
# 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


# DETEKCE JÍZDNÍCH PRUHŮ PRO AUTONOMNÍ VOZIDLA

LANE DETECTION FOR AUTONOMOUS VEHICLES

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Štěpán Holík**
AUTHOR

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

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

# Zadání diplomové práce

| | |
|---|---|
| Ústav: | Ústav automobilního a dopravního inženýrství |
| Student: | **Bc. Štěpán Holík** |
| Studijní program: | Automobilní a dopravní inženýrství |
| Studijní obor: | bez specializace |
| Vedoucí práce: | **doc. Ing. Pavel Kučera, Ph.D.** |
| Akademický rok: | 2023/24 |

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:

## Detekce jízdních pruhů pro autonomní vozidla

**Stručná charakteristika problematiky úkolu:**

Naprogramování řídicího algoritmu pro detekci jízdních pruhů na vozovce z kamer pro autonomní řízení vozidla využitím softwaru Simulink, nebo dalších programovacích jazyků C, C++, C#, atd. Konkrétně se jedná o detekci jízdních pruhů a tím určovat trajektorii pohybu autonomního. Experimentální ověření správné detekce jízdních pruhů a polohy vozidla.

**Cíle diplomové práce:**

Rešerše o detekci jízdních pruhů.
Naprogramování základní komunikace s kamerou.
Naprogramování algoritmu pro detekci.
Naprogramování algoritmu pro určování trajektorie vozidla.
Experimentální ověření správné detekce.

**Seznam doporučené literatury:**

HANSEN, John H. L. Digital signal processing for in-vehicle systems and safety. 1. London: Springer, 2012. ISBN 978-1441996060.

SOLOMON, Chris a Toby BRECKON. Fundamentals of digital image processing: a practical approach with examples in Matlab. 1. Hoboken, NJ: Wiley-Blackwell, 2011. ISBN 978-0470844731.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

| | |
|---|---|
| prof. Ing. Josef Štětina, Ph.D.<br>ředitel ústavu | doc. Ing. Jiří Hlinka, Ph.D.<br>děkan fakulty |

# ABSTRAKT

Tato diplomová práce se zaměřuje na návrh a experimentální ověření systému pro detekci jízdních pruhů, určování trajektorie a polohy vozidla. Cílem bylo vyvinout systém složený z algoritmů pro dílčí části systému. K tomuto účelu bylo využito dat nashromážděných kamerou ZED 2, modelu neuronové sítě U-Net a prvků počítačového vidění ke snížení falešně pozitivních detekcí pomocí časového okna. K určení polohy vozidla vůči trajektorii byly využity trigonometrické výpočty a parametry kamery. Součástí práce bylo také rozšíření datové sady TuSimple o data nashromážděná kamerou.

Experimentální ověření prokázalo funkčnost systému s vysokou spolehlivostí detekce v jednoduchých modelových situacích typu jízda po rovném úseku silnice. S náročností modelových situací spolehlivost systému klesá.

Navzdory těmto nedostatkům experimenty ukázaly, že systém je schopen detekovat hranice jízdních pruhů a navrhovat optimální trajektorii vozidla. Algoritmy pro určení trajektorie a polohy vozidla závisí na úvodní predikci hranic jízdního pruhu, jsou ale funkční a účinné.

## KLÍČOVÁ SLOVA

Detekce jízdních pruhů, určování trajektorie, poloha vozidla, U-Net, neuronové sítě, počítačové vidění, časové průměrování, zpracování obrazu

# ABSTRACT

This thesis focuses on the design and experimental verification of a system for lane detection, trajectory estimation and vehicle position. The goal was to develop a system composed of algorithms with its respective functions. Data collected with ZED 2 camera, the U-Net neural network model, and computer vision were used to reduce false positive predictions using a temporal window. Trigonometric calculations and camera parameters were used to estimate the vehicle's position relative to the trajectory. One of the outcomes of this thesis is TuSimple dataset extension with the data captured with ZED 2 camera.

Experimental verification demonstrated the system's functionality with high detection reliability in simple model situations, such as driving on a straight road segment. As the complexity of the model situations increased, the system's reliability decreases.

Despite these shortcomings, the experiments showed that the system is able to detect lane boundaries and estimate an optimal vehicle trajectory. The algorithms for trajectory and vehicle position determination depend on the initial prediction of the lane boundaries, but they are functional and effective.



## KEYWORDS

Lane detection, trajectory estimation, vehicle position, U-Net, neural networks, computer vision, temporal averaging, image processing

# BIBLIOGRAFICKÁ CITACE

HOLÍK, Štěpán. *Detekce jízdních pruhů pro autonomní vozidla* [online]. Brno, 2024 [cit. 2024-05-19]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/154076. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automobilního a dopravního inženýrství. Vedoucí práce Pavel Kučera.

# ČESTNÉ PROHLÁŠENÍ

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

V Brně dne 24. května 2024 ……..……..

Jméno a příjmení

# PODĚKOVÁNÍ

Chtěl bych poděkovat vedoucímu mé diplomové práce doc. Ing. Pavla Kučery, Ph.D. za jeho cenné rady a trpělivost během psaní mé diplomové práce. Dále bych chtěl poděkovat své rodině a přátelům za jejich podporu a pochopení, které mi umožnily se soustředit na studium a tuto práci dokončit.

# OBSAH

# Úvod

Autonomní vozidla v dnešní době představují perspektivní oblast vývoje dopravy a mobility. Jejich úspěšné nasazení vyžaduje spolehlivý systém detekce prostředí, ve kterém se pohybují, a správnou interpretaci těchto dat pro bezpečné řízení. Jedním z klíčových prvků této detekce je schopnost identifikace jízdních pruhů a určení optimální trajektorie pohybu vozidla.

Cílem této diplomové práce je praktická implementace algoritmu pro detekci jízdních pruhů a určování trajektorie vozidla s využitím kamerových dat. Práce je rozdělena do několika hlavních částí, které se postupně zabývají klíčovými aspekty tohoto úkolu.

Úvodní kapitola představuje kontext autonomních vozidel a význam jejich úrovní automatizované jízdy. Dále je zde uveden přehled existujících projektů v oblasti autonomního řízení s důrazem na implementaci detekce jízdních pruhů.

Další část práce se soustředí na problematiku detekce jízdních pruhů. Jsou zde představeny metody detekce založené na počítačovém vidění, včetně konvenčních přístupů a metod využívajících neuronové sítě.

Následující část práce se zaměřuje na praktickou implementaci navržených algoritmů. Tato část popisuje komunikaci s kamerou, trénink neuronové sítě pro detekci jízdních pruhů a samotný algoritmus detekce jízdních pruhů a určování trajektorie a polohy vozidla.

V závěrečné kapitole práce jsou prezentovány výsledky experimentálního ověření navrženého systému. Tato práce představuje konkrétní implementaci detekce jízdních pruhů pro autonomní vozidla a poskytuje pohled na problematiku s důrazem na praktickou aplikaci a experimentální ověření systému.

# 1 AUTONOMNÍ VOZIDLA

Autonomní vozidla představují revoluci v dopravním průmyslu a mají potenciál zásadně změnit způsob, jakým se dnes přepravují nejen lidé, ale také zboží. Tato vozidla využívají pokročilé technologie, jako jsou senzory pro vnímání okolí a sofistikované algoritmy zajišťující navigaci, řízení a parkování bez lidského zásahu. Vysoce automatizovaná a plně autonomní jízda má potenciál zlepšit bezpečnost na silnicích, snížit dopravní zácpy, optimalizovat spotřebu paliva a umožnit cestujícím věnovat čas jiným činnostem, jako je práce nebo odpočinek. V této kapitole se podrobněji seznámíme s úrovněmi automatizované jízdy a několika projekty zaměřenými na vývoj a implementaci autonomních vozidel ve světě.

## 1.1 ÚROVNĚ AUTOMATIZOVANÉ JÍZDY

Automatizace jízdy se dá rozdělit do několika úrovní podle stupně kontroly asistenčních systémů nad vozidlem a mírou zapojení řidiče do řízení vozu. Vegia a spol. [1] uvádí, že od padesátých let 20. století vzniklo již 12 systémů, které automatizaci jízdy dělí na různý počet úrovní. Dnes se většinou setkáme s rozdělením dle:

- NHTSA (National Highway Traffic Safety Administration)
- BASt Expert Group (Bundesanstalt für Straßenwesen)
- SAE International Standard J3016 (Society of Automotive Engineers)

Na *Obr. 1* lze vidět porovnání všech tří taxonomií. Všechny tyto taxonomie se snažily lépe definovat funkční limity automatizovaných subsystémů pracujících na různých úrovních autonomie se specifickou aplikací v automobilovém průmyslu. Přestože NHTSA i BASt definují pět úrovní automatizace, NHTSA spoléhá na číselný systém v rozsahu od úrovně 0 („žádná automatizace“) do úrovně 4 („plně autonomní řízení“), zatímco skupina odborníků BASt používá rozdělení v rozsahu od „Pouze řidič“ do „Plná autonomie“. Naproti tomu SAE, jak můžeme vidět na *Obr. 1* a *Obr. 2*, rozlišuje šest úrovní automatizace od nuly („žádná automatizace“) do pěti („plná autonomie“).

Panuje všeobecná shoda, že s rostoucí úrovní automatizace se řidič stále více vzdaluje fyzickým a kognitivním úkolům řízení. To se odráží v tabulce na *Obr. 1*, která ukazuje, jak jsou některé z primárních úkolů řízení rozděleny mezi řidiče a automatizované podsystémy. [2]

| Level | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| SAE | No Automation | Driver Assistance | Partial Automation | Conditional Automation | High Automation | Full Automation |
| NHTSA | No Automation | Function-specific Automation | Combined Function Automation | Limited Self-Driving Automation | Full Self-Driving Automation | |
| BASt | Driver only | Driver Assistance | Partial Automation | Highl Automation | Full Automation | |

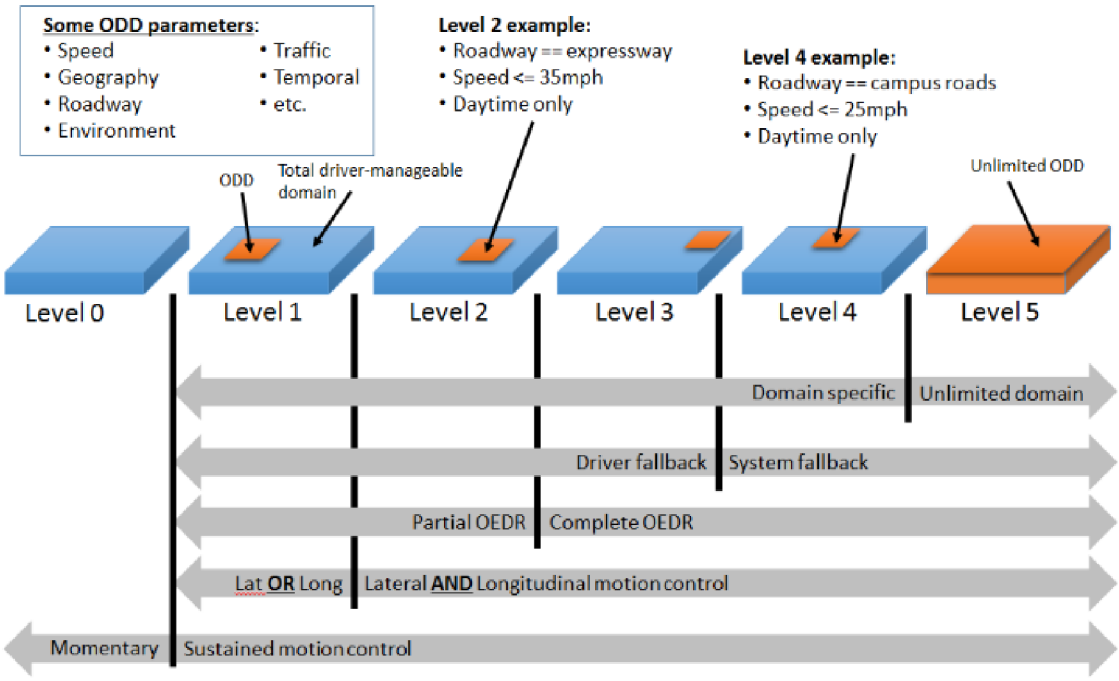
*Obr. 1* Porovnání úrovní automatizované jízdy dle taxonomií SAE, NHTSA a BASt [3]

### ÚROVNĚ AUTOMATIZOVANÉ JÍZDY DLE SAE J3016:

Jak již bylo zmíněno dříve, jedno z nejrozšířenějších a nejpoužívanějších rozdělení upravuje norma SAE J3016. Poslední aktualizace normy proběhla v roce 2021. Cílem tohoto rozdělení je popsat vztah řidiče a prvků automatizace jízdy, respektive jejich role během jízdy.

Norma definuje pojem DDT (Dynamic driving task). Jedná se o hromadné označení funkcí vozidla probíhajících v reálném čase, potřebných pro bezpečný provoz vozidla v silničním provozu. Mezi tyto funkce řadíme například identifikaci potenciálních rizik při jízdě, rozpoznávání překážek, brzdění, dodržování bezpečné vzdálenosti mezi vozidly, udržování vozu v jízdním pruhu atp.

Dalším důležitým pojmem je také ODD (Operational design domain), česky doména operačního návrhu. Jsou to provozní podmínky, pro které je navržen daný ADS (Automated driving system). Specifika jednotlivých úrovní automatizace a provozní podmínky shrnuje schéma na *Obr. 2*. [4]




*Obr. 2* ODD a úrovně automatizované jízdy [4]

Úroveň 0 neobsahuje žádné prvky automatizace. Jedná se tedy o manuální řízení a vozidlo může být doplněno o prvky aktivní bezpečnosti. Ty patří do varovných prvků ADS a nemají vliv na ovládání vozidla.

První dvě úrovně automatizace jízdy (úroveň 1 a 2) se týkají případů, kdy systém automatizace jízdy provádí dílčí úkoly řízení trvalého podélného a/nebo příčného řízení pohybu vozidla při DDT. Řidič tak nečiní, je za daný DDT odpovědný a v případě potřeby jej dokončí. Mezi tyto systémy jsou řazeny např. první generace asistentů jízdy v dopravní koloně, sledování jízdních pruhů, adaptivní tempomat nebo jejich kombinace.

Pokud systém automatizace jízdy provádí DDT, uživatel tak nečiní. Nicméně, pokud je připraven záložní DDT, očekává se, že uživatel DDT převezme, když dojde k poruše systému

související s výkonem DDT nebo pokud se při řízení automatizační systém chystá opustit svou doménu operačního návrhu (ODD). Asistenční systém například špatně vyhodnotí překážku nebo tvar jízdního pruhu a chystá se strhnout vozidlo do protisměru. Pak se očekává, že uživatel bude pohotově reagovat a je schopen obnovit výkon DDT. Toto rozdělení rolí odpovídá úrovni 3, kdy se již jedná o vysoce automatizovanou jízdu.

V případě, že systém automatizace řízení může provést celý DDT i záložní DDT buď v rámci předepsané ODD (Úroveň 4) nebo ve všech situacích provozu na silnici ovládaných řidičem (Úroveň 5), pak jsou všichni uživatelé přítomní ve vozidle, zatímco je zapnut ADS, cestující. Při úrovni 5 již tedy můžeme hovořit o autonomní jízdě a vozidlo tak nemusí být vybaveno volantem, pedály a dalšími ovládacími prvky spojenými s řízením. To pro úroveň 4 neplatí kvůli zmíněné ODD, mimo kterou není ADS funkční. [4] [5]

Obecně lze říci, že dnes vyvíjená autonomní vozidla dosahují buďto úrovně 2 (semiautonomní vozy), nebo úrovně 4 (plně automatizovaná jízda). Před jízdou se však každé vozidlo nachází v počátečním stavu na úrovní 0. Tzn. každé vozidlo je řízeno manuálně, dokud není řidičem (člověkem) spuštěn ADS. [6]

## 1.2 AUTONOMNÍ JÍZDA A PROJEKTY AUTONOMNÍCH VOZIDEL

Logické rozšíření tohoto vývoje úrovní automatizované jízdy je tedy systém, který dokáže trvale vykonávat všechny úkoly řízení, aniž by řidič musel systém kontrolovat. Řidič se stává cestujícím a může využít čas cesty k jiným činnostem. Mezi funkce automatizované jízdy patří samostatné hledání parkovacího místa (Automatizované parkování s obsluhou), jízda do garáže nebo odvoz dětí do školy bez řidiče. V neposlední řadě je zde naděje, že se sníží počet nehod díky rychlejším a spolehlivějším reakcím počítačem řízeného systému, na který, narozdíl od řidiče, nepůsobí únava. [5]

Řízení bez řidiče bylo několik desetiletí rozvíjeno v řadě výzkumných projektů. Již v roce 1986 profesor Dickmanns (Univerzita Federálních ozbrojených sil, Mnichov) představil „robotické vozidlo“, které bylo uzpůsobeno pro automatizovanou jízdu po rychlostní silnici nebo po dálnici. V roce 1995 tým profesora Dickmannse řídil další vozidlo z Mnichova do Kodaně a zpět s automatickým podélným a bočním vedením. Systém byl k dispozici po 95 % doby cesty. V roce 2004 se v USA konala DARPA Grand Challenge (Defense Advanced Research Projects Agency), první soutěž na delší vzdálenosti pro autonomní vozidla. [2]

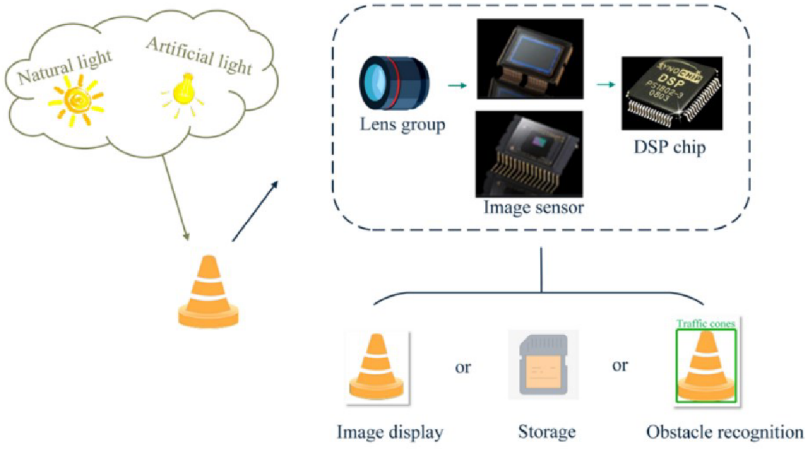
V posledních letech se vývoj autonomních vozidel značně urychlil, což vede k většímu počtu testovacích projektů a pilotních programů. Technologické giganty jako Google, Tesla a Uber, stejně jako tradiční automobiloví výrobci (Audi, BMW, Volvo), aktivně rozvíjejí autonomní technologie. Mezi nejznámější projekty patří například Waymo (Google) nebo Tesla Autopilot. Také existují různé pilotní programy ve městech a na univerzitách, jako jsou samořiditelné autobusy (Roboauto, Centrum dopravního výzkumu) nebo robotizované taxislužby. Přestože se průmysl rychle pohybuje směrem k plně autonomním vozidlům, stále čelí technologickým, právním a etickým výzvám.

# 2 SNÍMAČE AUTONOMNÍCH VOZIDEL

Tak jako se člověk (řidič) při řízení spoléhá na své smysly, díky kterým se dokáže orientovat na silnici, zaznamenat překážky atd., autonomní vozidla využívají senzory. Každé autonomní vozidlo využívá určitý set senzorů. V této kapitole jsou popsány základní typy snímačů využívané u autonomních vozidel. Detailněji je zde popsáno využití kamer, jelikož jsou také součástí praktické části práce.

U setu senzorů je důležitá redundance senzorů a využití různých typů senzorů. Tzn. že pro každý senzor existuje záložní řešení. Tím je zaručena kontrola daného snímače. Příkladem může být spojení kamery a radaru, kdy při snímání kamerou vozidlo nedokáže stoprocentně rozeznat stíny na vozovce od překážek. Současně s kamerou však okolí snímá také radar, který stíny vůbec nezaznamenává. Funguje tedy jako zpětná vazba, díky níž dokáže vozidlo správně reagovat. Takovémuto spojení snímačů se také říká fúze snímačů a je popsána v kapitole 3.1.1.

Snímání autonomních vozidel se dělí do třech kategorií s čímž souvisí také rozdělení senzorů [7]. Ty můžeme rozdělit na propriocepční (např. IMU), které se využívají k autodetekci vozu. Dále jsou zde lokalizační senzory (GPS, IMU), pomocí nichž je určena globální a lokální poloha vozidla. Posledním typem jsou senzory exterocepční (kamery, radary, LIDAR, atd.). Exterocepční senzory zajišťují snímání okolí (tzv. mapování), ale také aktuální stav řidiče [8]. Základní snímače autonomních vozidel jsou kamery, radary, lidary, ultrazvukové snímače, GNSS, IMU.




*Obr. 3* Schéma funkce kamery vozidla [9]

## 2.1 KAMERY

Kamery snímají obraz okolního prostředí za pomoci přirozeného nebo umělého osvětlení. Snímaný obraz považujeme za dvourozměrný, časově závislý, spojitý optický signál. Ten prochází soustavou čoček (objektivem) a pomocí obrazového snímače je převeden na signál elektrický. Poté je signál přes A/D (analogově-digitální převodník) a DSP (digital signal processor) transformován na digitální obrazový signál. Nakonec je snímek převeden do požadované podoby a je připraven k přímému zobrazení, uložen na paměťovou kartu,

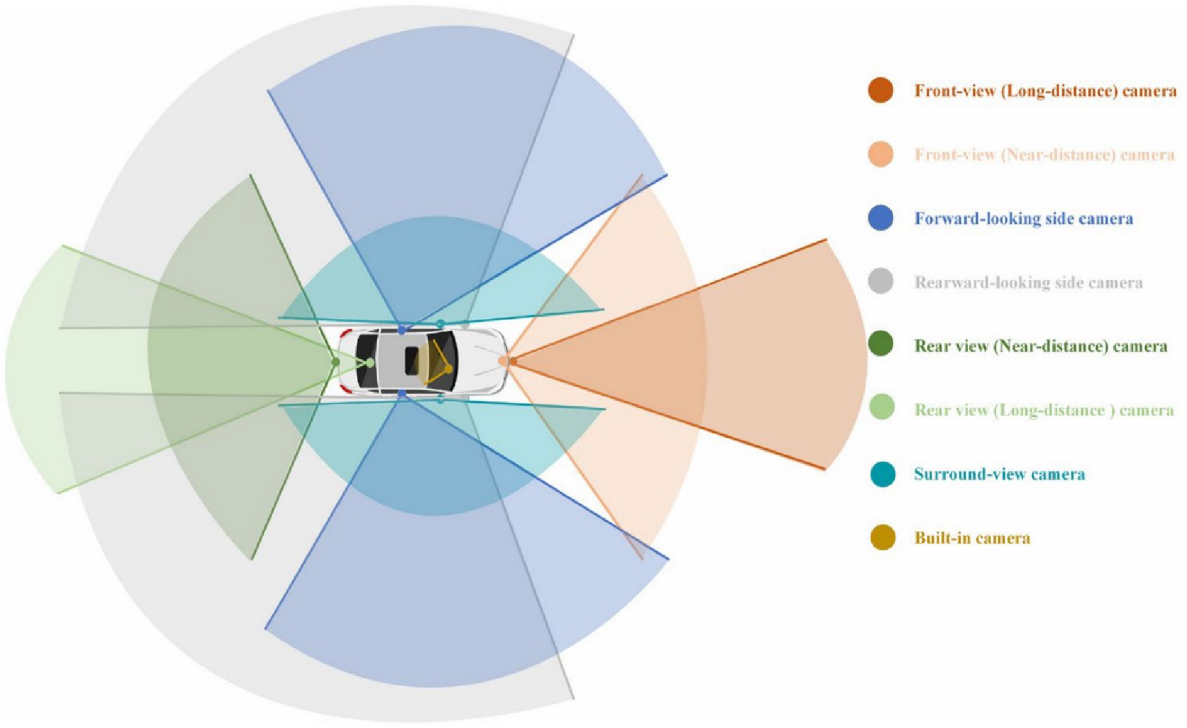
případně dál zpracován algoritmem pro automatické rozpoznávání stavu vozovky. Tento proces lze vidět na

*Obr. 3*.

Na kamery používané ve vozidlech jsou v praxi kladeny poměrně vysoké nároky. Vzhledem k tomu, že vozidla jezdí v různých klimatických podmínkách a na silnicích různé kvality, musí tyto náročné podmínky zvládnout také kamery ve vozidlech. K nejnáročnějším podmínkám, při nichž musí kamery fungovat, patří nízké a vysoké teploty, vlhkost, slabé nebo příliš silné osvětlení, vibrace atd. Nejdůležitější vlastnosti kamer jsou [9]:

1. Zorné pole – maximální úhel na straně objektu, při kterém může být objekt ještě reprodukován
2. Depth accuracy – rozdíl mezi naměřenými hodnotami a skutečnou vzdáleností ukazuje přesnost vzdálenosti
3. Rozlišení – hlavní faktor pro hodnocení výkonnosti kamery
4. Minimum illuminance – minimální potřebná osvětlenost snímače
5. Poměr signálu a šumu – poměr výstupního napětí signálu a šumu
6. Dynamický rozsah – poměr nejsvětlejších a nejtmavších odstínů, jaké je kamera schopna zobrazit v každém snímku

V automobilech můžeme vidět dva druhy aplikací kamer. Prvním z nich jsou systémy indikačních displejů, jako jsou například kamery zpětného pohledu. Zde je kladen důraz na vysoce kvalitní obraz a reprodukci barev. Obraz je většinou zpracován tak, aby se dosáhlo požadovaného dojmu na displeji. [9]




*Obr. 4* Rozdělení kamer vozidla podle jejich funkcí a umístění

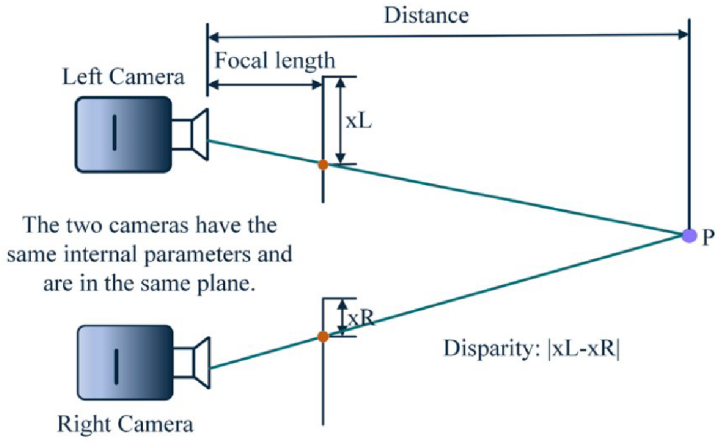
Druhým typem aplikací jsou systémy pro zpracování obrazu. Tyto aplikace využívají počítačové zpracování obrazu (computer vision) k extrakci relevantních dat. Zde již není nutné, aby zpracovaný obraz vypadal realisticky. Důležité je upravit obraz tak, aby byl algoritmus schopen správně vyhodnotit relevantní objekty. Tyto kamery mohou být využity jak pro informativní účely, například v systému automatického rozpoznávání značek, tak pro sběr dat pro ovládání akčních členů (automatické stmívání světlometů, parkovací asistent, senzory automatizovaného nebo autonomního řízení). S rostoucím stupněm automatizace jízdy vozidel rostou také nároky na výkon použitých kamer. [5]

Na *Obr. 4* můžeme vidět rozdělení kamer vozidla podle jejich funkcí a umístění do pěti kategorií: přední kamera, boční kamera, zadní kamera, panoramatická kamera a built-in camera.

Přední kamera slouží ke snímání objektů a světelné signalizace před vozidlem. Mezi nejčastěji používané typy patří **monokulární kamera**, binokulární kamera, kamery s více pohledy a hloubková kamera. Problém monokulární kamery je ten, že u ní neznáme měřítko zvětšení, a tak data získané takovou kamerou obsahují pouze informace o scéně. Tři a více monokulárních kamer tvoří **více pohledovou** kameru, kde jsou zastoupeny kamery různých ohniskových vzdáleností a dosahů. Je tak pokryta daleko větší oblast a zároveň umožněna detekce překážek a určování vzdáleností mezi nimi.

**Hloubkové kamery** zahrnují binokulární stereoskopické vidění, strukturované světlo a ToF (time of flight) kamery. Princip funkce stereo kamery je znázorněn na *Obr. 5*.

Pomocí dvou kamer jsou zachyceny dva obrazy (princip podobný lidskému zraku), extrahují se styčné body v obraze a na základě společných rysů se triangulací stanoví odchylka obou snímků a díky zvolené konverzní metodě je získána informace o hloubce. Fúzí obou snímků tedy dostaneme barevný hloubkový obraz.

