadném systému. Počátek byl zvolen tak, aby celá simulace probíhala do 1 km od počátku.

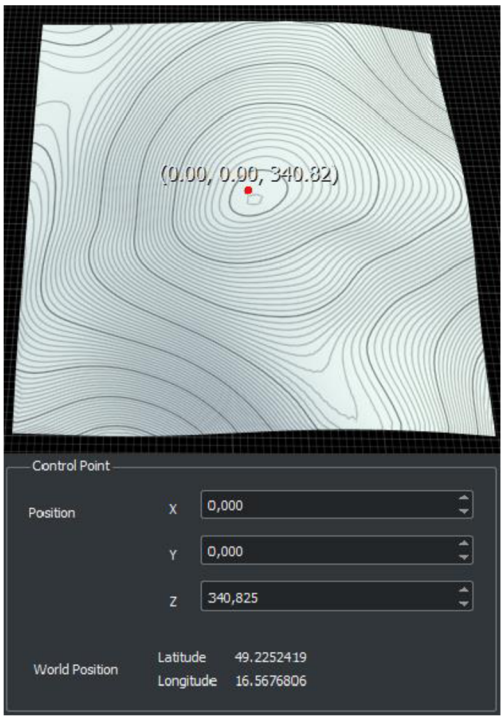
2.1.1 SIMULOVANÉ MÍSTO

Na základě doporučení vedoucího práce byla pro simulaci vybrána zatáčka ve městě Polička na silnici č. 34. Vymezení trasy simulace, okolo které je vymodelované virtuální prostředí, je znázorněno na obrázku *Obr. 19*.

Obr. 19 Vymezení simulovaného místa. [52]

2.1.2 PODKLADY PRO TVORBU MODELU

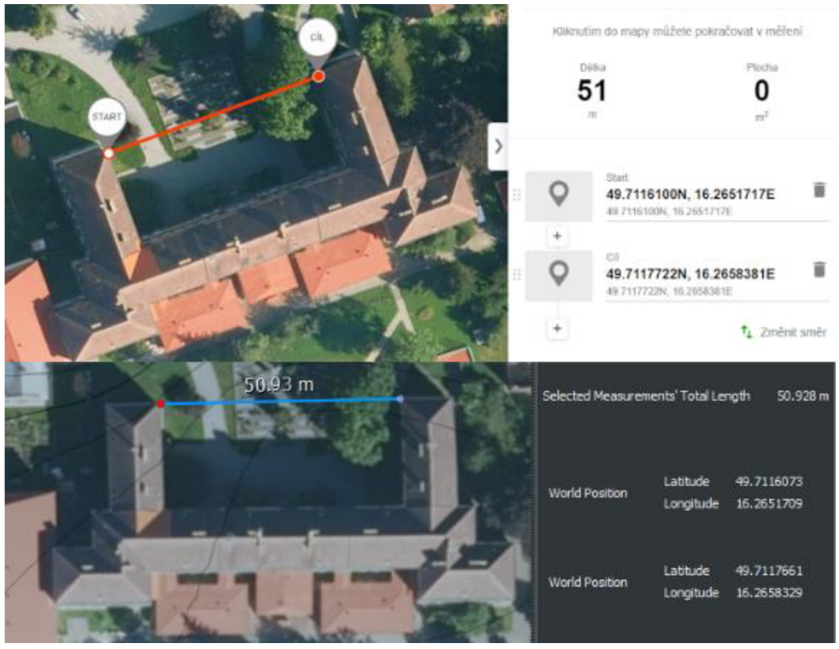
Prvním podkladem pro vytvoření virtuálního okolí byl digitální model reliéfu (DEM). Ten je možný zakoupit pro českou republiku na Geoportálu Českého úřadu zeměměřického a katastrálního. [23] Byl však využit evropský DEM, který má vrstevnice v ekvidistanci 25 m. Je zároveň financován EU a díky tomu je zdarma. [24] Evropský model reliéfu nebylo nutné nastavovat, protože má předdefinované světové zobrazení. Pro ověření správného promítání byli souřadnice a nadmořská výška porovnány s významným bodem, Palackého vrchem v Brně. Souřadnice i nadmořká výška odpovídají.



Obr. 20 Kontrola umístění DEM pomocí Palackého vrchu

Druhým použitým podkladem byl letecký snímek. Byl zakoupený na Geoportálu ČÚZK. [23] Letecký snímek využívá souřadnicový systém S-JTSK. [25] Po vložení leteckého snímku byla provedena kontrola pomocí dvou bodů, zobrazené na *Obr. 21*

Dalším možným způsobem získání leteckého snímku je použitím výstřižků a jejich spojením v grafickém editoru. Poté se importuje jako letecký snímek a nastaví podle 2 definovatelných míst, obdobně jako při kontrole. Tato možnost, nebyla zvolena z důvodu velké časové náročnosti.



Obr. 21 Kontrola přichycení leteckého snímku v RoadRunner dole s mapy.cz nahoře [52]

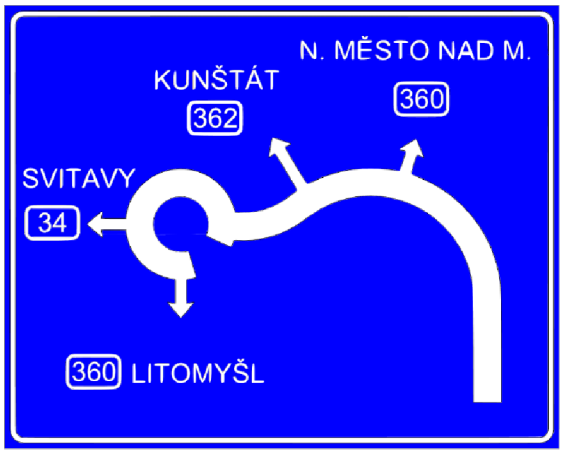
2.1.3 MODELOVÁNÍ VIRTUÁLNÍHO OKOLÍ

Při vytváření modelu byla nejdříve vytvořena silnice. Tvar a šířka silnice byly modelovány pomocí leteckého snímku, promítaného na DEM. Na modelu reliéfu se uchycují body nástrojů pro vytváření silnice a povrchu. Po dokončení modelu silnice byl vymodelován povrch a přiřazen vhodný materiál. Na povrch potom byly přidány rekvizity, např. stromy, lampy nebo domy. Pro přiblížení realitě byly v programu předpřipravené domy upraveny pomocí měřítka, aby korespondovaly s obrysem domů z leteckého snímku. Dále bylo vytvořeno značení za stejným účelem.

Obr. 22 Porovnání modelu okolí v UE vlevo s reálnou fotkou vpravo z mapy.cz [52]

VYTVOŘENÍ ZNAČENÍ

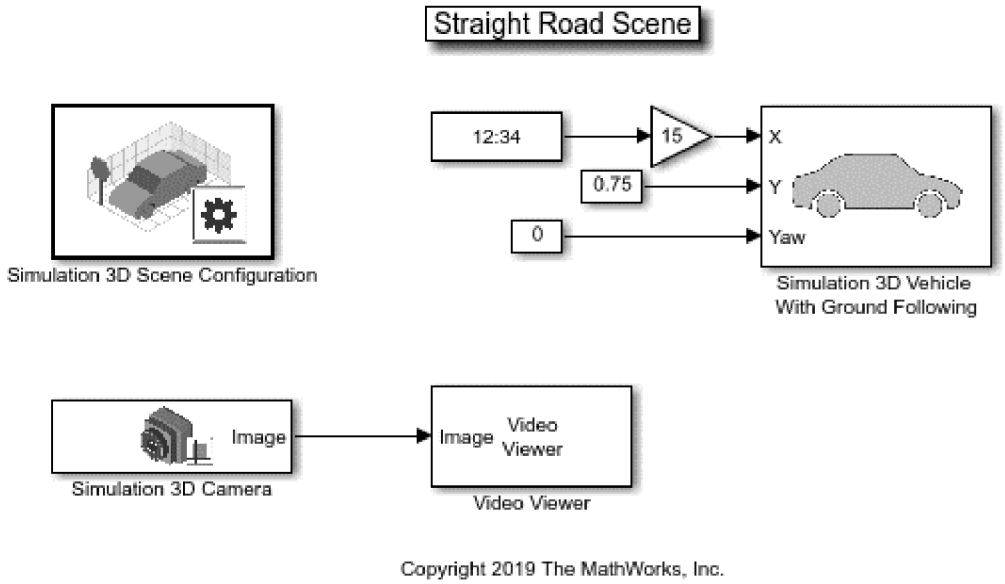
Pro vytvoření svislého dopravního značení byl použit Autocad pro vytvoření obrysů. [26] Obrys byl vyexportován do programu InkScape. [27] V InkScape byl vybarven, vyexportován a vložen do RoadRunner. Vodorovné dopravní značení bylo vytvořeno úpravou kopie již předpřipravených značení. Rozměry byly upraveny podle technických podmínek ministerstva dopravy. [28]



Obr. 23 Vytvořené svislé dopravní značení

2.2 Ověření funkčnosti simulace na jednoduchém modelu v Simulinku

Po nainstalování přesných verzí nutných programů podle [29] proběhlo vytvoření projektu pomocí návodu. [30] V předpřipraveném základním okolí předchystaného projektu byla ověřena celková funkčnost simulace pomocí jednoduchého modelu zobrazeného na *Obr. 24*. Na tomto modelu byla potvrzena funkčnost kooperace UE se Simulinkem a následně byl model rozšířen o vytvořené virtuální okolí.



Obr. 24 Jednoduchý model [30]

2.3 Vložení virtuálního okolí do Unreal Engine

Po vymodelování virtuálního okolí bylo vyexportováno ve formátu pro UE. Pro importování okolí do UE byl využitý návod. [31] Po nahrání mapy byly zjištěny drobné chyby. Největší z nich je chybějící dopravní ostrov znázorněný na *Obr. 25* vlevo v porovnání s modelem v RoadRunneru vpravo. Tato chyba je pouze estetická a na celou simulaci nemá vliv, proto se nebylo chybou dále zabýváno.

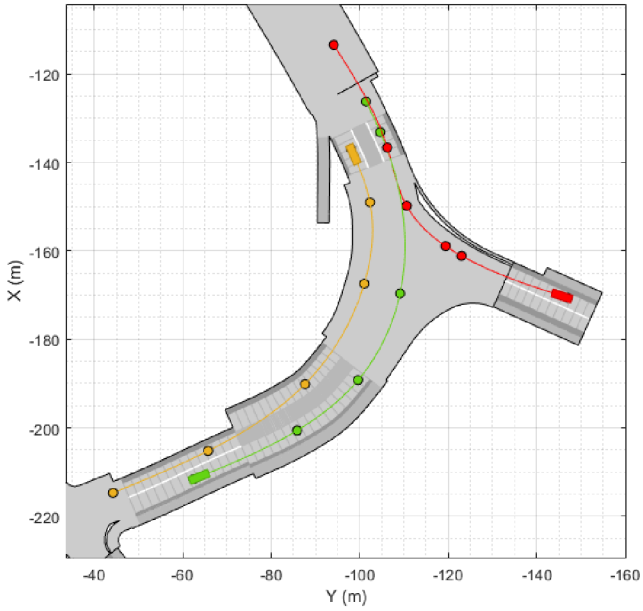
Obr. 25 Nevytvořený dopravní ostrov vlevo při importu do Unreal engine

2.4 VYTVOŘENÍ SCÉNY V DRIVING SCENARIO DESIGNER

Pro vytvoření jízdní scény v Driving Scenario Designer je zapotřebí soubor ASAM opendrive, který byl vyexportován z RoadRunneru pro vymodelované okolí. Tento soubor zobrazí vytvořené silnice, na kterých se vytvoří pohyb vozidel. Tvary silnic jsou v Driving Scenario Designer pouze informativní, jelikož simulace probíhá v UE. Pohyby vozidel jsou definovány trajektorií a rychlostí v každém bodě, zobrazeném na *Obr. 26* a *Tab. 2*. Vytvořená jízdní scéna byla uložena jako soubor MATLAB data, obsahující informace o pohybu vozidel. Celá navrhnutá scéna jízdy je ukončena v 9,2 s.

Tab. 2 Zápis časů bodů vozidel a jejich rychlostí v bodech

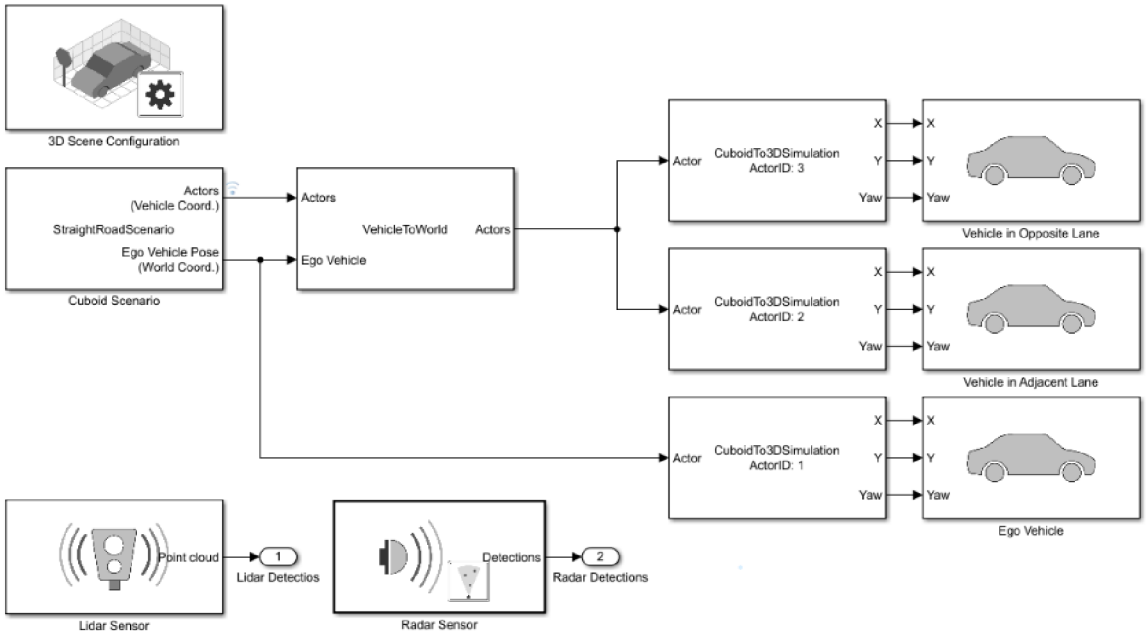
Vozidlo	Rychlosti v bodech [m·s^{-1}]	Časy průjezdů body [s]
Hlavní zelené	[14, 14, 10, 10, 12, 14]	[0, 1,85; 3,35; 5,55; 8,94; 9,52]
Žluté	[14, 12, 10, 10, 14, 14]	[0; 0,99; 2,68; 5,34; 7,57; 9,25]
Červené	[14, 6, 2, 8, 12, 14]	[0; 2,58; 3,64; 6,5; 7.59; 9,62]



Obr. 26 Zobrazení scény jízdy

2.5 ROZŠÍŘENÍ SIMULAČNÍHO MODELU O JÍZDNÍ SCÉNU

Simulační model byl v Simulinku rozšířen o řízení vozidel pomocí souboru s jízdní scénou vytvořeného v Driving Scenario Designer. Pro řízení vozidel pomocí jízdní scény byl vybrán Simulink model, který je vyobrazen na *Obr. 27*. [32] Následně byly v tomto modelu změněny hodnoty bloků. Bylo nastaveno již odzkoušené virtuální okolí v bloku *3D Scene Configuration*. V bloku *Cuboid Scenario,* který čte informace o pohybu vozidel ze souboru jízdní scény, byla uložena cesta k souboru scény. Poté byl upraven *Simulation 3D Vehicle with Ground Following,* kde byla pro jednotlivá vozidla nastavena předvolená karoserie a bylo zvoleno, jaké vozidlo z jízdní scény daný blok reprezentuje. Tento Simulink model byl dále využíván a upravován až do výsledné verze.



Obr. 27 Rozšířený model [32]

3 IMPLEMENTACE SNÍMAČŮ

Pro správné fungování algoritmu je vhodné mít na vozidle vhodně nastavené parametry snímačů. Tyto parametry by měly být technologicky podobné parametrům aktuálně dostupných snímačů, aby algoritmu nebyly poskytovány příliš kvalitní data. Z tohoto důvodu jsou vybrány parametry na základě existujících senzorů.

3.1 LIDAR

Při hledaní vhodného LiDARu byly zvoleny základní kritéria. První kritérium bylo bezpečnost očí za jakýchkoliv okolností. Proto každý vybraný LiDAR musí mít laser třídy 1. Druhé kritérium bylo 360° snímání. Posledním kritériem je dosah, který by měl být minimálně 80 m.

3.1.1 PARAMETRY EXISTUJÍCÍCH LIDARŮ

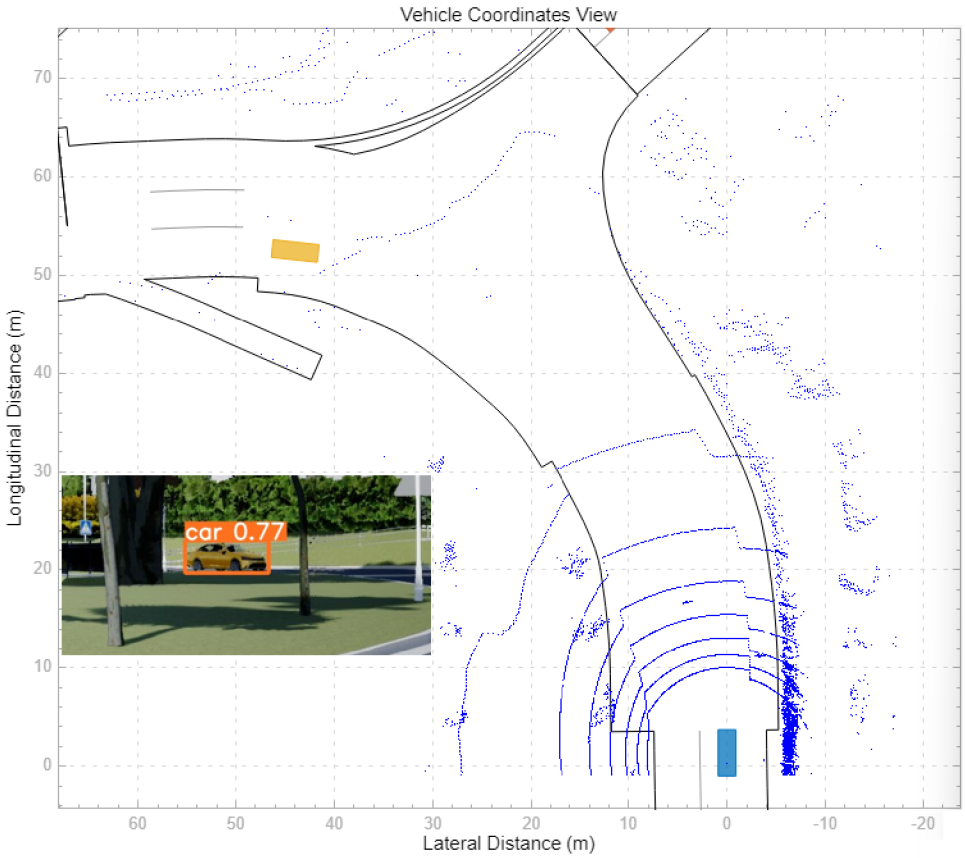
Z nalezených nabízených LiDARů na trhu bylo vybráno 5 LiDARů od dvou výrobců. Všechny zobrazené LiDARy v *Tab. 3* splňují výše zmíněná kritéria a zároveň jsou vhodné pro použití v automobilovém průmyslu.

Tab. 3 Aktuální LiDARy na trhu

Lidar 360°				
Výrobce	Název	Vertikální zorné pole / rozlišení	Hor. rozlišení	Dosah
Velodyne	HDL-32E [33]	41,33° (10,67° -30,67°) /1,33°	0,1° - 0,4°	100 m
Velodyne	PUCK [34]	30° (±15°) /2°	0,1° - 0,4°	100 m
Velodyne	PUCK Hi-Res [35]	20° (±10°) /1,33°	0,1° - 0,4°	100 m
Ouster	OS2 (REV. 7) [36]	22,5° (±11,25°) /0,35°	0,176°-0,7°	200 m
Ouster	OS1 (REV. 7) [37]	45° (±22,5°) /0,7°	0,176°-0,7°	90 m

3.1.2 ZVOLENÍ PARAMETRŮ LIDARU

Z prezentovaných LiDARů byly nejprve vyloučen Velodyne HDL-32E, protože má nadbytečně velké záporné zorné pole, které by nebylo využito z důvodu zasahování paprsků do vozidla. Velodyne LiDAR PUCK nebyl využit z důvodu velkého úhlu vertikálního rozlišení, které by bylo nedostatečné. Jako první byly nastaveny parametry podle Velodyne PUCK Hi-Res, který byl zvolen jako kompromis mezi výkonem a cenou. Bylo však zjištěno, že vertikální rozlišení 1,33° je nedostatečné již při 65 m, viz *Obr. 28*. Do obrázku byl přidán pohled na vozidlo, který odpovídá kroku, ve kterém bylo vozidlo špatně zachycené LiDARem. Z důvodu potřeby dostatečného rozlišení byly využívány parametry Ouster OS2.



Obr. 28 LiDAR s vertikálním rozlišením po 1,33°

3.2 KAMERA

Při výběru kamery bylo vycházeno z rozlišení obrazového snímače a podle potřeby úhlu zorného pole byla dopočítána ohnisková vzdálenost, která se zadává do bloku *Simulation 3D Camera*. V praxi by byl pro obrazový senzor vytvořen odpovídající objektiv.

3.2.1 DOSTUPNÉ OBRAZOVÉ SENZORY

K porovnávání byly vybrány optické senzory od výrobců Samsung [38] a Sony [39]. Od každého byly vybrány dva senzory pro vyhodnocení. Hlavním parametrem pro posuzování bylo rozlišení.

Tab. 4 Optické senzory

Název	Rozlišení	Filtr
Samsung ISOCELL 3B6	1920 x 1536 (2,95 Mpx)	RGGB/RCCB
Samsung ISOCELL Auto 4AC	1280 x 960 (1,23 Mpx)	RGB
Sony IMX390CQV	1920 x 1080 (2,07 Mpx)	RGB
Sony IMX224	1280 x 960 (1,23 Mpx)	RGB

3.2.2 VÝBĚR Z OBRAZOVÝCH SENZORŮ

Vybrány byly Sony IMX390CQV s rozlišením FullHD z důvodu dostatečného rozlišení. Snímač Samsung ISOCELL 3B6 má příliš velké vertikální rozlišení 1536 px, které by nebylo využité v celém rozsahu a zvyšovalo by nároky na zpracování obrazu. Ostatní obrazové snímače nedisponují dostatečným rozlišením.