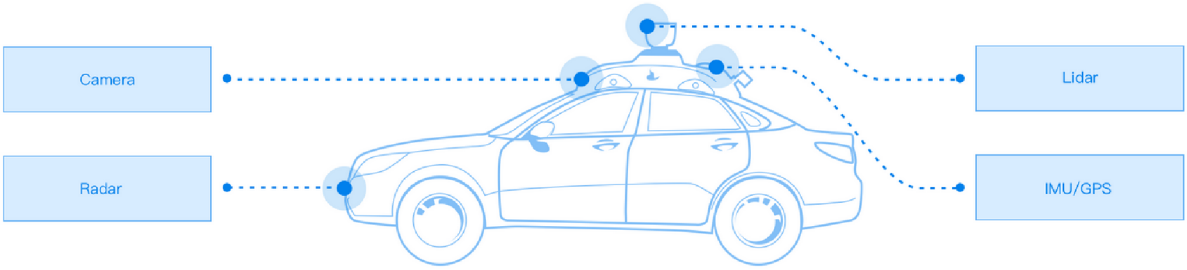



*Obr. 5* Principiální schéma funkce stereo kamery [9]

Informace o rozložení objektů v prostoru lze získat také za pomocí kamer se strukturovaným světlem a ToF kamer. Základem **kamer se strukturovaným světlem** jsou dva moduly, projekční a zobrazovací. Do prostoru jsou promítány paprsky světla o známé struktuře, které jsou poté snímány a na základě změn ve struktuře paprsků je vypočteno hloubkové rozložení pozorované scény.

**ToF kamery** pracují na principu měření doby letu odraženého světla od snímaného objektu. Výhodou je okamžitý 3D obraz a možnost snímání v reálném čase. Kamery poskytují v jednom záběru 2D i 3D obraz. [10]

Umístění přední kamery se může u různých typů vozidel lišit. Nejčastěji však bývá umístěna v horní části čelního skla vozidla nebo pod zpětným zrcátkem. Také může být součástí modulu na střeše vozu, tak jako u vozidel Apollo Robotaxi (viz *Obr. 6*)




*Obr. 6* Rozmístění percepčních snímačů Apollo Robotaxi [11]

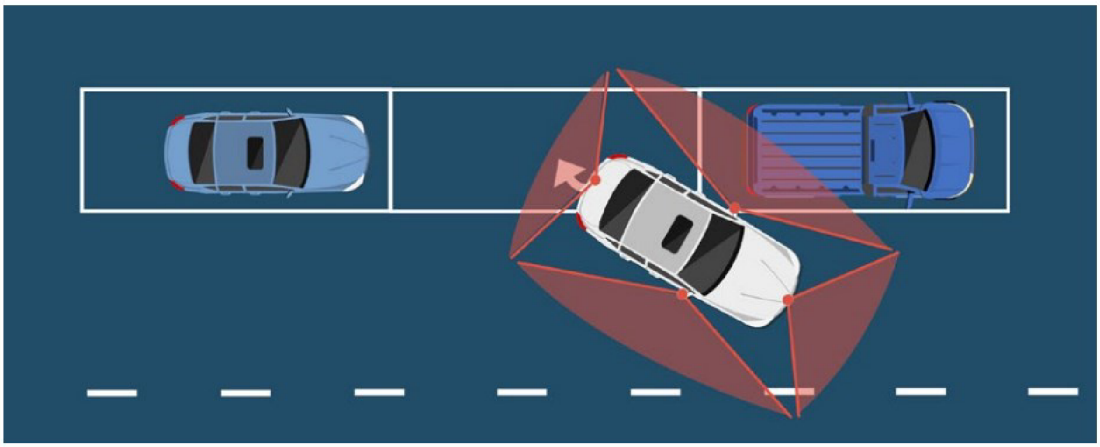
**Boční kamera** obvykle využívá širokoúhlý objektiv a snímá objekty kolem vozidla. Především řeší problém mrtvého úhlu vozidla. Běžně se přidává více bočních kamer, které včas zachytí blížícího se chodce nebo předjíždějící vozidlo. Boční kamery jsou rozmístěny po obou stranách karoserie vozidla.

Na B sloupcích vozidla bývají namontovány boční kamery s výhledem dopředu. Vybočení kamery je v rozmezí plus až mínus 60 stupňů a zorný úhel 90 stupňů. Tato kamera zaznamenává například vozidla, která se nečekaně zařadí do jízdního pruhu.

Pod bočními zpětnými zrcátky bývá instalována boční kamera s výhledem dozadu. Maximální pozorovací vzdálenost je 100 m a úhel vychýlení kamery musí být v rozmezí plus až minus 120 stupňů. Tato kamera sleduje mrtvý úhel za vozidlem.

**Zadní kamera** je užitečná při couvání, snímání parkovacích čar a překážek. Pro zajištění širokého zorného úhlu se běžně používají kamery s širokoúhlým objektivem. Umístěna bývá v blízkosti zadní registrační značky.

**Sférická kamera (panoramatická kamera)** slouží ke zvětšení pozorované oblasti a k eliminaci mrtvých úhlů vozidla. Obvykle využívá více objektivů s velmi širokým zorným úhlem a velkým soudkovým zkreslením (efekt rybího oka). Zkreslením zorné pole objektivu dosáhne až 220 stupňů, proto je ke složení celkového panorama potřeba méně snímků. Na *Obr. 7* níže je znázorněno zorné pole tohoto typu kamery.



*Obr. 7* Ukázka zorného pole panoramatické kamery – převzato [9]

**Built-in kamera** bývá umístěna u zpětného zrcátka vozidla, na volantu či A sloupku a je součástí DAS (driver alert system) nebo jiných systémů sledujících bdělost a koncentraci řidiče. Kamera snímá řidiče, následně je obraz vyhodnocen algoritmem, který je schopen odhadnout riziko spojené s chováním řidiče. [12]

Funkce klasické kamery je závislá na odrazu viditelného světla, což vede k častým chybám ve snímání za špatných světelných podmínek jako v noci nebo za špatného počasí. Tato vlastnost snižuje bezpečnost jízdy, proto se někdy používají také systémy nočního vidění, které umožňují sběr kvalitních dat za různých povětrnostních podmínek a intenzity světla. Systémy nočního vidění lze rozdělit do tří skupin podle principu, na kterém fungují: kamery se zesilovačem obrazu, aktivní infračervené noční vidění a termovize. Nejlepší schopnosti zabraňující dopravním nehodám mají systémy využívající termokamery, díky nimž dokáží rozpoznat chodce i v úplné tmě. Překážkou širokého využití termokamer je však jejich vysoká cena. Proto se v praxi spíše využívá kamer se zesilovačem obrazu. [9]

## 2.2 RADAR

Jak již bylo zmíněno, radar může při snížené viditelnosti doplnit kamerový systém autonomního vozidla a zlepšit jeho vnímání okolí. Tradičně se radar využívá k detekci lodí, letadel nebo v meteorologii. Funguje na základě vysílání radiových vln v pulzech. Jakmile se vlny odrazí od objektu, jsou zpětně zachyceny senzorem. [13]

Senzory radarů většinou využívají kmitočtově modulovanou nosnou vlnu (FMCW). Senzory dokáží díky zpoždění signálu a Dopplerova kmitočtového posuvu měřit a vyhodnocovat vzdálenost a radiální rychlost více cílů. Během zpracování signálu vygeneruje elektronika senzorů seznam cílů obsahující změřené pozice a rychlosti objektů zároveň s informací o jejich typu (vozidlo, chodec atd.). Seznam je poté zaslán do řídicí jednotky (ECU) vozidla. [14]

Stejně jako kamery ve vozidle jsou radarové senzory na vozidle umístěny tak, aby snímaly celé jeho okolí. Jsou schopny určovat rychlost a vzdálenost, nedokáží však rozeznávat například různé typy vozidel. [13]

Výhodou radaru je, že jeho funkce není, narozdíl od kamer, ovlivněna světelnými podmínkami. Také není tolik náchylná na povětrnostní podmínky. Jejich vliv nicméně není zanedbatelný a při využití radaru se musí počítat s určitým útlumem signálu vlivem srážek. [15]

## 2.3 LIDAR

Lidar (Light detection and ranging) je technologie založená na využití laserových paprsků k měření vzdáleností a trojrozměrnému mapování okolí. Lidar funguje na principu ToF podobně jako některé druhy kamer. Jedním modulem je vysílán laserový paprsek a druhým je přijímán odražený paprsek z okolí. Vzdálenost objektu je určena na základě doby letu paprsku dle následující rovnice níže, kde $R$ je vzdálenost, c rychlost světla a $\tau$ doba letu paprsku (ToF). [16]

$$R = \frac{1}{2} c\tau \tag{1}$$

Opakováním tohoto procesu je vytvořeno mračno bodů reprezentující prostor kolem vozidla ve 3D. K vytvoření 3D obrazu je potřeba jednotlivé objekty v okolí zachytit pod různými úhly.

Nejdůležitějšími měřítky pro hodnocení výkonnosti lidaru jsou jeho dosah, vysílaný výkon, vlnová délka a zorné pole, resp. pole záběru zorného pole (FoV), přesnost, rozlišení, rychlost pulzů a snímková frekvence. [16]

Vzdálenost, do které lze detekovat objekty, závisí především na vysílaném výkonu snímače, může být ovlivněna slunečním zářením nebo odrazivostí povrchů objektů. Vysílaný výkon je vypočítán na základě vlnové délky, průměru a pohybu paprsku. V aplikacích jako je autonomní řízení se udává maximální výkon laseru neboli maximální přípustná expozice laserem (MPE), kvůli interakci s očima chodců a dalších účastníků silničního provozu. [17]

Lidary autonomních vozů v současnosti využívají dvě hodnoty vlnové délky laseru, 905 nm a 1550 nm, které umožňují vysokou rozlišovací schopnost a rozlišení 3D zobrazení objektů v okolí vozidel. Vlnovou délku 905 nm využívají původní systémy Lidaru, které jsou levnější, mají však nižší dosah a jejich hlavní nevýhodou je samotná vlnová délka 905 nm. Ta se blíží vlnovým délkám viditelného světla a může tak proniknout až na sítnici a způsobit nevratné poškození očí. Vlnová délka 1550 nm se nachází za hranicí viditelného světla a je zde menší pravděpodobnost poškození. Senzory pracující s touto vlnovou délkou mají také vyšší dosah a přesnost. [16]

Vzdálenost objektů je nejčastěji měřena pomocí techniky pulzního nebo spojitého vlnění s využitím A/D převodníků. Jednodušší snímače často využívají implementaci detekce hran signálu. Jsou však méně přesné a náchylné na rušení okolními Lidary. Robustnější řešení na druhou stranu poskytují přesnější výsledky za cenu využití složitých obvodů a elektronických systémů. Všechna řešení využívají vysokofrekvenční signál v řádu jednotek GHz. [17]

Senzory LIDAR můžeme z pohledu konstrukce rozdělit do tří kategorií:

1. Mechanický LIDAR na bázi rotoru (Rotor-Based Mechanical LIDAR)
2. Skenující stacionární LIDAR (Scanning solid-state LIDAR)

3. Plně stacionární LIDAR (Full solid state LIDAR)


*Obr. 8* Velodyne Ultra Puck surround LIDAR [18]

Průkopníkem vývoje Lidarů je společnost Velodyne, jejíž Rotor-Based Mechanical LIDAR lze vidět na *Obr. 8* výše. Mezi další společnosti vyrábějící Lidary patří Hesai, Ouster nebo Robosense. V České republice se vývojem Lidarů zabývá například německá společnost Valeo.

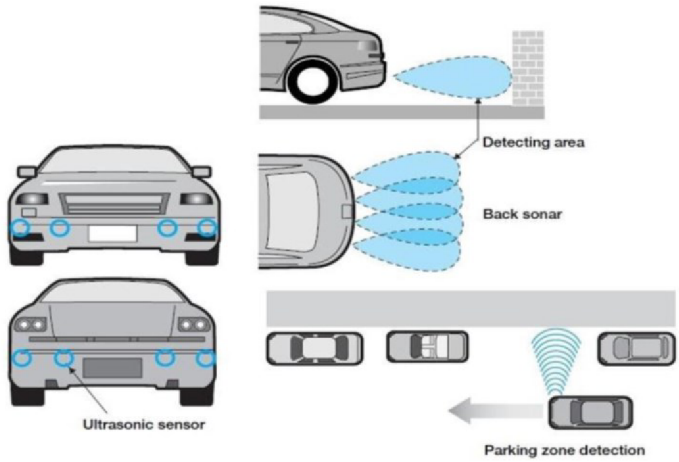
Využití Lidaru má oproti kameře a radaru řadu výhod. Je to zejména vysoká přesnost měření na dlouhé vzdálenosti až 300 m a zachycení a klasifikace malých objektů. Dále je oproti kameře velkou výhodou funkčnost systému za všech světelných podmínek. Narozdíl od radaru LIDAR dokáže bez problému klasifikovat objekty na základě obrysu a lze jej využít pro detekci jízdních pruhů.

Nevýhodou systému LIDAR jsou již zmiňované vysoké náklady, které zůstávají ze všech využívaných senzorů nejvyšší. Ve srovnání s kamerou má LIDAR nižší rozlišení obrazu a zároveň méně rozlišitelných tříd objektů. Také se potýká s nižší možností klasifikace na základě textury. [5]

Oproti radaru je detekce Lidaru náchylnější na povětrnostní podmínky. Největší problémy má systém s mlhou, sněhovou pokrývkou, sněžením nebo přímým oslněním sluncem. [17]

## 2.4 ULTRAZVUKOVÉ SNÍMAČE

Na *Obr. 9* lze vidět typické využití ultrazvukových snímačů. Ty se často využívají pro detekci překážek na krátkou vzdálenost a při nízkých rychlostech do 10 km/h [19]. Proto bývají součástí parkovacích asistentů. Jsou umístěny v náraznících vozidel a v případě, kdy se vozidlo přiblíží k překážce, je řidič upozorněn většinou zvukově nebo hapticky. Šestá generace ultrazvukových senzorů je také využívána pro funkci nouzového brzdění při detekci nečekaných překážek (například chodců). [5]




*Obr. 9* Aplikace ultrazvukových snímačů ve vozidlech [20]

Ultrazvukové senzory se nepoužívají pouze k varovným, ale také informativním účelům. Jelikož ultrazvukovými senzory nejsou vybavena pouze autonomní vozidla, nýbrž velká část nově vyrobených vozidel, některé aplikace využívají informace z těchto senzorů pro detekci volných parkovacích míst [21]. To znázorňuje *Obr. 9* výše, kde lze vidět také umístění těchto senzorů. Rychlost šíření zvuku ve vzduchu je řádově menší než šíření světla. Proto je možné je využívat při nízkých rychlostech. Rychlost zvukových vln je však závislá na teplotě, tlaku a vlhkosti zvuku, takže výkonnost ultrazvukových senzorů může být ovlivněna atmosférickými podmínkami. Některé systémy se proto snaží ultrazvukové senzory nahradit například radary. Automobilka Tesla ultrazvuk postupně nahrazuje svým systémem Tesla Vision, který by měl ultrazvukovou detekci zcela nahradit u nových modelů a částečně také u těch stávajících. [19]

## 2.5 GPS A IMU

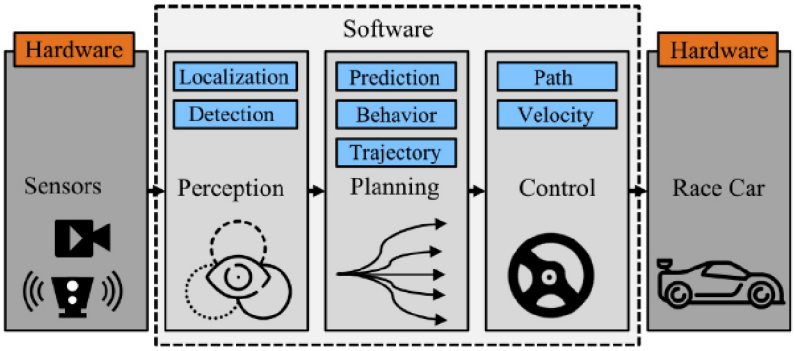
K určování polohy autonomní vozidla využívají systém GPS, který patří pod globální polohovací satelitní systém GNSS. Modul GPS pracuje jako pasivní přijímač a pro určení přesné polohy potřebuje informace o přesné poloze v čase alespoň tří satelitů. Na základě doby přenosu signálu a rychlosti šíření je dopočítána vzdálenost všech družic v dosahu a pomocí trilaterace je určena poloha, nadmořská výška, rychlost a směr pohybu vozidla. Pro přenos zpráv je využíván protokol NMEA. U systému GPS dochází k nepřesnostem vlivem: rozdílu času hodin družice a přijímače GPS, zpoždění signálu, nejistoty dráhy družice a vícecestného efektu šíření. [20]

K potlačení těchto nepřesností se data z přijímače GPS spojují s daty jiných přijímačů, jako je například inerciální měřící jednotka (IMU). Díky spojení těchto dvou senzorů se přesnost lokalizace posunula z 1 až 3 m na jednotky cm. [20]

IMU je jedním z nejdůležitějších snímačů u autonomních vozidel. Měří měrnou sílu, zrychlení a rychlost otáčení. Narozdíl od ostatních typů senzorů na něj nemají vliv povětrnostní a jiné podmínky, snímání také není ovlivněno vícecestným šířením narozdíl od GPS. IMU je složena z gyroskopů, akcelerometrů, případně magnetometrů. Jeden inerciální snímač dokáže měřit pouze podél jedné osy nebo kolem ní. Pro trojrozměrné měření se využívá tříosý inerciální snímač, což je sada tří jednoosých snímačů namontovaných do triády. Systém složený ze tří akcelerometrů a tří gyroskopů se také označuje jako šestiosý. [22]

# 3 DETEKCE JÍZDNÍCH PRUHŮ

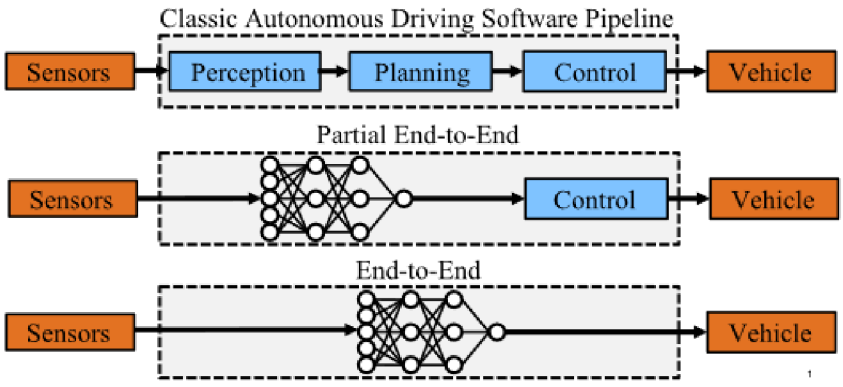
V této části bude představeno umístění detekce jízdních pruhů v kontrolním řetězci autonomního vozidla a dále přístupy a metody využívané pro detekci. Ty se dají rozdělit například podle způsobu zpracování dat nebo v závislosti na používaných algoritmech, či senzorech. Současně zde bude popsán proces detekce jednotlivých metod. Jelikož se tato práce zaměřuje na detekci pomocí kamery, odpovídají tomu také metody popisované v této části.




*Obr. 10* Schéma řídícího řetězce autonomního vozidla [23]

## 3.1 ŘÍDICÍ ŘETĚZEC AUTONOMNÍCH VOZIDEL

Autonomní řízení vozidla je dáno složitým systémem řízení, který zajišťuje jeho bezpečný pohyb po silniční komunikaci. Skládá se ze série senzorů, algoritmů a softwaru a jeho úkolem je v reálném čase vyhodnocovat kroky, schematicky znázorněné na *Obr. 10*. Obrázek znázorňuje klasické uspořádání úkonů řídícího řetězce.




*Obr. 11* Porovnání klasického, částečného end-to-end a úplného end to end přístupu v řídicím řetězci [23]

Existují však také další typy řídícího řetězce, které jsou spojeny s moderními metodami využívanými v autonomním řízení. Na *Obr. 11* níže lze vidět porovnání jednotlivých druhů uspořádání. Kromě klasického přístupu se využívá tzv. částečný end-to-end přístup a úplný end-to-end přístup. Částečný end-to-end přístup nahrazuje v řetězci pouze určité kroky (např. percepci a plánování) pomocí DNN (hluboká neuronová síť) a řízení vozidla je řešeno klasickým řídicím modulem [24]. Zatímco u úplného end-to-end přístupu je DNN nahrazen

celý řídicí řetězec. Znamená to tedy, že vstupem do DNN jsou data snímačů a výstupem jsou příkazy pro akční člen vozidla (například úhel natočení volantu nebo poloha akcelerátoru). [23]

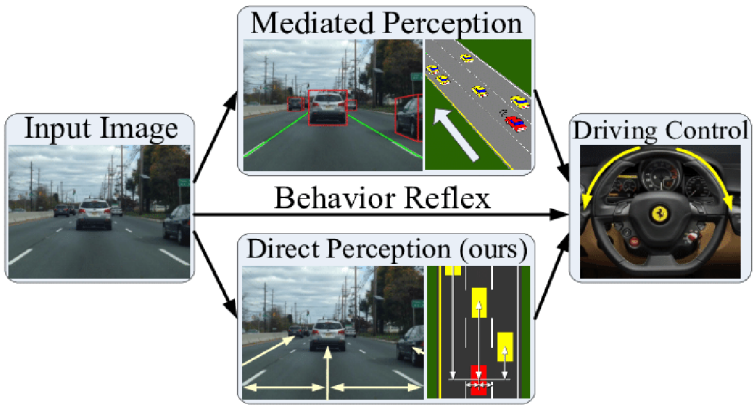
### 3.1.1 PERCEPCE

Detekce jízdních pruhů, jak lze vidět na *Obr. 10* Schéma řídícího řetězce autonomního vozidla , patří do kategorie vnímání okolí (percepce) vozidla. Senzory autonomního vozidla sbírají data z okolního prostředí a umožňují tak první krok řetězce algoritmů pro řízení autonomního vozu.

Percepce okolí AV je jeden z klíčových úkolů autonomního řízení. V zásadě musí tento krok poskytnout informace o tom, kde se vozidlo nachází v prostoru, identifikovat a klasifikovat okolní objekty a určit jejich polohu. Tyto informace musí být co nejkomplexnější a nejpřesnější, protože slouží jako vstupy pro plánování trajektorie a chování vozidla. Důležité je také poznamenat, že celý proces probíhá v reálném čase. Percepční systém AV je kritickým prvkem pro dosažení spolehlivého autonomního řízení. Percepce zahrnuje všechny algoritmy, které se starají o mapování okolí, lokalizaci vozidla nebo detekci jízdních pruhů a jiných prvků okolí vozu.

Na nejvyšší úrovni lze algoritmy vnímání rozdělit do tří kategorií dle *Obr. 12*:

1. Zprostředkované vnímání (Mediated perception)
2. Vnímání na základě reflexe chování (Behavior reflex perception)
3. Přímé vnímání (Direct perception)



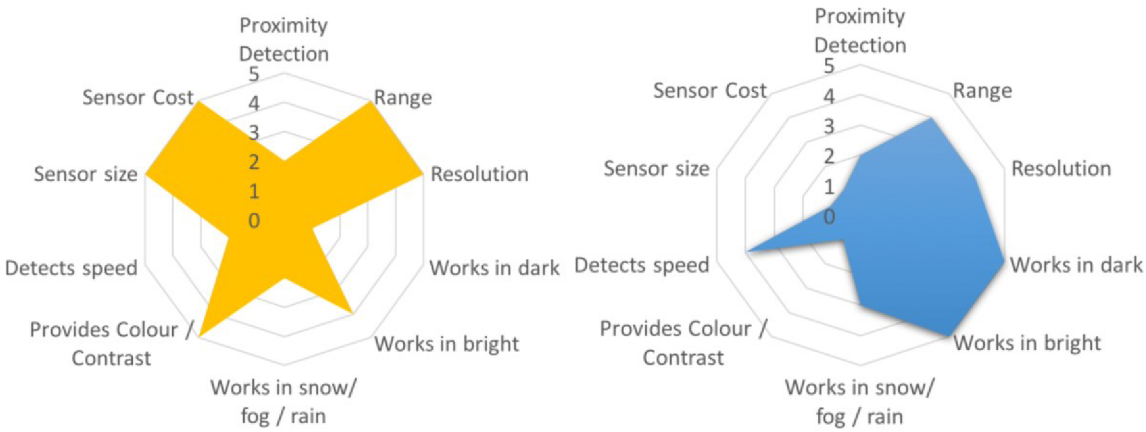

*Obr. 12* Tři kategorie algoritmů percepce [25]

#### ROZDĚLENÍ Z HLEDISKA VYUŽITÝCH SENZORŮ

Vnímání okolí může být zajištěno různými senzory (2). Prochází také přirozeným vývojem využití metod zpracování získaných informací (zpracování obrazu), díky rozvíjejícím se technologiím, zejména v růstu výpočetního výkonu, který umožnil implementaci složitějších a přesnějších algoritmů.

Z hlediska primárních senzorů se využívají buďto kamery, anebo přístupy mračen bodů, využívající LIDARy. Tyto přístupy se v dnešní době často kombinují tzv. fúzí snímačů.

Využití kamer je, hlavně v dnešní době, úzce spojeno s DNN. Tento přístup je do značné míry oblíbený hlavně pro svou cenovou dostupnost a možnost využití neuronových sítí. To vede k rychlejší implementaci v rámci systému a také vývoji. Pro využití kamer, coby hlavního percepčního snímače je však potřeba obrovské množství dat, které umožňují proces učení neuronových sítí. Pomocí těchto dat se tento systém dokáže v čase učit a zpřesňovat. Typickým zástupcem tohoto přístupu je například automobilka Tesla, která sází výhradně na systém kamer a nahrazuje jím i další snímače. [19]



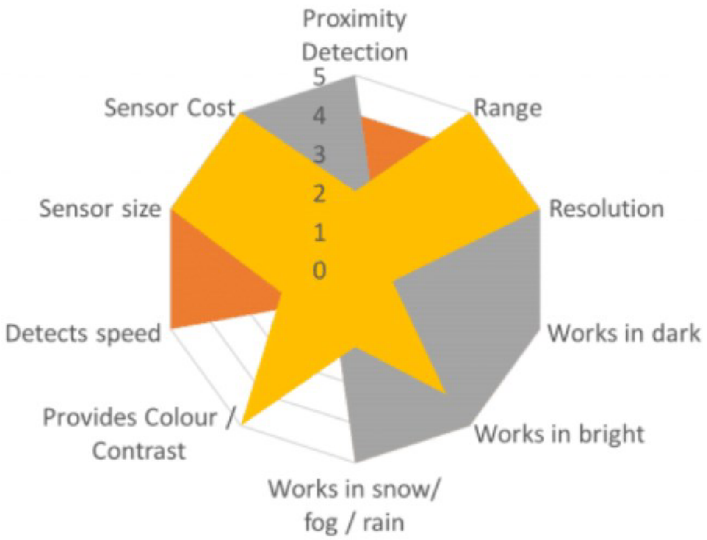

*Obr. 13* Relativní porovnání funkce kamery (vlevo) a lidaru (vpravo) pro využití k percepci AV [6]

Percepce založená na mračnech bodů na druhou stranu poskytuje konzistentnější výsledky a u případných chyb v detekci lze dohledat jejich původ. Ačkoliv je tento typ percepce velmi přesný, nedochází zde v průběhu času ke zlepšení výsledků. Často bývá kombinován s mapovacími technologiemi, aby vozidlo mělo přesnou představu o svém okolí. Opírá se převážně o data bodů nebo vzdáleností měřených objektů, shromážděných aktivními senzory (LIDAR). Algoritmy detekují objekty v okolí na základě hustoty, geometrie nebo vzoru mračna snímaných bodů. [26]

### FÚZE SNÍMAČŮ

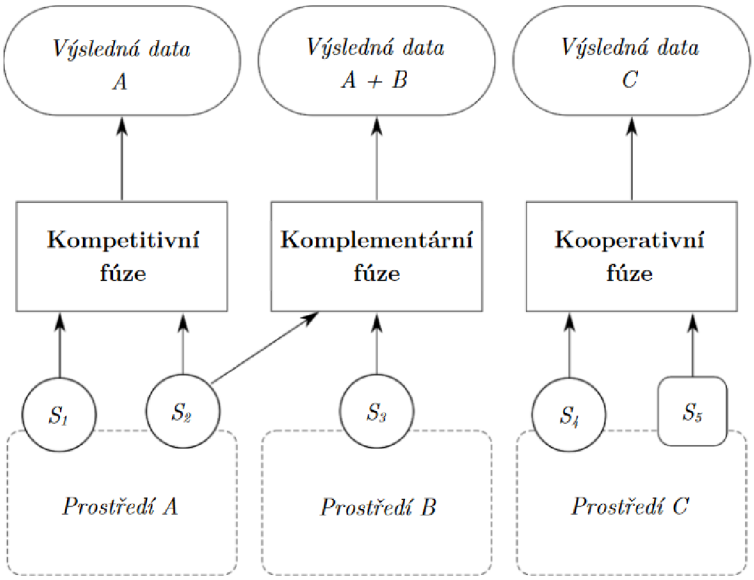
Jelikož mají všechny typy snímačů nějaké nedostatky, které při jejich využití zapříčiňují chyby v detekci, využívá se také přístup fúze senzorů. Ten spočívá kombinaci vstupů z více druhů senzorů, čímž se zvyšuje přesnost detekce. Můžeme porovnat využití samostatných senzorů na *Obr. 13*, kde byly jednotlivé vlastnosti senzorů ohodnoceny na škále od 0 (nejhorší) do 5 (nejlepší). Z uvedeného příkladu lze jasně vidět slabé a silné stránky kamer a lidarů. *Obr. 14* naproti tomu znázorňuje, jak může kombinace různých druhů snímačů přispět k eliminaci nedostatků a zvýšení výkonu percepce.

Výhody fúze senzorů jsou široce známé a multisenzorová fúze je častým tématem výzkumů. Typickým příkladem fúze senzorů je již zmíněná kombinace systému GPS a IMU. Další možností je využití několika senzorů stejného druhu snímajících stejné prostředí. Senzory tak zajišťují nezávislé měření a tím dochází k redundanci a zvýšení spolehlivosti snímání. Takovému typu fúze se říká kompetitivní [27].




*Obr. 14* Znázornění kombinace vlastností jednotlivých snímačů při fúzi (kamera žlutě, radar šedě, ultrazvuk oranžově) [6]

Komplementární fúze zajišťuje rozšíření zorného pole díky více nezávislých senzorů snímajících jinou část prostředí (např. kamery rozmístěny po obvodu vozu viz *Obr. 4* Rozdělení kamer vozidla podle jejich funkcí a umístění. Kamery a LIDAR spolu mohou také kooperovat, čehož se využívá často pro účely kalibrace [28]. Rozdělení fúze graficky popisuje *Obr. 15*.




*Obr. 15* Blokové schéma rozdělení senzorové fúze [28]

Fúze senzorů se využívá z mnoha důvodů, jako je zlepšení spolehlivosti výsledků, zvýšení pokrytí a použitelnosti, a především nižší počáteční náklady na set senzorů. I když s sebou nese vyšší výpočetní náklady, často se preferuje před použitím jednotlivých přesných senzorů,

jejichž počáteční náklady jsou obvykle vyšší než náklady na dva levnější senzory, které mohou výsledky srovnatelné s algoritmy jednotlivých senzorů dosáhnout prostřednictvím fúze dat. [8]

Matematicky lze dokázat, že kovariance měření dvou senzorů je menší než kovariance obou senzorů bez ohledu na jejich individuální rozptyly. Nicméně, i když je výhodné používat vysoce přesné snímače, musí být náklady na ně pro aplikace v automobilovém průmyslu stále přijatelné.

Fúze senzorů má různé způsoby řešení, které se liší v závislosti na konkrétní úloze. Neexistuje však univerzální postup pro sestavení modelu fúze. Existuje několik metod a algoritmů, které jsou často spojovány s tímto tématem. Kalmanův filtr je v oboru zpracování signálu nejznámější metodou pro fúzi dat. Jeho popularita spočívá v jednoduchosti a snadné implementaci. Podle výzkumu jsou metody fúze dat rozděleny do pravděpodobnostního přístupu, který se opírá o principy Bayesovské statistiky, a do fuzzy logiky, což je modernější přístup. [27]

### 3.1.2 PLÁNOVÁNÍ

Během kroku plánování autonomního vozidla je vytvořen plán pro budoucí pohyb vozidla. Tento plán se tvoří na základě předchozího kroku percepce okolního prostředí. Je zohledněn stav vozidla a cíl jízdy. Při predikci se analyzuje okolní prostředí a předpovídá se chování jiných účastníků silničního provozu. Tato data se používají k určení možných scénářů pro další pohyb vozidla.

Při určování chování vozidla se vyhodnocují různé možnosti pohybu a volí se ta, která nejlépe splňuje cíle jízdy a bezpečnostní požadavky. Mezi cíle jízdy patří dosažení cílové destinace nebo plnění dopravních předpisů. Po určení chování vozidla jsou zapojeny algoritmy pro určování trajektorie vozidla. Zde je vytvořen plán pohybu vozidla v čase. Tzn. rychlost, zrychlení a směr pohybu vozidla za současného plnění cílů jízdy.

Výsledkem kroku plánování je plán pohybu vozidla, který je následně využíván pro algoritmy řízení vozidla. Tyto plány se neustále aktualizují na základě nových informací o okolním prostředí a stavu vozidla. Je tedy důležité, aby algoritmy dokázaly pracovat v reálném čase.

### 3.1.3 ŘÍZENÍ

Krok řízení (Control) v autonomním řízení zahrnuje dva podkroky: určení dráhy (path) a rychlostního profilu (velocity). Tyto podkroky se používají k realizaci plánu pohybu vozidla vytvořeného v kroku plánování.

Během tohoto kroku je určen pohyb vozidla po určité trajektorii, která byla určena během kroku plánování. Cílem tohoto podkroku je dosáhnout určitého bodu na trajektorii a udržet se na ní. Zároveň je vypočtena správná rychlost vozidla na trajektorii. Cílem je dosáhnout určitého bodu trajektorie v daném čase a udržení přiměřené rychlosti tak, aby jízda splňovala vstupní a bezpečnostní požadavky. Tyto dva podkroky se provádějí současně a společně se starají o pohyb vozidla. Řízení vozidla je úzce spojeno s modelováním dynamiky vozu, nahrazením vozidla hmotným bodem, případně jednostopým či dvoustopým modelem a modely pneumatik. [23]

## 3.2 METODY DETEKCE JÍZDNÍCH PRUHŮ

### 3.2.1 ROZDĚLENÍ METOD V ZÁVISLOSTI NA POUŽITÝCH ALGORITMECH:

**Konvenční metody**: Tyto metody se obvykle používají jako základní krok při detekci jízdních pruhů. Mezi ně patří například Houghova transformace, Cannyho detektor hran, Sobelova operace nebo RANSAC.

**Metody založené na DNN**: Metody využívající algoritmy strojového učení, jako jsou například neuronové sítě. Tyto algoritmy se učí rozpoznávat jízdní pruhy na základě trénovacích dat a výsledky jsou poté použity pro detekci jízdních pruhů v reálném čase.

**Metody založené na příznacích**: Tyto metody využívají různé příznaky povrchu, jako jsou barva, textura nebo tvar, pro detekci jízdních pruhů. Tyto vlastnosti jsou získávány z obrazu a jsou poté použity k určení polohy jízdních pruhů.

**Metody založené na hloubkovém vidění:** Tyto metody využívají aktivní senzory jako jsou hloubkové kamery nebo lidary k získání informací o vzdálenosti od jízdních pruhů. Tyto informace jsou poté použity pro detekci jízdních pruhů.

**Kombinované metody**: Tyto metody kombinují několik přístupů pro detekci jízdních pruhů. Například mohou kombinovat konvenční metody s metodami založenými na strojovém učení nebo na příznacích.

### 3.2.2 ROZDĚLENÍ METOD V ZÁVISLOSTI NA ZPŮSOBU ZPRACOVÁNÍ DAT:

**Metody založené na klasickém počítačovém vidění** – tyto metody využívají tradiční počítačové vidění k detekci jízdních pruhů. Mezi tyto metody patří Houghova transformace, Sobelova operace, Cannyho detektor a další.

**Metody založené na DNN** – metody spoléhající na neuronové sítě a trénování modelů na detekci jízdních pruhů. Mezi tyto metody patří konvoluční neuronové sítě (CNN), rekurentní neuronové sítě (RNN), tzv. Region-based Convolutional Neural Networks (R-CNN) a další.

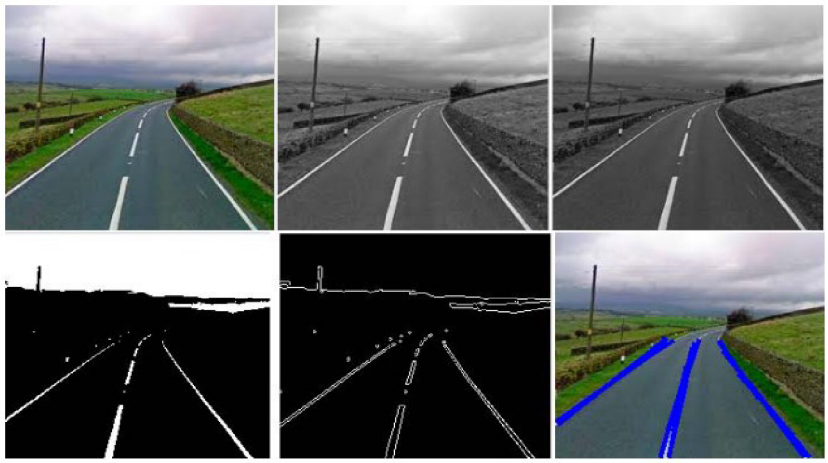
**Metody kombinující klasické počítačové vidění a DNN** – tyto metody kombinují prvky klasických počítačových vidění s využitím DNN pro zlepšení výkonu detekce jízdních pruhů.

## 3.3 KONVENČNÍ METODY DETEKCE ZALOŽENÉ NA POČÍTAČOVÉM VIDĚNÍ

Metody detekce jízdních pruhů založené na klasickém počítačovém vidění se spoléhají na tradiční techniky zpracování obrazu pro detekci hran, průsečíků a dalších geometrických vlastností jízdních pruhů. Mezi tyto metody patří například Houghova transformace, Sobelova operace, Cannyho detektor a další. Jsou založeny na matematických modelech a algoritmech,

které se snaží identifikovat hranice a oblasti, které odpovídají jízdním pruhům. Tyto metody mají výhodu v rychlosti zpracování dat, ale často mají problémy s robustností v neideálních podmínkách, jako jsou špatné světelné podmínky, špatný povrch silnice nebo nepřesné křivky jízdních pruhů. [29]

Na *Obr. 16* lze vidět příklad snímku, který prošel tímto algoritmem detekce. Vlevo nahoře vidíme pořízený záznam. Originální snímek byl převeden do odstínu šedi. Poté byl odfiltrován šum z obrazu. Následně zde byla provedena binarizace, detekce hran, a nakonec detekce pruhů. Jednotlivé metody předzpracování a následné detekce jsou mezi sebou různě kombinovány a výsledný algoritmus se tedy vždy mírně liší.



*Obr. 16* Průběh algoritmu detekce jízdních pruhů na zachyceném snímku [30]

#### OBLASTI ZÁJMU (REGIONS OF INTEREST)

Vstupem do algoritmu pro detekci jsou snímky kamery, kde je každý pixel reprezentován pomocí tří barevných komponent (RGB obraz). Tento obraz obsahuje pro algoritmus velké množství nepotřebných dat. Proto jsou ještě před zpracováním ve snímku nalezeny předpokládané oblasti zájmu (ROI). Místo zpracování celého obrazu se algoritmy detekce a sledování jízdních pruhů zaměřují pouze na specifické oblasti zájmu. Algoritmy v aktuálním snímku hledají oblasti zájmu pomocí výsledků předchozích zpracovaných snímků nebo uvažují předchozí znalosti o prostředí na silnici. [31]

### 3.3.1 PŘEDZPRACOVÁNÍ OBRAZU (PREPROCESSING)

Předzpracování obrazu zahrnuje odstranění šumu, vyhlazení obrazu, změnu barevné škály obrazu a případně jeho binarizaci. Předzpracování a filtrace obrazu jsou důležité zejména pro algoritmy využívající detekci hran.

#### PŘEVOD NA STUPNĚ ŠEDI

Běžnou praxí při zpracování obrazu je převod barevného obrazu na obraz ve stupních šedi, který je jednodušší na zpracování. Tento postup umožňuje snížit výpočetní nároky algoritmu, protože ke zpracování stačí pouze jeden kanál namísto tří. Převod na obraz ve stupních šedi se používá u algoritmů, kde barva obrazu není klíčová pro další zpracování, ale slouží pouze jako příprava pro následující kroky, např. pro hranovou detekci. Pro převod se využívá následující rovnice [32]:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B \tag{2}$$

#### VYHLAZENÍ OBRAZU (BILATERÁLNÍ FILTR)

Bilaterální filtr je neiterativní a jednoduchý způsob vyhlazení obrazu se zachováním hran. Tento filtr rozděluje obraz na velké prvky, jako jsou struktury, a malé prvky, jako je textura. Každý vzorek v obraze je nahrazen váženým průměrem jeho sousedů. Váhy jsou určeny dvěma faktory: vzdálenost od centrálního pixelu, což vede k větším váhám pro blízké pixely, a podobnost mezi sousedním a centrálním pixelem, což vede k větším váhám pro podobné pixely. [29]

#### BINARIZACE OBRAZU

Cílem binarizace je rozdělit obraz na dvě části – černou a bílou – takže lze jednoduše určit, kde jsou jízdní pruhy v obraze. Binarizace obrazu umožňuje snadnou identifikaci jízdních pruhů, protože tímto procesem se eliminuje mnoho nepotřebných informací v obraze, jako jsou například odlesky a stíny, což může způsobovat rušení detekce. Výsledkem binarizace obrazu je obraz s čistými bílými a černými pixely, které přesně vymezují jízdní pruhy. [30]

### 3.3.2 EXTRAKCE JÍZDNÍCH PRUHŮ A DETEKCE HRAN

V současných systémech pro detekci jízdních pruhů jsou nejčastěji využívány informace o gradientu a hranách obrazu. Tyto funkce vyžadují pouze malé množství výpočtů a umožňují extrahovat informace o pozici, kde se intenzity pixelů obrázku výrazně mění. Silně vykreslené jízdní pruhy produkují silné hrany, což usnadňuje jejich detekci. Avšak v závislosti na změnách prostředí mohou hrany jízdních pruhů být slabé a mohou být ovlivněny stíny nebo jinými faktory. Proto je potřeba vybrat vhodný detektor hran, který by dokázal zpracovat obraz i za těchto podmínek. Mezi nejlepší algoritmy pro detekci hran patří Cannyho detektor, který dokáže, v porovnání s ostatními algoritmy, extrahovat hrany s největší přesností. [33]

#### CANNYHO DETEKTOR

Cannyho detektor hran je metoda pro detekci hran v digitálních obrazech. Pro detekci jízdních pruhů se Cannyho detektor aplikuje na binarizovaný obraz, kde jsou jízdní pruhy bílé a pozadí černé. Cannyho detektor detekuje hranice mezi oblastmi s různou intenzitou a vytváří čáry, které označují hranice mezi jízdními pruhy a pozadím. Tento proces se skládá z několika kroků.

Obraz je vyhlazen pomocí Gaussova filtru, aby se snížil šum a vylepšila kvalita hran. Poté je určen gradient, na základě jednoduchého operátoru (např. Sobelův operátor). Následuje hledání lokálních maxim v gradientu v různých směrech a poté se všechna ostatní místa potlačí. Proto zůstanou jen ty body obrazu, které obsahují hrany. Posledním krokem je prahování s hysterezí, kdy dojde k eliminaci nevýznamných hran. Tím je zajištěna celistvost nalezených hran. [34]

### HOUGHOVA TRANSFORMACE

Houghova transformace využívá jednoduchý matematický model přímky ve dvou souřadnicových systémech k předpovídání přímek nebo jiných jednoduchých tvarů z obrazu. Základ Houghovy transformace představuje tato rovnice [32]:

$$r = x \cos\theta + y \sin\theta \tag{3}$$

Z algoritmu pro detekci hran dostaneme množinu bodů, které považujeme za součást vodorovného značení (čar). Houghova transformace tyto body využije k vytvoření přímek procházejících těmito body obrazu. Parametry $r$ a $\theta$ určují délku normály od přímky k souřadnému systému $r$ a úhel normály od osy $x$. Proměnné $x$ a $y$ reprezentují souřadnice konkrétního pixelu v obraze. Algoritmus postupně projde všechny body získané detekcí hran a v každém provede uvedený výpočet. Po vygenerování všech kombinací hodnot $r$ a $\theta$ ke každému bodu, nalezne hodnotu společnou všem bodům a tím i přímku, která body prochází. [32]

### CURVE FITTING

Protože Houghova transformace je určena k hledání přímek a jednoduchých tvarů, využívají se při detekci také další matematické funkce a modely jako jsou Beziérova křivka, Catmull-Romova křivka, Kubická B-křivka, parabola nebo klotoida. Tyto funkce se využívají pro aproximaci křivek v rovině, zejména se hodí pro detekci v ostrých zatáčkách. Tyto křivkové modely dosahují velmi dobrých výsledků, především v kombinaci s neuronovou sítí. [29]

### TRANSFORMACE OBRAZU NA PTAČÍ PERSPEKTIVU (IPM)

Další z možností je detekce jízdních pruhů na základě IPM (Inverse Perspective Mapping), což je technika transformace obrazu na ptačí perspektivu. Tato technika se používá pro zlepšení dosahu detekce. IPM transformace převádí obraz ze středového na ortogonální promítání, a tím odstraňuje efekt perspektivního zkreslení scény. Výsledkem je obraz se zmenšeným počtem informací, ale s větší přesností v horizontálním směru.

Výhodou IPM je, že umožňuje detekci jízdních pruhů na větší vzdálenosti a s vyšší přesností. Nevýhodou je zvýšená výpočetní náročnost a potřeba získat informace o výšce kamerového systému a geometrii vozovky. Hranové detektory jsou na druhou stranu méně výpočetně náročné než IPM, ale mají nižší přesnost a jsou náchylnější k chybám v důsledku šumu v obraze nebo změně prostředí. [35]

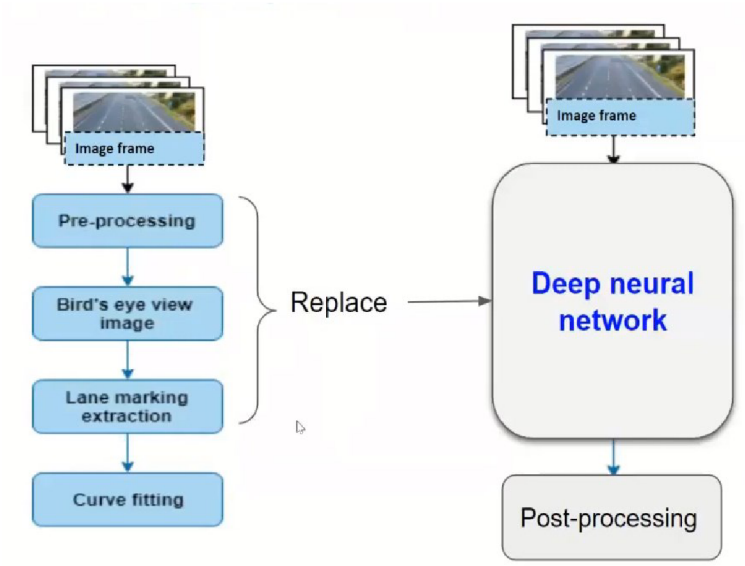
## 3.4 METODY DETEKCE ZALOŽENÉ NA KONVOLUČNÍCH A REKURENTNÍCH NEURONOVÝCH SÍTÍCH

Detekce jízdních pruhů je jedním z klíčových prvků autonomního řízení vozidel. Tradiční metody detekce jízdních pruhů jsou založeny na extrakci obrazových příznaků a následné klasifikaci obrazu pomocí konvenčních algoritmů. Nicméně, s vývojem DNN se stávají stále populárnější metody detekce jízdních pruhů založené na těchto sítích.

Nejčastěji používané modely jsou konvoluční neuronové sítě (CNN), rekurentní neuronové sítě (RNN) a Region-based Convolutional Neural Networks (R-CNN). Tyto modely jsou schopny zpracovávat obrazová data v reálném čase a vysoké přesnosti. Kromě detekce pruhů mohou být tyto modely také využity k sémantické segmentaci, která umožňuje rozlišit jednotlivé objekty na obrazu, včetně jízdních pruhů. [36]

Mezi populární metody pro detekci jízdních pruhů založené na neuronových sítích patří například U-Net nebo LaneNet, který využívá hluboké neuronové sítě pro detekci jízdních pruhů v reálném čase s vysokou přesností. LaneNet využívá kombinaci sémantické a instanční segmentace, aby dosáhl co nejlepších výsledků. [37]

Algoritmus pro detekci jízdních pruhů využívající DNN se již na první pohled liší od algoritmu konvenčních metod detekce. Záleží však také na zvolené metodě, jelikož algoritmy dělíme podle struktury na jednostupňové (end-to-end) a dvoustupňové. Jednostupňový přístup dokáže nahradit celý proces od vstupu až po postprocessing [36], zatímco u dvoustupňového probíhá nejdříve extrakce jízdních pruhů a poté v druhém kroku postprocessing. *Obr. 17* znázorňuje nahrazení IPM metody pro detekci dvoustupňovým algoritmem s DNN. Na obrázku lze vidět, že DNN nahradila jak fázi předzpracování, tak následné transformace a extrakce.




*Obr. 17* Porovnání řídicího řetězce konvenční metody a dvoustupňového přístupu [38]

CNN jsou založeny na sekvenci konvolučních operací, díky kterým jsou schopny z obrazu extrahovat složité tvary. Výborně se tedy hodí k extrakci jízdních pruhů z obrazu. Na začátku běžného procesu je zjednodušené předzpracování obrazu a na konci postprocessing.

Co se týče samotné detekce, přístupy k ní mohou být založené na segmentaci nebo predikci na základě kotev (Anchor based) [39]. Zheng a spol. [40] přidává navíc přístup k detekci s využitím parametrů pro modelování křivek jízdních pruhů.

### 3.4.1 PREPROCESSING

Ačkoliv DNN nahrazuje velkou část předzpracování, před vstupem do algoritmu je výhodné, podobně jako u konvenčních metod, vybrat ROI, se kterými bude DNN pracovat. Nepotřebné části jsou z obrazu odstraněny (např. část zabírající oblohu nebo kapotu vozidla). Procenta ušetřeného výkonu jsou pak přímo úměrná procentům odstraněné části obrazu.

Pro zlepšení procesu detekce je možné uměle rozšířit tréninkovou datovou sadu pomocí augmentačních technik. Mezi augmentační techniky se řadí například umělé přidání šumu do obrazu, pootočení snímků, jejich zrcadlení nebo zesvětlení. Rozšířený datový set má pozitivní vliv na robustnost detekce. Další možností je využití technik metaučení. [41]

### 3.4.2 POSTPROCESSING

Postprocessing u metod využívajících DNN slouží ke snížení počtu falešně-pozitivních výsledků a zpřesnění detekce. Techniky zpracování dat se liší v závislosti na konkrétní metodě detekce.

Obvykle během tohoto procesu dochází ke shlukování bodů náležících jednomu pruhu (clustering), filtraci pruhů pro odstranění falešně-pozitivní detekce. K vytvoření hladkých a přesných reprezentací pruhů se využívá např. polynomiální regrese.

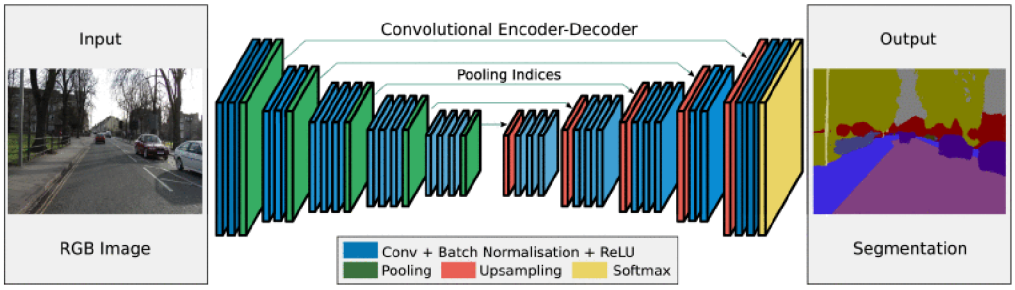
### 3.4.3 SEGMENTAČNÍ METODY

Segmentační metody detekce jízdních pruhů jsou založeny na rozdělení obrazu na části (segmenty) za účelem získání informací o scéně. Segmentační metody hodnotí parametry jednotlivých pixelů obrazu.

**Sémantická segmentace** každému pixelu přiřazuje label (štítek), jež jsou konkrétní typy objektů snímané scény. Tato metoda segmentuje celý obraz a jednotlivé objekty v obraze řadí do tříd. Sémantická segmentace narozdíl od instanční nedokáže rozlišit jednotlivé instance objektů. Všechny objekty stejného typu (např. jízdní pruhy, vozidla, budovy) jsou proto zařazeny do stejné třídy a síť jim přiřadí stejnou masku (barvu). Jízdní pruhy jsou detekovány pomocí hranic, nejčastěji v podobě čar na vozovce, které jsou rozpoznány na základě binární klasifikace každého pixelu, kdy algoritmus rozhoduje, zda se v dané části obrazu nachází či nikoliv. Další vlastností sémantické segmentace a CNN obecně je, že stejně jako konvenční metody pracují pouze s aktuálním snímkem.

Sémantická segmentace často pracuje s architekturou enkodér-dekodér (např. SegNet [42]), kde má síť enkodéru, např. pomocí funkce stride nebo max pooling, za úkol snížit rozlišení obrazu a umožňuje tvorbu souvislostí mezi jednotlivými pixely a generalizaci problému. Dekodér má

opačnou funkci – rozšířit vzorkovací frekvenci výstupu enkodéru. *Obr. 18* ukazuje průběh vzorkovací frekvence v enkodér-dekodér architektuře.



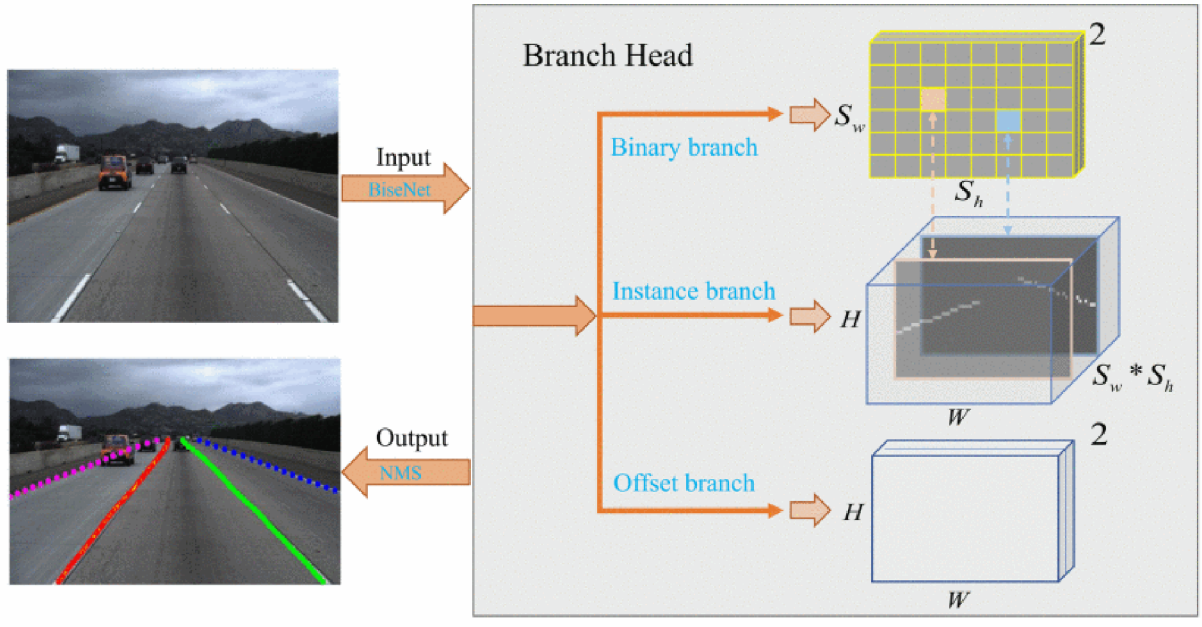

*Obr. 18* Ilustrace architektury SegNet na bázi enkodér-dekodér [42]

Jedním z problémů těchto metod je možná ztráta informací o prostorovém rozložení objektů, která je pro detekci pruhů klíčová. Tento nedostatek řeší například SCNN [43], kde konvoluce probíhají zvlášť pro každou část mapy příznaků. Tím umožňuje přenos informací mezi pixely v rámci jedné vrstvy. Výrazně tak zlepšuje výkon detekce jízdních pruhů, ale není vhodná u aplikací v reálném čase. RESA (REcurrent Feature-Shift Aggregator) využívá modul pro agregaci příznaků v reálném čase, díky kterému dokáže síť sbírat globální příznaky a zlepšovat výkon. RESA využívá silných apriorních informací o tvarech jízdních pruhů a zachycuje prostorové vztahy mezi pixely v řádcích a sloupcích. Přenos informací zde probíhá paralelním způsobem, což značně zvyšuje rychlost zpracování dat. [44]

Další metody využívají například distilaci znalostí nebo generativní modelování. Jejich výkon však pouze překonává původní metodu SCNN. Narozdíl od SCNN ale většinou předpokládají pevný počet jízdních pruhů. [41]

K rozlišení různého počtu jízdních pruhů lze využít dříve zmíněnou instanční segmentaci. Toho využívá např. LaneNet. LaneNet typicky potřebuje clustering dat v rámci postprocessingu pro vytvoření jednotlivých instancí. [36]

SoloLaneNet [45] patří také mezi metody **instanční segmentace**, ale jedná se o end-to-end metodu, využívající jednostupňový detektor. Namísto pixelů se zde k reprezentaci pruhů využívá klíčových bodů. SoloLaneNet tedy přímo generuje masky pro jednotlivé instance. Na *Obr. 19* lze vidět SoloLaneNet framework. SoloLaneNet využívá tři větve, kde první (binary branch) identifikuje, zda se v obraze vyskytují značení jízdních pruhů, druhá větev (instance branch) generuje klíčové body jednotlivých instancí jízdních pruhů, včetně masek. Třetí větev (offset branch) zajišťuje predikci relativních posunů klíčových bodů od levého horního rohu.