3.2.3 VÝPOČET ZORNÉHO POLE

Pro výpočet ohniskové vzdálenosti s definovaným zorným polem kamery je uvažován model dírkové kamery. Pro výpočet ohniskové vzdálenosti je využit upravený vztah (1). Po zjištění ohniskové vzdálenosti pomocí vzorce (6) bylo dopočítáno vertikální zorné pole rovnicí (7). Ohnisková vzdálenost se do bloku *Simulation 3D Camera* zadává pouze v pixelech, proto výpočty (6) a (7) probíhaly pouze v pixelech. Pro získání reálné ohniskové vzdálenosti je potřeba výsledek vynásobit velikostí pixelu.

$$f = \tan\left(\frac{\theta}{2}\right) \cdot \frac{x}{2} = \tan\left(\frac{60}{2}\right) \cdot \frac{1920}{2} = 1663\ px \tag{6}$$

$$\theta = 2 \cdot \tan^{-1}\left(\frac{y}{2 \cdot f}\right) = 2 \cdot \tan^{-1}\left(\frac{1080}{2 \cdot 1663}\right) = 35{,}98° \tag{7}$$

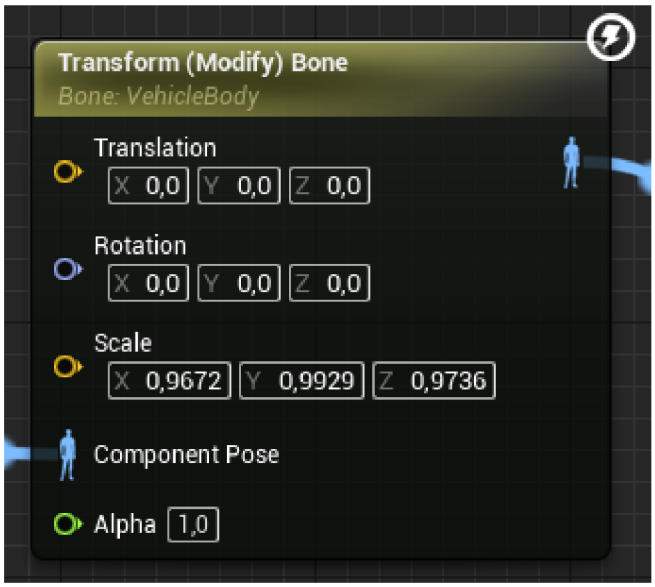
Kde x je horizontální rozlišení obrazového senzoru a y je vertikální rozlišení obrazového senzoru.

3.3 PŘIDÁNÍ SENZORŮ DO SIMULACE

Samotné bloky senzorů v Simulinku byly vybrány z knihovny Automated Driving Toolbox. [19] Senzory byly upraveny dle parametrů z vybraných senzorů z předchozích kapitol. Jejich umístění na vozidle bude detailněji popsáno v následujících kapitolách. Konkrétní umístění jednotlivých senzorů bylo měřeno z vnějších rozměrů karoserie.

3.4 ZVOLENÉ VOZIDLO A JEHO ÚPRAVA V SIMULACI

Pro tuto práci bylo zvoleno vozidlo Škoda Octavia 4. generace [40], protože v roce 2022 to byl nejprodávanější model na českém trhu. [41] Po zvolení vozidla bylo přistoupeno k jeho implementaci do simulace. Kvalitní síťové modely vozidla jsou prodávány za velmi vysokou cenu, byl proto použit předpřipravený model s nejpodobnější karoserií, kterým byl Sedan [42]. Poté byly poměrově upraveny vnější rozměry vozidla v UE Blueprint [43] pomocí bloku *Transform (Modify) Bone*.



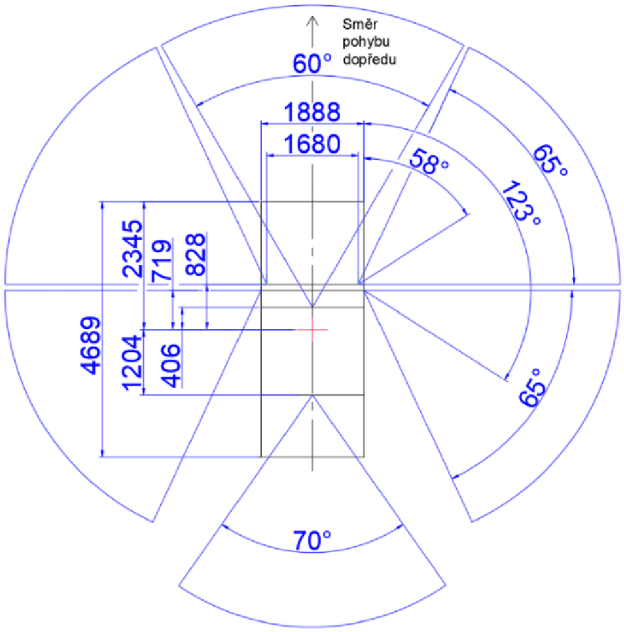
Obr. 29 Blok *Transform Bone*

3.5 NÁVRHY UMÍSTĚNÍ SNÍMAČŮ NA VOZIDLE

Existující varianty umístění senzorů na vozidlo jsou popsány v kapitole 1.2. V následujících podkapitolách jsou představeny vlastní návrhy umístění z důvodu aplikace na konkrétní vozidlo. Kótování senzorů je prováděno od geometrického středu vozidla, který je na *Obr. 30* a *Obr. 31* zobrazen červeným křížem a v simulaci je využíván pro získání souřadnic vlastního vozidla.

3.5.1 NÁVRH UMÍSTĚNÍ DO KAROSERIE

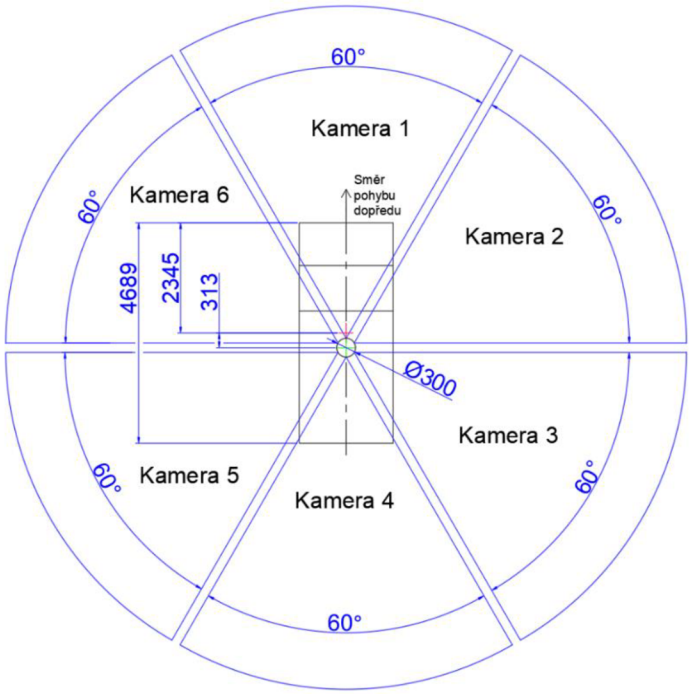
Pro toto osazení bylo použito 6 kamer pro získání 360° zorné pole znázorněných na *Obr. 30*. Jedna z kamer je namířena vpřed a umístěna do vnitřního zpětného zrcátka ve výšce 1350 mm se zorným úhlem 60°. První pár bočních kamer je umístěn ve vnějším zpětném zrcátku ve výšce 1020 mm se zorným úhlem 65°. Společně s přední kamerou zabírají úhel 180°. Další pár bočních kamer je umístěn taktéž ve vnějších zpětných zrcátkách a imituje pohled do vnějších zpětných zrcátek ve výšce 1040 mm se zorným úhlem 65°. Poslední kamera je umístěná u zadního vnitřního brzdového světla ve výšce 1350 mm se zorným úhlem 70°. Okolní kamery mají lehce zvětšené zorné pole, aby pokryly slepá místa v mezerách mezi kamerami. K tomuto rozmístění kamer bylo zvažováno umístění LiDARu na střechu.



Obr. 30 Návrh umístění do karoserie

3.5.2 NÁVRH UMÍSTĚNÍ NA STŘECHU

Vytvořený návrh umístění snímačů na střechu se skládá ze 6 kamer pro 360° zorné pole. Každá kamera má zorné pole 60° a jsou rozmístěny po 60°. Kamery jsou umístěné 80 mm nad střechou ve výšce 1557 mm. Zároveň je do středu senzorů, vyznačeným zeleně, umístěn LiDAR o 120 mm nad kamery. Vertikální úhel pro kamery byl zvolen 0°.



Obr. 31 Návrh umístění na střechu

3.5.3 VOLBA UMÍSTĚNÍ SNÍMAČŮ NA VOZIDLE

V návrhu do karoserie se umístění LiDARu na střechu ukázalo jako nevhodné řešení, protože při průjezdu kolem svislého dopravního značení bylo vozidlo detekováno, ovšem všechny body vzdáleností z LiDARu zasahovali do značení. Z tohoto důvodu je vhodné, aby byl LiDAR v blízkosti kamery. Proto by muselo být umístěno 6 lidarů se zorným polem podobným kamerám. Při přidání LiDARů by v tomto návrhu muselo být nově zpracováno jejich umístění.

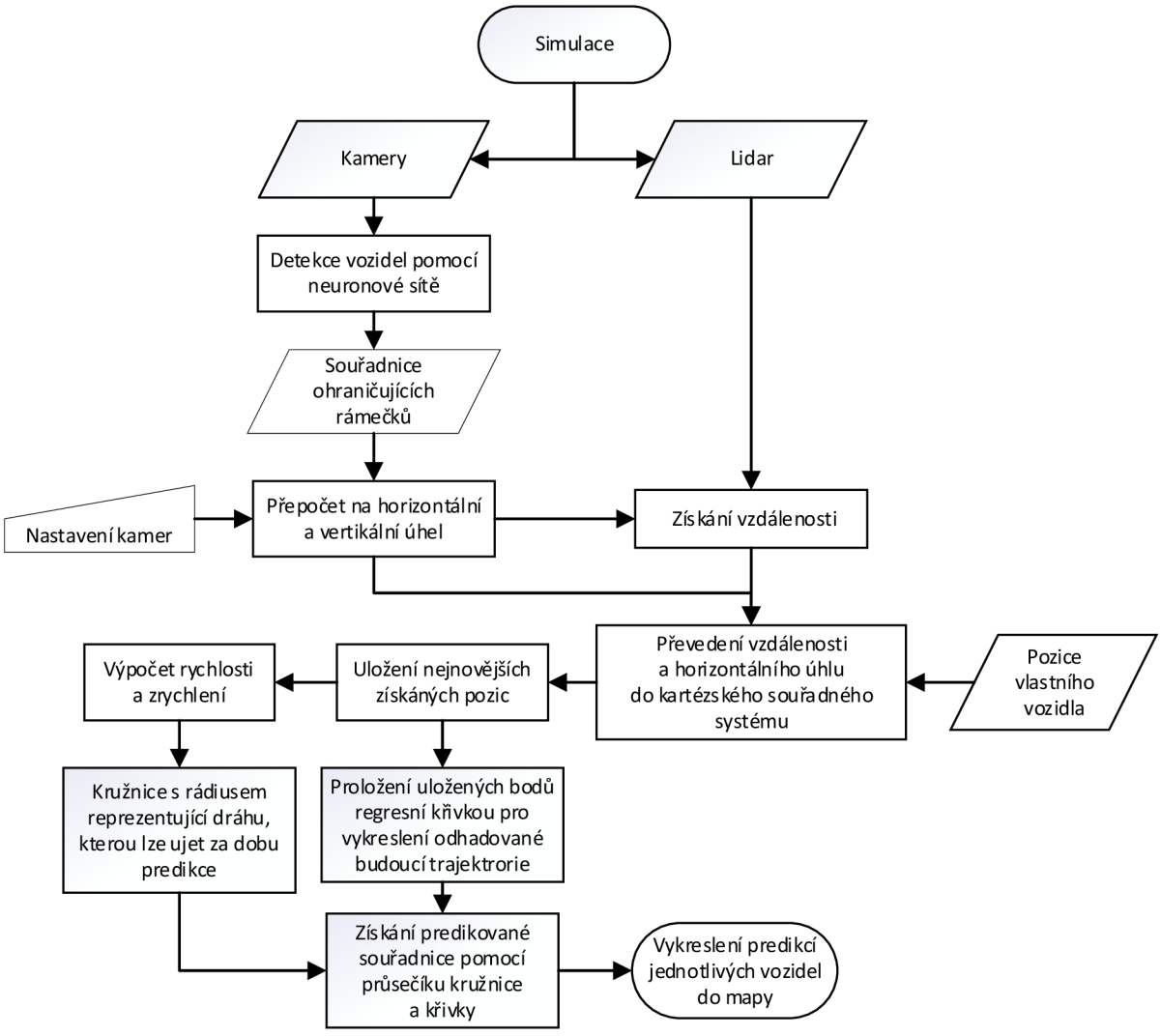
Senzory na střeše vozidla mají lepší viditelnost okolního prostředí. Tato zlepšená viditelnost je užitečná v raných fázích vývoje algoritmu, kdy je třeba shromáždit co nejvíce informací, pro pochopení chování systému a detekování potenciálních překážek. Udržením senzorů ve fixní poloze na horní části vozu se eliminuje potřeba simulovat složité pohyby potřebné pro senzory namontované na jiných částech vozidla.

Pro vývoj softwaru a ověření funkčnosti v reálném provozu je varianta umístění senzorů na střechu lepší z hlediska jednoduchosti osazení na již existující vozidla. Tuto variantu je možné vytvořit a připevnit k vozidlu jako celek. Díky tomu bude zároveň jednoduší kalibrace senzorů. Z těchto důvodů bylo zvoleno umístění senzorů na střechu, kterým muže být v provozu ověřena reálnost simulace. Z důvodu využívání tohoto umístění v simulaci, byl vytvořen vizuální koncepční návrh umístění kamer s LiDARem, který lze vidět na *Obr. 32*.

Obr. 32 Vizualizace koncepčního návrhu

4 VYTVOŘENÍ ALGORITMU PRO PREDIKCI POHYBU JEDNOTLIVÝCH OBJEKTŮ

Tato kapitola popisuje vytvořený algoritmus pro predikování pohybu detekovaných objektů. Logika algoritmu je založená na výstupních hodnotách z neuronové sítě, tedy souřadnic ohraničujících rámečků detekované třídy vozidla, které ukazují kde se vozidlo na kameře nalézá. Následně se souřadnice přepočítají na vertikální a horizontální úhel v kameře. Poté se za pomoci získaných úhlů a natočení kamery vůči vozidlu vyhledá vzdálenost pomocí LiDARu. Tyto hodnoty se přepočítají do kartézských souřadných systémů. Pomocí detekovaných souřadnic vozidel bude získána křivka regrese druhého řádu, která by měla odpovídat trajektorii, po které vozidlo nejspíše pojede a získán kruh reprezentující dráhu, kterou je vozidlo schopno ujet za dobu predikce. Prostřednictvím jejich průniku budou získány predikované souřadnice. Doba predikce je čas, za který se bude vozidlo nalézat na predikovaných souřadnicích. Samotná simulace bude probíhat při konstantních 32 snímcích za sekundu, protože se už 30 snímků za sekundu lidskému oku jeví jako plynulý obraz.



Obr. 33 Logika algoritmu

4.1 ZVOLENÍ NEURONOVÉ SÍTĚ PRO DETEKCI OBJEKTŮ

Z neuronových sítí, které byly zmíněny v kapitole 1.3.6. je nejvhodnější architekturou pro tuto závěrečnou práci YOLO, díky její rozmanitosti variant, vysokému mAP a především zpracovávání snímku v reálném čase. Mezi zvažované varianty patří všechny verze, které pracují na frameworku PyTorch, tedy YOLOv5-7, YOLOv6-2, YOLOv7 a YOLOv8.

Při výběru varianty YOLO pro detekci objektů byl zvážen výkon modelů jednotlivých neuronových sítí. Tento výkon je určen poměrem přesnosti a rychlosti zpracování snímku. Kromě toho bylo nutné zajistit propojitelnost se simulací.

Ze srovnání v kap. 1.3.6 na *Obr. 11* vyplývá, že nejlepší volbou je YOLOv8. YOLOv5-7 má oproti ostatním nižší přesnost mAP při stejné rychlosti zpracování jednoho snímku. YOLOv7 obsahuje pouze velké velikostní modely s velkým množstvím parametrů. To není vhodné, protože při jejich použití by byl nutný vysoký výpočetní výkon. Rozdíly mezi YOLOv8 a YOLOv6-2 jsou u malých modelů zanedbatelné. Od modelu M a větších je vhodnější verze 8, protože dosahuje vyšší přesnosti s kratší dobou zpracování snímku. Z tohoto důvodu bylo zvolena YOLOv8, konkrétně střední model YOLOv8m a nejmenší model YOLOv8n, které budou dále porovnány při použití v simulaci.

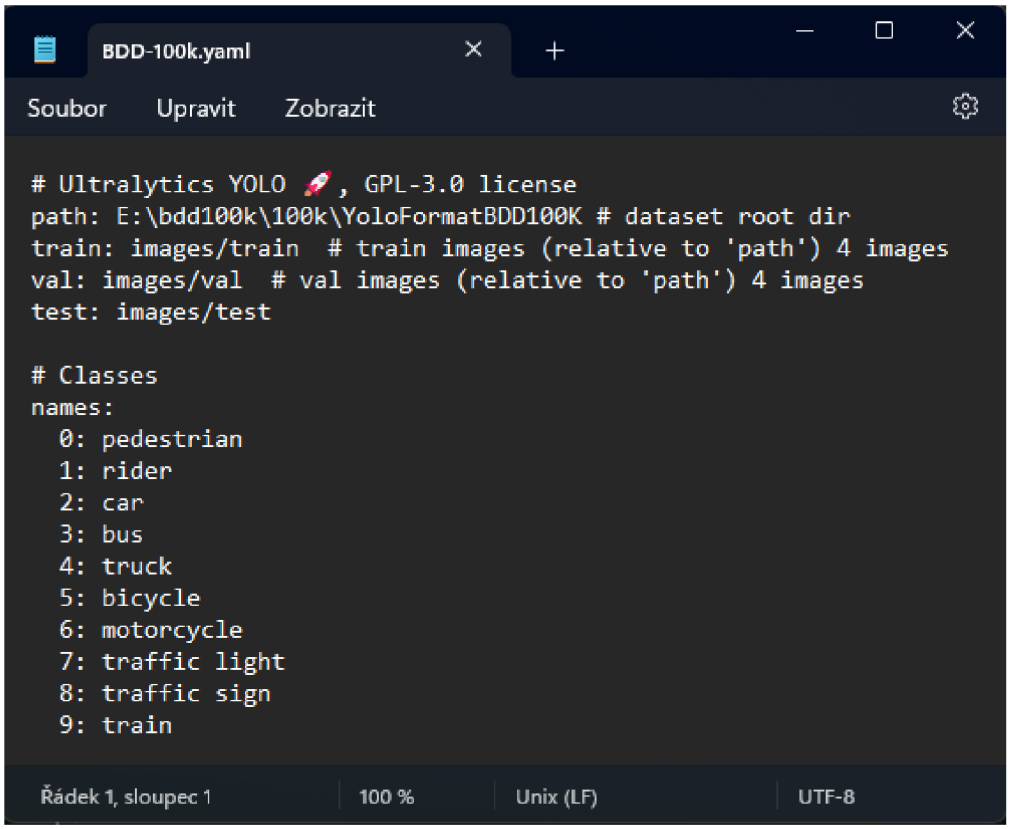
4.2 ZVOLENÍ DATASETU

Z datasetů popsaných v kapitole 1.4 byl vybrán BDD-100K, protože se jedná o rozsáhlý soubor dat, který nejlépe vystihuje prostředí, do něhož bude síť nasazena. Hlavním důvodem výběru je, že tento dataset je složen ze snímků různých typů vozidel pořízených v rozdílných městech s různými klimatickými podmínkami. BDD-100K také obsahuje více než dostatečný počet instancí třídy vozidlo.

4.3 INSTALACE A TRÉNOVÁNÍ SÍTĚ

Po zvolení vhodné neuronové sítě pro detekci objektů a datasetu bylo nutné nainstalovat platformu YOLOv8, která je napsána v Pythonu. Instalace byla provedena příkazem `pip install ultralytics`. [11] Tento příkaz v sobě zahrnuje instalaci repozitáře a všech potřebných knihoven pro správnou funkčnost. Následně byla naistalována kompatibilní verze platformy CUDA [44] a framework PyTorch [10]. Dále byla naistalována knihovna CUDNN [45].

Trénování probíhalo na stolní počítačové sestavě s procesorem AMD Ryzen 5 5600G, grafickou kartou Nvidia GeForce RTX 3060 12 GB a RAM 48 GB 3,6 GHz. Pro spuštění učení byl potřeba přístupový soubor obsahující cesty k obrázkům a textovým souborům s anotacemi z trénovací a validační složky datasetu. [46] Zvolený dataset obsahuje tyto informace ve formátu Scalabel. [47] Informace proto bylo nutné převést do správného formátu YOLO. Od autora BDD-100K existuje Python script pro převod do COCO formátu. [48] Pro převod do formátu YOLO byl k dispozici script v Pythonu od tvůrce YOLO. [49] Následně byl do vhodné složky vytvořen přístupový soubor *BDD-100k.yaml* pro učení modelů YOLOv8 na datasetu BDD-100K.



Obr. 34 Přístupový soubor *BDD-100k.yaml*

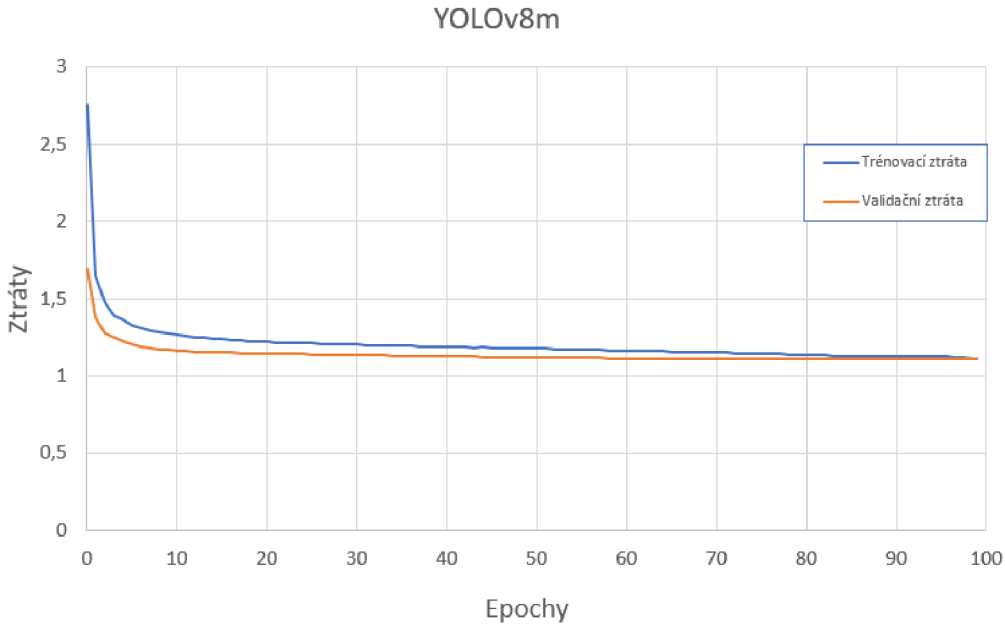
Po tomto kroku je možné spustit učení, které bylo provedeno pomocí příkazového řádku příkazem:

```
yolo detect train data=BDD-100k.yaml model=yolov8n.yaml epochs=100 imgsz=640 batch=60 device=0
```

V tomto příkazu je specifikováno, že platforma YOLOv8 bude trénovat model pro detekci objektů na datasetu popsaným v *BDD-100k.yaml*. Dále je v příkazu napsáno, který model bude trénován, počet epoch trénování, informace o velikosti formátu modelu, počet obrázků v dávce pro jednu iteraci a informaci o specifikaci zařízení které bude pro trénování použito, přičemž 0 znamená první grafická karta. Pro správné natrénování je potřeba hlídat trénovací a validační ztráty, aby nedošlo k přeučení sítě.