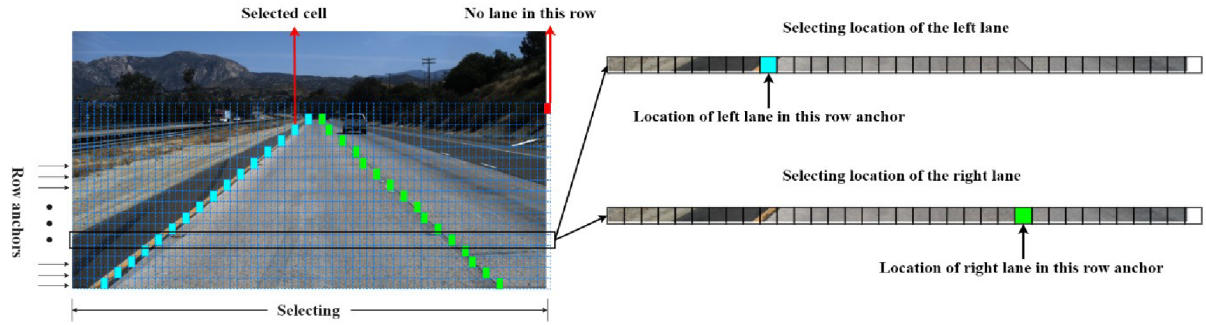



*Obr. 19* SoloLaneNet framework [45]

### 3.4.4 ANCHOR BASED PŘÍSTUP

Tento přístup je založený na předem definovaných kotvících rámečcích, rozmístěných po celé ploše obrazu. Kotvící rámečky (anchor boxes) mají předem definované rozměry a slouží k predikci polohy a tvaru objektů na základě relativního umístění objektů vůči nim. V prvním kroku jsou typicky pomocí CNN vypočteny pravděpodobnosti a regrese vztahující se ke každému kotvícímu rámečku. To nám umožňuje určit, zda se v rámečku nachází nějaký objekt. Poté je určeno skóre NMS (non-maximum suppression), které slouží k odstranění překrývajících se detekcí a získání konečného seznamu objektů v obraze.

Tento přístup využívají např. metody **row wise**. Row wise metody zpracovávají obraz řádek po řádku předem definované mřížky. Obraz je rozdělen na kotvící řádky, každý řádek na určitý počet buněk. Algoritmus pak lokalizuje ty buňky, které obsahují značení jízdních pruhů. Tento proces je vyobrazen na *Obr. 20*. [46]




*Obr. 20* Lokalizace buněk obsahujících značení jízdních pruhů u row-wise metody [46]

Row wise metody jsou v podstatě upravené segmentační metody. Metody však nepracují na úrovni pixelů a jsou tudíž podstatně rychlejší. Výborná latence je zde však vykoupena horší přesností. Metody jsou tedy vhodné pro algoritmy pracující v reálném čase s omezeným výpočetním výkonem. [39]

### 3.4.5 DETEKCE NA ZÁKLADĚ PARAMETRŮ

Metody založené na parametrech, jako je např. PolyLaneNet modelují křivky jízdních pruhů pomocí sady parametrů, které jsou využity jako stupeň polynomu u reprezentace křivek. PolyLaneNet využívá jako základnu (backbone) CNN k extrakci příznaků z obrazu. Následně jsou příznaky předány několika plně propojeným vrstvám k predikci koeficientu polynomu pro každý pruh. [47]

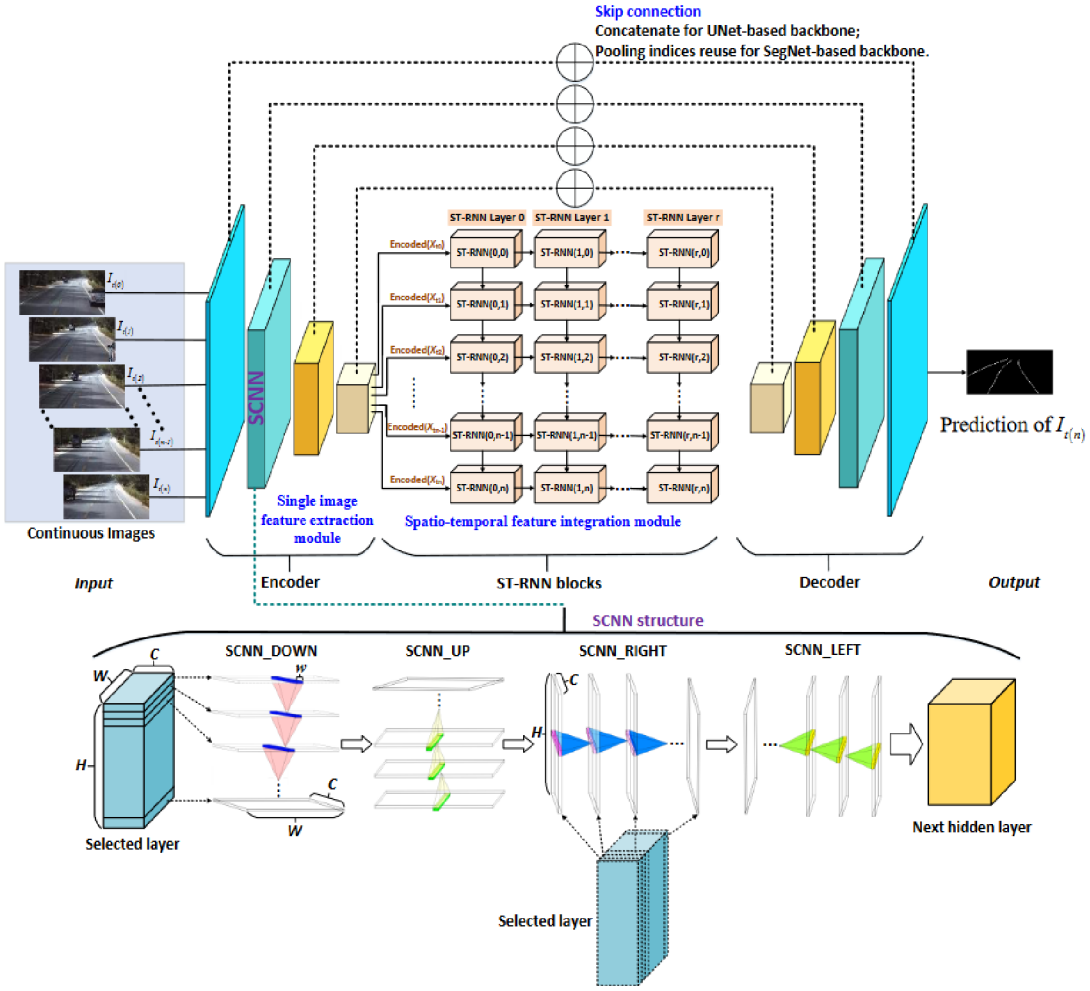
Tato metoda dosahuje, navzdory malého počtu parametrů pro reprezentaci křivek, vysoké efektivity. Jelikož je metoda založena na polynomiální regresi, lze k řešení využít lineární algoritmy. Metoda je však citlivá na chybné předpovědi koeficientů, což může způsobit změny tvaru detekovaných pruhů. Reprezentace křivek polynomem navíc nemusí být dostatečná pro zachycení složitých tvarů. [40]

### 3.4.6 HYBRIDNÍ METODY CCN A RNN

Donedávna se většina deep learning metod soustředila na extrakci příznaků pouze z jednoho aktuálního snímku díky CNN. Metody založené na RNN nebo spojení RNN a CNN však umožňují extrakci ze sekvencí snímků a následně analyzovat jejich temporální vztahy. Tato oblast však ještě není zcela prozkoumána.

Mezi tyto metody patří GRU (Gated recurrent unit), ConvGRU, LSTM (Long short term memory) nebo ConvLSTM. Zmíněné metody však dostatečně nevyužívají základní vlastnosti pruhů jako jsou dlouhé souvislé či přerušované pruhy. Stejně tak jako zde není využita závislost časoprostorových (spatio-temporal) závislostí ve snímané sekvenci. Proto jsou jejich výsledky v některých případech stále nedostatečné a existuje zde prostor pro zlepšení. [39]

Za tímto účelem byla navržena metoda SCNN_UNet_ConvLSTM2, která aktuálně drží prvenství při testování na TuSimple datasetu [48]. Jedná se o hybridní metodu využívající CNN a RNN, přičemž pro zahrnutí časoprostorových závislostí je zde zakomponován ST-RNN (Spatial-temporal recurrent neural network) modul. Na *Obr. 21* je vyobrazena celá architektura metody.




*Obr. 21* Architektura hybridní metody SCNN_UNet_ConvLSTM2 [39]

Jedná se o end-to-end metodu založenou na architektuře enkodér-dekodér, do které jsou implementovány dvě základny, U-Net a Seg-Net, využívající techniku skip connections [39]. Technika skip connection pomáhá zachovat informace o extrahovaných příznacích spojením vrstev enkodéru a dekodéru. Díky implementací SCNN do sítě enkodéru je zajištěna extrakce informací o prostorovém rozložení objektů v rámci jednotlivých snímků. Strukturu SCNN můžeme vidět ve spodní části *Obr. 21*. Podrobněji SCNN popisuje zmíněný Pan a spol. [43]. Modul ST-RNN je vložen mezi síť enkodéru a dekodéru, kde jeho vstupem jsou extrahované příznaky z enkodéru a výstupem časoprostorové závislosti dále vstupující do dekodéru. Pro modul ST-RNN byly použity různé typy RNN. Nejlepší výsledky však byly dosaženy s využitím ConvLSTM a ConvGRU.

## 3.5 KONVOLUČNÍ NEURONOVÁ SÍŤ

Jak již bylo zmíněno v předchozí kapitole o metodách detekce, CNN jsou jedním z nejpoužívanějších typů neuronových sítí pro úlohy klasifikace obrazu. Využívají se také pro binární segmentaci, a to zejména při práci s omezeným množstvím dat pro trénink sítě. [49] Jsou spojeny s využitím „pohybujícího se okna“, které prochází obrazem a identifikuje

specifické vzory. CNN však mohou být výpočetně náročné, což může být výzvou při jejich nasazení. Typicky se tento typ sítí skládá ze tří základních vrstev (částí):

1. Konvoluční vrstva (Convolution layer)
2. Sdružovací vrstva (Pooling layer)
3. Aktivační funkce (Activation function)

### 3.5.1 KONVOLUČNÍ VRSTVA

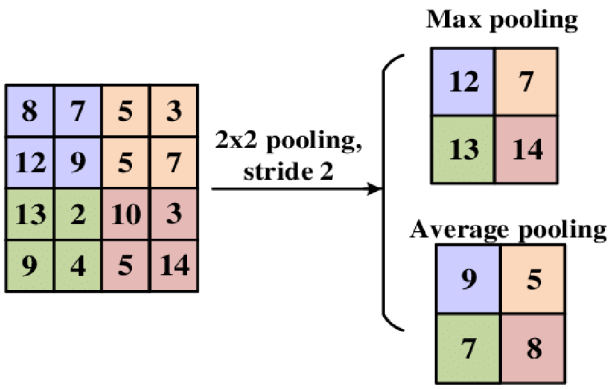
Zde probíhají konvoluční operace, které se učí lokální vzory v prostoru vstupních rysů. V případě obrázků jsou to vzory nalezené v malých 2D oknech (kernelech) definované velikosti, např. 3 x 3 px. Tato vlastnost umožňuje síti rozpoznat vzor, který se naučila v levé části obrázku i v pravé části obrázku, aniž by se musela daný vzor znovu učit. Konvoluční vrstvy se mohou také naučit prostorovou hierarchii vzorů. Tzn. první konvoluční vrstva se naučí malé lokální vzory, druhá už se učí abstraktnější vzory složené z rysů předchozí vrstvy. [49]

Konvoluce pracují s mapami vstupních rysů (feature maps), což jsou 3D tenzory s definovanou výškou, šířkou a hloubkou (osou kanálů). U RGB obrazu je rozměr hloubky 3, u černobílého 1. Okno konvoluce extrahuje ze své mapy vstupních rysů políčka (patches), použije na ně stejnou transformaci a vytvoří mapu výstupních rysů (output feature map), což je výsledek aplikace okna na vstupní obraz. Výstupní výška a šířka se může lišit v závislosti na kroku konvoluce (stride), hraničních faktorech a vycpávky (padding).

### 3.5.2 SDRUŽOVACÍ VRSTVA

Konvoluce s definovaným krokem (větší než 1) se v praxi používá jen zřídka, i když pro některé modely mohou být užitečné. Častěji se však využívá principu sdružování hodnot. [49] Standardně se jedná o sdružování hodnot dle maxima (max pooling) nebo průměrné hodnoty (average pooling).

Výsledkem těchto operací je snížení velikosti map rysů při zachování vlastností. Principem sdružování je určení maximální, respektive průměrné hodnoty z několika okolních hodnot v políčku viz *Obr. 22*. [49]



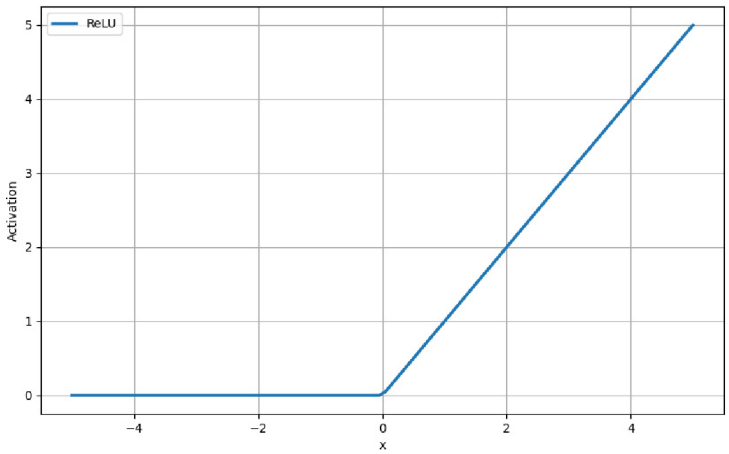

*Obr. 22* Ukázka sdružování hodnot dle maximální a průměrné hodnoty [50]

### 3.5.3 AKTIVAČNÍ FUNKCE

Aktivační funkce je spuštěna po každé konvoluční nebo plně propojené vrstvě sítě. Nejprve je tedy provedena lineární operace. To znamená, že vstupní data jsou vynásobena váhami a přičte se k nim zkreslení (bias). Výsledné hodnoty jsou předány aktivační funkci, která následně přetvoří hodnoty do nelineárního tvaru. Tento proces umožňuje neuronové síti naučit se složité vztahy v datech. Výstupy aktivační funkce poté vstupují do další vrstvy sítě.

#### RELU

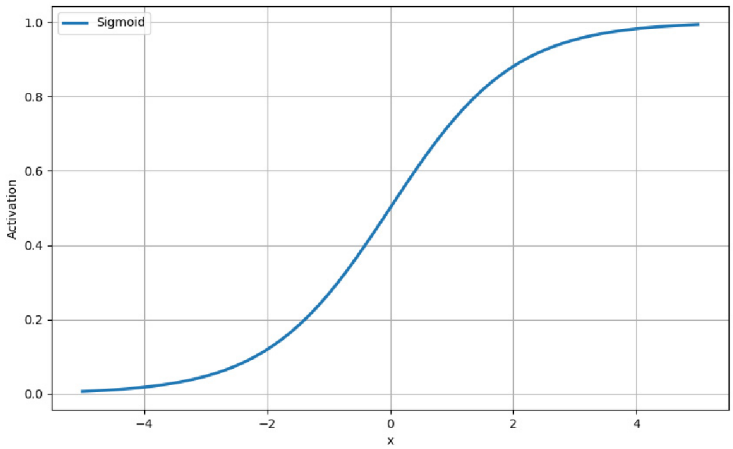
Aktivační funkce ReLU (Rectified Linear Unit) se nejčastěji používá u mezivrstev sítě. Je určená k vynulování záporných hodnot (viz *Obr. 23*). Kladné hodnoty jsou ReLU zachovány.




*Obr. 23* Aktivační funkce ReLU [51]

#### SIGMOID

Aktivační funkce sigmoid (vyobrazena na *Obr. 24*) je nejčastěji využívaná při binární klasifikaci. Jejím výstupem je pravděpodobnost (skóre mezi 0 a 1). Udává, s jakou pravděpodobností výstup přísluší pozitivní třídě. Aplikuje se v poslední vrstvě sítě.




*Obr. 24* Aktivační funkce sigmoid [51]

# 4 VÝBĚR METODY DETEKCE

Při detekci jízdních pruhů mohou vznikat problémy, které mohou ovlivnit přesnost detekce. Jedním z nich může být špatná viditelnost čar hranic pruhu v důsledku stárnutí barvy na vozovce. Problematická může být struktura povrchu vozovky, která může vést k nesprávné interpretaci dat. Dalším faktorem ovlivňujícím přesnost detekce je, že jízdní pruhy nejsou vždy vymezeny čarami. Jejich hranice mohou být označeny pouze obrubníky, svodidly nebo jinými značkami. Dále mohou být čáry překryty jiným značením, což může způsobit ztrátu informace o jízdním pruhu. Špatné světelné a povětrnostní podmínky také mohou ovlivnit detekci jízdních pruhů a snížit její přesnost.

Tyto problémy lze řešit různými způsoby. Problémy spojené se špatnou viditelností lze řešit správným zpracováním obrazu. Chybějící značení lze zase nahradit algoritmy, které berou v úvahu vlastnosti vozovky. Některé metody detekce jsou robustnější než jiné, ale ve většině případů je výhodné metody společně kombinovat.

Pro řešení detekce jízdních pruhů byla v této práci použita metoda využívající CNN. Konkrétně byl zvolen částečný end-to-end přístup využívající architekturu U-Net a skip connections. Výstupem modelu tedy nejsou příkazy pro jednotlivé části řízení, nýbrž detekované hranice jízdních pruhů, které následně slouží k návrhu vhodné trajektorie a výpočtu polohy vozidla vůči středu vozovky. Využití CNN namísto RNN bylo zvoleno s ohledem na časovou a výpočetní náročnost trénovacího procesu RNN. K natrénování CNN je rovněž potřeba méně dat a zároveň je zde předpoklad nižší výpočetní náročnosti při implementaci modelu neuronové sítě, a tedy lepší latence výsledného algoritmu. Pro zohlednění časových závislostí (největší přínos RNN) bylo do postprocesu zahrnuto časové průměrování, které bere v úvahu predikce z předchozích snímků. Podobného postupu bylo využito i při návrhu trajektorie a výpočtu polohy vozidla.

Využitím architektury U-Net se rozumí metoda sémantické segmentace. V tomto případě se jedná o binární segmentaci, jelikož má model za úkol určit, zda jednotlivé pixely náleží hranicím jízdního pruhu, či nikoliv. Metoda sémantické segmentace je sice spojena s vyšší výpočetní náročností, protože zohledňuje každý pixel obrazu, u navrženého algoritmu se však předpokládala vyšší robustnost než u konvenčních metod. Jednalo se tedy o kompromis mezi výpočetní náročností RNN a nestabilitou konvenčních metod detekce.

# 5 KOMUNIKACE S KAMEROU ZED 2

V této práci byla jako hlavní snímač detekce využita kamera, konkrétně kamera ZED 2 od společnosti Stereolabs (*Obr. 25*). Díky kameře mohla být nasbírána data pro experimentální ověření algoritmu detekce jízdních pruhů. Kamerou ZED 2 a počítače (PC) byl pořízen záznam vozovky, k jednotlivým snímkům vytvořena segmentační maska a algoritmus byl následně otestován pomocí těchto připravených dat. Aby bylo možné data nasbírat, bylo nejprve potřeba vytvořit skript pro komunikaci s kamerou, která snímá vozovku. Skript byl vytvořen ve vývojovém prostředí PyCharm v jazyce Python.


*Obr. 25* Kamera ZED 2 od firmy Stereolabs [52]

## 5.1 UMÍSTĚNÍ KAMERY

Kamera byla umístěna v testovacím vozidle Citroen Jumpy, tak jak je znázorněno na *Obr. 26*. Připevněna byla v držáku na palubní desce automobilu, v blízkosti čelního skla. Pravý objektiv kamery byl zarovnán na střed, aby bylo zajištěno, že střed snímaného obrazu bude reprezentovat podélnou osu vozidla.

*Obr. 26* Umístění kamery ZED 2 ve vozidle

## 5.2 KAMERA ZED 2

Pro práci bylo využito pouze jednoho objektivu kamery, nicméně kamera je primárně stereoskopická a dosahuje rozlišení s maximální snímkovou frekvencí dle Tab. 1. Zároveň je vybavena IMU, barometrem a snímači teploty včetně teplotní regulace. Další parametry obrazu jsou shrnuty v Tab. 2. Propojení kamery s PC bylo umožněno skrze USB 3.0 a vývojářský balíček ZED SDK 4.0. Kamera také podporuje Win 10 s minimálním požadavkem 4 GB RAM a grafické karty NVIDIA s minimálně 2 GB paměti.

Tab. 1 Maximální snímková frekvence kamery při zvoleném rozlišení

| Rozlišení | Max. snímková frekvence [s-1] |
|---|---|
| 2x (2208x1242) | 15 |
| 2x (1920x1080) | 30 |
| 2x (1280x720) | 60 |
| 2x (672x376) | 100 |

Pro práci s kamerou byl k dispozici Laptop ASUS GL502V vybavený grafickou kartou NVIDIA GEFORCE GTX 980 M. K zajištění kompatibility obou zařízení a IDE PyCharm, ve kterém byl algoritmus navrhován, byla do PC naistalována ZED Python API a balíček CUDA 10.2 Toolkit. Z důvodu kompatibility s Maxwell architekturou grafické karty nebylo možné využít nejnovější ovladače a ZED SDK.

Tab. 2 Parametry kamery pro rozlišení HD 720

| Rozsah detekce 2D [m] | Zorné pole H x V x D [ ° ] | Clona [ ] | Formát obrazu [ ] |
|---|---|---|---|
| 0,2 - 40 | 110 x 68 x 120 | f/1,8 | 16:09 |

### KALIBRACE KAMERY

Před využitím kamery ZED 2 není potřeba kalibrace. Manuální kalibrace je možná, avšak výrobcem nedoporučena [52]. Kamera je již zkalibrována z výroby a její parametry lze získat po oživení kamery využitím metody *getCameraCalibration*.

## 5.3 ALGORITMUS PRO KOMUNIKACI S KAMEROU

Úkolem algoritmu byla komunikace s kamerou ZED 2, pořizování záznamu vozovky a současné ukládání záznamu videa a jednotlivých snímků. Tento algoritmus navíc streamoval pořizované video na obrazovku PC. Tato funkce není při pořizování záznamu stěžejní, nicméně umožňovala vizuální kontrolu snímaného záznamu.

Pro algoritmus detekce jízdních pruhů je důležitá rychlost detekce, tzn. aby algoritmus detekoval oblast jízdního pruhu v reálném čase. Proto byl pořízen záznam videa o vzorkovací frekvenci 30 snímků za sekundu. Dosažení této hodnoty bylo pro prvotní návrh algoritmu problém. Jelikož dostupný hardware nestíhal ukládat záznam videa a jednotlivé snímky ve zvoleném rozlišení.

Pro optimalizaci algoritmu kamery bylo vyzkoušeno několik enkodérů. Pro formát .avi, konkrétně XVID, MJPG a H.264. Z pohledu propustnosti a výsledné vzorkovací frekvence dosáhl nejlepších výsledků enkodér MJPG. Při využití MJPG algoritmus zpracoval o 18 % více snímků než při využití enkodéru XVID. Podobný výsledek se dal předpokládat, jelikož u něj dochází k nejnižší míře komprimace ze všech jmenovaných. Z toho však vyplývá, že výsledné soubory také zabírají více místa na úložišti. Dalším problémem enkodéru MJPG je jeho náchylnost ke vzniku artefaktů, které mohou ovlivnit kvalitu snímků a výsledky algoritmu pro zpracování obrazu. Po otestování algoritmu pro snímání vozovky byl zvolen enkodér mp4 v kombinaci s formátem mp4. Tato kombinace dosáhla sice o 10 % nižší hodnoty zpracovaných snímků, ale pro zvolené rozlišení a vzorkovací frekvenci byly hodnoty dostatečné.

Další z možností, jak algoritmus zjednodušit a ušetřit výpočetní výkon, bylo odstranění zmíněné funkce streamování videa. Po hlubším zkoumání, s využitím modulu *cProfile*, však bylo zjištěno, že největší zátěž představuje ukládání jednotlivých snímků na disk. Aby tedy nedocházelo k zahlcení procesoru touto funkcí, bylo využito více vláken (multithreadingu), kdy byly funkce pro ukládání videa a snímků přiřazeny na samostatná vlákna a dále se již neovlivňovaly. Po této implementaci již byl algoritmus ukládat jak video, tak jednotlivé snímky při zachování zvolené vzorkovací frekvence 30 snímků za sekundu a rozlišení 1280x720 pixelů.

Výsledný skript se skládá z několika částí, importu knihoven, hlavní smyčky a funkcí pro ukládání videa a jednotlivých snímků. Každé funkci je přiřazena fronta pro ukládání dat na jednotlivých vláknech. Zvolené knihovny:

```
import cProfile
import pyzed.sl as sl
import cv2
import os
from datetime import datetime
from threading import Thread
import queue
```

V hlavní smyčce dochází nejprve k inicializaci a otevření kamery, nastavení parametrů kamery, inicializaci enkodéru a obou vláken. Poté následuje smyčka *while*, která běží, dokud není stisknuta klávesa q, čímž je ukončen celý program. Hlavním úkolem smyčky bylo zachycení snímku pravého objektivu kamery, načtení snímku a převedení do formátu BGR. To se děje za využití metod: *cam.grab(runtime), cam.retrieve_image(mat, sl.VIEW.RIGHT), cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)*.

Dále zajišťuje ukládání snímků do zmíněné fronty pomocí metody: *frame_queue.put(frame)* a zobrazení snímku na obrazovce, což má na starost metoda *cv2.imshow('ZED 2 Camera', frame)* Pokud je stisknuta klávesa „q”, smyčka je ukončena. Pokud během zachycení nastane chyba, program je ukončen a je vypsána chybová hláška. Celý skript je uložen v příloze práce.

# 6 UČENÍ NEURONOVÉ SÍTĚ PRO DETEKCI JÍZDNÍCH PRUHŮ

Jak je zmíněno v kap. 4, pro detekci jízdních pruhů byla zvolena metoda segmentace obrazu. Pro tento účel byla využita neuronová síť, konkrétně CNN s architekturou U-Net. Tato kapitola popisuje model sítě a kroky, které byly provedeny k získání funkčního modelu. Zejména se jednalo o výběr a přípravu datové sady a vytvoření skriptu pro trénink a ověření modelu. Zmíněny jsou také využité hyperparametry sítě a proces jejich optimalizace.