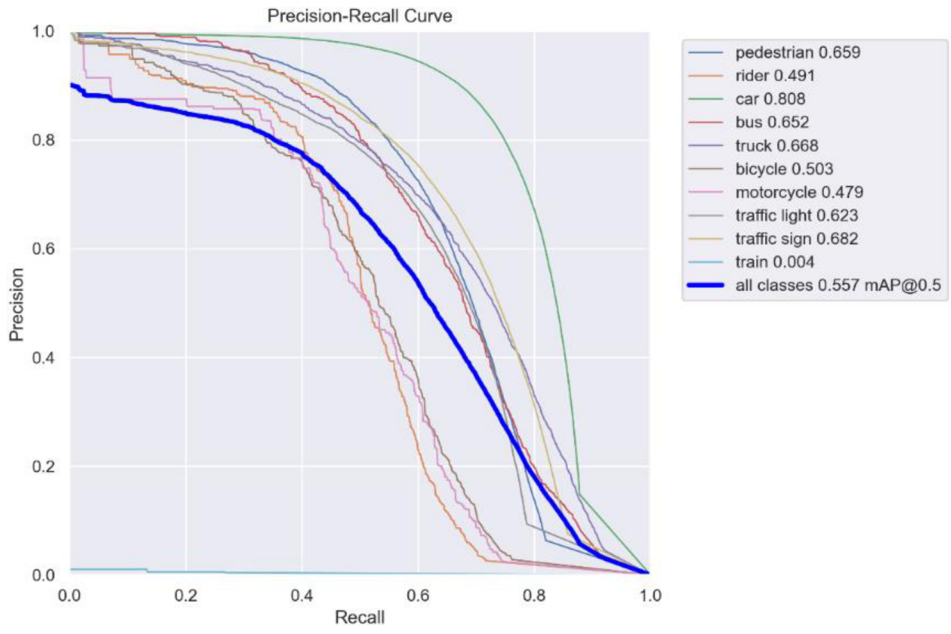
4.4 VÝSLEDKY TRÉNOVÁNÍ

Střední model YOLOv8m byl trénován 105 hodin. Z analýzy trénovacích a validačních ztrát na *Obr. 35* lze vyvodit, že přeučení modelu nenastalo a lze tedy předpokládat jeho správnou funkčnost.



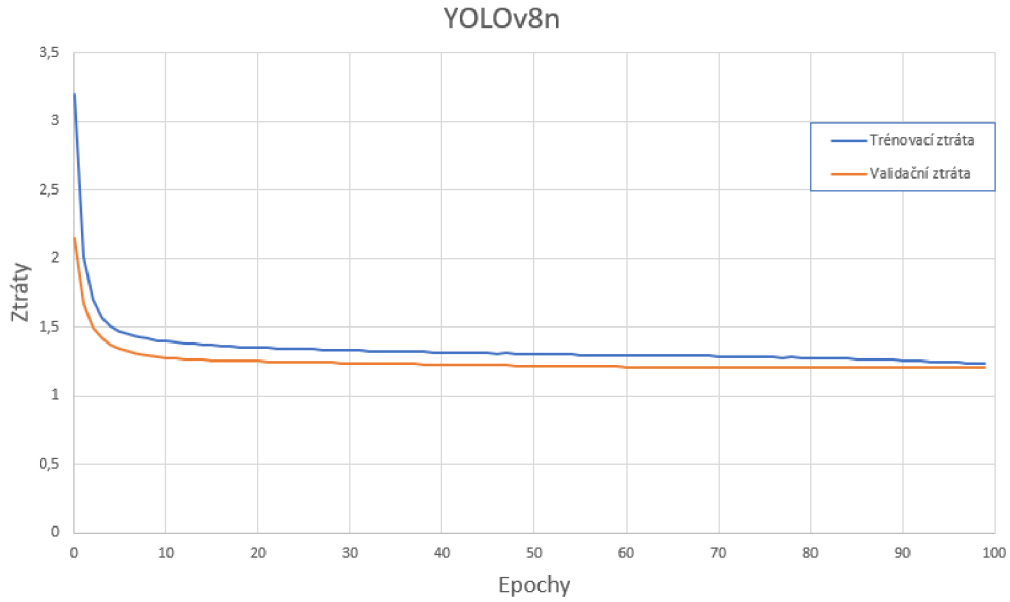
Obr. 35 Trénovací a validační ztráta modelu YOLOv8m

Na *Obr. 36* se nachází graf obsahující PR křivky, s hodnoty průměrné přesnosti (AP) pro trénované třídy, včetně průměru pro všechny třídy (mAP) při prahové hodnotě Intersection-over-Union (IOU) t=0,5. Z grafu vyplývá, že míra natrénování sítě odpovídá počtům instancí jednotlivých tříd. Toto lze nejlépe prezentovat na třídách vozidlo a vlak. Instancí třídy vozidlo bylo nejvíce, tudíž bylo její natrénování nejlepší a na druhé straně třída vlak, v níž je instancí minimum a síť je tak pro třídu vlak nepoužitelná. Celková hodnota natrénování sítě mAP@0.5 = 0,557 je horší v porovnání s hodnotou mAP@0.5 = 0,68, prezentovanou tvůrci platformy YOLO na datasetu COCO. [50] Pro třídu vozidlo je výsledná hodnota mAP@0.5 = 0,808, tudíž lze očekávat správnou detekci vozidel pomocí tohoto modelu.



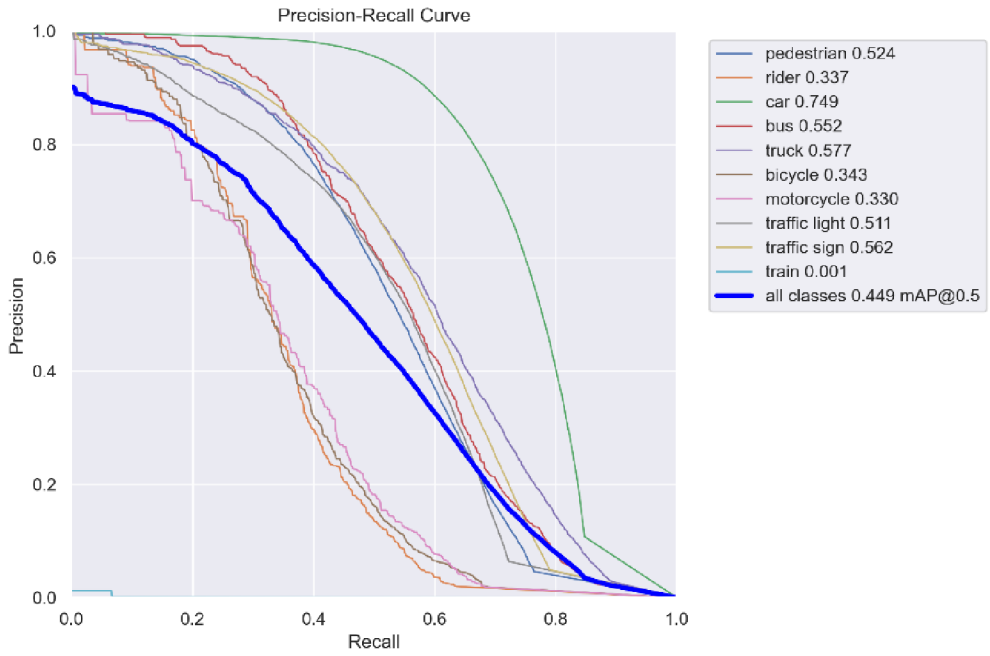
Obr. 36 Graf s PR křivkami pro YOLOv8m včetně hodnot AP a mAP

U nejmenšího modelu YOLOv8n trénování trvalo 30 hodin a probíhalo obdobně jako u středního modelu. Zde je vyšší trénovací a validační ztráta, která je způsobená menším modelem.



Obr. 37 Trénovací a validační ztráta modelu YOLOv8n

Podle ztrátové křivky bylo možné odhadnout, že i výsledné mAP bude menší v porovnání se středním modelem. Zde je opět nižší hodnota mAP@0,5 = 0,449, než mAP@0,5 = 0,53, které dle autorů dosahuje síť na datasetu COCO. Pro třídu vozidlo model dosahuje hodnoty mAP@0.5 = 0,749. Tudíž i u tohoto modelu lze očekávat správnou detekci vozidel.



Obr. 38 Graf s PR křivkami pro YOLOv8n včetně hodnot AP a mAP

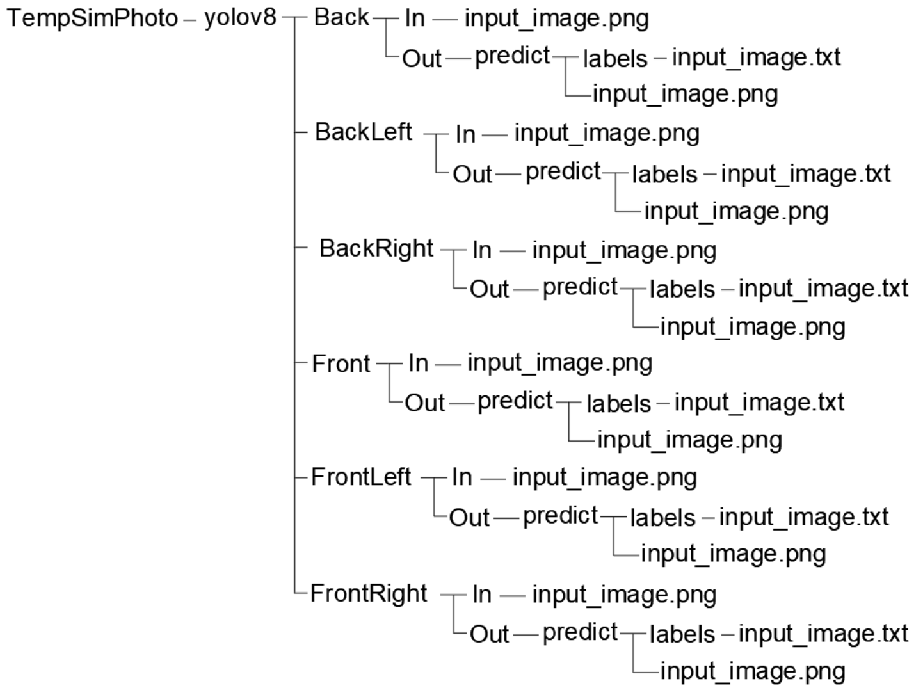
4.5 IMPLEMENTACE NEURONOVÉ SÍTĚ DO SIMULINKU

Pro vložení neuronové sítě do Simulinku byly pro jednotlivé kamery vytvořeny funkce, které v jednotlivých krocích ukládají fotky do přiřazeného adresáře zobrazeného na *Obr. 39*. Poté je příkazovým řádkem spuštěna platforma YOLOv8 pomocí funkce *system*, která zpracovává uložené snímky.

```
function [boxes, labels] = detectObjectYoloV8Front(image, time)

resWidth=1920;
resHeight=1080;

coder.extrinsic("imwrite","system","delete","readmatrix","exist","fullfile","cd")

labels=zeros(20,1);
boxes=zeros(20,5);

model=fullfile(cd, 'yoloWeight\best.pt');
imgPath = fullfile(cd, 'TempSimPhoto\yolov8\Front\In\input_image.jpg');
proj = fullfile(cd, 'TempSimPhoto\yolov8\Front\Out');
txtFile = fullfile(cd, ...
    'TempSimPhoto\yolov8\Front\Out\predict\labels\input_image.txt');

if exist(txtFile,"file")
    delete(txtFile)
end

imwrite(image, imgPath);

if not(isempty(image)) && time > 0.05
    % příkaz pro spuštění YOLO v cmd
    command = sprintf('yolo predict model="%s" source="%s" project="%s"
         save_txt=True exist_ok=True save=True conf=0.5', model, imgPath, proj);
    % Spustit příkaz
    [~, ~] = system(command);

    % Načíst výstup a extrahovat boundary boxy a popisky
    if exist(txtFile, 'file')
        boxes = readmatrix(txtFile);
        labels = boxes(:,1)+1;
        boxes(:,1) = boxes(:,2).*resWidht;
        boxes(:,2) = boxes(:,3).*resHeight;
        boxes(:,3) = boxes(:,4).*resWidht;
        boxes(:,4) = boxes(:,5).*resHeight;
        boxes(:,5) = 0;
    end
    boxes = boxes(any(boxes,2), :);
    labels = labels(any(labels,2), :);
end
```

Některé funkce použité v kódu nejsou podporovány pro kompilaci Simulinkem a proto musí být deklarovány jako vnější funkce pomocí *coder.extrinsic*. Následuje inicializace výstupních proměnných. Poté jsou v kódu nastaveny relativní cesty pro soubory. V první relativní cestě uložené v proměnné *model* je cesta sloužící k načtení naučeného modelu na platformě YOLOv8. Druhá relativní cesta uložená v *imgPath* slouží k uložení obrázku ke zpracování neuronovou sítí. Třetí relativní cesta uložená v *proj* slouží k uložení zpracovaných souborů,

kterými jsou obrázek pro vizualizaci a textový soubor s hodnotami souřadnic ohraničujících rámečků detekovaných objektů popsaných v kapitole 1.3.5. Poslední relativní cesta je ke zmíněnému textovému souboru a je uložená v *txtFile*. Při zpracování nového snímku se hodnoty ukládají na nové řádky ve stejném textovém souboru, je tedy třeba vždy smazat textový soubor pro vytvoření nového s aktuálními hodnotami. Poté je uložen nový snímek do příslušného adresáře pomocí funkce *imwrite*. Příkaz vkládaný do funkce *system* se skládá z příkazu spouštějící platformu YOLOv8 (`yolo`), poté následuje `predict` což znamená vyhodnocení predikce objektů na obrázku, poté jsou v příkazu vložené všechny relativní cesty k souborům s výjimkou *txtFile*. Následuje `save_txt`, sloužící k uložení textového souboru s hodnotami ohraničujících rámečků. Dále `exist_ok` značí ukládání do stejné složky a nevytváří inkrementálně nové složky. Následně `save` ukládá výstupní obrázek pro vizualizaci. Posledním je `conf`, který určuje minimální práh důvěryhodnosti objektů. Příkaz se spouští pouze v případě, že existuje snímek vygenerovaný simulací a čas simulace je větší než 0,05 s z důvodu inicializace simulace.



Obr. 39 Struktura adresáře

Po získání textového souboru s třídami a souřadnicemi detekovaných objektů zpracovaného snímku pomocí modelu YOLOv8 jsou informace ze souboru přečteny pomocí funkce *readmatrix*. Informace jsou následně rozděleny na očíslované třídy detekovaných objektů do proměnné *labels*. Hodnota všech tříd je zvýšena o 1, protože MATLAB využívá indexování od 1 a řádky s nulou jsou následně mazány. Ohraničující rámečky byly ponechány v proměnné *boxes* a jsou vynásobeny počtem pixelů v příslušném směru, aby odpovídaly souřadnicím původního obrazu. Byly ovšem posunuty na 1. až 4. sloupec v *boxes*. Následně byly vymazány řádky obsahující nuly. Proměnné *labels* a *boxes* slouží pro další zpracování.

4.6 POPIS ZÁKLADNÍCH FUNKCÍ ALGORITMU

V této kapitole jsou popisovány pouze významné části kódu jednotlivých funkcí. Celý algoritmus je uložen v příloze. Části, kde by byl popis funkcí zdlouhavý nebo složitý, jsou nahrazeny vývojovým diagramem. Pro vytvoření základu algoritmu bylo postupováno dle základní logiky, prezentované výše.

4.6.1 FUNKCE PRO VOZIDLO SE SENZORY

Funkce *EgoMove* slouží pro zpracování pohybu vlastního vozidla. Ze simulace byla získána data o poloze geometrického středu v lokálním absolutním kartézském systému. Pomocí získaných hodnot byla vypočítána rychlost, zrychlení a natočení vůči severu. Pro výpočet rychlosti byla použita rovnice (8) a pro výpočet zrychlení rovnice (9). Hodnoty jsou ve funkci ukládány do trvalých proměnných. Před uložením nové hodnoty jsou minulé hodnoty nejdříve posunuty o řádek níže pomocí funkce *circshift* a poslední řádek s hodnotami se vrátí na první a je přepsán nově získanými hodnotami. Tato metoda se nazývá FIFO (First In First Out).

$$v = \frac{\sqrt{\partial x^2 + \partial y^2}}{\partial t} \tag{8}$$

$$a = \frac{\partial v}{\partial t} \tag{9}$$

Kde v je rychlost, ∂x je změna polohy v ose X, ∂y je změna polohy v ose Y, ∂t je změna času, a je zrychlení, ∂v je změna rychlosti.

Natočení vozidla vůči severu bylo zjištěno pomocí rozdílů v osách X a Y. Nejdříve byl pomocí podmínky *if-elseif* zjištěn kvadrant směru pohybu a poté pomocí příslušné funkce dopočítáno přesné natočení vozidla, jak lze vidět na kódu níže.

```
if deltaX >= 0 && sign(deltaY) == 1 && not(deltaY == 0)
    orientCar = atand(deltaX/deltaY);
elseif deltaY == 0 && sign(deltaX) == 1
    orientCar = 90;
elseif sign(deltaX) == 1 && deltaY <= 0 && not(deltaX == 0)
    orientCar = -atand(deltaY/deltaX) + 90;
elseif deltaX == 0 && sign(deltaY) == -1
    orientCar = 180;
elseif deltaX <= 0 && sign(deltaY) == -1 && not(deltaY == 0)
    orientCar = atand(deltaX/deltaY) + 180;
elseif deltaY == 0 && sign(deltaX) == -1
    orientCar = 270;
else
    orientCar = -atand(deltaY/deltaX) + 270;
end
```

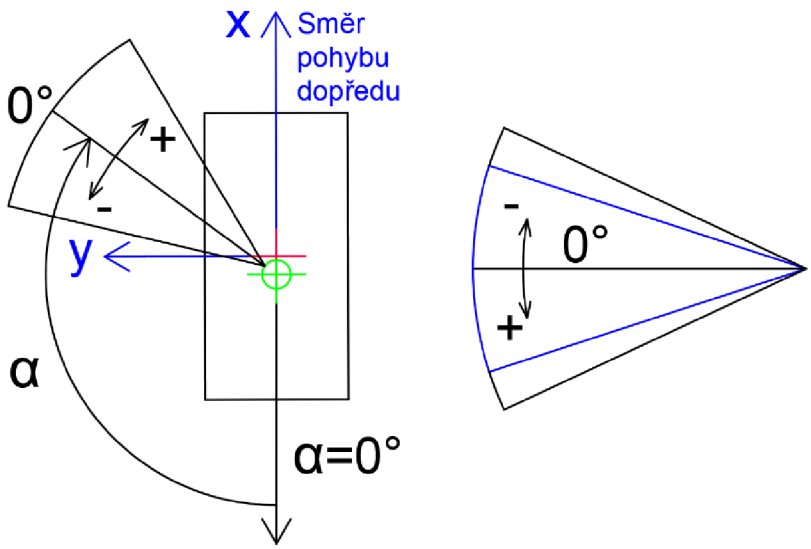
Hodnoty rychlosti, zrychlení a natočení byly uloženy do výstupní proměnné *ego* této funkce. Jsou využity pro predikování trajektorie vlastního vozidla ve funkci *AbsolutPredict,* která je popsána v kapitole 4.6.7. Dále je použito natočení vůči severu ve funkci pro převedení vzdálenosti a horizontálního úhlu do lokálního absolutního systému, která je popsaná v kapitole 4.6.6. Tato funkce *EgoMove* byla později nahrazena funkcí *EgoSimImu* popisovanou v kapitole 4.7.6.

4.6.2 FUNKCE PRO PŘEVOD Z OHRANIČUJÍCÍHO BOXU NA ÚHEL

Funkce *BoxToAngle* převádí souřadnice středů ohraničujících rámečků ze zpracovaných snímků pomocí modelu YOLOv8 na horizontální a vertikální úhly. Pro přepočítání je třeba znát zorné pole, rozlišení kamer a natočení kamer. Tyto hodnoty byly v algoritmu zadány jako konstanty.

```
midleOfBoxes = [(boxes (:,1)) (boxes (:,2))];

detectAngle(:,1) = (midleOfBoxes(:,1) .* horFov / imgSize(1)) - (horFov/2));
detectAngle(:,2) = (midleOfBoxes(:,2) .* vertFov / imgSize(2)) - (vertFov/2);
detectAngle(:,1) = detectAngle(:,1) + alfa; % otočení kamery
```

Nejprve byly hodnoty středů ohraničujících rámečků uloženy do proměnné *midleOfBoxes*. V prvním sloupci jsou uloženy horizontální souřadnice středů ohraničujících rámečků a ve druhém vertikální souřadnice středů polohy v kameře. Poté pomocí všech zmíněných údajů byly souřadnice přepočteny na horizontální a vertikální úhly a uloženy do proměnné *detectAngle*, která je výstupní proměnnou z funkce. Následně jsou horizontální úhly upraveny podle natočení kamery vůči LiDARu, který je znázorněn na následujícím obrázku. Horizontální natočení kamery je uloženo v proměnné *alfa*.



Obr. 40 Horizontální úhel ke kameře vpravo a vertikální zorný úhel kamery i LiDARu vlevo

4.6.3 FILTRACE TŘÍD DETEKOVANÝCH OBJEKTŮ

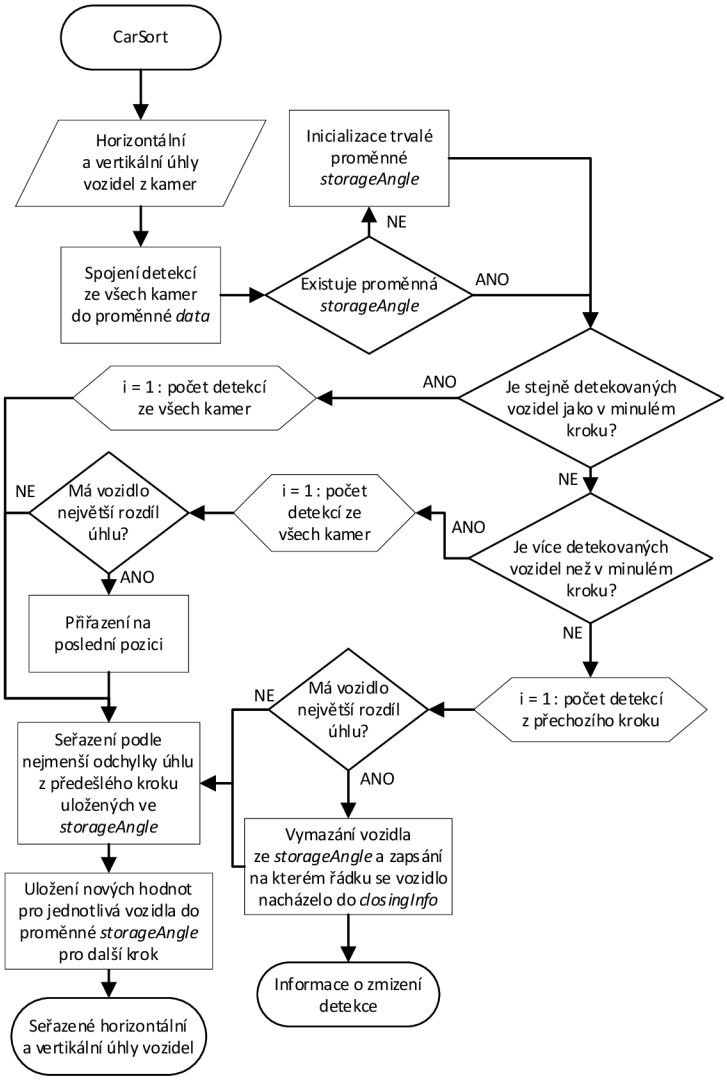
Tato práce je zaměřena na detekci a zpracování vozidel v obraze, proto je k tomuto účelu implementován kód, který filtruje vozidla ze všech detekovaných objektů. Tím v dalších krocích umožňuje práci pouze s vozidly.

```
for i = 1:size(boxes,1)
    switch labels(i,1)
        case 3
           carBBoxes(i,1:4) = boxes(i,1:4);
    end
end
```

V kódu byla použita podmínka *switch* z důvodu umožnění doplnění dalších tříd, jako například osob. Třídy jsou číselně označeny podle přístupového souboru, zobrazeného na *Obr. 34*, tudíž jsou přímo závislé na zvoleném datasetu. Informace o detekované třídě se nachází v proměnné *labels*. Tato podmínka *switch* byla vložena do cyklu *for*, který má počet iterací závislý na počtu detekovaných objektů. Tento kód se nachází na začátku ve funkci *BoxToAngle*. Po implementaci tohoto kódu byla upravena proměnná *midleOfBoxes*, protože jejím vstupem byla nově proměnná *carBBoxes*, namísto *boxes.*

4.6.4 FUNKCE TŘÍDĚNÍ HODNOT PRO JEDNOTLIVÁ VOZIDLA

V algoritmu byla vytvořena funkce *CarSort* třídění dat pro jednotlivá vozidla, protože model YOLOv8 neobsahuje jednoznačné určení detekovaných objektů. Nabízí informaci, jakou třídu detekoval, jak bylo znázorněno na *Obr. 9* v kapitole 1.3.5, ale nerozlišuje různé objekty jedné třídy. Touto funkcí *CarSort* se zamezí náhodnému přiřazování hodnot jednotlivým vozidlům. Zde je zobrazen zjednodušený vývojový diagram vyobrazující pouze základní logiku třídění. Funkce byla rozšířena a kompletní vývojový diagram je popsán v kapitole 4.7.4 na rozšířené verzi funkce.



Obr. 41 Logika třídění

Základní logika třídění vozidel je založena na třídění dle nejmenší odchylky horizontálního úhlu detekovaných vozidel oproti předchozímu kroku. Výstupem jsou v každém kroku seřazená vozidla podle řádků, na kterých se nacházely v minulem kroku. Samotné třídění začíná podmínkou IF, pomocí které je porovnáván počet vozidel aktuálně detekovaných s počtem aut detekovaných v minulém kroku. Z tohoto algoritmus získá informaci, zda je počet vozidel v aktuálním kroku stejný, větší nebo menší. Podle této informace se algoritmus rozhoduje, co bude provedeno, viz *Obr. 41*.

Při zmizení vozidla je poslána do funkce *AbsolutPredict* informace o tom, z jaké pozice vozidlo zmizelo. Data zmizelého vozidla jsou smazány pomocí posunutí všech následujících sloupců na pozici o jednu menší až do posledního inicializovaného sloupce a poslední je vynulován. Tím dojde k přepsání zmizelého vozidla. Toto mazání je aplikováno ve všech trvalých proměnných, formátu pole buněk.

4.6.5 FUNKCE PRO ZÍSKÁNÍ VZDÁLENOSTI

Další vytvořená funkce D*istanceFromLidar* slouží pro získání vzdálenosti detekovaného vozidla. Pro synchronizaci vertikálního a horizontálního úhlu zorného pole kamery s lidarem bylo vycházeno z *Obr. 40*. Poté byl vertikální a horizontální úhel detekovaného vozidla přepočítán na hodnoty odpovídající pozici řádku a sloupce ve kterých se hledaná vzdálenost nachází v matici vzdáleností generované LiDARem.

```
for i = 1:size(sortedAngle, 1)
    col = 1 + round(sortedAngle(i,1)/360 * (900-1));
    row = round((sortedAngle(i,2)/20) * 16)+8;

    distFromLid(i, 1) = lidarDist(row, col);
    distFromLid(i, 2) = sortedAngle(i, 1);
end
```

Hodnoty musely být zaokrouhleny, protože řádky a sloupce jsou pouze celočíselné hodnoty. Pomocí proměnných řádku *row* a sloupce *col* byla vyhledána vzdálenost pro jednotlivá i-tá vozidla. Hodnoty vzdáleností byly zapsány do výstupní proměnné *distFromLid*, která obsahuje informace o vzdálenosti v prvním sloupci a horizontálním úhlu pro jednotlivá detekovaná vozidla ve druhém sloupci.

$$\text{řádek} = 1 + round\left(\frac{m_{hor}}{360} * (Q_{hor} - 1)\right) \tag{10}$$

$$sloupec = round\left(\frac{m_{vert}}{FoV_{lidar}} * Q_{vert}\right) + \frac{Q_{vert}}{2} \tag{11}$$

Kde m_{hor} je horizontální úhel středu ohraničujícího rámečku, Q_{hor} počet generovaných bodů LiDARem v horizontálním směru, m_{vert} je vertikální úhel středu ohraničujícího rámečku, FoV_{lidar} je vertikální zorné pole lidaru, Q_{vert} počet generovaných bodů LiDARem ve vertikálním směru.

4.6.6 FUNKCE PŘEVEDENÍ VZDÁLENOSTI A HORIZONTÁLNÍHO ÚHLU DO LOKÁLNÍHO ABSOLUTNÍHO SYSTÉMU

Do této funkce *DetectXY* vstupuje proměnná *distFromLid* z předchozí funkce a natočení vlastního vozidla vzhledem k severu. Funkce nejprve převádí z polárních souřadnic dle *Obr. 40*, obsažených v *distFromLid,* do kartézských souřadnic vlastního vozidla dle normy SAE J670 s osou Z otočenou nahoru popsanou v kapitole 1.6. Získané souřadnice vlastního vozidla jsou v geometrickém středu, který je o 0.313 m před senzory, tudíž hodnota 0.313 m byla odečtena v podélném směru.

Poté funkce otáčí souřadnice detekovaných vozidel podle aktuálního natočení vlastního vozidla vůči severu v lokálních absolutních souřadnicích uloženého do *absRot.* Detekovaná vozidla jsou otočená o 90° proti směru hodinových ručiček z důvodu převodu souřadnic vlastního vozidla na absolutní kartézské souřadnice popsané v kapitole 1.6.1 na *Obr. 17.* Otočené souřadnice jsou uloženy ve výstupní proměnné *detCarPoss* z funkce.

```
for i = 1:size(distFromLid,1)
      vehiclePos(i,:) = [(distFromLid(i,1)*cosd(180-distFromLid(i,2)))-0.313
          (distFromLid(i,1)*sind(180-distFromLid(i,2)))];
end

absRot = (-ego(1,3)+90);

for i = 1:size(distFromLid,1)
   detCarPoss(i,1)= vehiclePos(i,1)*cosd(absRot) - (vehiclePos(i,2)*sind(absRot));
   detCarPoss(i,2)= vehiclePos(i,1)*sind(absRot) + (vehiclePos(i,2)*cosd(absRot));
end
```

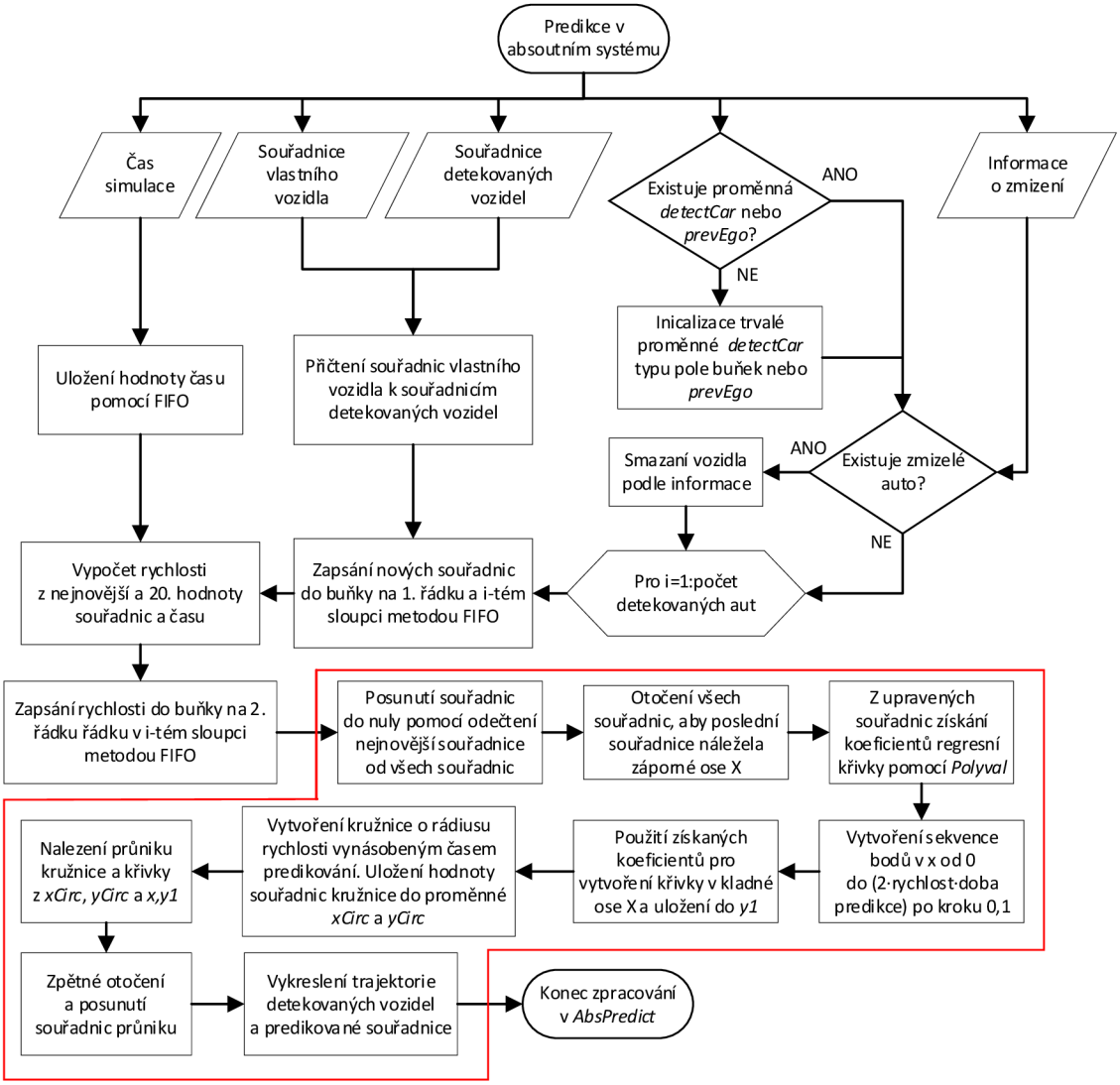
4.6.7 FUNKCE PREDIKCE TRAJEKTORIE V LOKÁLNÍM ABSOLUTNÍM SYSTÉMU

Funkce *AbsolutPredict* je funkce, ve které dochází k ukládání souřadnic detekovaných vozidel a vytváření predikce jejich trajektorie v lokální absolutní kartézské soustavě souřadnic prezentované v kapitole 1.6.1. Data všech detekovaných vozidel se ukládají do trvalé proměnné *detectCar*, která je ve formátu pole buněk. Pole buněk znamená, že každá buňka muže obsahovat matici hodnot. Jednotlivá detekovaná vozidla mají vlastní sloupec buněk pro výpočtové hodnoty. Jednotlivé sloupce odpovídají řádku detekovaného vozidla ze seřazených dat ve vstupní proměnné *detCarPoss.* Druhá vstupní proměnná obsahuje souřadnice vlastního vozidla pro vytvoření predikce trajektorie vlastního vozidla. Souřadnice vlastního vozidla jsou přičteny k detekovaným vozidlům, které poté mají správné souřadnice v lokálním absolutním systému. Další vstupní proměnnou je informace o ztraceném vozidle. Informace obsahuje, na kterém řádku, resp. v *detectCar* sloupci se vozidlo nalézalo, aby mohlo být smazáno. Mazání je prováděno stejně jako v kapitole 4.6.4. Poslední informací, vstupující do funkce, je čas pro výpočty. V simulaci byl použit čas simulace. Mimo simulaci by byly využity hodiny reálného času. Pro tento systém je potřeba mít informaci o poloze vlastního vozidla v každém kroku a definovaný nulový bod na mapě, ke kterému se poloha vlastního vozidla přepočítává. Poloha ze zeměpisných souřadnic by se získala pomocí rovnic (12) a (13). Tento výpočet je zjednodušen předpokladem, že země je koule. Zároveň je potřeba brát v potaz, že se zvětšující se vzdáleností od středu vzniká větší chyba, což by v aplikaci do provozu nebylo vhodné. Tato funkce byla vytvořena především pro získání hodnot odchylek detekovaných a predikovaných souřadnic vozidel ze simulace.

$$X = r * \tan(\lambda - \lambda_s) \tag{12}$$

$$Y = r * \tan(\varphi - \varphi_s) \tag{13}$$

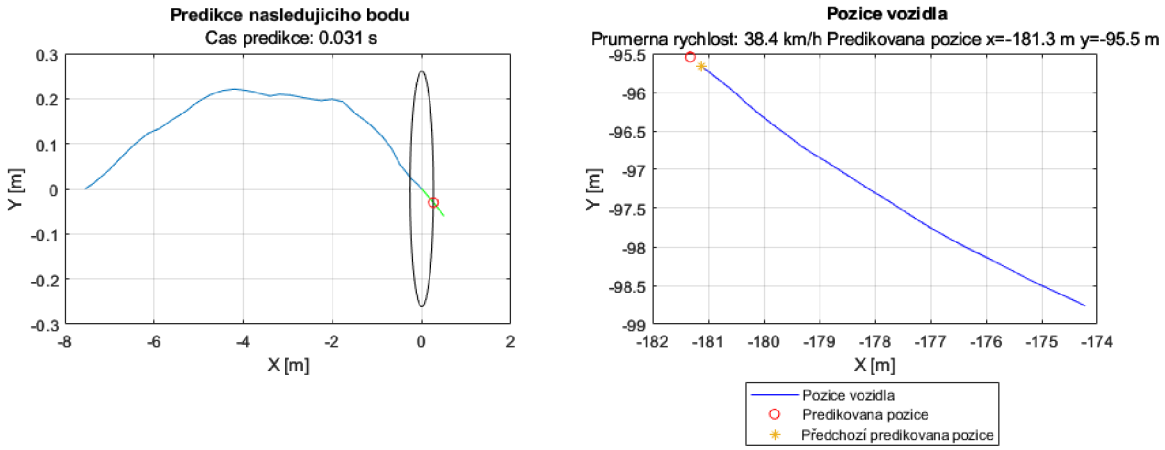
Kde X je osa X, r je rádius země, λ je zeměpisná délka vlastního vozidla, λ_s je zeměpisná délka nulového bodu, Y je osa Y, φ je zeměpisná šířka vlastního vozidla, φ_s je zeměpisná šířka nulového bodu.



Obr. 42 Diagram predikce v absolutním systému

Na tomto vývojovém diagramu je zobrazen postup vytváření predikcí pohybu pouze pro detekovaná vozidla. Pro výpočet rychlosti je použit rozdíl 1. a 20. nejnovější polohy detekovaných vozidel. Zároveň v této funkci probíhá predikování pohybu vlastního vozidla, které funguje analogicky jako u detekovaných vozidel s rozdílem, že z funkce *EgoMove* je již získána rychlost a zrychlení. Tyto hodnoty se společně se souřadnicemi zapisují do trvalé proměnné *prevEgo*. V novém kroku jsou nové hodnoty zapsány metodou FIFO zmíněnou v kapitole 4.6.1. Je zachycováno posledních 30 souřadnic detekovaných vozidel, které jsou využity k získání koeficientů regresní křivky. Pro predikci pohybu je využívána průměrná rychlost z 10 nejnovějších rychlostí.

Při programování funkce byly vytvořeny pomocné grafy, viz *Obr. 43*, pro vykreslení predikovaných souřadnic jednotlivých vozidel pro ověřování funkčnosti. Na těchto grafech lze názorně ukázat predikci souřadnic za definovanou dobu predikce. Pro vykreslení množiny bodů predikovaných souřadnic trajektorie, byla vytvořená třída *carPrediction*. Do této třídy byla vložená červeně ohraničená část kódu z *Obr. 42* . Následně červená část byla používaná jako statická metoda třídy. Statické metody jsou podobné funkcím, zjednodušují však programování, protože si není třeba pamatovat názvy funkcí, ale postačuje zadat například *carPrediction* a po napsání tečky za třídu již MATLAB nabídne metody, které třída obsahuje.



Obr. 43 Pomocné grafy pro predikci

VYTVOŘENÍ TŘÍDY S PREDIKCÍ

Dalším krokem bylo vytvoření třídy *carPrediction*, která obsahuje statickou metodu pojmenovanou *CarPredictRegres2,* predikující souřadnice. Pomocí metody je možné jednoduše měnit dobu predikce a díky tomu vytvořit množinu bodů souřadnic, které tvoří predikovanou trajektorii pro jednotlivá vozidla.

```
for j = 1:30
   [detectCar{5,i}(j,1), detectCar{5,i}(j,2)] =
      carPrediction.CarPredictRegres2(detectCar{1,i}, ...
          detectCar{2,i}, ...
          (j/10), ... %doba predikce [s]
          0); %číslo grafu
end
```

Na představené části kódu je ukázán princip tvorby predikované trajektorie pomocí nově vytvořené statické metody. Predikce souřadnic je vytvářena pro i-té vozidlo a je vytvořeno 30 predikovaných souřadnic od 0,1 do 3 sekund s časovým intervalem po 0,1 s. Vstupními daty jsou detekované souřadnice, rychlost, doba predikce a číslo grafu, který bude vykreslován. V případě zadání 0 nebude žádný graf vykreslen.

Zároveň nebylo vhodné, aby se již při 3 získaných bodech vykreslovala celá predikovaná trajektorie, proto byl definován minimální počet získaných bodů podle nerovnice (14). Tato nerovnice je využívána do získání 30 pozic detekovaného vozidla.

$$počet\ pozic > t_{Pred} * 10 \tag{14}$$

Kde t_{Pred} je čas predikce.

4.7 ROZŠÍŘENÍ ZÁKLADNÍHO ALGORITMU

Z důvodu zpřesnění algoritmu byla základní logika rozšiřována. Mezi rozšíření patří kontrola vzdálenosti detekovaných vozidel, predikce v relativním systému, spojení vozidel projíždějících mezi kamerami a zajištění pamatování falešně negativních vozidel.

4.7.1 VYTVOŘENÍ TŘÍDY S VÝPOČTOVÝMI METODAMI

Pro zvýšení přehlednosti kódu byla vytvořena třída *carCalcFunction*, do které byly vložené opakující se úryvky kódů potřebné pro výpočty. Z jednotlivých úryvků byly vytvořeny nové statické metody, které nahradily například výpočet rychlosti, zrychlení, úhel natočení od severu atd.

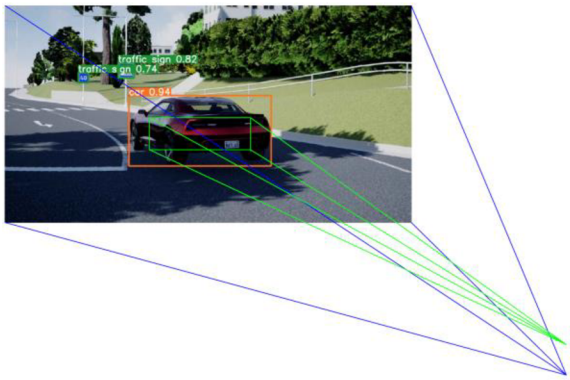
4.7.2 VÝSEČ Z MATICE VZDÁLENOSTÍ

Protože se na vozidle nenachází pouze jeden bod vzdálenosti získané z LiDARu byla provedena následující úprava. Díky ní je získáváno více hodnot vzdáleností, které se nacházejí na vozidle.

```
for i = 1:size(sortedAngle, 1)
  searchAngle(i,1) = 1 + ceil((sortedAngle(i,1)-(sortedAngle(i,3)/1.4))/360 * ...
      (size(LidarDist,2)-1));
  searchAngle(i,2) = 1 +floor((sortedAngle(i,1)+(sortedAngle(i,3)/1.4))/360 * ...
      (size(LidarDist,2)-1));
  searchAngle(i,3)=ceil((((sortedAngle(i,2)-(sortedAngle(i,4)/3.5))/fovLidar)* ...
      size(LidarDist,1))+ (size(LidarDist,1)/2));
  searchAngle(i,4)=floor((((sortedAngle(i,2)+(sortedAngle(i,4)/1.3))/fovLidar)*...
      size(LidarDist,1))+ (size(LidarDist,1)/2));
end

for i = 1:size(searchAngle, 1)
    %lidar výseč matice
    lidarBox = LidarDist(searchAngle(i,3):searchAngle(i,4), ...
       searchAngle(i,1):searchAngle(i,2));
    filteredPoints = carPointsFilter(lidarBox); %filtrace
    distFromLid(i, 1) = mean(filteredPoints,"all","omitnan");
    distFromLid(i, 2) = sortedAngle(i, 1);
end
```

V kódu bylo upraveno samotné přepočítávání, protože již není nutné hledat hodnoty středních úhlů pozic rámečků, ale jsou vyhledávány horizontální a vertikální úhly okrajů rámečků. Také byly využité jiné funkce zaokrouhlování, kterými jsou *ceil*, která slouží k zaokrouhlení na nejbližší vyšší a *floor* na nejbližší nižší. Výpočet znázorněný v kódu je zobrazen pomocí rovnic (15) a (16). Následně byly úhly k okrajům rámečků zmenšeny o konstanty. Pro pravý a levý okraj byla experimentálně zjištěna hodnota 1,4, pro spodní 1,3 a pro vrchní 3,5. Konstanta k vrchnímu okraji byla největší z důvodu skla, nalézajícího se na horní části vozidla, které je pro LiDAR neviditelné. Proměnná *searchAngle* obsahuje v 1. sloupci zmenšenou levou stranu rámečku, ve 2. pravou stranu, ve 3. se nachází horní a ve 4. dolní strana. Tyto hodnoty zmenšených rámečků, uložených v *searchAngle* jsou dále využity pro vyhledávání v řádkách a sloupcích. Při hledání v řádkách je začínáno vrchním řádkem, protože 1. řádek je v nejvyšší poloze. Vyhledávání ve sloupci je od levého po pravý. Hodnoty z matice vzdáleností jsou uloženy v proměnné *lidarBox*.