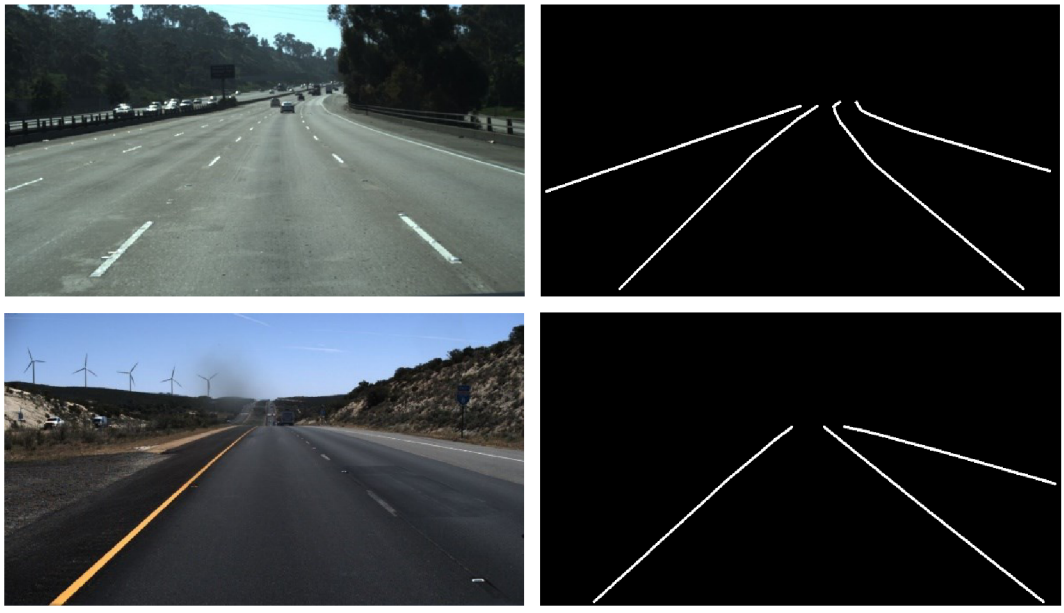
## 6.1 PŘÍPRAVA DATOVÉ SADY

Pro jakoukoliv úlohu využívající neuronové sítě je důležitý výběr správné datové sady (datasetu) využité při tréninku. Nesprávně zvolená či chybná data negativně ovlivňují výkon výsledného modelu.

Datová sada využita pro trénink v této práci se skládala ze dvou částí. První z nich byla známá datová sada TuSimple. Ta byla doplněna daty nasbíranými kamerou ZED 2.

### 6.1.1 TUSIMPLE DATASET

Tato datová sada obsahuje snímky pořízené zejména na dálnicích a rychlostních silnicích. Snímky jsou pořízeny za různých povětrnostních a světelných podmínek. Z datasetu bylo vybráno 3781 anotovaných snímků vozovky. Vybrána byla jen relevantní data a byla jim přiřazena odpovídající struktura tak, aby mohla být efektivně využita v trénovacím procesu. Rozlišení snímků je 1280x720 px. Ukázku vybraného snímku a jeho segmentační masky lze vidět na *Obr. 27*. Data byla zíkána na platformě Kaggle [53].

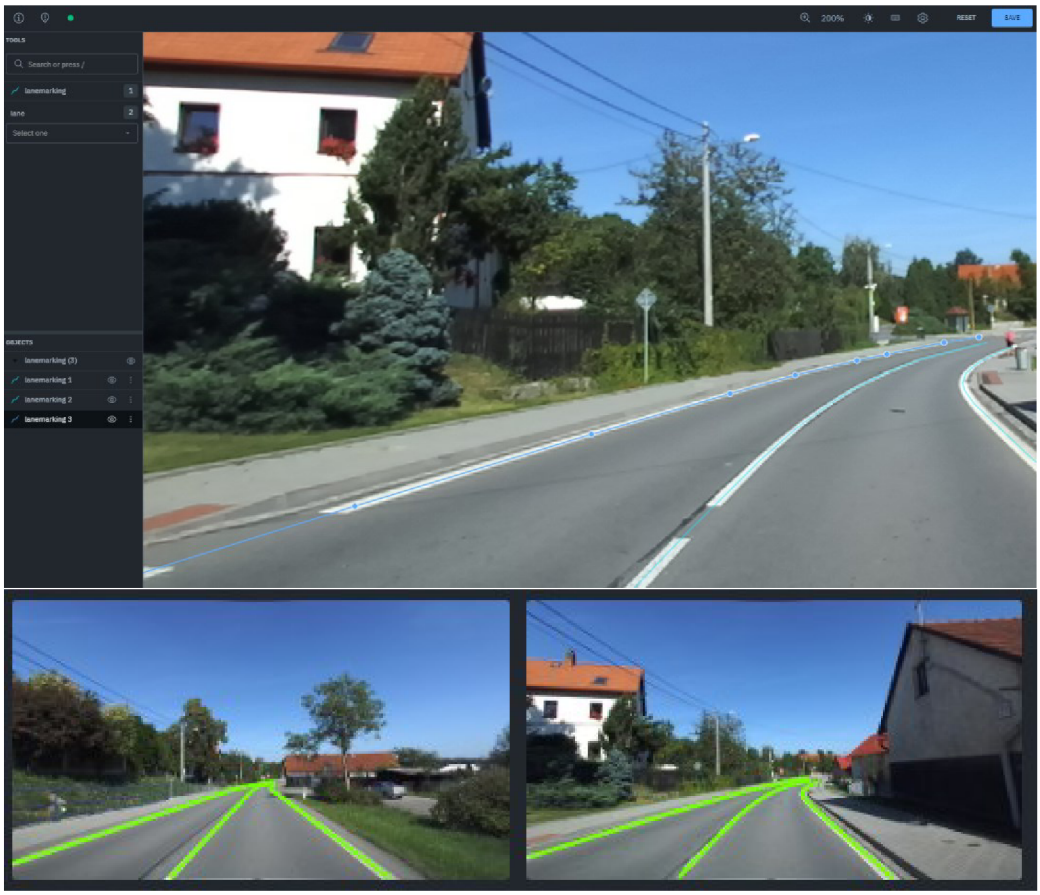


*Obr. 27* Ukázka snímku (vlevo) a segmentační masky (vpravo) TuSimple datasetu [53]

### 6.1.2 ROZŠÍŘENÍ DATOVÉ SADY NASBÍRANÝMI DATY

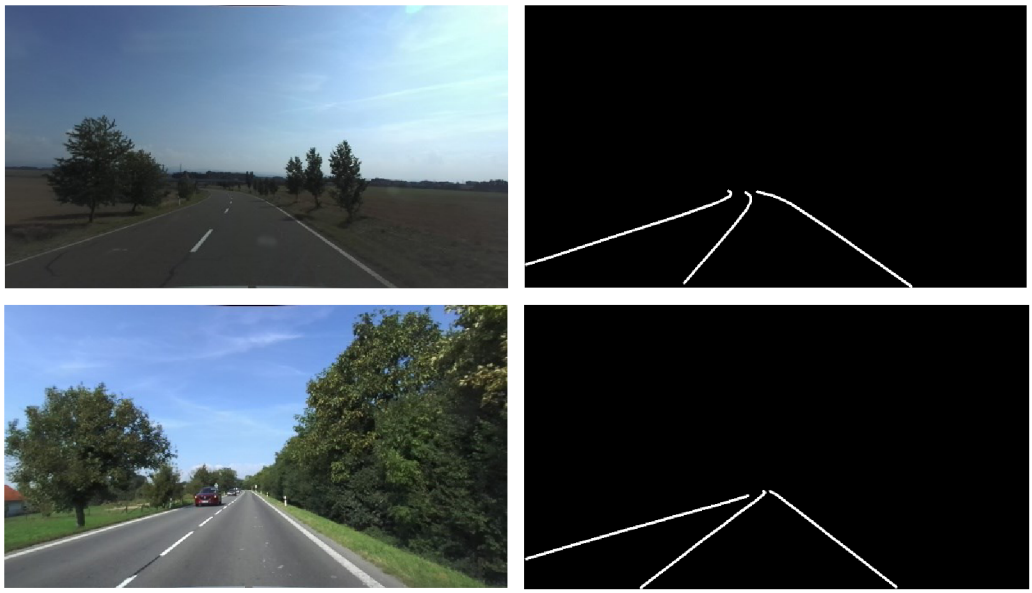
TuSimple dataset neobsahuje žádné snímky silnic nižších tříd ani příměstské komunikace. Byl proto doplněn daty nasbíranými na místních komunikacích. Aby mohla být využita data nasbíraná kamerou ZED 2, bylo nutné nejprve vytvořit segmentační masky jednotlivým snímkům určeným.

Z nasbíraných dat bylo tedy vybráno 590 snímků, které byly manuálně anotovány. K anotaci bylo využito prostředí Labelbox a příslušných anotačních nástrojů. Pro vytvoření labelů byla zvolena metoda „lanes and splines“, díky níž se podařilo dosáhnout požadované reprezentace hranic jízdních pruhů. Ukázku z prostředí Labelbox a vytvořené labely lze vidět na *Obr. 28*.



*Obr. 28* Ukázka vlastní anotace z prostředí Labelbox

Následně byly labely vyexportovány ve formátu .ndjson a za pomoci připraveného skriptu vytvořeny masky k jednotlivým snímkům. Ukázku snímku a k němu vytvořené masky můžete vidět na *Obr. 29*. Snahou bylo zachovat styl anotace datové sady TuSimple.



*Obr. 29* Ukázka vlastního snímku a vytvořené segmentační masky

### 6.1.3 ROZDĚLENÍ DATASETU

Doplněný dataset obsahoval 4372 jedinečných snímků byl následně rozdělen na tři části:

1. Tréninkový dataset
2. Validační dataset
3. Testovací dataset

Tomuto rozdělení se také říká „training split“ a udává, v jakém poměru jsou data pro dané části tréninkového procesu rozdělena. Pro trénink modelu byl zvolen poměr 80:10:10, kdy 80 % dat náleželo tréninkové sadě, 10 % validační a 10 % testovací sadě. Tréninková a validační data byla poté použita k tréninku sítě, zatímco testovací data sloužila ke konečnému vyhodnocení modelu.

### 6.1.4 AUGMENTACE DAT

Tréninková datová sada byla dále rozšířena pomocí technik augmentace obrazu, což jsou techniky umělého rozšíření datové sady. Pro augmentaci byly využity funkce knihovny Pytorch. Tento postup umožnil využití vyššího počtu dat pro trénink sítě. Augmentace tréninkových dat zlepšuje generalizaci výsledného modelu. Využity byly tyto dva druhy augmentace:

1. Rotace snímku o náhodný úhel v daném rozmezí
2. Horizontální převrácení snímku

## 6.2 Model neuronové sítě U-Net

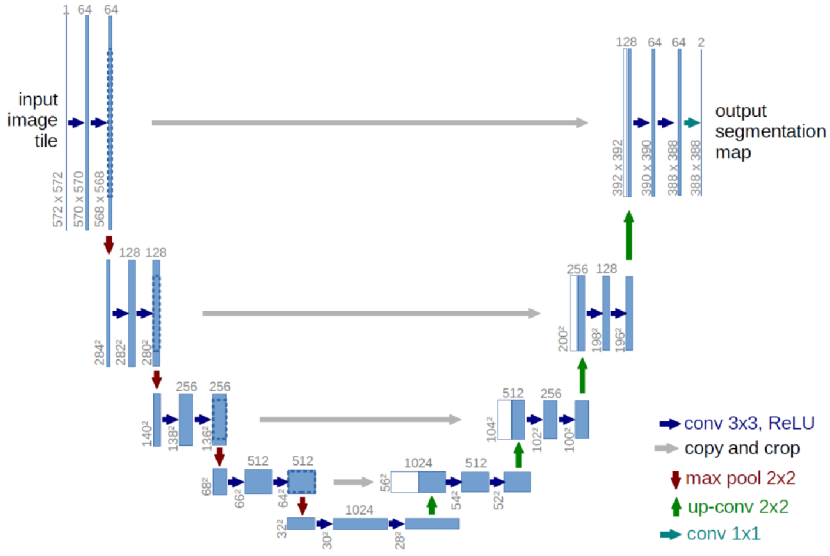
Použitý model U-Net se skládá ze tří základních komponent, z bloků, enkodéru a dekodéru (viz *Obr. 30*) Každá z nich hraje svou roli při zpracování obrazu a tvorbě segmentační masky.

### 6.2.1 Bloky

Základní stavební jednotkou modelu jsou bloky, které obsahují dvě konvoluční vrstvy s oknem (kernelem) o velikosti 3 x 3. Ty provádějí filtraci vstupních dat a extrakci příznaků z obrazu. Mezi nimi se nachází aktivační funkce ReLU.

### 6.2.2 Enkodér

V enkodéru dochází k postupné redukci vstupního obrazu pomocí sdružovacích vrstev a získání širšího kontextu. Tzn. umožňuje tvorbu souvislostí mezi jednotlivými pixely. Kernel pro sdružování má velikost 2 x 2. Počet kanálů se v enkodéru postupně zvyšuje ze 3 (vstupní obraz) na 64.




*Obr. 30* Ukázka architektury modelu U-Net [54]

### 6.2.3 Dekodér

Zde dochází k převzorkování mapy vlastností pomocí transponovaných konvolucí 2 x 2. Během nich se zvětší rozměry obrazu a počet kanálů se v každém bloku dekodéru sníží na polovinu. Mapa vlastností je rovněž spojena s mapou vytvořenou na stejné úrovni enkodéru. Při výstupu z dekodéru je aplikována bodová konvoluce, která zredukuje počet kanálů poslední vrstvy na počet tříd. Ten se při binární segmentaci rovná 1. Je zde také využita interpolace pro zachování vstupních rozměrů obrazu a funkce sigmoid. Výstupem modelu je potom segmentační mapa, pro binární segmentaci také nazývaná binární mapa obrazu.

## 6.3 TRÉNINK NEURONOVÉ SÍTĚ

Trénink modelu neuronové sítě probíhal v prostředí Google Collaboratory s využitím knihovny Pytorch a NumPy. Pro automatizaci procesu učení byly implementovány frameworky Optuna a Weights and Biases (W&B).

### 6.3.1 KÓD PRO TRÉNINK NEURONOVÉ SÍTĚ

Trénovací algoritmus byl rozdělen na několik skriptů dle funkce jednotlivých metod a tříd:

1. *config*
2. *dataset*
3. *model*
4. *train*

V této kapitole budou popsány pouze důležité části kódu, zejména popis funkce jednotlivých částí a smyček. Celý kód bude umístěn v příloze.

V konfiguračním souboru *config* byly definovány cesty k textovým souborům pro tvorbu datasetu, hodnoty globálních proměnných a defaultní hodnoty tréninkových parametrů.

#### DEFINICE DATASETU

Modul *dataset* definoval pomocnou funkci *read_txt_file*, pro čtení textových souborů s cestami k jednotlivým snímkům. Tato funkce bere jako vstup cestu k textovému souboru, iniciuje prázdný seznam, otevře soubor pro čtení a pomocí nekonečné smyčky *while* čte soubor po řádcích. Řádek rozdělí na základě obsažených mezer a přidá jej do vytvořeného seznamu. Po přečtení souboru vrátí vytvořený seznam (cest k souborům):

```
def read_txt_file(file_path):
    img_list = []
    with open(file_path, 'r') as file_to_read:
        while True:
            lines = file_to_read.readline()
            if not lines:
                break
            item = lines.strip().split()
            img_list.append(item)
    file_to_read.close()
    return img_list
```

Dále je zde definována třída *SegmentationDataset(Dataset)*. Ta je navržena pro práci s daty pro segmentaci obrazu, kdy je dataset rozdělen na vzorky obsahující jednotlivé páry obrázku a segmentační masky. Metody třídy jsou kompatibilní s knihovnou Pytorch.

Tato třída obsahuje konstruktor pro její inicializaci:

```
def __init__(self, file_path, transforms, num_samples=None):
        # Uložení cest k souborům s obrázky a maskami
```

```
        self.img_list = read_txt_file(file_path)
        # Omezení počtu vzorků, pokud je specifikován počet vzorků
        if num_samples is not None:
            self.img_list = random.sample(self.img_list, num_samples)
        # Odstranění obrázků nebo masek, které nelze načíst
        self.img_list = [img_path for img_path in self.img_list
                         if cv2.imread(img_path[0]) is not None
                         and cv2.imread(img_path[1], 0) is not None]
        # Uložení velikosti datasetu po filtraci
        self.dataset_size = len(self.img_list)
        # Uložení specifikovaných transformací
        self.transforms = transforms
```

Dále metodu pro získání počtu vzorků datasetu a metodu pro vrácení dvojici transformovaného obrázku a masky:

```
def __len__(self):
        return self.dataset_size

def __getitem__(self, idx):
        # Extrahování cest k souborům obrázku a masky pro daný index
        img_path_list = self.img_list[idx]
        image_path, mask_path = img_path_list
        # Načtení obrázku z disku, změna z BGR na RGB
        # a načtení přidružené masky z disku v režimu odstínu šedi
        image = cv2.imread(image_path)
        mask = cv2.imread(mask_path, 0)
        # Převod obrázku z BGR na RGB formát
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # Převod na PIL obrázky, aby mohly být transformovány
        image = Image.fromarray(image)
        mask = Image.fromarray(mask)
        # Kontrola, specifikací transformací a jejich aplikace
        if self.transforms is not None:
            image = self.transforms(image)
            mask = self.transforms(mask)
        # Vrácení n-tice obsahující obrázek a jeho masku
        return image, mask
```

### DEFINICE ARCHITEKTURY MODELU

Soubor *model* obsahuje architekturu modelu U-Net. Pro každou část je zde vytvořena samostatná třída. Třída *Block* definuje konvoluční vrstvy a ReLU vrstvy modelu. Dále jsou zde třídy *Encoder* a *Decoder*. Nakonec se zde nachází třída U-Net, která kombinuje jednotlivé části k vytvoření celkového modelu U-Net pro segmentaci obrazu. Architektura modelu je známá, došlo zde tedy k úpravě původního kódu [55] a jednotlivých tříd tak, aby byly kompatibilní s ostatními částmi trénovacího algoritmu.

### DEFINICE SMYČKY PRO TRÉNINK

Modul *train* obsahuje hlavní trénovací smyčku a jsou zde implementovány zmíněné frameworky. Po importu důležitých knihoven jsou inicializovány parametry W&B pro sledování tréninkového procesu. Poté jsou definovány operace pro předzpracování dat (transformace obrazu) a inicializována tréninková a validační datová sada.

Také je definovaná funkce *objective* (cílové funkce) frameworku Optuna, sloužící k optimalizaci hyperparametrů. V této funkci je zapouzdřena celá tréninková smyčka. Probíhá zde tedy inicializace modelu, optimalizace hyperparametrů, trénink modelu a současně jsou sledovány důležité metriky učení modelu.

V hlavní trénovací smyčce probíhá trénink na základě nastavených hyperparametrů. Ty jsou optimalizovány pomocí frameworku Optuny, kdy je frameworku zadán počet pokusů (*trials*), parametry pro optimalizaci a jejich rozsah.

```
# Nastavení parametrů pro optimalizaci pomocí Optuna
if trials is not None:
    params = {'INIT_LR': trials.suggest_categorical('INIT_LR',[0.001]),
              # Další parametry pro optimalizaci
    }
```

Optuna poté optimalizací zadaných hyperparametrů snižuje nebo zvyšuje návratovou hodnotu definované funkce *objective*. Průběh jednotlivých metrik je po epochách logován do W&B, kde lze zvolené metriky sledovat v reálném čase (viz
*Obr. 31.*) Skript importuje knihovny a třídy z ostatních modulů:

```
from dataset import SegmentationDataset, read_txt_file
from model import UNet
import config
from config import parse_arguments
```

Ve funkci *objective* je poté definován postup tréninku po krocích. Nejprve je iniciován model U-Net, který je přiřazen definovanému zařízení (pokud možno GPU):

```
unet = UNet().to(config.DEVICE)
```

Inicializace parametrů W&B:

```
wandb_kwargs = {"project": "Unet1-F1_1.2-TuSimple_Extended"}
wandbc = WeightsAndBiasesCallback(wandb_kwargs=wandb_kwargs,
as_multirun=True)
```

Definice transformací pro trénink a validaci. Zde jsou definovány transformace pro augmentaci vstupních snímků před vstupem do modelu:

```
train_transforms = v2.Compose([
```

```
    # Transformace pro trénink
])
transforms = v2.Compose([
    # Transformace pro validaci
])
```

Inicializace parseru pro zpracování argumentů. Defaultní hodnoty jsou uloženy v modulu *config*:

```
args = parse_arguments()
```

Vytvoření detesetu pro trénink a validaci:

```
# Vytvoření tréninkového a validačního datasetu
train_dataset = SegmentationDataset(file_path=config.TRAIN_PATH,
transforms=transforms, num_samples=args.NUM_SAMPLES_TRAIN)
validation_dataset = SegmentationDataset(file_path=config.VALIDATION_PATH,
transforms=transforms, num_samples=args.NUM_SAMPLES_TEST)
```

Inicializace optimizéru a scheduleru:

```
# Inicializace optimizeru a plánovače učícího poměru
optim = Adam(unet.parameters(), lr=args.INIT_LR)
scheduler = CosineAnnealingWarmRestarts(optim, T_0, T_mult=2, eta_min=1e-5)
```

Poté je zahájen trénink modelu. Pomocí smyčky *for* probíhají iterace přes jednotlivé epochy:

```
for e in tqdm(range(args.NUM_EPOCHS), unit="epoch"):
```

Po každé epoše je model ověřen pomocí validačních dat a jsou zaznamenány metriky učení:

```
with torch.no_grad():
    for (x, y) in validationLoader:
        # Kód pro validaci
```

Na konci tréninku jsou výsledky zalogovány pro další analýzu do W&B:

```
wandb.log({
    # Logování metrik učení
})
```

Po doběhnutí smyčky je model uložen na disk:

```
torch.save(unet, config.MODEL_PATH)
```

Optimalizace je spuštěna vytvořením objektu studie Optuna a následné zavolání funkce *study.optimize*:

```
study = optuna.create_study(direction='maximize')

study.optimize(objective, n_trials=1, callbacks=[wandbc])
```

### 6.3.2 SLEDOVANÉ METRIKY UČENÍ MODELU

Během procesu učení neuronové sítě byly sledovány následující kvalitativní metriky:

1. Trénovací a Validační ztráty (Train Loss a Test Loss)
2. Accuracy
3. Precision
4. Recall
5. F1 skóre

Metriky byly implementovány pomocí knihovny Pytorch. Zpočátku byl kladen důraz na co nejvyšší přesnost modelu a snížení hodnoty ztrátové funkce. To vedlo k rychlé konvergenci ztrátových hodnot a vysoké hodnotě přesnosti, avšak při kontrole ostatních metrik a vizuální kontrole predikce byla zjištěna vysoká míra falešně negativní predikce. Jelikož zastoupení jízdních pruhů (značení jízdních pruhů) je ve snímcích přibližně 2%, i při 100% negativní predikci (tj. černá obrazovka) byla přesnost predikce 98 %.

Z toho důvodu byla jako hlavní metrika využito hodnocení F1 skóre, které je nejvhodnější pro úlohy s nerovným zastoupením tříd. Tato metrika byla využita při optimalizaci modelu a nastavena jako cílová funkce frameworku Optuna. Toto nastavení omezilo výskyt falešně negativních predikcí, které jsou v tomto případě považovány za nejdůležitější. K objektivnějšímu hodnocení detekce byla sledována další kritéria, precision a recall, které zároveň slouží k výpočtu F1 skóre.

### 6.3.3 ZTRÁTOVÁ FUNKCE (LOSS FUNCTION)

Pro trénink neuronové sítě byla zvolena ztrátová funkce Binary Cross Entropy Loss with Logits, která je často využívaná u problémů binární segmentace. Kombinuje totiž výhody sigmoid aktivační funkce a logit transformace. Je tedy robustnější než samotná Binary Cross Entropy Loss (Binární křížová entropie).

### 6.3.4 VÁHY JEDNOTLIVÝCH TŘÍD (CLASS_WEIGHT) VE ZTRÁTOVÉ FUNKCI

Jak již bylo zmíněno, jedná se o úlohu s nerovným zastoupením tříd. Proto byl při tréninku nastaven parametr *class_weight* (váhy třídy) ve stejném poměru, jako je zastoupení pixelů třídy v obraze. Tzn. 0,02 pro *lane*=True a 1,02 pro *lane*=False. Pokud by tento parametr chyběl, model by nebral pruhy v obraze v potaz.

### 6.3.5 OPTIMALIZÁTOR (OPTIMIZER)

Při tréninku byly využity dva typy optimalizátoru Adam a SGD. Při ladění modelu bylo zjištěno, že lepších výsledků dosahuje optimalizátor Adam, který byl využit při tréninku výsledného modelu.

### 6.3.6 UČÍCÍ RYCHLOST (LEARNING RATE)

Ke změně učící rychlosti v průběhu tréninku byly využity dva typy scheduleru: StepLR s exponenciálně klesající učící rychlostí a CosineAnnealingWarmRestarts s klesajícím cosinovým průběhem a restartem hodnoty učící rychlosti po zvolené periodě. Při prvotní optimalizaci pomocí Optuna bylo zjištěno, že scheduler CosineAnnealingWarmRestarts dosahuje lepších výsledků a byl tedy využit při závěrečné fázi ladění hyperparametrů. Konečný model byl trénován s počáteční hodnotou learning rate 0,001.

### 6.3.7 UČÍCÍ DÁVKA (BATCH SIZE)

Pro trénink modelu byla zvolena učící dávka *batch_size* = 4. Tato hodnota opět dosahovala nejlepších výsledků při optimalizaci s Optuna. Během optimalizace sice probíhal trénink s hodnotou učící dávky 4, 8, 16, 128 a 256, hodnota F1 skóre však buďto nebyla dostatečně vysoká nebo docházelo k přeučení sítě či přetížení systému.

### 6.3.8 POČET EPOCH

Prvotní iterace tréninku byly provedeny při nižším počtu epoch tréninku pro ušetření tréninkového času. V závěrečné fázi ladění hyperparametrů a tréninku konečného modelu byla neuronová síť učena při 62 epochách.

### 6.3.9 PRAHOVÁNÍ (THRESHOLDING)

Během učení neuronové sítě proběhla také řada experimentů s různou hodnotou pro prahování. Práh (threshold) je hodnota, se kterou je porovnána hodnota pravděpodobnosti výskytu dané třídy. Má tedy přímý vliv na detekci. Standardně se při binární segmentaci používá hodnota *Threshold* = 0,5. Lze však využít i jiné hodnoty. Během tréninku byly využity hodnoty *Threshold* 0,5; 0,6 a 0,7. Hodnota *Threshold* = 0,7 byla zvolena z důvodu nedostatečných výsledků metriky precision při defaultní hodnotě.

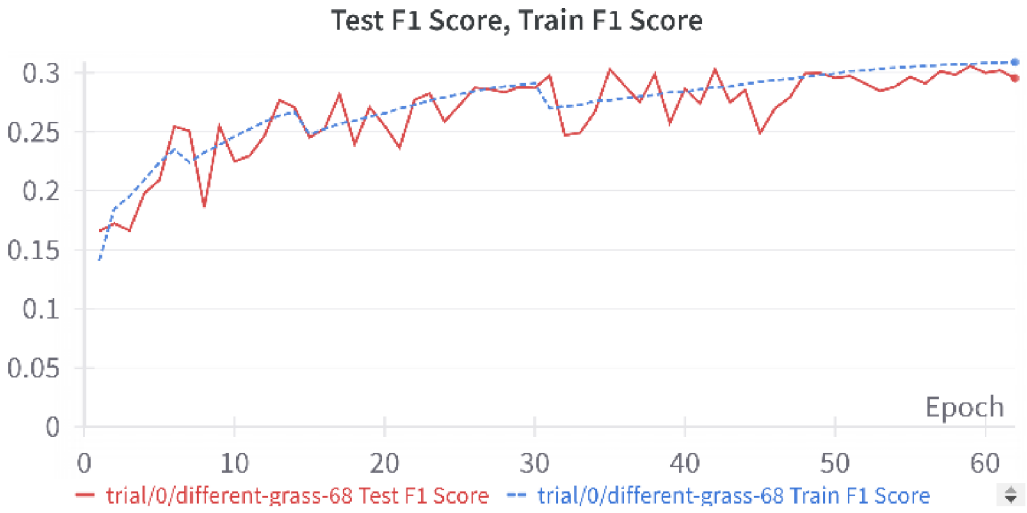
### 6.3.10 VYHODNOCENÍ TRÉNINKU A PARAMETRŮ NAUČENÉHO MODELU U-NET

Trénink modelu U-Net v prostředí Google Colab trval 3 hodiny a 45 minut. Model byl trénován na grafické kartě Tesla T4. V *Tab. 3* lze vidět konečné hodnoty sledovaných metrik modelu, počet epoch, učící dávku a počáteční hodnotu učící rychlosti. Tréninková sada obsahovala 3498 jedinečných snímků.


*Obr. 31* Průběh trénovací a validační ztráty během tréninku modelu U-Net v prostředí W&B

Na grafu (

*Obr. 31*) je zobrazen průběh trénovacích a validačních ztrát během tréninku, který byl zastaven po 62 epochách, kdy se začal narůstat rozdíl mezi trénovací a validační ztrátou, aby nedošlo k přeučení sítě. Při následném vyhodnocení sítě na testovací datové sadě byly potvrzeny výsledky dosažené na validační sadě. V grafu je také vidět body, kdy došlo k restartu hodnoty učící rychlosti.