Poté bylo nutné tyto hodnoty filtrovat. Pro funkci filtrace je vyčleněna následující podkapitola. Poté byl z vyfiltrovaných hodnot vytvořen průměr, který je výstupní hodnotou vzdálenosti pro dané vozidlo. Zároveň byla automatizována změna parametrů LiDARu. Při jeho změně se přepíše pouze vertikální rozlišení podle nově zvoleného LiDARu.



Obr. 44 Znázornění vyhledávané výseče vzdáleností v LiDARu uložené v *lidarBox*

```
toBondary = [(carBBoxes(:,3)/2) (carBBoxes(:,4)/2)];

detectAngle(:,3) =  (toBondary(:,1) .* horFov / imgSize(1));
detectAngle(:,4) =  (toBondary(:,2) .* vertFov / imgSize(2));
```

Proto aby mohla být tato část kódu implementována, musely být ve funkci *BoxToAngle,* představené v kapitole 4.6.2, přidány do výstupní proměnné *detectAngle* informace o úhlech od středů ke krajům rámečků. Úhly od středů ke krajům rámečků jsou získány přepočítáním hodnot velikostí rámečků pomocí úhlu zorného pole kamery v jednotlivých osách. Velikosti rámečků byly získány z textového souboru a uloženy do *toBoundary*.

$$Levý\ kraj\ = 1 + ceil\left(\frac{m_{hor} - \frac{b_{hor}}{k}}{360} * (Q_{hor} - 1)\right) \tag{15}$$

$$Horní\ kraj\ = ceil\left(\frac{m_{vert} - \frac{b_{vert}}{k}}{FoV_{lidar}} * Q_{vert}\right) + \frac{Q_{vert}}{2} \tag{16}$$

Kde b_{hor} je horizontální úhel od středu ke kraji, k je zmenšující koeficient, b_{vert} je vertikální úhel od středu ke kraji.

FUNKCE FILTROVÁNÍ HODNOT VZDÁLENOSTÍ

Jak již bylo zmíněno v minulé kapitole, aby získání vzdálenosti ve výseči bylo co nejpřesnější, je třeba odfiltrovat všechny okolní body, v případě že se nenachází na vozidle. Z tohoto důvodu byla vytvořena funkce s názvem *carPointsFilter*.

```
value = mode(round(lidarBox/10)*10,"all");

filteredPointsHard = zeros(size(lidarBox,1),size(lidarBox,2));
```

```
for k = 1:size(lidarBox, 1)
    for j = 1:size(lidarBox, 2)
        if lidarBox(k,j) > value-10 && lidarBox(k,j) < value+10
           filteredPointsHard(k,j)=lidarBox(k,j);
        else
           filteredPointsHard(k,j)=NaN;
        end
    end
end
```

Do funkce vstupuje proměnná *lidarBox,* ve které jsou uloženy vzdálenosti získané výsečí popsané v předchozí kapitole. Před zjištěním nejpočetnější hodnoty pomocí *mode* byly vzdálenosti zaokrouhleny na desítky. Tím je zajištěno, že se nejčastěji vyskytující hodnota vzdálenosti bude nalézat na vozidle. Všechny hodnoty, které přesahují vzdálenost 10 metrů od nejpočetnější hodnoty jsou odfiltrovány. Tato hodnota 10 m byla zjištěna experimentálně jako nejvhodnější. Podmínce vzdálenosti od nejpočetnější hodnoty byly všechny body vystaveny pomocí smyčky *for* s vnořenou smyčkou *for*. Když hodnota nevyhovovala podmínce, byla nastavena na hodnotu NaN. Poté byl tento filtr aplikován ještě jednou se zaokrouhlením na jednotky metrů a vzdáleností od nejpočetnější hodnoty 5 m.

4.7.3 SPOJENÍ VOZIDEL PROJÍŽDĚJÍCÍCH MEZI KAMERAMI

Toto vylepšení algoritmu řeší problém s projíždějícím vozidlem mezi kamerami, pomocí sloučení dvou rámečků ze sousedních kamer. Před implementací tohoto kódu docházelo při projíždění vozidel mezi kamerami k mystifikaci algoritmu, který vyhodnocoval, že se v obraze nachází více vozidel. Kód se nalézá ve funkci *CarSort* před tříděním.

```
if size(preDetAngles,2)>=2
    for i = 1 : size(preDetAngles,1)-1
        for j = i+1 : size(preDetAngles,1)
            if (preDetAngles(i,6) > 1870 && ...
                    preDetAngles(j,5) < 50 && ...
                    not(preDetAngles(i,7)==0) && ...
                    ((preDetAngles(i,7)==(preDetAngles(j,7)-1)) || ...
                    (preDetAngles(i,7)==6 && preDetAngles(j,7)==1)))
                %ošetření po směru
                preDetAngles(i,1) = (preDetAngles(i,1)+preDetAngles(j,1))/2;
                preDetAngles(i,2) = (preDetAngles(i,2)+preDetAngles(j,2))/2;
                preDetAngles(i,3) = preDetAngles(i,3)+preDetAngles(j,3);
                preDetAngles(i,4) = (preDetAngles(i,4)+preDetAngles(j,4))/2;

                preDetAngles(j,:) = 0;
                break
            elseif (preDetAngles(i,5) < 50 && ...
                    preDetAngles(j,6) > 1870 && ...
                    not(preDetAngles(i,7)==0) && ...
                    (preDetAngles(i,7)==(preDetAngles(j,7)+1) || ...
                    (preDetAngles(i,7)==1 && preDetAngles(j,7)==6)))
                %ošetření v protisměru
                preDetAngles(i,1) = (preDetAngles(i,1)+preDetAngles(j,1))/2;
                preDetAngles(i,2) = (preDetAngles(i,2)+preDetAngles(j,2))/2;
                preDetAngles(i,3) = preDetAngles(i,3)+preDetAngles(j,3);
                preDetAngles(i,4) = (preDetAngles(i,4)+preDetAngles(j,4))/2;
```

```
                preDetAngles(j,:) = 0;
                break
            else
                continue
            end
        end
    end
    %odstranění již nepotřebných a vynulovaných hodnot
    preDetAngles = preDetAngles(any(preDetAngles,2),:);
    data = preDetAngles(:,1:4);
end
```

Celý zmíněný kód začíná podmínkou, která spouští kód při detekci 2 a více vozidel. Následně vnořená smyčka *for* do druhé smyčky *for* páruje jednotlivé dvojice vozidel bez jejich opakování. Podmínka if porovná, zdali se dvojice nachází na sousedních kamerách na jejich odpovídajících stranách. Tedy že pravý okraj 1. rámečku vozidla se nachází na pravém okraji kamery a levý okraj 2. rámečku vozidla na levém okraji sousední kamery. Zároveň kamera 2. rámečku musí být o jedno vyšší než kamera 1. rámečku, nebo musí být 1. rámeček vozidla v 6. kameře a 2. rámeček vozidla v 1. kameře. Tato podmínka kontroluje vozidla pouze po směru hodinových ručiček. Proto ještě tato část kódu obsahuje analogickou podmínku *elseif*, která kontroluje vozidla v protisměru. Když je podmínka splněna, jsou středy ohraničujících rámečků zprůměrovány a horizontální úhly ke krajům sečteny. Proměnná *preDetAngles* vznikla spojením detekcí ze všech kamer. V 7. sloupci *preDetAngles* se nachází informace, z jaké kamery detekce pochází. Číslování kamer je znázorněno v kapitole 3.5.2 na *Obr. 31* .

Proto aby mohla být tato část kódu implementována, musely být ve funkci *BoxToAngle* přidány další 2 sloupce do proměnné *detectAngle*, které jsou zobrazeny níže. Přidané sloupce obsahují hodnoty souřadnic levé a pravé strany ohraničujících rámečků v pixelech.

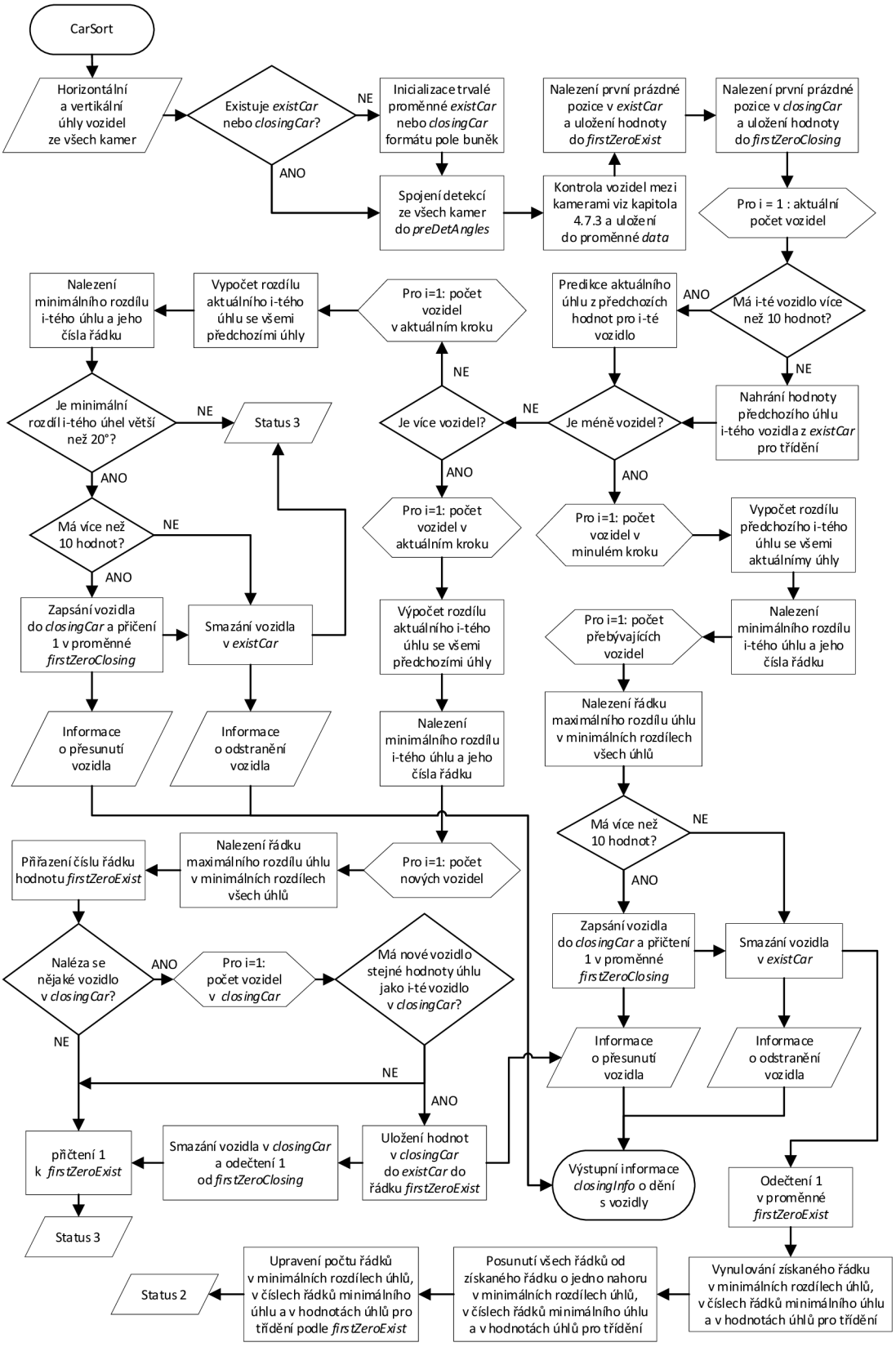
```
detectAngle(:,5) = carBBoxes(:,1)-(carBBoxes(:,3)/2);
detectAngle(:,6) = carBBoxes(:,1)+(carBBoxes(:,3)/2);
```

4.7.4 TŘÍDĚNÍ VOZIDEL S PAMATOVÁNÍM VOZIDEL ZMIZELÝCH Z OBRAZU

Předchozí třídění vozidel nebylo dostatečné z důvodu potřeby zachování informací o falešně negativních vozidlech, které se nalézaly v kameře, ale nebyly detekovány. Z důvodu velikosti byl diagram rozdělen na dvě části. První obsahuje logiku třídění s vyhledáním řádku s nejmenší odchylkou a druhá část ošetření zmizelých vozidel a seřazení detekovaných vozidel do výstupní proměnné.

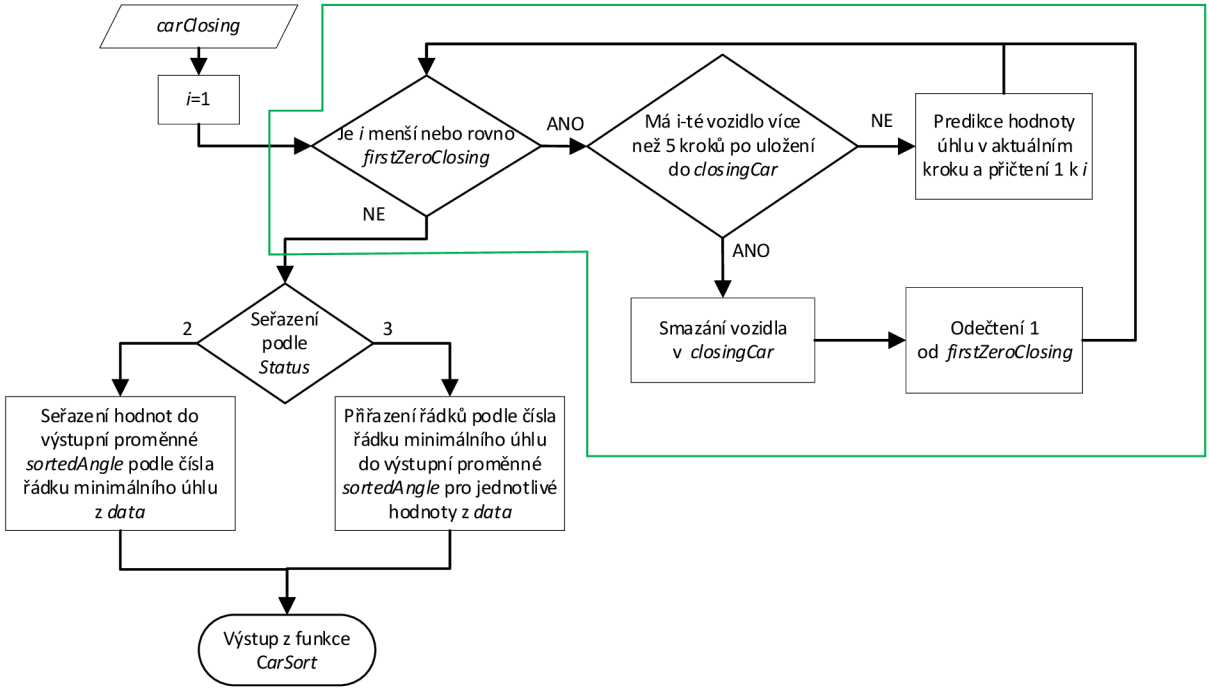
Ve všech lokálních proměnných, vstupních a výstupních jsou hodnoty jednotlivých vozidel řazeny v řádkách, ale v trvalých proměnných *existCar* a *closingCar,* které jsou ve formátu pole buněk, jsou vozidla řazeny do sloupců. Výstupní proměnná *closingInfo* obsahuje jednotlivé přesuny vozidel mezi proměnnými *existCar* a *closingCar*. Tyto přesuny jsou zapsány zvlášť na řádek v *closingInfo*, který má 4 sloupce a slouží pro informování ostatních funkcí. První sloupec znamená přesunutí z *existCar* na první volný sloupec do *closingCar* a uvedená hodnota značí na jakém řádku, v případě trvalých proměnných sloupci, se vozidlo nacházelo. Druhý a třetí sloupec slouží pro vrácení vozidla z *closingCar* do *existCar*. Druhý sloupec obsahuje informace, ve které pozici se nalézá vozidlo v *closingCar* a třetí do kterého

sloupce v *existCar* má být přesunuto. V posledním sloupci je informace o přímém mazání vozidla v *existCar* když má méně než 10 hodnot získaných souřadnic.



Obr. 45 Logika rozšířené funkce *carSort*

Diagram na *Obr. 45* je ukončen statusem, který slouží ke správnému seřazení výstupních hodnot pro jednotlivá vozidla, viz *Obr. 46*, protože nelze vždy porovnávat jednotlivé aktuálně získané úhly s předchozími. Když je například v aktuálním kroku méně vozidel než v předchozím kroku, je potřeba porovnat jednotlivé předchozí úhly s aktuálními, aby bylo nalezeno vozidlo, které zmizelo z obrazu. Poté musí být hodnoty aktuálních úhlů seřazeny vhodným způsobem. V následujícím obrázku s diagramem je zobrazen zbylý postup funkce. Nejprve je v *closingCar* provedena zeleně ohraničená smyčka, která vytvoří nové hodnoty nebo v případě, že vozidlo zmizelo před více než 5 kroky, je smazáno. Poté jsou hodnoty detekovaných vozidel vhodně seřazeny do výstupní proměnné.



Obr. 46 Dodatek funkce *CarSort*

Po rozšíření funkce *CarSort* musely být upraveny všechny funkce obsahující trvalou proměnou s daty detekovaných vozidel, aby byly schopné mazat, přesouvat a vytvářet nové hodnoty pro zmizelé vozidla a k tomu provádět smyčku zobrazenou zeleným rámečkem s predikcí hodnot potřebných v dané funkci.

4.7.5 KONTROLA SPRÁVNOSTI VZDÁLENOSTI

Dalším krokem pro rozšíření celého algoritmu bylo přidání možnosti vložení odhadované vzdálenosti namísto reálné při neočekávané situaci. Kvůli tomu byla vytvořena ve funkci *DistanceFromLidar* trvalá proměnná *distCar*, formátu pole buněk, která ukládá hodnoty vzdáleností jednotlivých vozidel. K tomu do třídy *carPrediction* byla vytvořena statická metoda *PredictDistFromMat* pro získání následující pravděpodobné vzdálenosti detekovaného vozidla. Tato část kódu se nalézá ve stejném cyklu pod výsečí z LiDARu popsanou v kapitole 4.7.2.

```
PredDist = carPrediction.PredictDistFromMat(distCar{1,i}(:,1));
distCar{1,i}= circshift(distCar{1,i}, [1, 0]);
```

```
cont = size(distCar{1,i}(any(distCar{1,i}(:,3),2),3),1);

if (meanDist < (distCar{1,i}(2,1) - (distCar{1,i}(2,1)*0.2)) || ...
  meanDist > (distCar{1,i}(2,1) + (distCar{1,i}(2,1)*0.2))) && cont>10
    distCar{1,i}(1,1) = PredDist(1,1);
    distCar{1,i}(1,2) = sortedAngle(i,1);
    distCar{1,i}(1,3)= 0;
else
    distCar{1,i}(1,1) = meanDist;
    distCar{1,i}(1,2) = sortedAngle(i,1);
    distCar{1,i}(1,3)= 1;
end
```

Hned po získání zprůměrované vzdálenosti je z předchozích hodnot vzdáleností proveden odhad vzdálenosti pro aktuální krok, která slouží pro porovnání s aktuální detekovanou hodnotou vzdálenosti.

Poté jsou hodnoty posunuty o řádek níže, aby pomocí principu FIFO mohla být vložena nová hodnota vzdálenosti, než je vložená jsou provedeny následující operace: spočítání řádků obsahujících 1 ve třetím sloupci, značící vzdálenost získanou z LiDARu a 0 značí vložení odhadované vzdálenosti. Tento počet řádků obsahujících 1 je uložen do proměnné *cont*. V případě více než 10 získaných vzdáleností je následující vzdálenost porovnávána s odhadovanou a v případě odchylky větší než 20 % je vložena vzdálenost odhadovaná, v opačném případě je vložená vzdálenost detekovaná. Jelikož i nadále hodnoty vzdálenosti oscilovali, byl aplikován plovoucí průměr.

STATICKÁ METODA *PREDICTDISTFROMMAT*

Kód začíná vymazáním, nulových řádků. Poté následuje vytvoření pozic v záporné ose X, kterým je přiřazena vzdálenost jako hodnota Y. Poté je možné vyhodnotit koeficienty křivky. Pomocí nich je následně získána vzdálenost v bodě X = 1, která odpovídá odhadované vzdálenosti. Tento přístup funguje pouze s konstantním časem zpracování kroků.

```
function y_DistPred = PredictDistFromMat(matDistFromLid)
   distPoints = matDistFromLid(any(matDistFromLid(:,1),2),1);
   xForPoints = 0:-1:-14;
   distFitCurve = polyfit(xForPoints(1:size(distPoints,1)), distPoints,2);
   y_DistPred = polyval(distFitCurve, 1);
end
```

4.7.6 NOVÁ FUNKCE PRO VOZIDLO SE SENZORY SIMULUJÍCÍ IMU

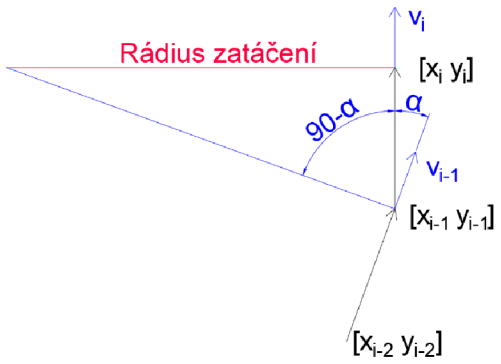
Po vytvoření třídy s výpočtovými metodami v kapitole 4.7.1 byla funkce *EgoMove* přebytečná a z tohoto důvodu se výpočet zmíněný v kapitole 4.6.1 přesunul do funkce *AbsolutPredict*. Funkce *EgoMove* byla nahrazena funkcí *EgoSimImu*. Tato funkce zpracovává pohyb vlastního vozidla pro simulování IMU senzoru. Ze známých poloh v čase je možné vytvořit podélné i příčné zrychlení, úhlovou rychlost v ose Z a úhel natočení vůči severu. Podélné zrychlení a natočení vůči severu je získáno stejným způsobem jako v kapitole 4.6.1 pomocí výpočtových metod. Pro výpočet bočního zrychlení je vycházeno z rovnice (17), pro kterou je potřeba získat rádius zatáčení. Pro zjištění rádiusu zatáčení bylo využito zjednodušujícího předpokladu, že rádius zatáčení je v každém bodě kolmý na rychlost vozidla, jak je znázorněno na *Obr. 47*. Pro získání úhlové rychlosti okolo osy Z je vycházeno

z rovnice (18). Je třeba brát v potaz, že tato funkce simuluje IMU senzor pouze teoreticky. Při získávání hodnot z reálného IMU by hodnoty byly zatíženy chybou senzoru, který by způsoboval chybu v pohybu vlastního vozidla. Výstupy této funkce budou využity v následující kapitole pro predikci v relativním systému. Natočení vůči severu je využito ve funkci *DetectXY*.

$$a_n = \frac{v^2}{r} \tag{17}$$

$$\omega_Z = \frac{\partial \alpha}{\partial t} \tag{18}$$

Kde a_n je normálové zrychlení, r je rádius zatáčení, v je rychlost, ω_Z je úhlová rychlost, $\partial\alpha$ změna natočení vlastního vozidla.



Obr. 47 Znázornění zjednodušeného výpočtu rádiusu zatáčení

4.7.7 PREDIKCE V RELATIVNÍM SYSTÉMU

Poslední vytvořená funkce slouží k predikci pohybu v lokálním relativním kartézském systému prezentovaným v kapitole 1.6.2. V tomto systému může být využita inerciální měřící jednotka. Proto byla vytvořena funkce *EgoSimImu* ze které jsou čerpány informace o zrychlení a úhlové rychlosti vozidla.

Na začátku simulace je provedeno inicializování rychlosti a směru, protože rychlost vlastního vozidla je na začátku simulace 14 m/s a pro zachování os je vhodné inicializovat natočení vozidla vůči severu. Pomocí následujících rovnic jsou získány informace o dráze vlastního vozidla mezi minulým a aktuálním krokem.

$$\varphi_i = \omega_Z \cdot \partial t + \varphi_{i-1} \tag{19}$$

$$a_x = a_F \cdot \sin(\varphi_i) + a_n \cdot \sin(\varphi_i - 90) \tag{20}$$

$$a_y = a_F \cdot \cos(\varphi_i) + a_n \cdot \cos(\varphi_i - 90) \tag{21}$$

$$v_{x,i} = v_{x,i-1} + a_x \cdot \partial t \tag{22}$$

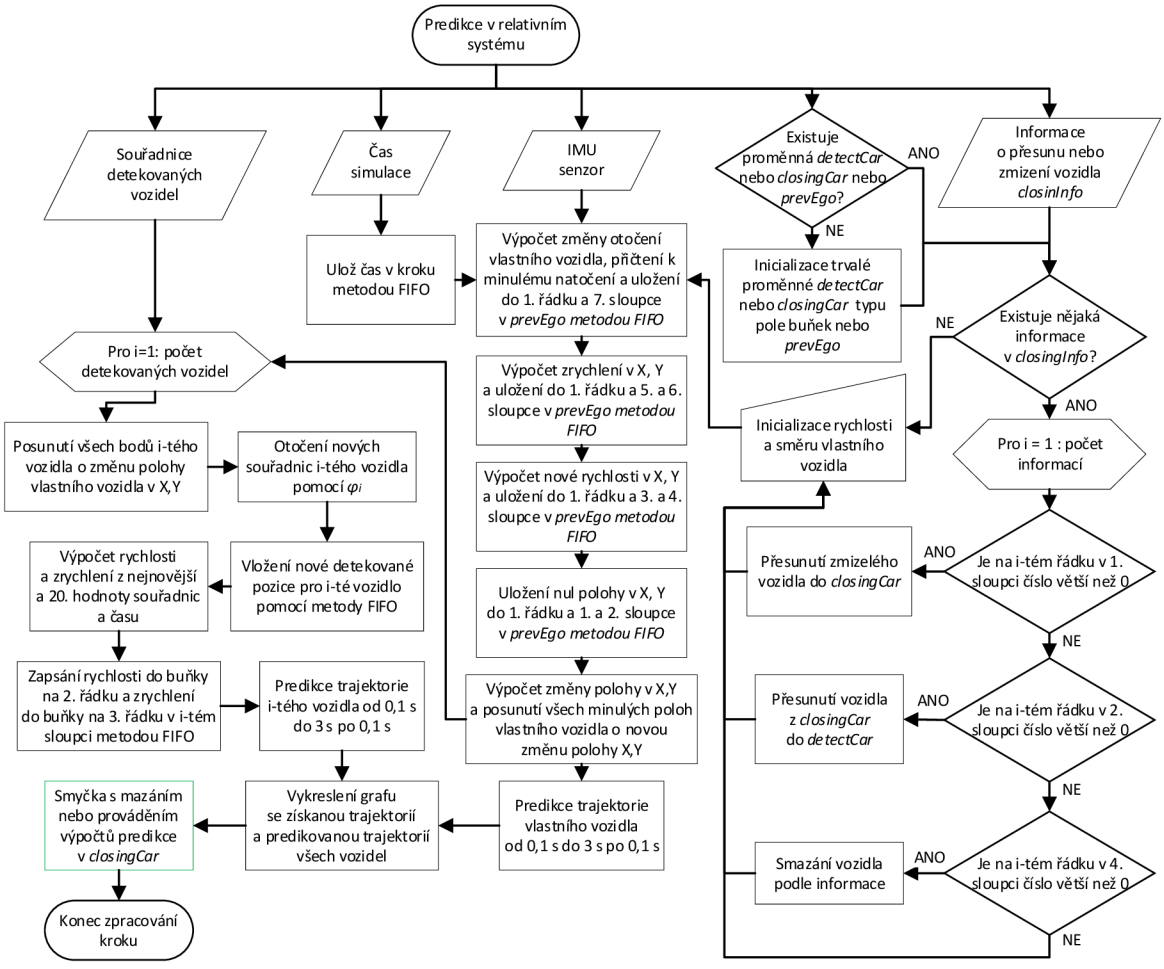
$$v_{y,i} = v_{y,i-1} + a_y \cdot \partial t \tag{23}$$

$$\partial X = \frac{a_x}{2} \cdot \partial t^2 + v_{x,i} \cdot \partial t \tag{24}$$

$$\partial Y = \frac{a_y}{2} \cdot \partial t^2 + v_{y,i} \cdot \partial t \tag{25}$$

Kde φ_i je aktuální natočení vozidla, ω_Z je úhlová rychlost za změnu času ∂t, φ_{i-1} je natočení vozidla v minulém kroku, a_x je zrychlení v ose X, a_F zrychlení v podélném směru, a_y je zrychlení v ose Y, $v_{x,i}$ je rychlost v ose X v aktuálním kroku, $v_{x,i-1}$ je rychlost v ose X v minulém kroku, $v_{y,i}$ je rychlost v ose Y v aktuálním kroku, $v_{y,i-1}$ je rychlost v ose Y v minulém kroku, ∂X posunutí vozidla v ose X, ∂Y posunutí vozidla v ose Y.

Vypočítané hodnoty dráhy jsou využity pro přepočítání všech minulých hodnot souřadnic vlastního vozidla i detekovaných vozidel, jelikož nulový bod zůstává na vlastním vozidle, musí být posunuty všechny minulé souřadnice o ujetou dráhu vlastním vozidlem. Pro predikci v relativním systému byl ve funkci *DetectXY* vytvořen nový přepočet do kartézských souřadnic, do kterého není zahrnut posun ke středu vozidla. Výstupní souřadnice detekovaných vozidel z funkce *DetectXY* jsou v souřadném systému vozidla dle *Obr. 15* v kapitole 1.6. Jelikož inicializace natočení vlastního vozidla proběhla od severu, což je osa Y je nutné detekovaná vozidla otočit o 90°. V relativním systému by probíhala predikce pohybu při aplikaci do provozu.



Obr. 48 Vývojový diagram relativního systému

5 OVĚŘENÍ FUNKČNOSTI

Pro ověření funkčnosti algoritmu, jsou představeny získané výsledky simulace, které jsou v následujících podkapitolách rozebírány podrobněji. Výsledky ze simulace jsou vyobrazeny na příslušných grafech.

5.1 POROVNÁNÍ MODELŮ YOLO

Pro porovnání modelů naučených na platformě YOLOv8 bylo spočítáno počet výskytů vozidel, počet správných detekcí vozidel, počet falešných detekcí vozidel a počet nedetekovaných vozidel i přesto, že se nacházely v obraze. Celkový počet získaných snímků byl 1770. Hodnoty jsou vyobrazeny v *Tab. 5*.

Tab. 5 Porovnání naučených modelů YOLO

Model	Počet výskytů vozidel	Počet správných detekcí	Počet falešně negativních	Počet falešně pozitivních	Procento správných detekcí	Procento falešně pozitivních
YOLOv8n	507	459	48	18	90,5 %	3,6 %
YOLOv8m	507	499	8	2	98,4 %	0,3 %

Dále pro modely YOLOv8m i YOLOv8n byly zjištěny rychlosti, kterých dosahují při zpracovávání snímků na grafické kartě NVIDIA RTX 3060 12 GB. Pro menší model byla průměrná rychlost 25 ms při 32 % využití grafické karty. Pro střední model byla průměrná rychlost 29,9 ms při 36 % využití grafické karty. V přepočtu 38,5 FPS pro malý model a 33,4 FPS pro střední. Na základě získaných rychlostí zpracování bylo zjištěno, že pro zpracování pomocí obou modelů, by byla potřeba dvounásobná výpočetní kapacita grafické karty pro naučené modely pro práci při 30 snímcích za sekundu. Protože střední model se jeví jako vhodnější, výsledky simulace budou získány pomocí modelu YOLOv8m. Rychlosti zpracování jsou specifikovány níže. První rychlost zpracování snímků je pro menší model YOLOv8n.

```
Speed: 1.6ms preprocess, 18.4ms inference, 5.0ms postprocess per image at shape
(1, 3, 640, 640)

Speed: 0.6ms preprocess, 24.6ms inference, 4.7ms postprocess per image at shape
(1, 3, 640, 640)
```

Obr. 49 Ukázka snímků zpracovaných pomocí YOLOv8m

5.2 VYKRESLENÍ DETEKOVANÝCH VOZIDEL S TRAJEKTORIÍ PREDIKCE

Prvním výsledným grafem je graf trajektorie detekovaných vozidel s predikcí jejich trajektorie do 3 s, vykreslená každou sekundu. Na této mapě lze vidět, že predikování trajektorie vozidel je funkční a přibližuje se skutečné trajektorii po které vozidlo pojede. Zároveň si lze všimnou že došlo k překřížení predikcí trajektorie vlastního vozidla a vozidla červený muscle.

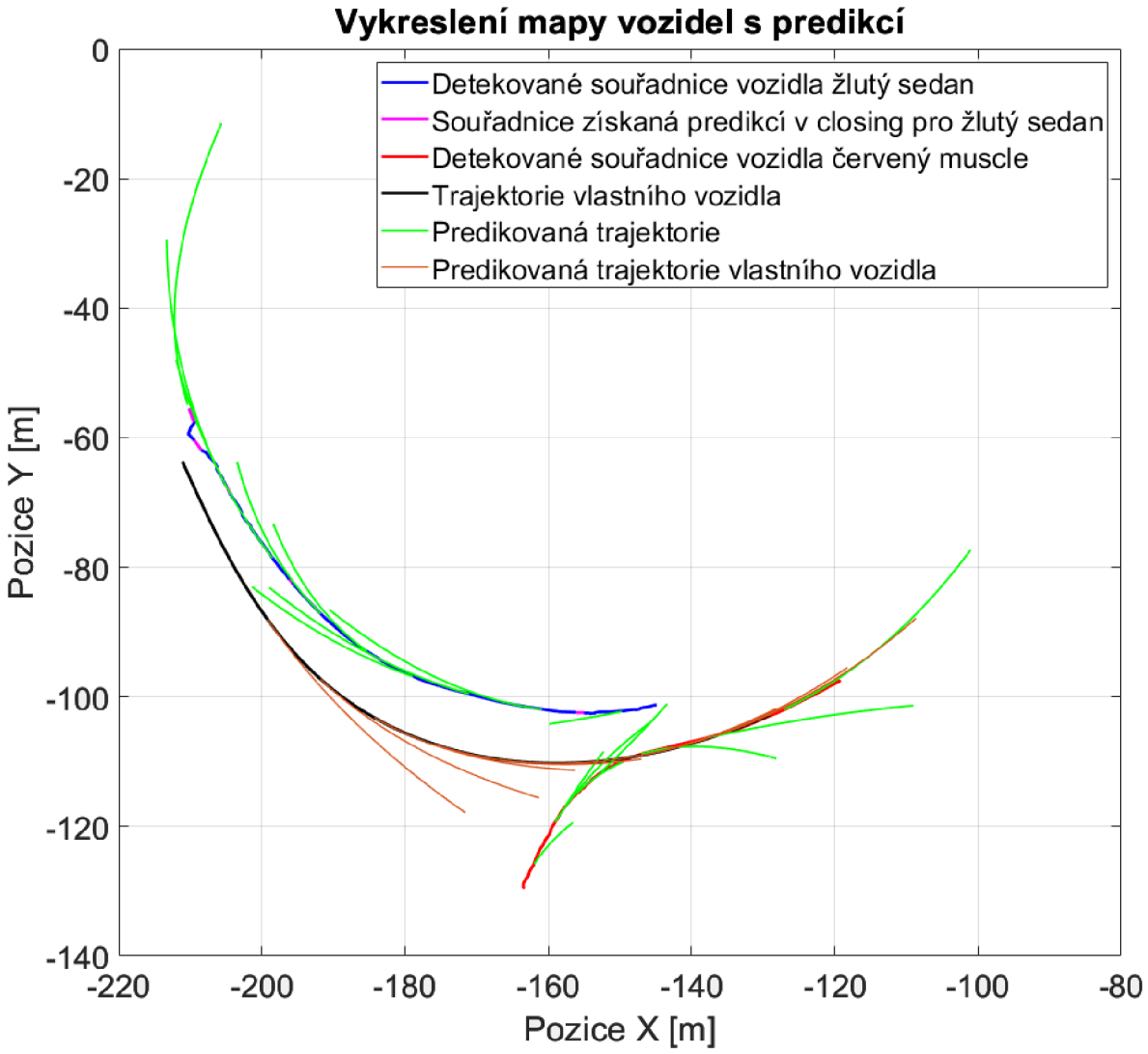
Tento graf je vykreslen pomocí lokálního absolutního systému. Pro tento systém byla vypočtena chyba pro 1 km od nulového bodu pomocí rovnice (26). Chyba je menší než 1 mm, proto může být v této simulaci zanedbávána.

$$l_{ch} = l - (\tan^{-1}\left(\frac{l}{r}\right) * \frac{\pi}{180} * r) \tag{26}$$

$$l_{ch} = 1 - (\tan^{-1}\left(\frac{1}{6378}\right) * \frac{\pi}{180} * 6378)$$

$l_{ch} = 8{,}2 * 10^{-9}$ km

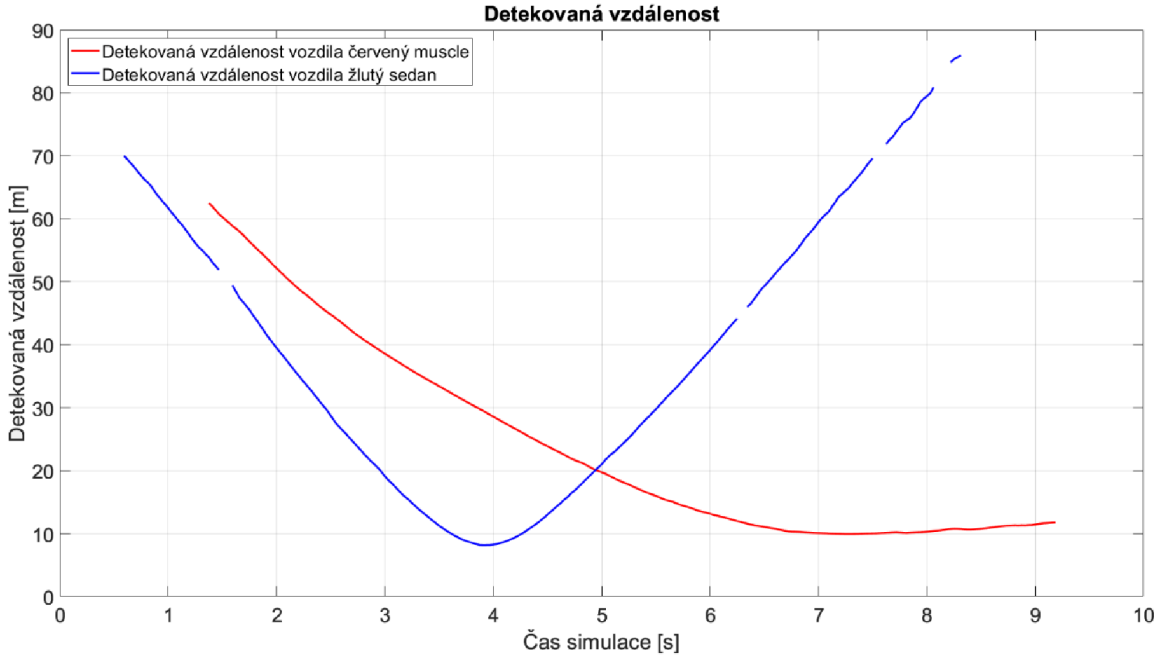
Kde l_{ch} je chyba zpětné projekce na zeměkouli, l je vzdálenost v kartézské 2D mapě, r poloměr zeměkoule.



Obr. 50 Vykreslení souřadnic vozidel s jejich predikcí

5.3 DETEKOVANÁ VZDÁLENOST VOZIDEL

Pro vykreslení vzdálenosti byla použita výstupní proměnná z funkce *DistanceFromLidar*, která obsahuje jen vzdálenosti detekovaných vozidel a zmizelá vozidla nejsou jejím výstupem. Pro jejich vykreslení by tedy bylo třeba jiné výstupní proměnné. Na grafu níže si lze všimnout oscilací při vzdálenosti větší než 50 m.



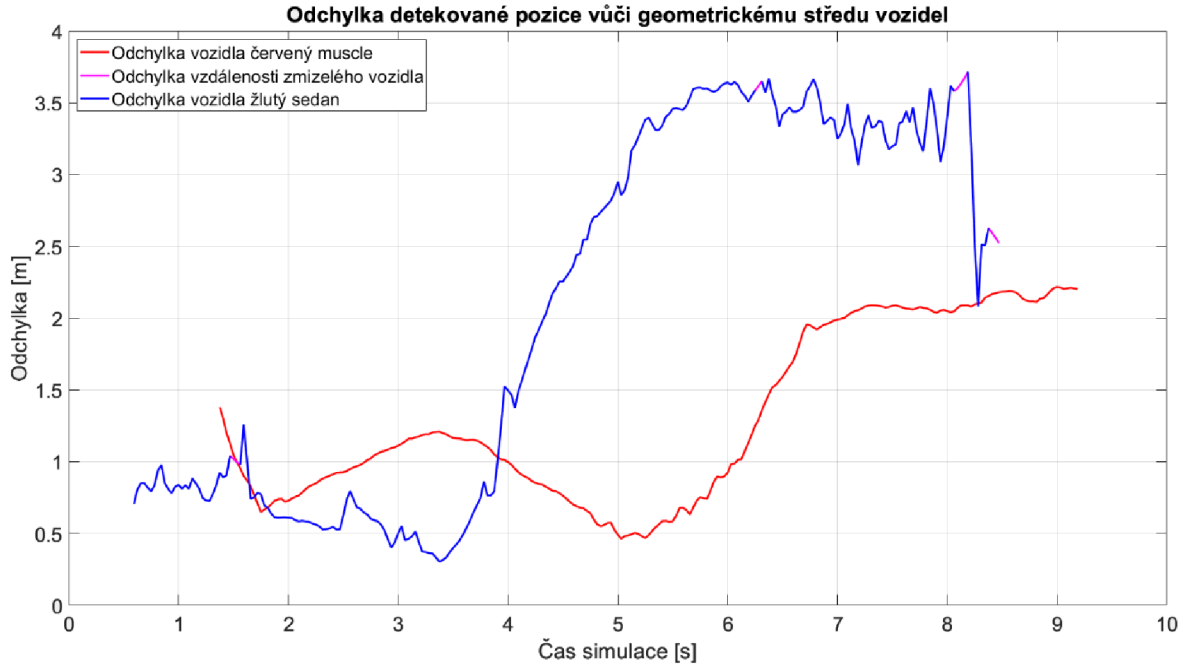
Obr. 51 Detekovaná vzdálenost

5.4 ODCHYLKA DETEKOVANÉ VZDÁLENOSTI

Na grafu na *Obr. 52* lze vidět odchylku detekované vzdálenosti. Tato odchylka vzdálenosti byla získána pomocí absolutního systému. Detekované souřadnice vozidel v každém kroku byly v jednotlivých osách odečteny od souřadnice v geometrickém středu vozidla. Rozdíly v jednotlivých osách pro detekovaná vozidla lze vidět na *Obr. 54*. Celková odchylka detekované vzdálenosti byla vypočítána Pythagorovou větou jako přepona, přičemž odchylky v jednotlivých osách byly odvěsny.

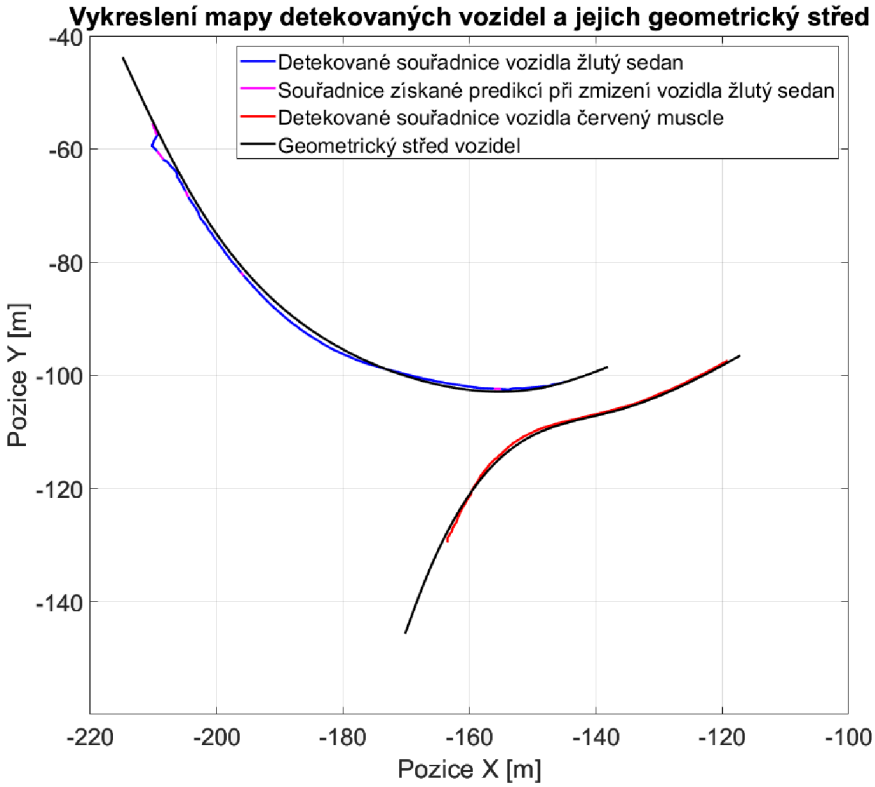
Odchylka, kterou lze vidět u žlutého sedanu je způsobena detekcí okrajů vozidla a zároveň plovoucím průměrem, který byl aplikován z důvodu oscilace získané vzdálenosti. Chyba způsobena detekcí okrajů by při čelní detekci měla nabývat hodnoty okolo 2,3 m a při boční detekci okolo 1 m. Při jízdě proti sobě je tato chyba nižší. Chyba začíná narůstat až po průjezdu vozidla mezi kamerami ve 4. sekundě, jelikož projíždí vozidlo kolem vlastního vozidla a je tak detekována jeho strana, které odpovídá chyba 1 m.