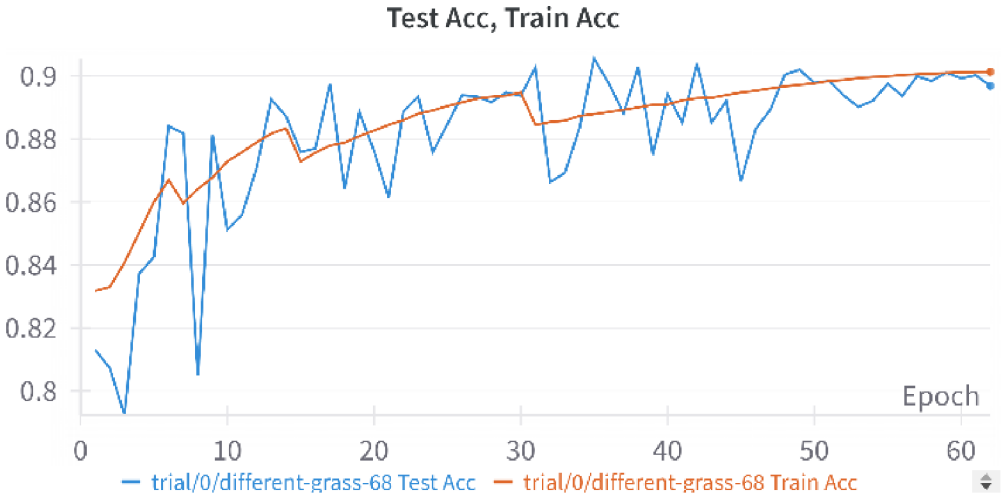

*Obr. 32* Průběh F1 skóre během tréninku modelu U-Net v prostředí W&B

Natrénovaný model dosahuje hodnoty F1 skóre 0,29 (*Obr. 32*), což není dobrá hodnota této metriky. Způsobovala to nízká hodnota metriky precision, model tedy vykazuje vysokou míru falešně pozitivní predikce. Tento problém bude při implementaci modelu řešen dodatečnou filtrací, případně časovým průměrováním predikce.

*Tab. 3* Konečné hodnoty metrik modelu U-Net

| BATCH | LR | SCHEDULER TYPE | Epoch | Acc | F1 Score | Test Loss | Precision | Recall |
|---|---|---|---|---|---|---|---|---|
| 4 | 0,001 | CosineAnnealingWarmRestarts | 62 | 0,9 | 0,29 | 0,63 | 0,18 | 0,88 |

Na *Obr. 33* je průběh hodnoty accuracy, která i přes nepříznivý výsledek hodnoty F1 skóre, dosáhla 90 %. Tento výsledek je dán především zmiňovaným nerovnoměrným zastoupením pozitivních a negativních pixelů v obraze.




*Obr. 33* Průběh hodnoty accuracy během tréninku modelu U-Net v prostředí W&B

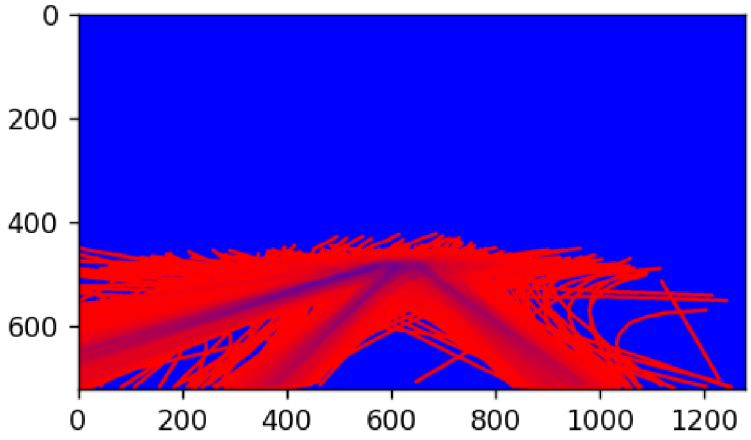
# 7 ALGORITMUS DETEKCE JÍZDNÍCH PRUHŮ

Natrénovaný model neuronové sítě byl implementován do algoritmu detekce. Zde má neuronová síť za úkol extrahovat hranice jízdních pruhů z obrazu snímaného kamerou. Plní tak percepční funkci řídicího řetězce. Algoritmus detekce tedy přijímá data snímaná kamerou nebo data nasbíraná kamerou uložená ve video formátu. Upraví jednotlivé snímky videa pro vstup do neuronové sítě, která z obrazu extrahuje hranice jízdních pruhů v obraze v podobě binární segmentační masky. S tou pracuje další část algoritmu, kde jsou odfiltrovány případné falešně pozitivní detekce a jsou vybrány hranice jízdního pruhu, v němž se vozidlo nachází. Tyto hranice, reprezentovány polynomem druhého stupně, jsou vykresleny na obrazovku.

Algoritmus detekce využívá moduly *lane_detection*, které obsahují funkce k získání binární segmentační masky a práci s ní. Dále je zde modul *projection*. Ten získává kontury z masky, dále je třídí, filtruje a provádí zmíněnou projekci na originální snímek. Také jsou zde dva moduly, které byly využity také v algoritmu pro trénink sítě, *model* a *config*. Ty umožňují využití modelu neuronové sítě. Pro lepší čitelnost kódu jsou některé metody uloženy v modulu *helper_functions*, odkud jsou poté volány jednotlivými moduly.

## 7.1 PREPROCESSING

Aby byla zajištěna správná funkčnost modelu, byly použity stejné vstupní parametry jako při tréninku sítě. Než může být snímek zpracován modelem, je proto upravena jeho velikost s využitím bikubické interpolace, obraz je převeden do RGB formátu a je normalizován. Je také aplikován ROI, při kterém je ořezána vrchní část obrazu neobsahující vozovku. Při pohledu na *Obr. 34* je vidět, že výskyt značení ve vrchní polovině obrazu je nepravděpodobný. Nedochází zde tedy ke ztrátě kontextu scény.




*Obr. 34* Vykreslení hranic jízdních pruhů z masek datové sady

## 7.2 POSTPROCESSING

Po segmentaci obrazu neuronovou sítí probíhá prahování hodnot predikce. Při získání masky můžeme v některých případech pozorovat značné množství šumu. K těmto situacím dochází například při průjezdu vozidla v protějším jízdním pruhu nebo výskytu stínů či nečistot

na povrchu vozovky. Z toho důvodu bylo využito časového okna, které je aplikováno ještě před samotným prahováním.

### 7.2.1 PRŮMĚROVÁNÍ PREDIKCE MODELU POMOCÍ ČASOVÉHO OKNA

Technikou FIFO byl vytvořen seznam pěti posledních predikcí, z jejichž hodnot je získán vážený průměr s exponenciálním poklesem vah dle rovnice:

$$Vážený\ průměr = \frac{\sum_{i=1}^{n}(w_i x_i)}{\sum_{i=1}^{n} w_i}, \tag{4}$$

kde n je počet hodnot (predikcí) v časovém okně, $x_i$ je i-tá hodnota predikce a $w_i$ je váha přiřazená i-té hodnotě.

Váha predikce aktuálního snímku $w_1 = 1$. Váhy dalších snímků jsou vypočteny následovně:

$$w_i = r\, w_{i-1} \text{ pro } i = 2, 3, \dots, n, \tag{5}$$

kde byla zvolena míra poklesu $r = 0{,}7$.

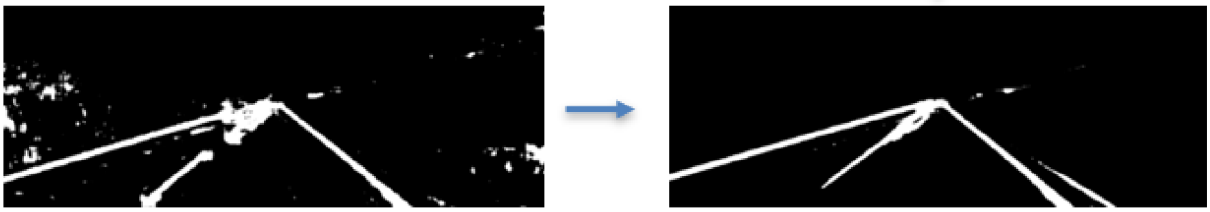
Váhy jsou následně normalizovány:

$$w_i' = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{6}$$

Vážený průměr je poté vypočten rovnicí:

$$Vážený\ průměr = \sum_{i=1}^{n} (w_i' x_i) \tag{7}$$

Výstupem je tedy binární segmentační maska vážená tímto průměrem. Jak můžeme vidět na *Obr. 35*, tímto způsobem byla odstraněna velká část falešně pozitivních predikcí. Aplikací časového okna však do predikce vnášíme zkreslení, jejíž míra se odvíjí od velikosti časového okna, tj. počtu predikcí vstupujících do výpočtu váženého průměru.



*Obr. 35* Aplikace časového okna na binární masku

Navržená metoda *temporal_avg* vypadá následovně:

```
def temporal_avg(self):
    if len(self.previous_predictions) > self.temporal_window:
        self.previous_predictions.pop(0)
    num_predictions = len(self.previous_predictions)
```

```
    if num_predictions <= 1:
        return None
    # Decay = series of products
    decay_rate = 0.7
    weights = [1.0]
    for i in range(1, num_predictions):
        weights.append(weights[-1] * decay_rate)

    total_weight = sum(weights)
    normalized_weights = [j / total_weight for j in weights]
    weighted_predictions = [p * j for p, j in
zip(self.previous_predictions, normalized_weights)]
    avg_prediction = sum(weighted_predictions)
    return avg_prediction
```

### 7.2.2 FILTRY A MORFOLOGICKÉ OPERACE

Na *Obr. 35* je také vidět jeden z problémů, který nastával při predikci. V blízkosti horizontu, kde se sbíhají hranice pruhů, se pruhy spojí do jedné kontury a jednotlivé hranice již není možné proložit polynomem. Na získanou masku jsou proto dále aplikovány mediánový filtr a morfologické operace pro zajištění přesné reprezentace hranic jízdních pruhů. Tyto operace samotné nedokázaly problém odstranit, maska je tudíž ořezána těsně pod místem, kde dochází ke spojení. Problémové situace poté řeší metody pro filtrování kontur.

```
def post_process_prediction(self, prediction, avg_prediction,
threshold=config.THRESHOLD):
    # Aplikace prahování
    if avg_prediction is not None:
        binary_mask = (avg_prediction > threshold) * 255
    else:
        binary_mask = (prediction > threshold) * 255
    binary_mask = binary_mask.astype(np.uint8)
    # Definice velikosti filtru dilatace a eroze
    kernel_ero = np.ones((5, 5), np.uint8)
    kernel_dil = np.ones((4, 4), np.uint8)
    # Ořezání masky
    binary_mask = binary_mask[280:, :]
    # Aplikace eroze, mediánového filtru a dilatace
    binary_mask = cv2.erode(binary_mask, kernel_ero, iterations=1)
    binary_mask = cv2.medianBlur(binary_mask, 5)
    binary_mask = cv2.dilate(binary_mask, kernel_dil, iterations=1)

    return binary_mask
```

## 7.3 KLASIFIKACE HRANIC A OŠETŘENÍ HRANIČNÍCH PŘÍPADŮ

Po získání zpracované masky bylo potřeba pracovat s konturami jednotlivých hranic. Jeden z problémů, který zde nastával, byl už zmíněný případ, kdy pixely kontury nelze proložit křivkou polynomu. Dalším případem, který může nastat, je situace, kdy v části obrazu není detekována žádná hranice. Oba tyto případy bylo nutné ošetřit pro větší robustnost algoritmu.

Navržené metody filtrace a ošetření případů mají pozitivní dopad na celkový výkon algoritmu. Vyhodnocením jejich implementace se blíže věnuje kapitola 10, kde lze vidět přínos jednotlivých metod pro detekci.

### 7.3.1 EXTRAKCE KONTUR Z BINÁRNÍ SEGMENTAČNÍ MASKY

Nejprve byly extrahovány kontury nacházející se v masce pomocí funkce *cv2.findContours*. Byl zde definován parametr pro počet extrahovaných kontur kvůli výpočtové náročnosti operace. Po extrakci z masky byly s využitím parametru *correction_shift* upraveny hodnoty y tak, aby odpovídaly rozměrům snímku z kamery. Metoda extrakce byla navržena s touto logikou:

```
def extract_lane_markings(self, num_contours):
    resized_mask = self.binary_mask
    # Extrahuj kontury v obraze
    contours, _ = cv2.findContours(resized_mask, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)
    # Seřaď kontury
    contours = sorted(contours, key=cv2.contourArea, reverse=True)
    # Vyber definovaný počet největších kontur
    self.largest_contours = contours[:num_contours]
    # Uprav souřadnice y-souřadnice bodů všech kontur
    for contour in self.largest_contours:
        for point in contour:
            point[0][1] += self.correction_shift
```

### 7.3.2 SEGMENTACE KONTUR

První problém byl vyřešen rozdělením kontur na menší segmenty, u těch byla určena poloha těžiště, která byla následně proložena polynomem. Tato logika byla implementována do metody *split_contours*. Metoda pomocí třech cyklů *for* rozděluje body kontur na daný počet segmentů definované výšky. Jednotlivé segmenty jsou následně vráceny v seznamu určenému k dalšímu zpracování:

```
def split_contours(self):
    num_segments = 80
    segment_height = self.frame.shape[0] // num_segments
    contours = self.largest_contours.copy()
    contours_segments = []

    for contour in contours:
        contour_segments = []

        for i in range(num_segments):
            start_y = i * segment_height
            end_y = min(start_y + segment_height, self.frame.shape[0])
            segment_points = []

            for point in contour:
                if start_y <= point[0][1] < end_y:
                    segment_points.append(point)

            contour_segment = np.array(segment_points)

            if np.any(contour_segment):
                contour_segments.append(contour_segment)
        contours_segments.append(contour_segments)
    return contours_segments
```

### 7.3.3 FILTRACE HRANIC

Zmíněné body bylo potřeba dále vyfiltrovat a rozdělit podle toho, jakým hranicím náleží. K tomuto účelu byl naprogramován třífázový proces skládající se z těchto částí:

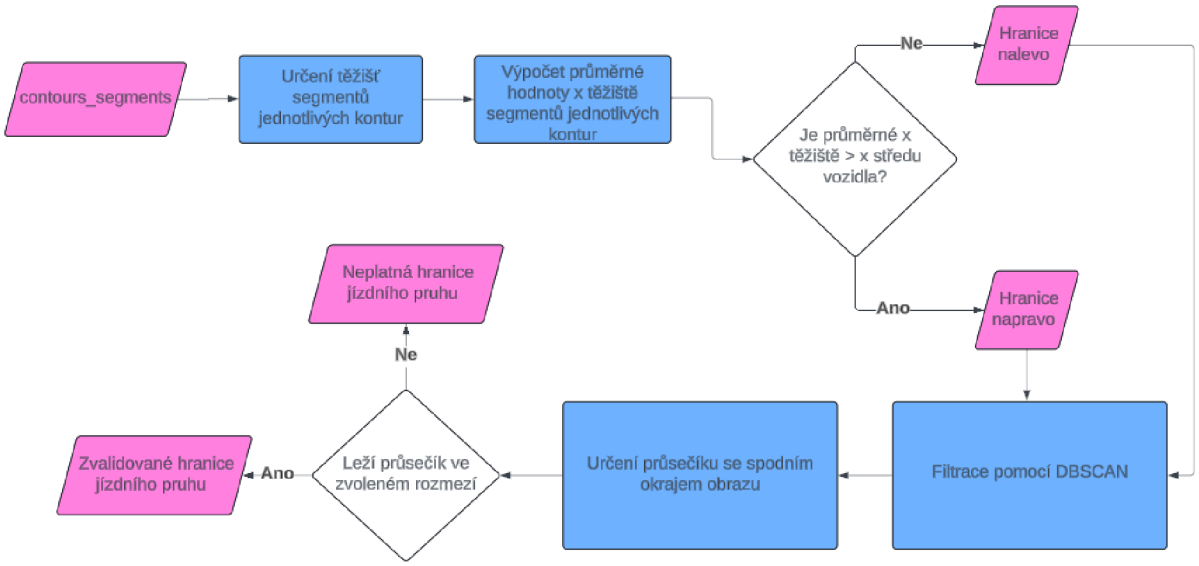
1. Analýza těžišť na základě polohy na ose x obrazu
2. Vytvoření klastrů pomocí DBSCAN
3. Validace polohy na základě orientace hranic

Analýzou polohy těžišť jsou polohy hranice klasifikovány na pravé a levé na základě průměrné hranice těžišť jejich kontur na ose x. Pokud je průměrná poloha těžišť v levé části obrazu, je kontura klasifikována jako levá hranice, a pokud je v pravé části, je klasifikována jako pravá hranice.

Následně je na těžiště aplikován DBSCAN, který identifikuje shluky (klastry) bodů těžišť v obraze na základě jejich hustoty a umožní vyřadit ty kontury, které nejsou reprezentovány dostatečným počtem segmentů nebo jsou příliš vzdáleny od ostatních.

V poslední fázi je vypočten průsečík přímek procházející jednotlivými konturami a spodním okrajem obrazu a jsou vybrány pouze kontury se správnou orientací. Tzn. u hranic, které leží nalevo od vozidla, se předpokládá, že jejich průsečík leží v levé části obrazu a obdobně pro pravé hranice v pravé části obrazu.

Logiku této metody lze pro názornost vidět na diagramu *Obr. 36*. Kód metody je dostupný v příloze.

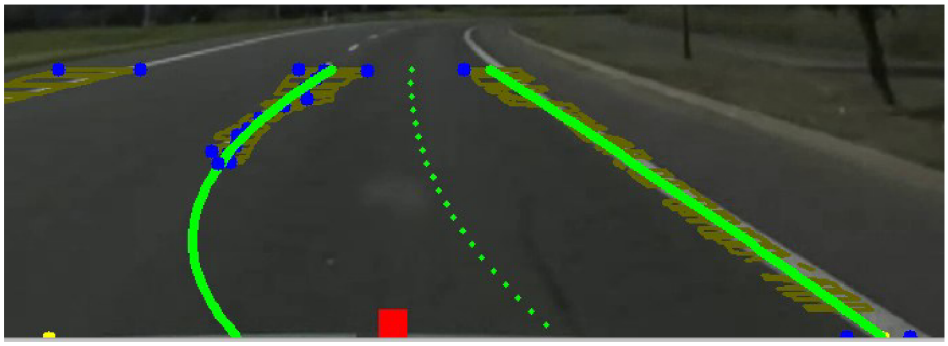



*Obr. 36* Blokový diagram filtrace hranic

### 7.3.4 OŠETŘENÍ FALEŠNĚ NEGATIVNÍ DETEKCE

Průsečíky poté hrají důležitou roli při falešně negativní detekci. V případech, kdy algoritmus není schopen detekovat celé hranice jízdního pruhu, docházelo často k tomu, že následná reprezentace hranice polynomem byla zkreslená. Tento případ je znázorněn na *Obr. 37*, kde je

hranice značena přerušovanou čarou. Ve spodní hranici zde není detekován žádný pruh, z obrazu je však zřejmé, že zmíněný průsečík (žlutý bod v obraze) může být za hranici považován. Tyto případy jsou proto ošetřeny zahrnutím průsečíku do funkce pro proložení křivkou.



*Obr. 37* Ukázka případu nepřesné reprezentace hranice jízdního pruhu polynomiální křivkou

K tomuto účelu byla implementována následující metoda:

```
def check_detection(self):
    if (not self.left_intersection_points or not
    self.right_intersection_points):
        return
    # Definice rozsahu y-hodnot pro spodní část obrázku
    y_max = self.frame.shape[0]
    y_min = y_max - 80
    # Kontrola, zda jsou v body left_lane a right_lane určeném rozsahu
    left_bottom = any(y_min <= point[1] <= y_max for
                      point in self.left_filtered_centers)
    right_bottom = any(y_min <= point[1] <= y_max for
                       point in self.right_filtered_centers)
    # Přidání bodu do seznamu průsečíků, pokud není detekován žádný střed
    if not left_bottom:
        add_bottom_point(self.left_intersection_points,
                         self.left_filtered_centers, y_max)
    if not right_bottom:
        add_bottom_point(self.right_intersection_points,
                         self.right_filtered_centers, y_max)
```

Metoda *check_detection* zkontroluje souřadnice y bodů, kterými má být proložena křivka. Pokud se ve spodní části obrazu (blízkosti vozidla) nenachází žádný bod, zavolá navrženou pomocnou funkci, která do seznamu bodů přidá bod určený z průsečíků kontur reprezentujících hranici se spodním okrajem obrazu podle této logiky:

```
def add_bottom_point(intersection_points, lane_centers, y_max):
    # Vypočítá průměrnou x-ovou hodnotu průsečíků
    points_array = np.array(intersection_points)
    x_values = points_array[:, 0]
    avg_x = np.mean(x_values)
    bottom_point = [avg_x, y_max]
    # Přidá bod do seznamu středů pruhů pro aproximaci polynomem
    lane_centers.append(bottom_point)
    return
```

### 7.3.5 VYKRESLENÍ HRANIC JÍZDNÍHO PRUHU

#### PROLOŽENÍ HRANIC KŘIVKOU

Filtrované hranice v podobě dedikovaných seznamů bodů pro levou a pravou stranu jízdního pruhu vstupují do metody *fit_polynomial_curve*, ta zajistí výpočet koeficientů polynomu druhého řádu pro aproximaci křivkou. Metoda následně vrátí koeficienty pro levou a pravou stranu.

#### STABILIZACE

Pro stabilizaci reprezentace hranic jízdního pruhu byla navržena metoda *avg_polynomials*, která s využitím FIFO vypočítá průměrné koeficienty polynomu, počet průměrovaných předpovědí je dán hodnotou parametru *window*. Při zahrnutí této logiky se předpokládá, že hranice jízdního pruhu jsou kontinuální čáry, jejichž geometrie se skokově nemění. Tato úvaha vychází i z faktu, že k návrhu geometrie vozovky se využívá klotoidních přechodnic mezi různě zakřivenými úseky. Přesto nemůže být velikost zásobníku příliš velká, aby nedocházelo k zásadnímu zkreslení.

```
def avg_polynomials(self, window=3):
    global avg_coefficients
    # Vyhlazení detekce s časovým průměrováním
    for j, prev in enumerate(curves_for_avg):
        if len(prev) <= 1:
            return None
        if len(prev) > window:
            prev.pop(0)
        # Výpočet průměrné hodnoty koeficientů
        avg_coeffs = np.mean(prev, axis=0)
        avg_coefficients[j] = avg_coeffs
    return avg_coefficients
```

#### VYKRESLENÍ NA ORIGINÁLNÍ SNÍMEK

K vykreslení hranic na originální snímek byla navržena metoda *project_lane_marking*. Ta bere průměrované koeficienty a pomocí funkcí knihovny OpenCV a cyklu *for* vykresluje jednotlivé křivky hranic jízdního pruhu do originálního snímku:

```
def project_lane_marking(self):
    for i, curve_coefficients in enumerate(avg_coefficients):
        # Generování hodnot y
        y_curve = np.linspace(start=(self.frame.shape[0] - 1),
                              stop=self.correction_shift)  # Korekce posunu

        # Výpočet hodnot x s polynomiálními koeficienty
        x_curve = np.polyval(curve_coefficients, y_curve)
        # Převod x, y na celá čísla
        curve_points = np.column_stack((x_curve.astype(int),
                                        y_curve.astype(int)))
        # Vykreslení křivky na snímek
        cv2.polylines(self.frame, [curve_points], isClosed=False,
                      color=(0, 255, 0), thickness=5)
```

# 8 URČOVÁNÍ TRAJEKTORIE VOZIDLA

Modul pro detekci jízdních pruhů, který detekuje hranice jízdního pruhu, tedy v řídicím řetězci zastává funkci percepce (kap. 3.1). Získané hranice můžeme dále využít pro identifikaci optimální trajektorie vozidla. Je potřeba si uvědomit, že v tomto případě je funkce percepce značně zjednodušená, protože nebere v úvahu polohu chodců, ostatních vozidel a jiných překážek na vozovce. Tomu odpovídá také návrh optimální trajektorie vozidla, za který je v tomto případě považován střed jízdního pruhu, kde se v daném čase vozidlo nachází.

Na základě detekovaných hranic jízdního pruhu byla pomocí samostatného modulu určena ideální trajektorie vozu v podobě centrální čáry mezi těmito hranicemi. Tato trajektorie byla následně vykreslena na obrazovku. Výstupem modulu byla také informace o poloze ideální trajektorie pro zpracování v další části řídicího řetězce.

## 8.1 ALGORITMUS PRO URČOVÁNÍ TRAJEKTORIE

Pro určování trajektorie byl navržen modul *trajectory_estimator* a v něm třída *TrajectoryEstimator*. Využívá koeficienty polynomu pravé a levé hranice jízdního pruhu, ve kterém se vozidlo nachází a, určí ideální trajektorii vedoucí středem pruhu. Ta je následně vyobrazena na daném snímku.

Program tedy načte koeficienty obou hranic a vypočte středové body mezi nimi. Tyto body jsou následně proloženy polynomem druhého stupně, který reprezentuje ideální trajektorii vozidla. Trajektorie je na snímku vyobrazena pomocí pravidelně rozmístěných bodů v obraze.

### 8.1.1 POPIS METOD TŘÍDY TRAJECTORYESTIMATOR

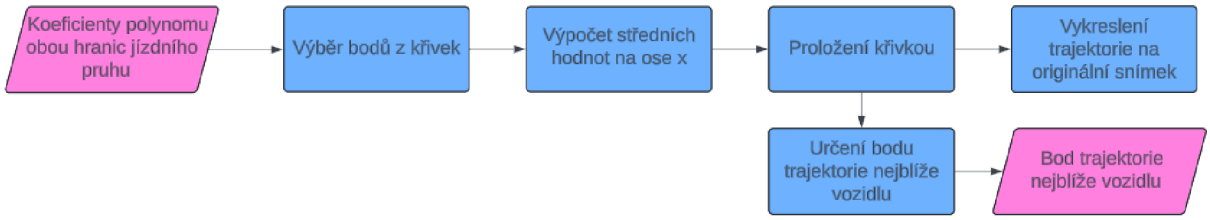
#### KONSTRUKTOR

Instance třídy je inicializovaná konstruktorem s atributy *left_coeffs* a *right_coeffs* v podobě seznamu koeficientů jednotlivých hranic.

#### METODY TŘÍDY

Výpočet trajektorie, její vykreslení a určení bodu nejblíže vozidlu poté řeší metoda *calculate_trajectory*, která výpočet provede s aktuálními hranicemi jízdního pruhu. Její implementaci znázorňuje diagram na *Obr. 38*. Hranice pruhu aktualizuje metoda *update_boundaries*:

```
def update_boundaries(self, left_coeffs, right_coeffs):
    self.left_line_coeffs = left_coeffs
    self.right_line_coeffs = right_coeffs
```




*Obr. 38* Diagram metody pro určení trajektorie

Poslední metodou třídy je *get_bottommost_trajectory_point*, která zkontroluje, zda je instanci třídy přiřazen bod trajektorie nejblíže vozidlu *bottommost_trajectory_point*. Tento bod také představuje návratovou hodnotu této metody.

```
def get_bottommost_trajectory_point(self):
    if self.bottommost_trajectory_point:
        return self.bottommost_trajectory_point
```

# 9 URČOVÁNÍ POLOHY VOZIDLA

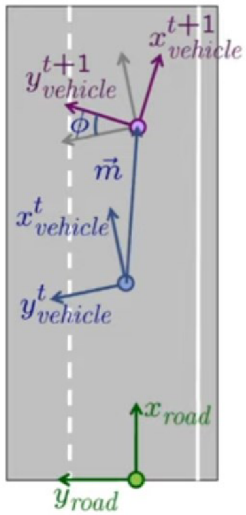
Posledním cílem práce bylo určit polohu vozidla v jízdním pruhu. Přesněji se jednalo u určení jeho relativní polohy vůči určené ideální trajektorii. Tento výstup je stěžejní informací pro regulátor či řídicí algoritmus vozidla, který určuje hodnoty pro aktuátory vozu tak, aby se přiblížilo ideální trajektorii.

### 9.1.1 PŘÍSTUPY VÝPOČTU POLOHY VOZIDLA

Pro určování polohy byly zvažovány následující přístupy:

1. Přírůstková lokalizace vozidla v jízdním pruhu
2. Přímý odečet vzdáleností s využitím IPM
3. Trigonometrický přístup

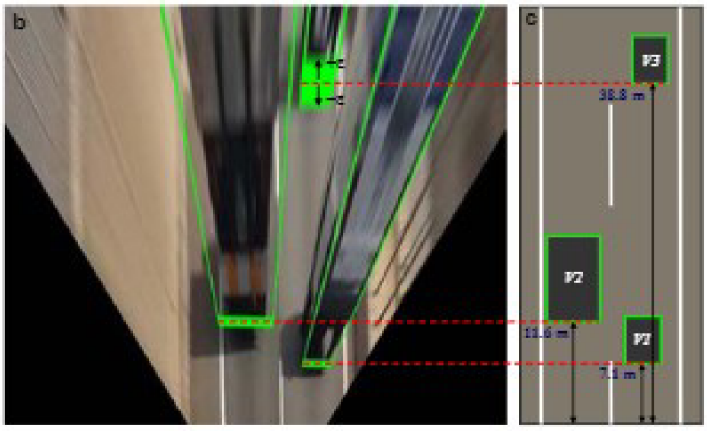
Prvním přístupem bylo využití metody přírůstkové lokalizace vozidla v jízdním pruhu, kdy by detekované hranice jízdního pruhu sloužily k vytvoření modelu jízdního pruhu. Tato metoda vyžaduje metodu IPM a rovněž sledování jízdního pruhu v čase. Model jízdního pruhu musí být zvlášť vytvořen pro jízdu po rovině a jízdu zatáčkou a zapojení Kalmanova filtru pro řešení nelineárních rovnic, jelikož model pro jízdu zatáčkou nelze popsat lineárními rovnicemi. Tato metoda je však spíše vhodná pro odhad budoucí polohy vozidla a směru pohybu, viz např. *Obr. 39*.




Slide Credits: Martin Lauer

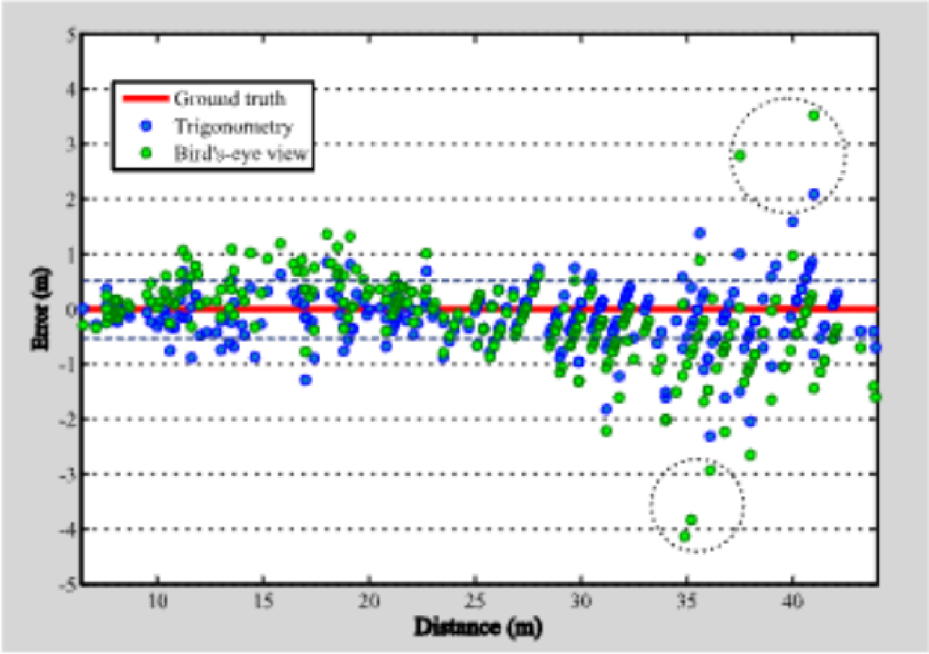
*Obr. 39* Ukázka modelu rovného jízdního pruhu přírůstkové lokalizace vozidla [56]

Další metodou bylo využití metody IPM pro převod obrazu s detekovanými hranicemi a navrženou trajektorií do ptačí perspektivy k přímému odečtu vzdáleností, jak ukazuje *Obr. 40*. Tato metoda se zdá na první pohled přímočará, avšak taková transformace obrazu je výpočtově náročná. Jak navíc ukazuje [57], u techniky výpočtu za pomoci IPM, roste míra chybovosti s rostoucí vzdáleností. A to kvůli značnému zkreslení obrazu během transformace.




*Obr. 40* Odečet vzdáleností po transformaci obrazu do ptačí perspektivy [57]

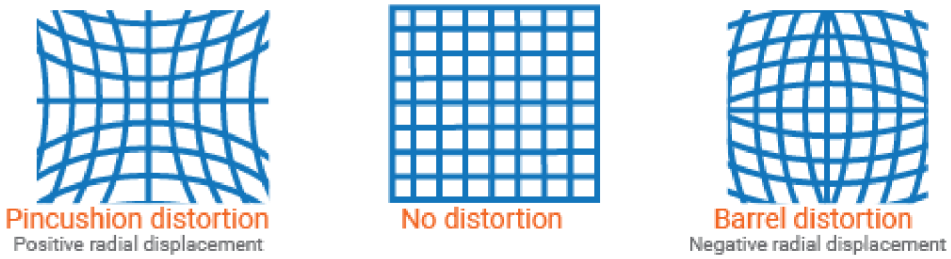
Na grafu v *Obr. 41* lze vidět porovnání chyby při výpočtu vzdálenosti za pomoci IPM a třetího uvažovaného přístupu (přepočet vzdáleností pomocí trigonometrie). Je zřejmé, že se zvyšující se vzdáleností roste chyba u obou metod. Nicméně i pro výpočet relativní polohy vozidla se zdá výhodnější využití trigonometrie. Nejen že je metoda přesnější pro menší vzdálenosti, ale výpočet je více přímočarý a výpočtově méně náročný. Tj. vyhneme se transformaci dat ptačí perspektivy.




*Obr. 41* Porovnání chyby při výpočtu vzdálenosti pomocí trigonometrie a metody IPM [57]

IPM je tedy výhodnější pro případy, kdy dochází ke sledování jízdního pruhu v průběhu času a může zde dojít ke korekci chyb vzniklých při transformaci. Protože zde jde pouze o výpočet relativní polohy vozidla vůči středu vozovky v daném čase, není v tomto případě IPM potřeba. Výpočet tedy mohl být zjednodušen využitím trigonometrických metod a euklidovské vzdálenosti bodů v obraze.

Využitím trigonometrie se předpokládá, že snímky prošly rektifikací obrazu, aby se do výpočtu nevnášela chyba zkreslení zkosením, které lze vidět např. na *Obr. 42*. Tento předpoklad byl splněn, jelikož kamera ZED 2 poskytuje již zpracovaná obrazová data a program tedy pracoval se snímky v přesném měřítku.




*Obr. 42* Ukázka porovnání radiálně zkreslené mřížky obrazu a mřížky bez zkreslení [58]

Dalším předpokladem pro správný výpočet bylo umístění kamery ve středu vozidla, tzn. kamera byla upevněna v rovině *xz* vozidla. Při analýze snímku se tedy střed vozidla nachází přesně uprostřed osy *x* obrazu. Poloha tohoto bodu je poté porovnávána s polohou bodu *C*, nejbližší snímaný bod trajektorie vozu.

### 9.1.2 VÝPOČET RELATIVNÍ VZDÁLENOSTI VOZIDLA OD IDEÁLNÍ TRAJEKTORIE

Do výpočtu vstupovaly následující proměnné: šířka snímku *Bi* [px], ohnisková vzdálenost kamery *f* [px], vzdálenost vozidla od trajektorie na ose x snímku $B_p$ [px], vertikální zorné pole kamery $\alpha$ [°], úhel sklonu kamery vůči ose *z* vozovky $\theta_c$ [°], umístění kamery nad povrchem vozovky *H* [m]. Schéma pro výpočet je vyobrazeno na *Obr. 43*.

Spodní okraj obrazu, a tedy i nejbližší snímaný bod vozovky je snímán pod úhlem $\frac{\alpha}{2}$ vzhledem k ose z kamery. Vzdálenost $l_c$ můžeme tedy vypočítat pomocí rovnice:

$$l_c = \frac{H}{\cos\left(\theta_c - \frac{\alpha}{2}\right)} \tag{8}$$

Dále je za pomoci $B_i$ a $B_{Cp}$ a *f* vyjádřeno $tan(\beta_C)$, kdy $\beta_C$ je úhel, který svírá rovina *yz* kamery se spojnicí počátku souřadného systému kamery s bodem *C*. $tan(\beta_C)$ lze vyjádřit také jako poměr $B_C$ (vzdálenost bodu od osy kamery v metrech) a $l_c$. Tato vyjádření popisují rovnice:

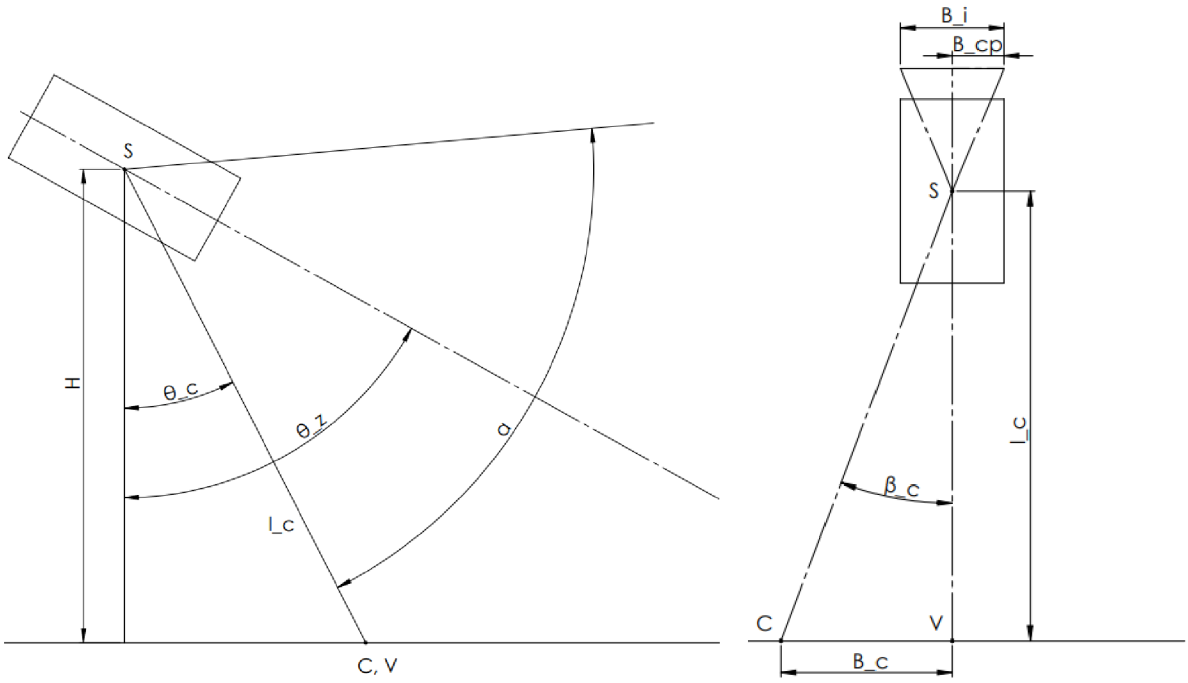
$$\tan\beta_C = \frac{\frac{B_i}{2} - B_{Cp}}{f} \tag{9}$$

$$\tan\beta_L = \frac{B_C}{l_c} \tag{10}$$

Z těchto rovnic můžeme následně vyjádřit $B_C$, což je vzdálenost středu kamery, respektive vozidla od středu vozovky:

$$B_C = l_c \cdot \tan \beta_L \tag{11}$$

Obdobným způsobem lze následně vyjádřit také vzdálenost od hranic jízdního pruhu.




*Obr. 43* Schéma výpočtu relativní polohy kamery (vozidla) vůči navržené ideální trajektorii

## 9.2 ALGORITMUS PRO URČOVÁNÍ POLOHY VOZIDLA

Modul *vehicle_position* a třída *VehiclePositionEstimator* slouží k určení polohy vozidla vzhledem k navržené trajektorii. Do metod třídy byl implementován výpočet vzdálenosti vozidla od trajektorie popsaný výše.

### 9.2.1 POPIS METOD TŘÍDY VEHICLEPOSITIONESTIMATOR

#### KONSTRUKTOR

Konstruktor inicializuje instanci třídy s následujícími atributy:

1. *focal_length* = ohnisková vzdálenost kamery
2. *image_width* = šířka obrazu kamery
3. *pitch_angle* = úhel náklonu kamery v radiánech
4. *camera_height* = výška kamery nad zemí
5. *V_FoV* = vertikální zorné pole kamery v radiánech

### METODY TŘÍDY

Metoda *get_relative_position* počítá vzdálenost polohy vozidla na snímku vůči poloze trajektorie, pro kterou uvažuje bod trajektorie nejblíže vozidlu (*bottommost_trajectory_point*). Metoda poté vrací tuto vzdálenost. Implementace vypadá následovně:

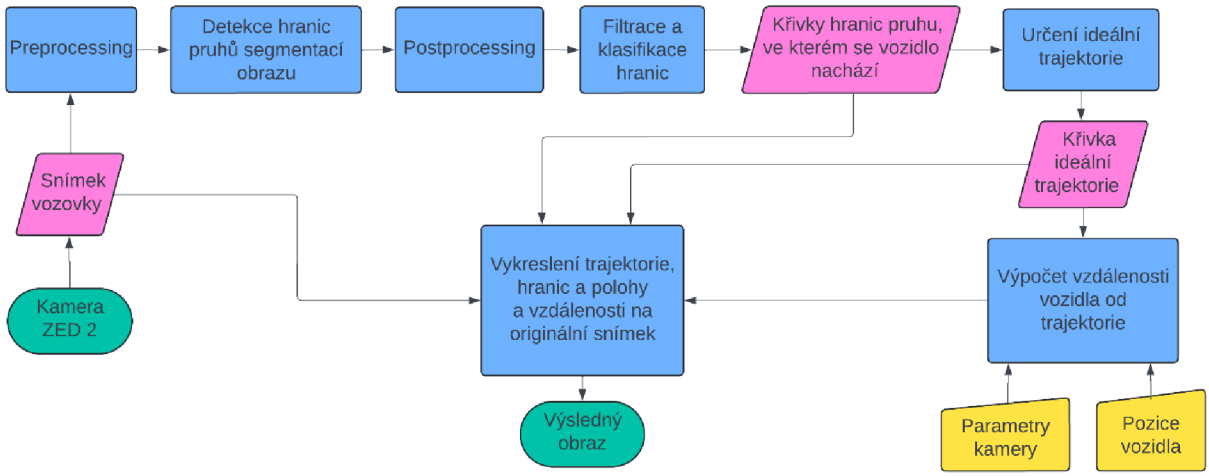
```
def get_relative_position(self, bottommost_trajectory_point):
    # Pozice vozidla ve směru osy x
    vehicle_position_x = self.vehicles_position_x
    # Bod trajektorie nejblíže k vozidlu
    bottommost_trajectory_x = bottommost_trajectory_point[0]
    # Horizontální vzdálenost v pixelech
    px_distance = vehicle_position_x - bottommost_trajectory_x
    # Převod vzdálenosti v pixelech na metry
    rw_distance = self.px_to_rw(px_distance)
    return rw_distance
```

*px_to_rw* je metodou pro převod vzdálenosti na snímku na skutečnou vzdálenost. Převádí tedy hodnotu v pixelech na metry. V této metodě je aplikován zmíněný výpočet. Návratová hodnota metody je vzdálenost převedená na metry a informace o tom, zda se vozidlo nachází nalevo nebo napravo od středu vozovky.

```
def px_to_rw(self, px_distance):
    theta_b = self.pitch_angle - self.V_FoV / 2
    l_c = self.camera_height / np.cos(theta_b)
    beta_c = np.arctan(px_distance / self.focal_length)
    rw_distance = abs(l_c * np.tan(beta_c))
    if px_distance == 0:
        # Text pro vizualizaci
        text = f"Vozidlo je umístěno v centru"
        return rw_distance, text
    elif px_distance < 0:
        side = "vlevo"
    else:
        side = "vpravo"
    # Text pro vizualizaci
    text = (f"Vozidlo je posunuto {rw_distance:.3f} m "
            f"{side} od středové čáry")
    return rw_distance, text
```

# 10 EXPERIMENTÁLNÍ OVĚŘENÍ SYSTÉMU

Na *Obr. 44* lze vidět diagram funkce navrženého systému, ve kterém jsou implementovány všechny navržené algoritmy. Pro ověření funkce algoritmu bylo využito dat nashromážděných kamerou. Tato kapitola popisuje ověření využitelnosti navrženého systému, natrénovaného modelu neuronové sítě a vyhodnocení jejich využitelnosti pro detekci jízdních pruhů.




*Obr. 44* Diagram funkce navrženého systému

## 10.1 BĚH NAVRŽENÉHO SYSTÉMU

Všechny části navrženého systému jsou spojeny v hlavní smyčce systému. Skript *main*, ve kterém se smyčka nachází propojuje jednotlivé moduly systému, iniciuje objekty tříd a volá jejich metody. Nejprve importuje potřebné knihovny a moduly, které jsou používány v hlavním skriptu a jeho souvisejících modulech.

Definuje se cesta k modelu U-Net pro detekci pruhů a načte se tento model. Hlavní funkce *main* obsahuje hlavní logiku systému. Inicializuje se objekt pro detekci pruhů, objekt pro zachycení snímků videa, objekt pro určení polohy vozidla a objekt pro odhad trajektorie.

Poté následuje hlavní smyčka, která zpracovává snímky videa, provádí detekci a určuje ideální trajektorii. Každý snímek videa je předzpracován a je aplikován ROI. Provádí se detekce hranic jízdního pruhu pomocí modelu U-Net a aplikuje se časové průměrování masky. Následně jsou získány kontury hranic pruhu, které jsou dále zpracovávány. Probíhá filtrace získaných bodů těžišť segmentů kontur. Poté dochází k aproximaci hranic polynomiálními křivkami. Ty jsou promítnuty do původního snímku.

Dále se aktualizují hranice třídy pro určování trajektorie a jsou získána trajektorie, která je také promítnut do snímku. Systém také určuje relativní pozici vozidla vůči středové čáře. Získaná poloha vozidla je rovněž zobrazena na obrazovce. Po skončení smyčky se uvolní všechny používané prostředky.

## 10.2 DATA PRO OVĚŘENÍ FUNKCE SYSTÉMU

Díky kameře ZED 2 a příslušného algoritmu pro komunikaci s PC, bylo nashromážděno značné množství dat, které mohly být využity k ověření systému. Část nasbíraných dat byla použita

na tvorbu datové sady pro trénink modelu neuronové sítě (viz. Příprava datové sady), kde se jednalo o snímky vozovky. Zde bylo využito videosouborů, u nichž bylo zajištěno, že se data neshodují s daty pro trénink neuronové sítě. Soubory byly seskupeny podle dopravních situací, které obsahují. Účelem bylo ověření funkčnosti v daných situacích. Testovány byly následující situace:

1. Pravotočivá zatáčka
2. Levotočivá zatáčka
3. Dvě po sobě jdoucí zatáčky („esovitá zatáčka“)
4. Rovina
5. Vozovka překrytá vozidlem z výhledu
6. Oslnění kamery
7. Stíny na vozovce
8. Přechod pro chodce
9. Vozovka bez značení

Každému modelovému příkladu byla přiřazena sekvence 150 snímků, na kterém byla ověřena správnost detekce. Na *Obr. 45* lze vidět příklad jednoho snímku zpracovaného navrženým systémem. Na příkladový snímek jsou promítnuty navržené křivky (zeleně) reprezentující hranice jízdního pruhu. Dále lze vidět přerušovanou křivku navržené ideální trajektorie (žlutě), označenou polohu středu vozidla (červeně) a nakonec ve vrchní části snímku se nachází výpis udávající aktuální polohu vozidla vzhledem k navržené trajektorii.


*Obr. 45* Ukázka vizualizace výstupů jednotlivých částí systému při jízdě po rovině – hranice jízdního pruhu (zeleně), navržená trajektorie (žlutě), poloha středu vozidla (červeně), aktuální poloha vozidla vůči trajektorii (bílý text)

## 10.3 VÝPOČETNÍ ČAS

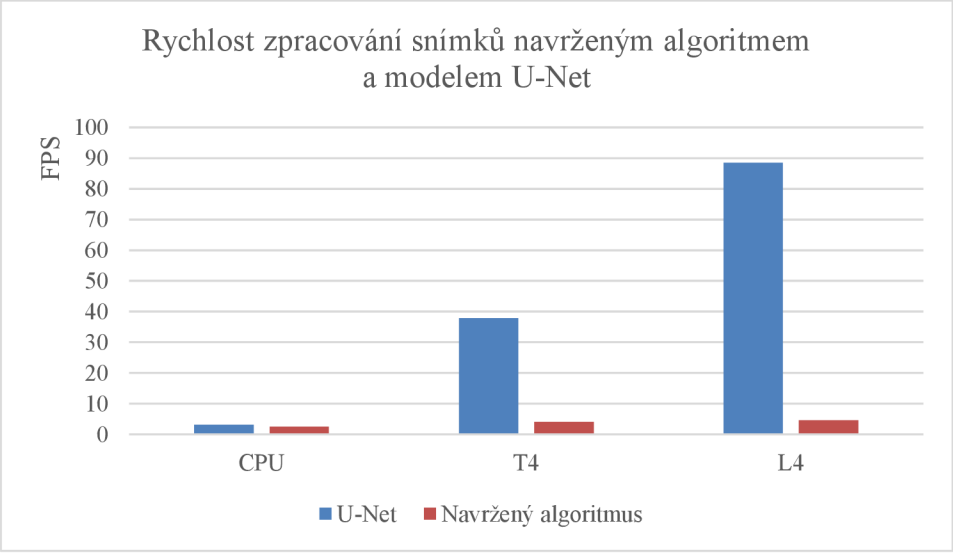
U algoritmu i samotného modelu U-Net byl změřen čas zpracování jednoho snímku. Pro algoritmus se jednalo o dobu běhu jedné iterace hlavní smyčky. Celkově byl čas měřen na třech

typech hardwaru – CPU využitého PC (hw kamera) a dvou typech GPU v prostředí Google Colab (Tesla T4, Tesla L4). Výsledky měření lze porovnat na grafu (*Obr. 46*) a v *Tab. 4*.

*Tab. 4* Rychlost zpracování snímku

| Typ zpracování | Metrika | CPU | T4 | L4 |
|---|---|---|---|---|
| Algoritmus | FPS | 2,5 | 4,1 | 4,6 |
| | Čas [ms] | 400,4 | 242,2 | 218,7 |
| Model U-Net | FPS | 3,1 | 37,9 | 88,5 |
| | Čas [ms] | 319,0 | 26,4 | 11,3 |

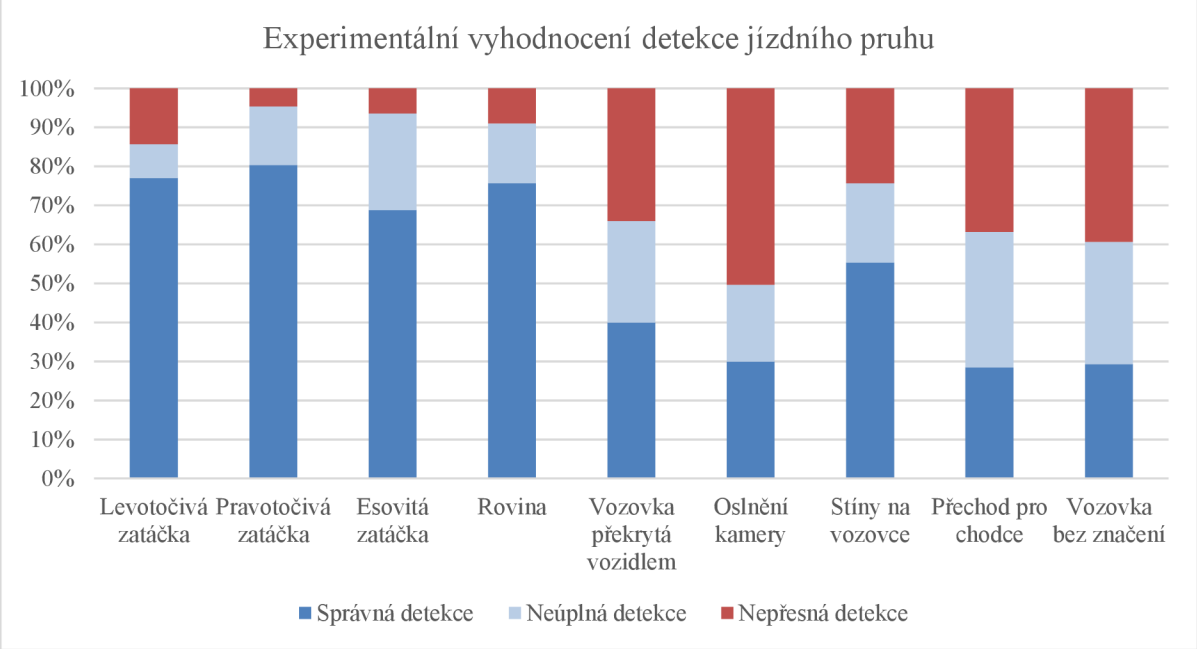
Z výsledků je patrné, že využitím GPU se podstatně zkracuje čas zpracování jednoho snímku. Nicméně snímková frekvence algoritmu, která dosáhla nejvyšší hodnoty 4,6 fps, je z pohledu využití v reálném čase nedostatečná. Z porovnání výsledků hodnot modelu U-Net je zřejmé, že největší podíl na výpočetním čase představuje postprocessing predikce modelu a další operace implementované v systému. Model U-Net dosáhl na Tesla T4 37,9 fps a na Tesla L4 dokonce 88,5 fps, což lze prohlásit za vyhovující hodnoty.




*Obr. 46* Graf udávající počet snímků zpracovaných navrženým algoritmem a modelem U-Net za využití daného HW

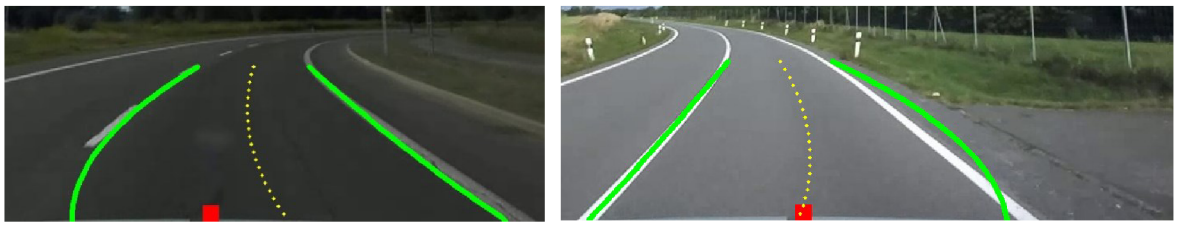
## 10.4 VYHODNOCENÍ DETEKCE JÍZDNÍCH PRUHŮ

Výsledky správnosti detekce byly vyhodnoceny do tří tříd – správná detekce (křivky polynomu přesně kopírují hranice jízdního pruhu), částečná detekce (křivky kopírují jen část hranice) a špatná detekce (křivky polynomu jsou deformovány nebo vůbec nereprezentují hranice pruhu). Výsledky tohoto ověření jsou shrnuty v grafu na *Obr. 47*.




*Obr. 47* Graf vyhodnocení správnosti detekce hranic jízdního pruhu navrženým algoritmem

Nejvyšší úspěšnost dosahoval algoritmus při průjezdu zatáčkou konstantního zakřivení, kde při průjezdu vozidlem pravotočivou zatáčkou dosáhla úspěšnost 80 %. Jak lze vidět vlevo na *Obr. 48*, systém měl problémy zejména s přerušovanou čarou značení jízdního pruhu. Toto je problém falešně negativní detekce modelu. I přesto, že je zde vidět určitá funkčnost navržené funkce algoritmu detekce, která doplnila chybějící bod pro proložení křivkou do spodu snímku (křivka není zcela deformovaná), nelze zde detekci považovat za správnou.



*Obr. 48* Falešně negativní detekce (vlevo) a falešně negativní detekce (vpravo) při průjezdu zatáčkou

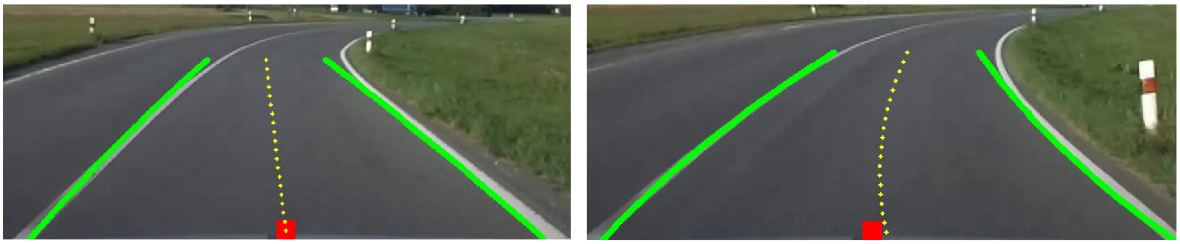
*Tab. 5* ukazuje procentuální úspěšnost detekce tohoto modelového případu, která byla vypočtena ze 150 výskytů pro každou hranici.