Při vjíždění vozidel do obrazu nastává problém se správným detekováním z důvodu detekce pouze části vozidla, což lze vidět u vozidla červený muscle. I u červeného vozidla se vyskytuje odchylka způsobená plovoucím průměrem, ovšem při vyrovnání rychlostí se po 7 s odchylka minimalizuje. Dále je patrné, že se od 50 m detekované vzdálenosti začínají objevovat drobné oscilace ve vzdálenosti, které se zvětšují se zvětšující se vzdáleností.



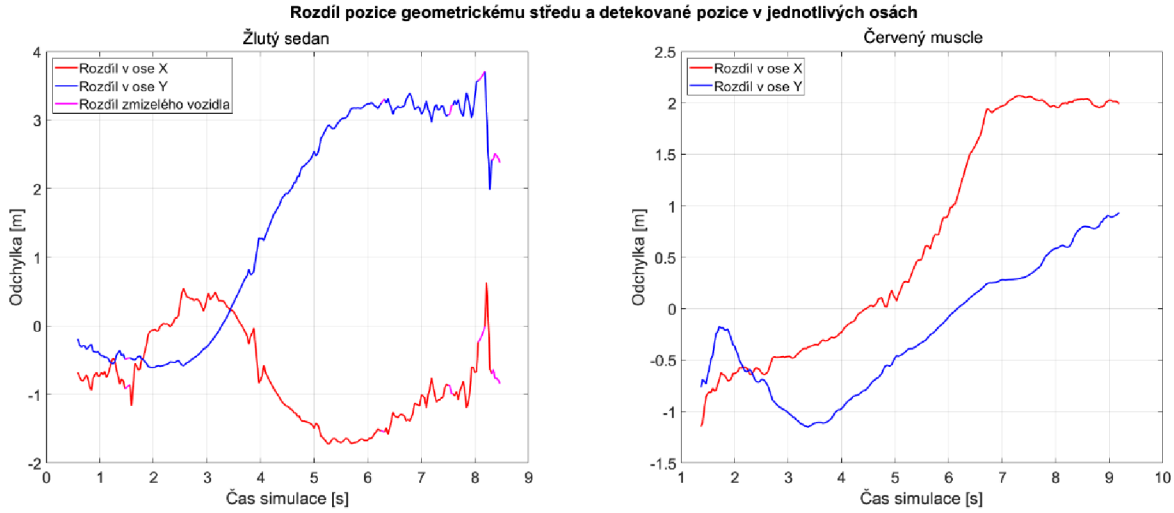
Obr. 52 Odchylka detekované vzdálenosti

Na následujícím grafu je vykreslena mapa detekovaných souřadnic vozidel a souřadnic geometrických středů vozidel. S grafem korespondují odchylky na *Obr. 52* a *Obr. 54*.



Obr. 53 Mapa detekovaných souřadnic vozidel a souřadnice geometrických středů vozidel

Na grafu níže jsou odchylky jednotlivých vozidel v jednotlivých osách, které vznikají vypočítáním rozdílu detekovaných souřadnic vozidel v každém kroku a souřadnic v geometrickém středu vozidla. Z těchto odchylek byla vypočítána celková odchylka.

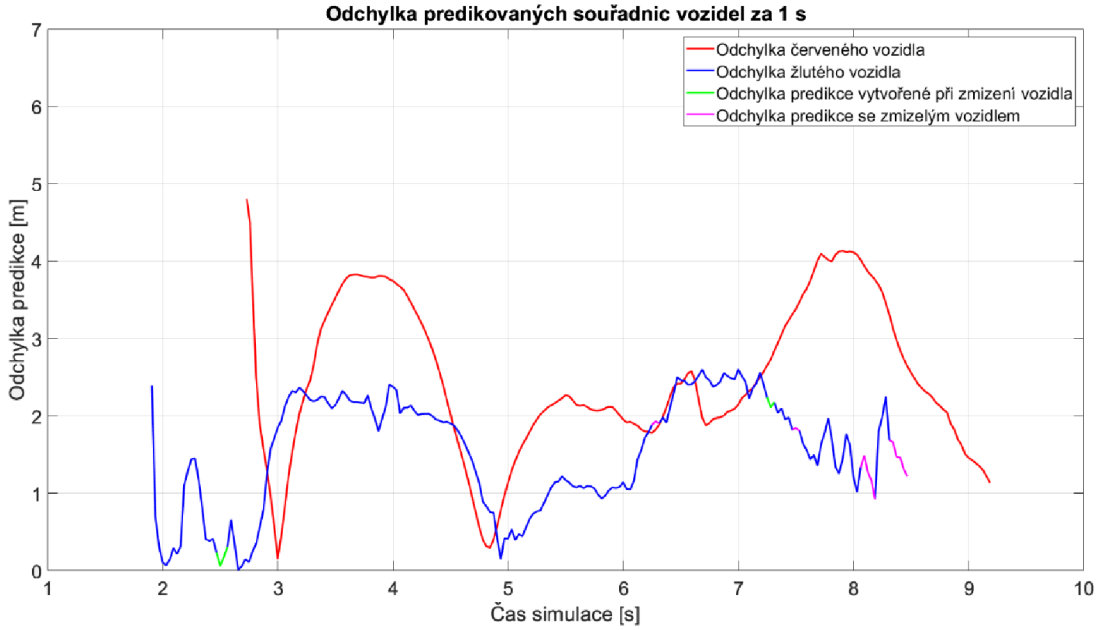


Obr. 54 Odchylka v osách vozidel

5.5 ODCHYLKA PREDIKOVANÝCH SOUŘADNIC ZA 1 S

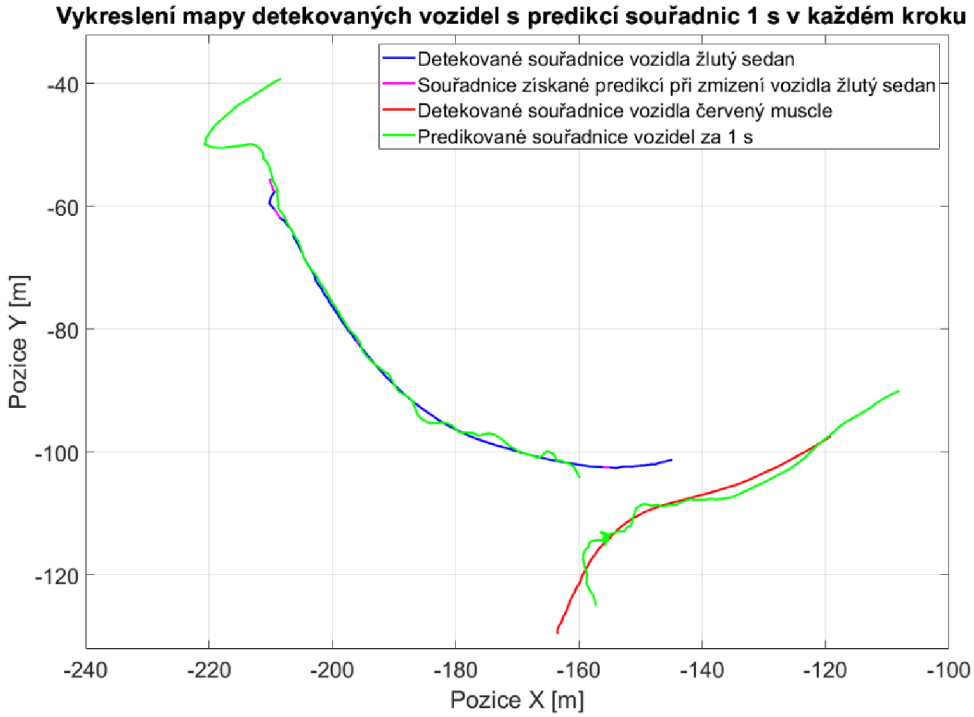
Na *Obr. 55* se nachází odchylka predikovaných souřadnic za 1 sekundu v porovnání s místem, kde je vozidlo detekováno za 1 s. Odchylka byla vypočtena stejným způsobem, jako u odchylky vzdálenosti. Odchylka u žlutého vozidla je na začátku způsobena oscilací vzdálenosti. I přesto je odchylka malá. Od 3 do 5 sekund je chyba způsobena především zpožděním vzdálenosti zapříčiněným plovoucím průměrem po průjezdu mezi kamerami. Před 5. sekundou odchylka klesla pod 1 m. Po 5. sekundě začalo vozidlo akcelerovat, proto se odchylka opět zvýšila.

Odchylka u červeného vozidla je zpočátku způsobená vjížděním do záběru. První minimální odchylka predikce ve 3 s byla získána vytvořením predikce při špatně inicializované rychlosti při vjíždění vozidla do záběru, se shodnou průměrnou rychlostí po dobu následující 1 s. V opačném případě by odchylka byla od začátku detekovaní vozidla konstantní kvůli deceleraci do 4. sekundy. Odchylka mezi 3. a 5. sekundou je způsobená decelerací vozidla. Následná minimální odchylka před 5. sekundou se vytvořila kvůli tomu, že rychlost vozidla, při které byla vytvářena predikce byla po dobu následující sekundy průměrně stejná. Poté již odchylka odpovídá akceleraci vozidla. Z grafu lze vidět, že odchylka ke konci simulace klesá. V případě konstantní rychlosti a delší simulace by se ustálila u nulové hodnoty.



Obr. 55 Odchylka predikovaných souřadnic za 1 s

Na následujícím obrázku jsou do mapy s detekovanými souřadnicemi vozidel promítnuté predikované souřadnice za dobu predikce 1 s. Z grafu je patrné, že trajektorie, po které se vozidlo pohybovalo za 1 s, je velice blízká predikované trajektorii. Největší odchylku predikce způsobuje akcelerace a decelerace. Lze také vidět chybně vytvořenou predikci při detekci nad 80 m, způsobenou malým počtem bodů vzdáleností na vozidle zachycených pomocí LiDARu a špatně inicializované červené vozidlo při vjíždění do záběru.

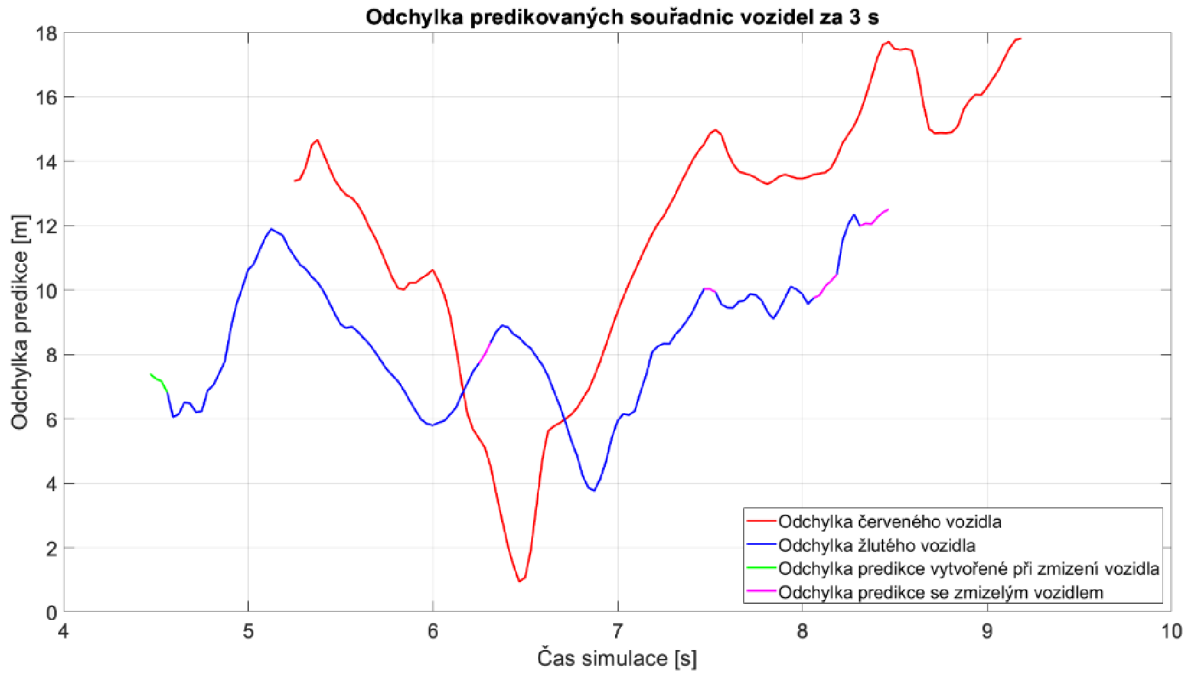


Obr. 56 Mapa detekovaných souřadnic a predikované souřadnice za 1 s

5.6 ODCHYLKA PREDIKOVANÝCH SOUŘADNIC ZA 3 S

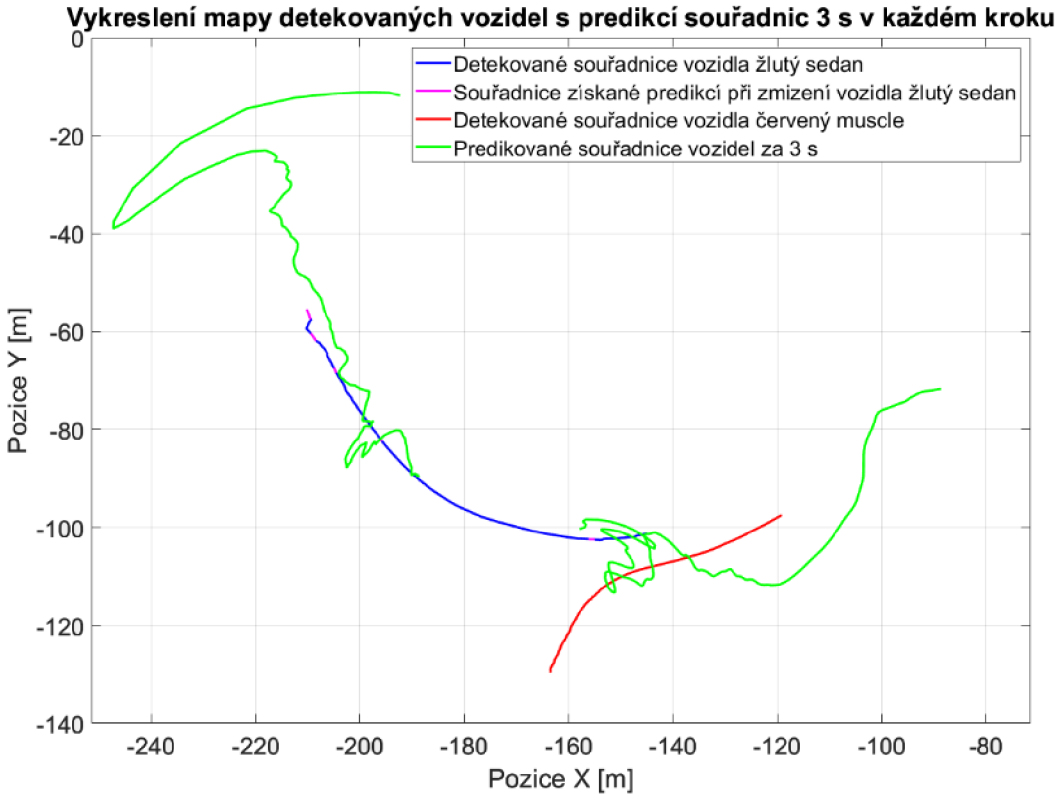
Největším problémem při predikci souřadnic na delší čas je předvídání dynamické jízdy. Proto je potřeba většího množství souřadnic získaných v co nejkratším čase a získaní přesnějších dat.

U červeného vozidla je vzniklá odchylka způsobena decelerací. Minimální odchylka predikce v 6,5 s se vytvořila kvůli tomu, že rychlost vozidla, při které byla vytvářena predikce byla po dobu následujících 3 sekund průměrně stejná. Poté se odchylka predikce začala opět zvyšovat v důsledku akcelerace a změně pohybu, kterou je obtížné předvídat.



Obr. 57 Odchylka predikované pozice za 3 s

Na vykreslené mapě s dobou predikce souřadnic za 3 s lze vidět, že se pro vozidlo žlutý sedan predikovaná trajektorie nachází poblíž detekované. V případě červeného vozidla je predikce horší v důsledku změny směru a rychlosti pohybu.



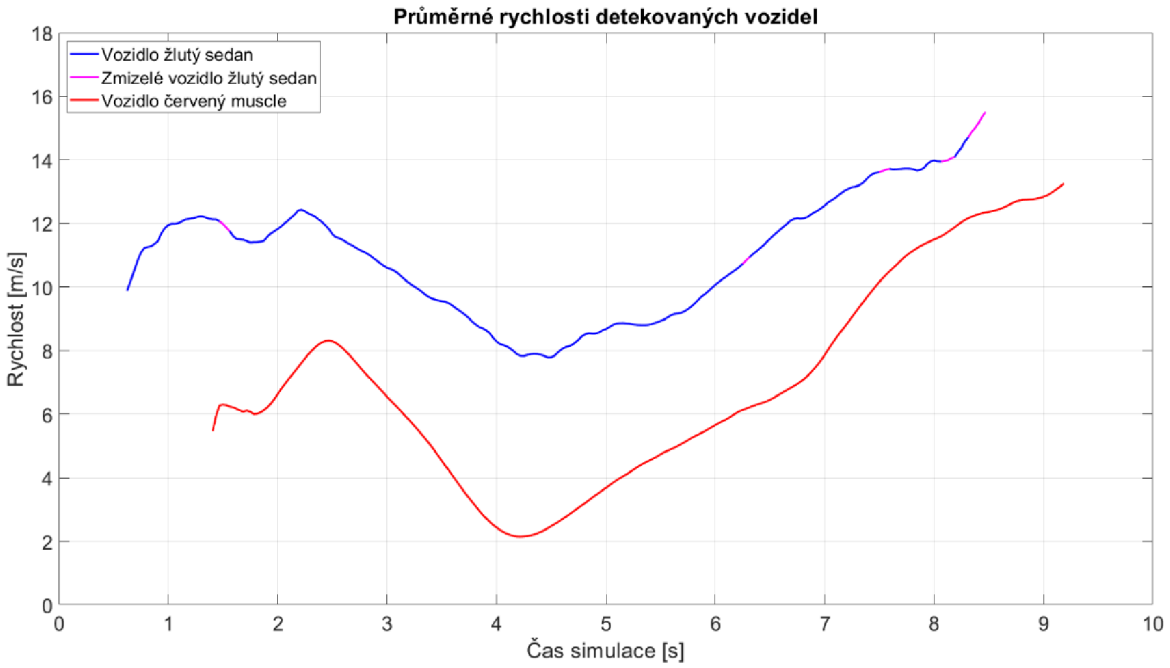
Obr. 58 Mapa detekovaných souřadnic a predikované souřadnice za 3 s

5.7 VYOBRAZENÍ RYCHLOSTÍ

V grafu je zobrazená průměrná rychlost detekovaných vozidel, která je využívána pro vytváření predikce. Hodnoty rychlosti přibližně odpovídají *Tab. 2*. V grafu lze vidět zpoždění 0,6 s, které je způsobené průměrováním hodnot.

Také je zde možné pozorovat problém u vozidel vjíždějících do scény. Tyto vozidla se jeví jako pomalejší, protože je detekována jen jejich část. Proto se před plným vjezdem vozidla do scény střední horizontální úhel mění pomaleji.

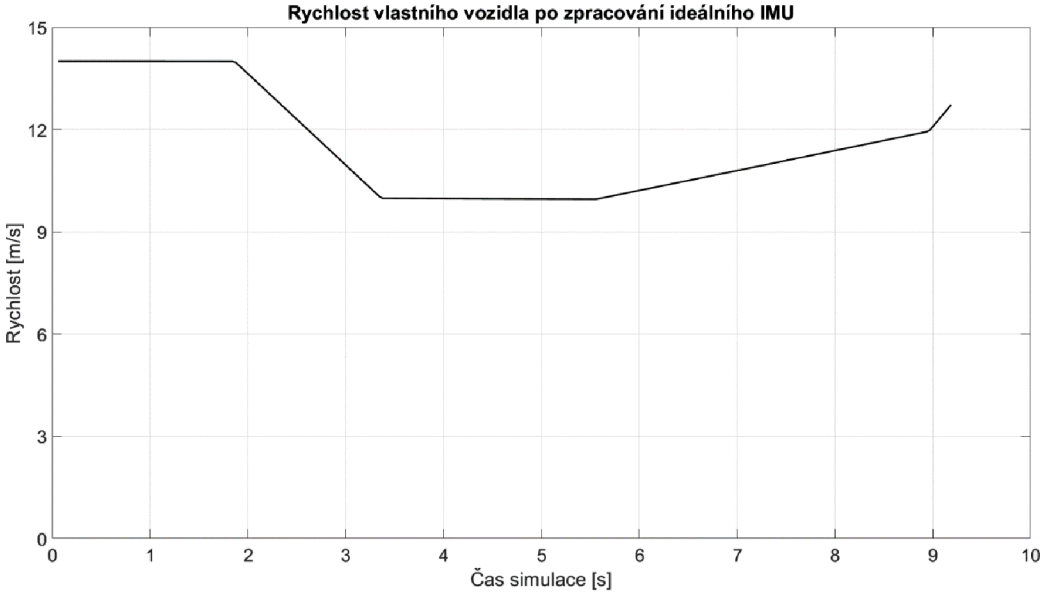
Od 2,5 s rychlost červeného vozidla se zpožděním odpovídá tabulkovým hodnotám. U žlutého vozidla si lze povšimnout snížení rychlosti o 2 m/s při projíždění kolem vlastního vozidla. Tato chyba je způsobená plovoucím průměrem. Zde se také projevuje chyba nad 80 m, kdy začíná detekovaná vzdálenost oscilovat a tím i zvyšovat rychlost. Rychlost byla získána z relativního systému.



Obr. 59 Průměrná rychlost vozidel

Níže je vyobrazen graf rychlosti vlastního vozidla po zpracování dat z IMU senzoru v relativním systému. Rychlost byla vypočítaná pomocí Pythagorovy věty jako přepona z rychlostí v jednotlivých osách. Jelikož se jedná o dokonale ideální IMU tak rychlost po zpracování přímo odpovídá tabulkovým hodnotám.

Při inicializaci natočení vlastního vozidla vůči severu vykreslení predikce trajektorie všech vozidel v relativním souřadném systému odpovídá průběhu predikce v absolutním souřadném systému, viz video v příloze. V případě inicializace pouze rychlosti by predikce probíhala obdobně, ovšem s relativními i osy což by na funkčnost nemělo mít vliv.



Obr. 60 Rychlost vlastního vozidla

ZÁVĚR

Cílem této práce bylo vytvoření predikce trajektorie okolních objektů především vozidel ve virtuálním okolí autonomního vozidla. Za tímto účelem bylo vytvořeno virtuální okolí pomocí RoadRunner, které bylo využito při simulaci, probíhající v Simulinku v kooperaci s Unreal Enginem. Dále byly umístěny kamery a LiDAR na vozidle. Pomocí nich je vytvořený algoritmus schopný získat informaci o poloze okolních vozidel a predikovat jejich trajektorii.

V první části práce byli na základě rešerše popsány využité senzory a jejich možnosti implementace do vozidla. Dále pak byl vytvořen teoretický základ pro získání objektu v okolí vozidla pomocí neuronových sítích pro počítačové vidění. V souvislosti s neuronovými sítěmi byly představeny datasety pro jejich učení. V závěru kapitoly byli představeny použité souřadné systémy ke zpracování pohybu detekovaných vozidel.

Modelování virtuálního okolí bylo provedeno na základě podkladů, a to digitálního modelu reliéfu a leteckého snímku. Následně je popsáno samotné vytvoření virtuálního okolí a jsou prezentovány snímky pro porovnání s realitou. Z RoadRunner bylo vyexportováno prostředí do Unreal Engine a soubor ASAM OpenDRIVE pro Driving Scenario Designer k vytvoření jízdní scény, na které bude algoritmus ověřován.

V další části diplomové práce jsou popsány kritéria pro výběr parametrů LiDARu a blíže prezentovány vybrané LiDARy splňující kritéria pro aplikaci do autonomních vozidel a z nich se jevil nejvhodněji Ouster OS2, ze kterého byly převzaty parametry. Dále jsou představeny existující obrazové senzory, ze kterých byl vybrán nejvhodnější s FullHD rozlišením. Poté jsou popsány navrhnuté koncepce implementace snímačů do karoserie a na střechu. Z návrhů byla vybrána koncepce umístění kamer s LiDARem na střechu, pro kterou byla vytvořena vizualizace.

Následně byl vytvořen algoritmus pro predikci pohybu vozidel v okolí autonomního vozidla. Základem pro naprogramování algoritmu je navrhnutá obecná logika fungování algoritmu. Pro detekci objektů v okolí vozidla pomocí kamery byly zvoleny dva modely neuronové sítě, YOLOv8n a YOLOv8m, které byly učeny na datasetu BDD-100k, který je vhodný pro aplikaci do prostředí, ve kterém se pohybují vozidla. Dále jsou prezentovány a zhodnoceny výsledky natrénování modelů YOLOv8. Bylo zjištěno že neuronová síť je vhodně natrénována na detekci všech tříd, které se vyskytují v datasetu BDD-100k, kromě třídy vlak. Podle základní logiky byly popsány vytvořené funkce pro zpracování výstupu z neuronové sítě a LiDARu až po predikci. Zároveň bylo naprogramováno správné přiřazení hodnot jednotlivým vozidlům.

Na závěr bylo provedeno zhodnocení naučených modelů YOLOv8n a YOLOv8m ze kterých se ukázal být vhodnější model YOLOv8m. Po vykreslení detekovaných souřadnic s predikcemi lze pozorovat, že predikovaná trajektorie kopíruje detekovanou trajektorii vozidla. Predikce trajektorie je tedy dostatečná, aby nastalo překřížení predikovaných trajektorii a mohla by tak být zjištěna potenciální kolize. Následně byly vyobrazeny odchylky detekované vzdálenosti a predikcí za 1 a 3 sekundy a popsány jejich příčiny. Při predikci přesných souřadnic za 1 s je odchylka z největší části způsobena změnou rychlosti. Predikce přesných souřadnic za 3 sekundy dokáže algoritmus předvídat spíše pro ustálené děje, ovšem predikce přesných souřadnic za 3 sekundy při dynamickém ději může být složitá

i pro člověka. Poslední hodnocenou veličinou byla vypočtená průměrná rychlost, u které bylo zjištěno zpoždění o 0,6 s oproti tabulkovým hodnotám, způsobené průměrováním.

Pokud by měl být vytvořený algoritmus využit v autonomním řízení, bylo by potřeba definovaného konstantního kroku zpracování. Dále by musela být nahrazena funkce *EgoImuSim*, protože informace by byly získávány z reálného IMU. Následně by čas simulace byl nahrazen hodinami reálného času. Dalším zpřesněním při využití v provozu by mělo být opravení zjištěné odchylky, vznikající plovoucím průměrem. Odchylka by byla snížena rychlejším zpracováním snímků, nebo pravděpodobněji korekčním součinitelem, který by reflektoval rychlost zpracování snímků, rozdíl rychlostí a natočení vlastního vozidla vůči detekovaným vozidlům. Tento součinitel by zároveň mohl sloužit pro korekci vzdálenosti, kterou vozidlo ujede za čas zpracování kroku. Poslední úpravou by byla jiná implementace neuronové sítě YOLOv8. To by zpracovávalo obrazová data přímo z kamer, na rozdíl od simulace, kde je neuronová síť YOLOv8 spouštěna v každém kroku zvlášť.

POUŽITÉ INFORMAČNÍ ZDROJE

[1] DIETSCHE, Karl-Heinz a Konrad REIF. *Automotive Handbook*. 11th Edition. Karlsruhe: Robert Bosch GmbH, 2022. ISBN 978-1-119-91190-6.

[2] ČSN EN 60825-1. *Bezpečnost laserových zařízení - Část 1: Klasifikace zařízení a požadavky*. Praha: Uřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2015.

[3] *Tesla: About Autopilot* [online]. In: . [cit. 2023-02-18]. Dostupné z: https://www.tesla.com/ownersmanual/modely/en_us/GUID-EDA77281-42DC-4618-98A9-CC62378E0EC2.html

[4] MOU, Shenyu, Yan CHANG, Wenshou WANG a Ding ZHAO. *An Optimal LiDAR Configuration Approach for Self-Driving Cars* [online]. 7 [cit. 2023-04-29]. Dostupné z: https://arxiv.org/abs/1805.07843

[5] AGGARWAL, Charu C. *Neural Networks and Deep Learning: A Textbook* [online]. Cham, Switzerland: Springer International Publishing AG, 2018 [cit. 2023-04-28]. ISBN 978-3-319-94463-0. Dostupné z: https://doi.org/10.1007/978-3-319-94463-0

[6] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[7] CHOLLET, François. *Deep learning with Python*. Shelter Island, NY: Manning, 2018. ISBN 978-161-7294-433.

[8] REN, Shaoqing, Kaiming HE, Ross GIRSHICK a Jian SUN. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [online]. 14 [cit. 2023-04-29]. Dostupné z: https://arxiv.org/abs/1506.01497

[9] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. *You Only Look Once: Unified, Real-Time Object Detectio* [online]. 10 [cit. 2023-04-29]. Dostupné z: https://arxiv.org/abs/1506.02640

[10] *PyTorch* [online]. [cit. 2023-04-21]. Dostupné z: https://pytorch.org/

[11] *JOCHER, Glenn. ultralytics YOLOv8.* [online]. 2023 [cit. 2023-04-16]. Dostupné z: https://github.com/ultralytics/ultralytics

[12] Tips for Best Training Results. In: *Ultralytics YOLOv8 Docs* [online]. [cit. 2023-04-22]. Dostupné z: https://docs.ultralytics.com/yolov5/tips_for_best_training_results/#dataset

[13] LIN, Tsung-Yi, Michael MAIRE, Serge BELONGIE, Ross GIRSHICK, James HAYS, C.Lawrence ZITNICK a Piotr DOLLÁR. *Microsoft COCO: Common Objects in*

Context [online]. 15 [cit. 2023-04-22]. Dostupné z: https://arxiv.org/pdf/1405.0312.pdf

[14] DOMINGUEZ-SANCHEZ, Alex, Sergio ORTS-ESCOLANO, Jose GARCIA-RODRIGUEZ a Miguel CAZORLA. *A New Dataset and Performance Evaluation of a Region-based CNN for Urban Object Detection* [online]. 8 [cit. 2023-04-22]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8489478

[15] YU, Fisher, Haofeng CHEN, Xin WANG, Wenqi XIAN, Yingying CHEN, Fangchen LIU, Vashisht MADHAVAN a Trevor DARRELL. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning* [online]. 2020, 14 [cit. 2023-04-22]. Dostupné z: https://arxiv.org/pdf/1805.04687.pdf

[16] CHATURVEDI, Devendra K. *Modeling and Simulation of System Using MATLAB and Simulink*. Boca Raton: CRC Press, 2010. ISBN 978-1-4398-0672-2.

[17] Automated Driving Toolbox. In: *MathWorks* [online]. [cit. 2023-04-30]. Dostupné z: https://www.mathworks.com/products/automated-driving.html

[18] Scenario Simulation. In: *MathWorks Help Center* [online]. [cit. 2023-04-30]. Dostupné z: https://www.mathworks.com/help/driving/scenario-simulation.html

[19] Choose a Sensor for Unreal Engine Simulation. In: *MathWorks Help Center* [online]. [cit. 2023-04-29]. Dostupné z: https://www.mathworks.com/help/driving/ug/choose-a-sensor-for-3d-simulation.html

[20] RoadRunner. In: *MathWorks* [online]. [cit. 2023-05-15]. Dostupné z: https://www.mathworks.com/products/roadrunner.html

[21] SAE J670. *Vehicle Dynamics Terminology*. SAE International, 2008.

[22] Coordinate Systems in Automated Driving Toolbox. In: *Mathworks Help Center* [online]. [cit. 2023-05-08]. Dostupné z: https://www.mathworks.com/help/driving/ug/coordinate-systems.html

[23] *Český úřad zeměměřický a katastrální: Geoportál* [online]. In: . [cit. 2023-02-17]. Dostupné z: https://geoportal.cuzk.cz/

[24] *Copernicus: EU-DEM* [online]. In: . [cit. 2023-02-17]. Dostupné z: https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1

[25] *Český úřad zeměměřický a katastrální: Geoportál Ortofoto České republiky* [online]. In: . [cit. 2023-02-17]. Dostupné z: https://geoportal.cuzk.cz/(S(rvjrp1urfwdsz1gtoxqniy3t))/Default.aspx?mode=TextMeta&side=ortofoto&metadataID=CZ-CUZK-ORTOFOTO-R&productid=63410&mapid=83&menu=231

[26] Autocad. In: *Autodesk* [online]. [cit. 2023-05-15]. Dostupné z:

https://www.autodesk.cz/products/autocad/

[27] *InkScape* [online]. [cit. 2023-05-15]. Dostupné z: https://inkscape.org/cs/

[28] *ZÁSADY PRO VODOROVNÉ DOPRAVNÍ ZNAČENÍ NA POZEMNÍCH KOMUNIKACÍCH*. In: . Praha: Ministerstvo dopravy odbor pozemních komunikací, 2013. Dostupné také z: https://pjpk.rsd.cz/data/USR_001_2_8_TP/TP_133.pdf

[29] Mathworks Help Center1: Unreal Engine Simulation Environment Requirements and Limitations. In: *Mathworks* [online]. [cit. 2023-02-19]. Dostupné z: https://www.mathworks.com/help/driving/ug/3d-visualization-engine-requirements.html

[30] Mathworks Help Center2: Install Support Package for Customizing Scenes. In: *Mathworks* [online]. [cit. 2023-02-19]. Dostupné z: https://www.mathworks.com/help/driving/ug/install-and-configure-support-package-for-customizing-scenes.html

[31] Mathworks Help Center4: Export to Unreal Using Filmbox (.fbx) File. In: *Mathworks* [online]. [cit. 2023-02-19]. Dostupné z: https://www.mathworks.com/help/roadrunner/ug/export-to-unreal-fbx.html

[32] Scenarios in Simulink. In: *Mathworks Help Center* [online]. [cit. 2023-04-22]. Dostupné z: https://www.mathworks.com/help/driving/scenarios-in-simulink.html

[33] HDL-32E. In: *Velodyne Lidar* [online]. [cit. 2023-05-05]. Dostupné z: https://www.mapix.com/wp-content/uploads/2018/07/97-0038_Rev-M_-HDL-32E_Datasheet_Web.pdf

[34] Puck. In: *Velodyne Lidar* [online]. [cit. 2023-05-05]. Dostupné z: https://velodynelidar.com/wp-content/uploads/2019/12/63-9229_Rev-K_Puck-_Datasheet_Web.pdf

[35] Puck Hi-Res. In: *Velodine Lidar* [online]. [cit. 2023-05-05]. Dostupné z: https://velodynelidar.com/wp-content/uploads/2019/12/63-9229_Rev-K_Puck-_Datasheet_Web.pdf

[36] Ouster OS2. In: *AutonomouStuff* [online]. [cit. 2023-05-05]. Dostupné z: https://autonomoustuff.com/products/ouster-os2

[37] Ouster OS1. In: *AutonomouStuff* [online]. [cit. 2023-05-05]. Dostupné z: https://autonomoustuff.com/products/ouster-os1

[38] Automotive Image Sensors. In: *Samsung* [online]. [cit. 2023-04-22]. Dostupné z: https://semiconductor.samsung.com/image-sensor/automotive-image-sensor/

[39] Image Sensor for Automotive Use. In: *SONY* [online]. [cit. 2023-04-22]. Dostupné z:

https://www.sony-semicon.com/en/products/is/automotive/automotive.html

[40] Octavia IV. In: *Škoda Storyboard* [online]. [cit. 2023-04-30]. Dostupné z: https://www.skoda-storyboard.com/cs/skoda-model/nova-octavia/octavia-iv/

[41] Škoda Auto dodala v roce 2022 celosvětově 731 300 automobilů. In: *Škoda Storyboard tiskové zprávy archív* [online]. [cit. 2023-04-30]. Dostupné z: https://www.skoda-storyboard.com/cs/tiskove-zpravy-archiv/skoda-auto-dodala-v-roce-2022-celosvetove-731-300-automobilu/

[42] Sedan. In: *Mathworks Help Center* [online]. [cit. 2023-04-30]. Dostupné z: https://www.mathworks.com/help/driving/ref/sedan.html

[43] Introduction to Blueprints. In: *Unreal Engine* [online]. [cit. 2023-04-30]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/

[44] *CUDA Toolkit Archive* [online]. [cit. 2023-04-21]. Dostupné z: https://developer.nvidia.com/cuda-toolkit-archive

[45] *NVIDIA cuDNN* [online]. In: . [cit. 2023-04-21]. Dostupné z: https://developer.nvidia.com/cudnn

[46] *JOCHER, Glenn. ultralytics YOLOv5 Docs Train Custom Data.* [online]. [cit. 2023-04-17]. Dostupné z: https://docs.ultralytics.com/yolov5/train_custom_data/

[47] YU, Fisher. Label Format. In: *BDD100K documentation* [online]. [cit. 2023-04-21]. Dostupné z: https://doc.bdd100k.com/format.html

[48] Bdd100k/bdd100k/label/to_coco.py. In: *GitHub* [online]. [cit. 2023-04-21]. Dostupné z: https://github.com/bdd100k/bdd100k/blob/master/bdd100k/label/to_coco.py

[49] JOCHER, Glenn. ultralytics/JSON2YOLO. In: *GitHub* [online]. [cit. 2023-04-21]. Dostupné z: https://github.com/ultralytics/JSON2YOLO

[50] Detect. In: *Ultralytics* [online]. [cit. 2023-04-22]. Dostupné z: https://docs.ultralytics.com/tasks/detect/

[51] The New York Times. In: *Archive* [online]. [cit. 2023-04-20]. Dostupné z: https://archive.nytimes.com/www.nytimes.com/imagepages/2017/04/07/business/08waymo1.html

[52] *Seznam Mapy* [online]. [cit. 2023-05-01]. Dostupné z: https://mapy.cz/

SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

FoV_{lidar}	[°]	Vertikální úhel zorného pole lidaru
Q_{hor}	[–]	Počet generovaných bodů LiDARem v horizontálním směru
Q_{vert}	[–]	Počet generovaných bodů LiDARem ve vertikálním směru
a_F	[m·s^{-2}]	Zrychlení v podélném směru
a_n	[m·s^{-2}]	Normálové zrychlení
a_x	[m·s^{-2}]	Zrychlení v ose X
a_y	[m·s^{-2}]	Zrychlení v ose Y
l_{ch}	[km]	Chyba při zpětné projekci na zeměkouli
m_{hor}	[°]	Horizontální úhel středu ohraničujícího rámečku
m_{vert}	[°]	Vertikální úhel středu ohraničujícího rámečku
r_z	[m]	Rádius zatáčení
t_{Pred}	[s]	Doba predikce
$v_{x,i}$	[m·s^{-1}]	Rychlost v ose X v aktuálním kroku
$v_{x,i-1}$	[m·s^{-1}]	Rychlost v ose X v minulém kroku
$v_{y,i}$	[m·s^{-1}]	Rychlost v ose Y v aktuálním kroku
$v_{y,i-1}$	[m·s^{-1}]	Rychlost v ose X v minulém kroku
w_i	[–]	Váha pro vstupní uzel
x_i	[–]	Hodnota ze vstupních uzlů
λ_s	[°]	Zeměpisná délka nulového bodu
φ_i	[°]	Aktuální natočení vlastního vozidla
φ_{i-1}	[°]	Natočení vlastního vozidla v minulém kroku
φ_s	[°]	Zeměpisná šířka nulového bodu
ω_Z	[°·s^{-1}]	Úhlová rychlost okolo oy
ΔX	[m]	Posunutí vozidla v ose X od minulého kroku
ΔY	[m]	Posunutí vozidla v ose Y od minulého kroku
Δt	[s]	Rozdíl časů
Δv	[m·s^{-1}]	Rozdíl rychlostí
Δx	[m]	Rozdíl dráhy v ose X
Δy	[m]	Rozdíl dráhy v ose Y
absRot	[°]	Proměnná – úhel otočení detekovaných vozidel
ADAS		Advanced driver-assistance system
alfa	[°]	Konstanta – horizontální úhel natočení kamery vůči lidaru
AP		Average Precision
BDD-100K		Berkley Deep Drive – 100000

boxes	[px]	Proměnná – ohraničující rámečky
carBBoxes	[px]	Proměnná – ohraničující rámečky pouze vozidel
closingCar		Trvalá proměnná – informace zmizelých vozidel
closingInfo		Proměnná – informace o zmizení nebo objevení vozidel
CMOS		Complementary Metal Oxide Semiconductor
col		Proměnná – sloupec vyhledávané vzdálenosti
comand		Proměnná – příkaz pro volání platformy YOLOv8
cont		Proměnná – počet řádků se vzdáleností získanou z LiDARu
CUDA		Compute Unified Device Architecture
ČSN EN		Česká verze evropské normy
ČÚZK		Český úřad zeměměřický a katastrální
data	[°]	Proměnná – úhly vozidel ze všech kamer
deltaX	[m]	Proměnná – rozdíl dráhy v ose X
deltaY	[m]	Proměnná – rozdíl dráhy v ose Y
DEM		Digitální model reliéfu (Digital Elevation Model)
detCarPoss		Proměnná – otočené souřadnice vozidel bez posunutí
detectAngle		Proměnná – úhly vozidel a strany rámečků
detectCar		Trvalá proměnná – informace detekovaných vozidel
distCar		Trvalá proměnná – hodnoty vzdáleností vozidel
distFitCurve		Proměnná – koeficienty křivky
distFromLid		Proměnná – vzdálenost a horizontální úhel vozidel
distPoints	[m]	Proměnná – hodnoty vzdáleností vozidla
ego		Proměnná – rychlost, zrychlení a natočení vlastního vozidla
EU		Evropská unie
existCar		Trvalá proměnná – informace detekovaných vozidel
FIFO		First In First Out
filteredPoints	[m]	Proměnná – vyfiltrované vzdálenosti
filteredPointsHard		Proměnná – poprvé filtrované hodnoty
FPS		Frame per Second
FullHD		Full High Definition
GB		Gigabajt
GHz		Giga Hertz
GPS		Globální polohový systém
horFov	[°]	Konstanta – horizontální zorný úhel kamery

image		Proměnná – snímek z kamery
imgPath		Proměnná – relativní cesta pro uložení snímku
imgSize	[px]	Konstanta – rozlišení kamery
IMU		Inerciální měřící jednotka
IOU		Intersection over Union
labels		Proměnná – detekované třídy
LiDAR		Light Detection and Ranging
lidarBox	[m]	Proměnná – výseč vzdáleností na vozidle
LidarDist	[m]	Proměnná – vzdálenosti získané lidar
m		Metr
mAP	[–]	mean Average Precision
meanDist	[m]	Proměnná – průměrná hodnota vzdálenosti detekovaného vozidla
MEMS		Micro Electro Mechanical Systems
midleOfBoxes	[px]	Proměnná – střed ohraničujícího rámečku
mm		milimetry
model		Proměnná – relativní cesta k naučeným váhám YOLOv8
Mpx		Mega pixel
MS COCO		Microsoft Common Objects in Context
NMS		Non Max Suppression
orientCar	[°]	Proměnná – Natočení od severu
PR		Precision-Recall
PredDist	[m]	Proměnná – predikovaná hodnota vzdálenosti
preDetAngles		Proměnná – úhly vozidel a strany rámečků ze všech kamer
proj		Proměnná – relativní cesta pro uložení souborů
px		Pixel
Radar		Radio detection and ranging
RAM		Random Access Memory
R-CNN		Region-based Convolutional Neural Network
ReLU		Rectified Linear Unit
resHeight	[px]	Konstanta – vertikální rozlišení kamery
resWidth	[px]	Konstanta – horizontální rozlišení kamery
RGB		Red Green Blue
row		Proměnná – řádek vyhledávané vzdálenosti
RPN		Region Proposal Network

s		Sekunda
SAE		Society of Automotive Engineers
searchAngle		Proměnná – řádky a sloupce stran ohraničujících rámečků
S-JTSK		Systém jednotné trigonometrické sítě katastrální
sortedAngle	[°]	Proměnná – roztříděné hodnoty úhlů
SPZ		Státní poznávací značka
time	[s]	Proměnná – čas simulace
toBondary	[px]	Proměnná – vzdálenost od středu ke kraji ohraničujícího rámečků
txtFile		Proměnná – relativní cesta k textovému souboru
UE		Unreal Engine
value	[m]	Proměnná – nejpočetnější hodnota
vehiclePoss		Proměnná – souřadnice vozidel v souřadném systému vozidla
vertFov	[°]	Konstanta – vertikální zorný úhel kamery
Vires VTD		Vires Virtual Test Drive
VOC		Visual Object Classes
xForPoints		Konstanta – pozice v X pro vzdálenost
X-Y-Z	[m]	Osy
y_*DistPred*	[m]	Proměnná – Predikovaná vzdálenost
YOLO		You Only Look Once
Φ	[–]	Aktivační funkce
F	[N]	Síla
a	[m·s^{-2}]	Zrychlení
b	[–]	Zkreslení
c	[N·m^{-1}]	Tuhost pružiny
f	[px]	Ohnisková vzdálenost
k		Konstanta zmenšující strany ohraničujícího rámečků
l	[km]	Vzdálenost v kartézském systému
m	[kg]	Hmotnost
n	[–]	Počet vstupních uzlů
r	[m]	Rádius země
$round$		Zaokrouhlení na jednotky
v	[m·s^{-1}]	Rychlost
x	[m]	Stlačení pružiny
x	[px]	Horizontální rozlišení

y	[px]	Vertikální rozlišení
θ	[°]	Úhel zorného pole
λ	[°]	Zeměpisná délka vlastního vozidla
φ	[°]	Zeměpisná šířka vlastního vozidla

SEZNAM PŘÍLOH

- Algoritmus pro predikci trajektorie
- Video ze simulace s relativním kartézským systémem
- Natrénované váhy YOLOv8n