*Tab. 5* Procentuální úspěšnost detekce – levotočivá a pravotočivá zatáčka

| **Levotočivá zatáčka** | | | **Pravotočivá zatáčka** | | |
|---|---|---|---|---|---|
| **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** | **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** |
| Správná | 91 % | 63 % | Správná | 69 % | 91 % |
| Částečná | 9 % | 9 % | Částečná | 25 % | 5 % |
| Špatná | 0 % | 29 % | Špatná | 5 % | 4 % |

Vpravo na *Obr. 48* lze na druhou stranu vidět příklad falešně pozitivní detekce, která byla pozorována při průjezdu levotočivou zatáčkou. Tento problém se zřejmě objevil nedostatečným

zastoupením podobných snímků v datasetu. Lze totiž vidět, že algoritmus zde považuje za jízdní pruh část křižovatky. Celková úspěšnost detekce u tohoto testu dosáhla 77 %. Poměrné procentuální zastoupení správnosti detekce pro jednotlivé hranice lze opět vidět v *Tab. 5*.



*Obr. 49* Ukázka zkreslení detekce vlivem využití časového okna a průměrování koeficientu polynomů

Při průjezdu vozidlem dvěma zatáčkami za sebou dosáhl systém úspěšnosti 69 %. Na *Obr. 48* lze vidět jedna ze slabin navrženého algoritmu, a to předpokládané zkreslení detekce. Změna geometrie během průjezdu vozidla esovitou zatáčkou ukázala, že využitá filtrace šumu využitím časového okna a následné průměrování koeficientů polynomu není ideální cestou. Možným způsobem, jak odstranit tento problém, by mohla být další optimalizace výpočetního času algoritmu, a tedy využití vyšší vzorkovací frekvence video vstupu z kamery. Nerovnoměrnost úspěšnosti detekce mezi levou a pravou hranicí pruhu (viz *Tab. 6*) zde může být přisouzena prudší změně poloměru zakřivení pravé hranice vozovky při nájezdu do druhé zatáčky.

*Tab. 6* Procentuální úspěšnost detekce – esovitá zatáčka, rovina

| **Esovitá zatáčka** | | | **Rovina** | | |
|---|---|---|---|---|---|
| **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** | **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** |
| Správná | 79 % | 59 % | Správná | 89 % | 63 % |
| Částečná | 15 % | 35 % | Částečná | 9 % | 22 % |
| Špatná | 7 % | 6 % | Špatná | 3 % | 15 % |

Porovnáním hodnot obou hranic při jízdě po rovině (*Tab. 6*) si lze všimnout odlišných hodnot úspěšnosti. Algoritmus zde měl problém s detekcí vnější hranice vozovky (pravá hranice pruhu). Nicméně u jízdy po rovině můžeme pozorovat vysokou úspěšnost detekce i při přerušovaném značení jízdního pruhu, kdy se správně uplatňuje navržená metoda pro přidání bodu průsečíku (viz kap. Ošetření falešně negativní detekce).

Ačkoliv se při jízdě za vozidlem z výhledu na vozovce současně vyskytovaly stíny na vozovce a celková procentuální úspěšnost dosáhla pouze 40 %, lze si všimnout, že algoritmus byl schopen pruh detekovat (viz *Obr. 50*). Křivky hranic tedy nebyly zcela deformované. Jednalo se nicméně o jízdu po rovině a v těchto případech je možné připsat zásluhu časovému průměrování detekce a průměrování koeficientů polynomů. Vyhodnocení obou hranic znázorňuje *Tab. 7*.

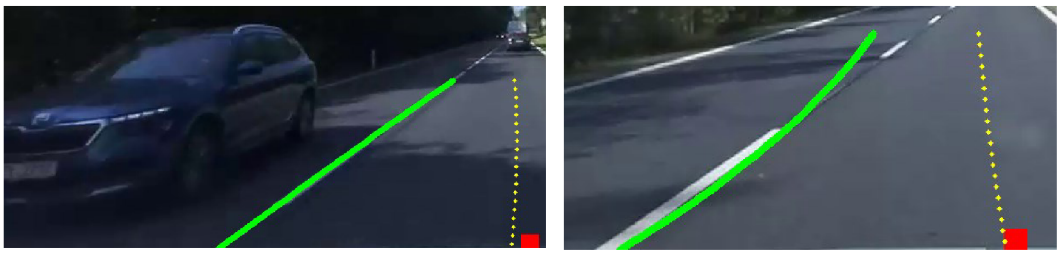
*Obr. 50* Ukázka detekce při jízdě v blízkosti vozidla z výhledu (vlevo) a detekce při oslnění kamery (vpravo)

Jeden z největších problémů pro algoritmus představovalo oslnění kamery slunečním zářením (*Obr. 50*), kdy úspěšnost detekce klesla až na 30 %. Těmto situacím by se pravděpodobně dalo předejít vhodnějším umístěním kamery ve vozidle.

*Tab. 7* Procentuální úspěšnost detekce – vozovka překrytá vozidlem, oslnění kamery

| **Vozovka překrytá vozidlem** | | | **Oslnění kamery** | | |
|---|---|---|---|---|---|
| **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** | **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** |
| Správná | 42,7 % | 37,3 % | Správná | 35,3 % | 24,7 % |
| Částečná | 32,7 % | 19,3 % | Částečná | 22,7 % | 16,7 % |
| Špatná | 24,7 % | 43,3 % | Špatná | 42,0 % | 58,7 % |

Při jízdě po rovné vozovce, na které se vyskytovaly stíny dosáhl systém celkové úspěšnosti detekce 55 %. Z vizualizovaných snímků je zřejmé, že vozovka překrytá stínem nepůsobí systému problémy. To se však nedá říct o stínech, částečně zakrývající povrch vozovky, kdy v obraze vznikají hrany. Na *Obr. 51* níže můžete vidět také příklad detekce při průjezdu vozidla v protisměru.



*Obr. 51* Ukázka detekce se stíny na vozovce a vozidlem projíždějícím v protisměru

Nejnižší úspěšnost detekce dosáhl algoritmus při posledních dvou situacích – jízda po vozovce bez značení jízdního pruhu a průjezd přes přechod pro chodce. Tyto dvě situace nebyly dostatečně zastoupeny v trénovací sadě modelu, proto pro algoritmus představovaly extrémní situaci. Poměr detekce obou hranic u těchto dvou situací lze vidět v *Tab. 8* a

*Tab. 9*.

Celková úspěšnost detekce všech zmíněných situací (viz *Obr. 47*) dosáhla 59,9 % (viz *Tab. 9*). Pokud bychom uvažovali pouze méně náročné případy detekce, tj. bez přechodu pro chodce, oslnění slunce a uvažovali bychom pouze značené jízdní pruhy, dostaneme úspěšnost detekce 68 %.

*Tab. 8* Procentuální úspěšnost detekce – stíny na vozovce, přechod pro chodce

| Stíny na vozovce | | | Přechod pro chodce | | |
|---|---|---|---|---|---|
| **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** | **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** |
| Správná | 59,3 % | 51,3 % | Správná | 27,3 % | 29,7 % |
| Částečná | 20,0 % | 20,7 % | Částečná | 30,0 % | 39,3 % |
| Špatná | 20,7 % | 28,0 % | Špatná | 42,7 % | 31,0 % |

## 10.5 Funkčnost určování trajektorie a polohy vozidla

Algoritmy pro určování trajektorie a polohy vozidla pracují na základě dat získaných při detekci hranic jízdního pruhu. Je tedy zřejmé, že přesnost těchto výpočtů je závislá na přesnosti detekce. Tento případ lze vidět například na *Obr. 50*, kdy vpravo vidíme případ deformovaných hranic pruhu. V podobných situacích tedy dochází také ke zkreslení predikované trajektorie a výpočtu vzdálenosti vozidla od ní. V situacích, kdy jsou hranice jízdního pruhu detekovány správně (viz *Obr. 45*), lze vidět také správně určenou trajektorii a vzdálenost vozidla od této určené trajektorie.

*Tab. 9* Procentuální úspěšnost detekce – vozovka bez značení, celková úspěšnost

| Vozovka bez značení | | | Celková úspěšnost | |
|---|---|---|---|---|
| **Hodnocení detekce** | **Levá hranice** | **Pravá hranice** | **Hodnocení detekce** | **Pravá hranice** |
| Správná | 35,3 % | 26,3 % | Správná | 56,9 % |
| Částečná | 18,0 % | 38,0 % | Částečná | 20,0 % |
| Špatná | 46,7 % | 35,7 % | Špatná | 23,0 % |

## 10.6 Zhodnocení detekce jízdních pruhů modelem U-Net

Funkčnost detekce modelem U-Net byla ověřena s využitím části datové sady popsané v kap. 6.1. Jednalo se o část dat určených k otestování modelu (testovací dataset). Tento dataset obsahoval 438 vzorků z datové sady TuSimple rozšířené daty nasbíranými kamerou ZED 2.

```
# Unet
[INFO:] found 438 original examples in the test set...
[INFO:] loading up model...
100%|██████████| 110/110 [15:33<00:00,  8.49s/batch]
Average Test Loss: 0.636
Average Test Accuracy: 0.899
Average Test Precision: 0.182
Average Test Recall: 0.864
Average Test F1 Score: 0.301
```

Obr. 52 Výsledek predikce modelu U-Net na testovacím datasetu z prostředí Google Colab

Výsledky dosažené při predikci na testovacím datasetu (viz Obr. 52) potvrzují schopnost generalizace modelu, tj. replikovat výsledky na nových datech. Přetrvává zde však problém s falešně pozitivní predikcí (nízká hodnota precision). Následný výskyt šumu v segmentační masce představoval obtíže při dalším zpracování obrazu. Jelikož z analýzy výpočetního času je patrné, že při využití GPU nebyl problém dosáhnout predikce modelu v reálném čase, možným řešením by mohlo být rozšíření modelu sítě. Zpřesnění predikce modelem, by poté ušetřilo výpočetní čas při dalším zpracování obrazu.

# ZÁVĚR

Tato diplomové práce se zabývala návrhem a experimentálním ověřením systému pro detekci jízdních pruhů, určováním trajektorie a polohy vozidla. Výstupem práce je navržený systém, využívající model neuronové sítě U-Net, prvky počítačového vidění, časového okna k redukci výskytu falešně pozitivní detekce. K výpočtu polohy vozidla vůči navržené trajektorii bylo využito trigonometrie a parametrů kamery. Dalším z výstupů bylo experimentální ověření navrženého algoritmu. V rámci práce vzniklo také rozšíření datové sady TuSimple o data nashromážděná kamerou ZED 2.

Z dat ověření výpočetního času systému je zřejmé, že celkový algoritmus není schopen detekce v reálném čase, jelikož při využití GPU Tesla L4, dosahuje rychlosti zpracování snímků pouze 4,6 fps. Protože samotný model U-Net za stejných podmínek zpracovává snímky s rychlostí 88,5 fps, je zde prostor pro rozšíření modelu neuronové sítě pro dosažení kvalitnější predikce modelu a snížení počtu operací postprocessingu.

Výsledky experimentálního ověření detekce jízdních pruhů ukázaly, že má systém vysokou úspěšnost detekce v jednodušších situacích jako jsou rovné úseky silnic nebo průjezd zatáčkou konstantního poloměru zakřivení. Naopak komplikovanější případy, například jízda při oslnění kamery slunečními paprsky či při překrytí vozovky stíny, způsobily výrazné snížení spolehlivosti detekce.

Hlavní omezení navrženého systému spočívají v nedostatečném zastoupení kritických situací v datové sadě, což vedlo nižší k úspěšnosti u těchto situací. Dále byl zjištěn problém vysokého výskytu falešně pozitivních detekcí. Ty komplikovaly následné zpracování obrazu. Při analýze predikce však byla prokázána užitečnost navržených metod pro ošetření výskytu falešně pozitivních detekcí, ale také falešně negativních.

I přes zjištěné nedostatky, experimenty prokázaly schopnost systému detekovat hranice jízdních pruhů a následně navrhovat ideální trajektorii vozidla v podobě středu mezi navrženými hranicemi. Výsledky algoritmu pro určení trajektorie a polohy vozidla jsou závislé na úvodní predikci hranic jízdního pruhu, navržené algoritmy jsou však funkční.

Hlavní potenciál pro zlepšení výsledků navrženého systému spočívá v rozšíření datové sady a využití komplexnějšího modelu neuronové sítě. Možným rozšířením by mohla být implementace modulu RNN do modelu. Po zajištění spolehlivější detekce neuronovou sítí, by mohla být dále provedena optimalizace výpočetního času pro fungování systému v reálném čase. Celkově tato diplomová práce prokázala, že je možné dosáhnout vysoké přesnosti detekce jízdních pruhů pomocí využitých metod.

# POUŽITÉ INFORMAČNÍ ZDROJE

[1] VEGIA, Marialena; TRANSETH, Aksel a FJERDINGEN, Sigurd. A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed?. online. Dostupné z: https://doi.org/10.1016/j.apergo.2015.09.013. [cit. 2023-03-17].

[2] BANKS, Victoria a STANTON, Neville. *Automotive Automation: Distributed Cognition on the Road*. online. 1. Boca Raton: CRC Press, 2017. ISBN 9781315295657. Dostupné z: https://doi.org/10.1201/9781315295657. [cit. 2023-03-17].

[3] JAHROMI, B. S. Levels of Automation for Autonomous Ground Vehicles. online. In: *Medium*. Medium. Dostupné z: https://medium.com/@BabakShah/levels-of-automation-for-self-driving-cars-d410a4f679b7. [cit. 2023-03-18].

[4] J3016. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Doplněné 2021. Geneva: SAE International, 2014.

[5] DIETZE, Karl-Heinz a REIF, Konrad. *Automotive Handbook*. 11. vydání. Postfach, Německo: Robert Bosch GmbH, 2022. ISBN 978-1-119-91190-6.

[6] FRIDMAN, Lex. *MIT Deep Learning and Artificial Intelligence Lectures*. online. In: . USA: MIT. Dostupné z: https://deeplearning.mit.edu/. [cit. 2023-03-17].

[7] MAURER, Markus; GERDES, J.; LENZ, Barbara a WINNER, Hermann. *Autonomous Driving: Technical, Legal and Social Aspects*. online. Ladenburg: Springer Nature, 2015. ISBN 978-3-662-48847-8. [cit. 2023-03-17].

[8] BRUMMELEN, Jessica; O'BRIEN, Marie; GRUYER, Dominique a NAJJARAN, Homayoun. Autonomous vehicle perception: The technology of today and tomorrow. online. *Transportation Research: Part C: Emerging Technologies*. 2018, s. 384-406. Dostupné z: https://doi.org/10.1016/j.trc.2018.02.012. [cit. 2023-03-17].

[9] WANG, Chaoyang; HU, Hao; LIANG, Yanxue a SHEN, Gang. On the Application of Cameras Used in Autonomous Vehicles. online. Dostupné z: https://doi.org/10.1007/s11831-022-09741-8. [cit. 2023-03-17].

[10] *ATEsystem: Time-of-Flight*. online. Ostrava: ATEsystem s.r.o. Dostupné z: https://eshop.atesystem.cz/. [cit. 2023-03-17].

[11] *Apollo: Apollo Perception*. online. Apollo, 2020. Dostupné z: https://developer.apollo.auto/platform/perception.html. [cit. 2023-03-17].

[12] SAJDL, Jan. *DAC (Driver Alert Control)*. online. In: Autolexicon.net. autolexicon.net. Dostupné z: https://www.autolexicon.net/cs/articles/dac-driver-alert-control/. [cit. 2023-03-17].

[13] BURKE, Katie. *How Does a Self-Driving Car See?: Camera, radar and lidar sensors give autonomous vehicles superhuman vision.* online. In: NVIDIA Blog. USA: NVIDIA Corporation. Dostupné z: https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/. [cit. 2023-03-17].

[14] REIL, Andreas a HEUEL, Steffen. *Radarová technologie v automobilech*. online. In: Průmyslové spektrum. MM Průmyslové spektrum, 2001. Dostupné z: https://www.mmspektrum.com/clanek/radarova-technologie-v-automobilech. [cit. 2023-03-17].

[15] ARAGE, Alabel; STEFFENS, Wolf; KUEHNLE, Goetz a JAKOBY, Rolf. Effects of Water and Ice Layer on Automotive Radar. online. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=65a40d1c9af3fd15eaff1b735d7ee89d8fe7afcf. [cit. 2023-03-17].

[16] RORIZ, R.; CABRAL, J. a GOMES, T. Automotive LiDAR Technology: A Survey. online. *TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9455394. [cit. 2023-03-17].

[17] RASSHOFER, R. a SPIES, H. Influences of weather phenomena on automotive laser radar systems. online. *Advances in Radio Science*. 2011, s. 49-60. Dostupné z: https://doi.org/10.5194/ars-9-49-2011. [cit. 2023-03-17].

[18] *Velodyne Lidar*. online. Velodyne Lidar, 2023. Dostupné z: https://velodynelidar.com. [cit. 2023-03-17].

[19] *Tesla: Tesla Vision Update: Replacing Ultrasonic Sensors with Tesla Vision*. online. USA: Tesla, 2022. Dostupné z: https://www.tesla.com/support/transitioning-tesla-vision. [cit. 2023-03-17].

[20] VARGAS, J.; ALSWEISS, S.; TOKER, A.; RAZDAN, R. a SANTOS, J. An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions. online. Dostupné z: https://doi.org/10.3390/s21165397. [cit. 2023-03-17].

[21] *Otomoto: How Ultrasonic Sensor Data is Powering Automotive IoT*. online. Frankfurt, Germany: Otomoto, 2021. Dostupné z: https://otonomo.io/blog/ultrasonic-data-automotive-iot/. [cit. 2023-03-17].

[22] *Vectornav: Inertial Navigation Articles*. online. Vectornav, 2023. Dostupné z: https://www.vectornav.com/resources/inertial-navigation-articles. [cit. 2023-03-17].

[23] BETZ, J.; ZHENG, H.; LINIGER, A.; ROSOLIA, U.; KARLE, P. et al. Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing. online. *Intelligent Transportation Systems*. Dostupné z: https://doi.org/10.1109/OJITS.2022.3181510. [cit. 2023-03-17].

[24] WEISS, T. a BEHL, M. DeepRacing: Parameterized Trajectories for Autonomous Racing. online. Dostupné z: https://doi.org/10.48550/arXiv.2005.05178. [cit. 2023-03-17].

[25] CHEN, Ch.; SEFF, A.; KORNHAUSER, A. a XIAO, J. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. online. 2015. Dostupné z: https://doi.org/10.1109/ICCV.2015.312. [cit. 2023-03-17].

[26] NGUYEN, A. a LE, B. 3D point cloud segmentation: A survey. online. *Conference on Robotics, Automation and Mechatronics (RAM)*. Dostupné z: https://doi.org/10.1109/RAM.2013.6758588. [cit. 2023-03-17].

[27] DURANT-WHYTE, H. F. Sensor Models and Multisensor Integration. online. In: *Autonomous Robot Vehicles*. New York: Springer, 1990. ISBN 978-1-4613-8997-2. Dostupné z: https://doi.org/10.1007/978-1-4613-8997-2_7. [cit. 2023-03-17].

[28] ELMENREICH, W. An Introduction to Sensor Fusion. online. Dostupné z: https://www.researchgate.net/profile/Wilfried-Elmenreich/publication/267771481_An_Introduction_to_Sensor_Fusion/links/55d2e45908ae0a3417222dd9/An-Introduction-to-Sensor-Fusion.pdf. [cit. 2023-03-17].

[29] GOEL, A. Lane Detection Techniques - A Review. online. *IJCA*. ISSN 2320–088X. Dostupné z: https://ijcsmc.com/docs/papers/February2014/V3I2201491.pdf. [cit. 2023-03-17].

[30] ZOICAN, S.; VOCHIN, M.; ZOICAN, R. a GALATCHI, D. Lane Departure Warning System Implementation using the Blackfin Microcomputer. online. *ISETC*. Dostupné z: https://doi.org/10.1109/ISETC.2016.7781044. [cit. 2023-03-17].

[31] CHENG, H. *Autonomous Intelligent Vehicles: Theory, Algorithms, and Implementation*. online. New York, USA: Springer, 2011. ISBN 978-1-4471-2280-7. Dostupné z: https://doi.org/10.1007/978-1-4471-2280-7. [cit. 2023-03-17].

[32] VISWANATHAN, V. a HUSSEIN, R. Applications of Image Processing and Real-Time embedded Systems in Autonomous Cars: A Short Review. online. *IJIP*. 2017. [cit. 2023-03-17].

[33] GUO, J.; WEI, Z. a MIAO, D. Lane Detection Method Based on Improved RANSAC Algorithm. online. *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*. 2015. Dostupné z: https://doi.org/10.1109/ISADS.2015.24. [cit. 2023-03-17].

[34] FISHER, R. *Canny Edge Detector*. online. In: . Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm. [cit. 2023-03-17].

[35] YING, Z. a LI, G. Robust lane marking detection using boundary-based inverse perspective mapping. online. *ICASSP*. ISSN 2379-190X. Dostupné z: https://doi.org/10.1109/ICASSP.2016.7472011. [cit. 2023-03-17].

[36] WANG, Z.; REN, W. a QIU, Q. LaneNet: Real-Time Lane Detection Networks for Autonomous Driving. online. Dostupné z: https://doi.org/10.48550/arXiv.1807.01726. [cit. 2023-03-18].

[37] CVIJETIC, N. Read Between the Lines: How We Taught Neural Nets to Predict Lane Lines. online. In: *NVIDIA Blog*. USA: Nvidia. Dostupné z: https://blogs.nvidia.com/blog/2019/07/10/drive-labs-neural-nets-predict-lane-lines/. [cit. 2023-03-18].

[38] GETAHUN, T. *Lane Detection for Autonomous Driving: Conventional and CNN approaches*. online. In: Access Laboratory. Access Lab., 2021. [cit. 2023-03-18].

[39] DONG, Y.; PATIL, S.; AREM, B. a FARAH, H. A hybrid spatial–temporal deep learning architecture for lane detection. online. *Computer-Aided Civil and Infrastructure Engineering*. 2022, s. 67–86. Dostupné z: https://doi.org/10.1111/mice.12829. [cit. 2023-03-18].

[40] ZHENG, T.; HUANG, Y.; LIU, Y.; TANG, W.; YANG, Z. et al. CLRNet: Cross Layer Refinement Network for Lane Detection. online. *CVPR*. s. 888-897. Dostupné z: https://doi.org/10.1109/CVPR52688.2022.00097. [cit. 2023-03-18].

[41] MAMUN, A.; PING, E.; HOSSEN, J.; TAHABILDER, A. a JAHAN, B. A Comprehensive Review on Lane Marking Detection Using Deep Neural Networks. online. *Sensors*. ISSN 1424-8220. Dostupné z: https://doi.org/10.3390/s22197682. [cit. 2023-03-18].

[42] BADRINARAYANAN, V.; KENDALL, A. a CIPOLLA, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. online. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. s. 2481-2495. ISSN 1939-3539. Dostupné z: https://doi.org/10.1109/TPAMI.2016.2644615. [cit. 2023-03-18].

[43] PAN, X.; SHI, J.; LUO, P.; WANG, X. a TANG, X. Spatial As Deep: Spatial CNN for Traffic Scene Understanding. online. *AAAI Conference on Artificial Intelligence*. Dostupné z: https://doi.org/10.1609/aaai.v32i1.12301. [cit. 2023-03-18].

[44] ZHENG, T.; FANG, H.; ZHANG, Y.; TANG, W.; YANG, Z. et al. RESA: Recurrent Feature-Shift Aggregator for Lane Detection. online. *AAAI Conference on Artificial Intelligence*. s. 3547-3554. Dostupné z: https://doi.org/10.1609/aaai.v35i4.16469. [cit. 2023-03-18].

[45] ZHANG, H.; GU, Y.; WANG, X.; WANG, M. a PAN, J. SOLOLaneNet: Instance Segmentation-Based Lane Detection Method using Locations. online. *ITSC*. Dostupné z: https://doi.org/10.1109/ITSC48978.2021.9564795. [cit. 2023-03-18].

[46] QIN, Z.; WANG, H. a LI, X. Ultra Fast Structure-aware Deep Lane Detection. online. *ECCV: Lecture Notes in Computer Science*. Dostupné z: https://doi.org/10.1007/978-3-030-58586-0_17. [cit. 2023-03-18].

[47] TABELINI, L.; BERRIEL, R.; PAIXãO, T.; BADUE, C.; SOUZA, A. et al. PolyLaneNet: Lane Estimation via Deep Polynomial Regression. online. *ICPR*. Dostupné z: https://doi.org/10.1109/ICPR48806.2021.9412265. [cit. 2023-03-18].

[48] *Papers with code: Lane Detection*. online. 2023. Dostupné z: https://paperswithcode.com/task/lane-detection#task-home. [cit. 2023-03-18].

[49] FRANçOIS, Chollet. *Deep learning v jazyku Python: Knihovna Keras, TensorFlow*. 2. Praha: Grada Publishing, 2023. ISBN 978-80-271-5133-2.

[50] YINGGE, H.; IMRAN, A. a KANG-YOON, L. Deep Neural Networks on Chip - A Survey. online. *IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2020. ISSN 2375-9356. Dostupné z: https://doi.org/10.1109/BigComp48618.2020.00016. [cit. 2024-05-20].

[51] ROBOTDREAMS. *Robotdreams*. online. 2020. Dostupné z: https://robotdreams.cz/. [cit. 2024-05-10].

[52] STEREOLABS. *Stereolabs: ZED 2*. online. San Francisco, 2024. Dostupné z: https://www.stereolabs.com/products/zed-2. [cit. 2024-03-24].

[53] SRIDHARA, M. TuSimple Dataset: Ace the Lane Detection Challenge. online. In: GOOGLE. *Kaggle*. Dostupné z: https://www.kaggle.com/datasets/manideep1108/tusimple?resource=download. [cit. 2024-05-24].

[54] RONNEBERGER, O.; FISCHER, P. a BROX, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. online. 2015. Dostupné z: https://doi.org/https://doi.org/10.48550/arXiv.1505.04597. [cit. 2024-05-20].

[55] CHANDHOK, S. *Pyimagesearch: U-Net: Training Image Segmentation Models in PyTorch*. online. 2021. Dostupné z: https://pyimagesearch.com/2021/11/08/u-net-training-image-segmentation-models-in-pytorch/. [cit. 2024-05-20].

[56] LAUER, Martin a GEIGER, Andreas. *Mathematisch-Naturwissenschaftliche Fakultät - Self-Driving Cars*. online. In: . Dostupné z: https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/self-driving-cars/. [cit. 2024-05-10].

[57] REZAEI, Mahdi; TERAUCHI, Mutsuhiro a KLETTE, Reinhard. Robust Vehicle Detection and Distance Estimation Under Challenging Lighting Conditions. online. *IEEE Transactions on Intelligent Transportation Systems*. 2015, s. 2723-2743. ISSN 1558-0016. Dostupné z: https://doi.org/10.1109/TITS.2015.2421482. [cit. 2024-05-20].

[58] THE MATHWORKS. *MathwWorks - Help Center*. online. 1994-2024. USA. Dostupné z: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [cit. 2024-05-20].

# SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

| | | |
|---|---|---|
| *2D* | | Dvojrozměrný |
| *3D* | | Trojrozměrný |
| *A/D* | | Analogově-digitální |
| *ADS* | | Automated driving system |
| *AV* | | Autonomní vozidlo |
| *BASt* | | Bundesanstalt für Straßenwesen |
| *BC* | [m] | Vzdálenost středu kamery, respektive vozidla od středu vozovky |
| *BGR* | | Blue Green Red |
| *Bi* | [px] | Šířka snímku |
| *Bp* | [px] | Vzdálenost vozidla od trajektorie na ose x snímku |
| *CUDA* | | Compute Unified Device Architecture |
| *DARPA* | | Defense Advanced Research Projects Agency |
| *DAS* | | Driver alert system |
| *DDT* | | Dynamic driving task |
| *DNN* | | Hloboká neuronová síť |
| *DSP* | | Digital signal processor |
| *ECU* | | Electronic Control Unit |
| *FIFO* | | First In First Out |
| *FMCW* | | Kmitočtově modulovaná nosná vlna |
| *FoV* | | Zosrné pole |
| *GNSS* | | Global Navigation Satellite System |
| *GPS* | | Globální polohový systém |
| *GRU* | | Gated recurrent unit |
| *H* | [m] | Umístění kamery nad povrchem vozovky |
| *IMU* | | Inerciální měřící jednotka |
| *IPM* | | Inverse Perspective Mapping |
| *lc* | [m] | Vzdálenost kamery od nejbližšího promítaného bodu trajektorie |
| *LIDAR* | | Light Detection and Ranging |
| *lr* | | Učící rychlost |
| *LSTM* | | Long short term memory |
| *MPE* | | Maximální propustná expouice laserem |
| *ndjson* | | New line delimited JSON |

| | | |
|---|---|---|
| *NHTSA* | | National Highway Traffic Safety Administration |
| *NMS* | | Non-maximum suppression |
| *ODD* | | Operational design domain |
| *PC* | | Počítač |
| *px* | | Pixel |
| *R* | [m] | Vzdálenost |
| *RAM* | | Random Access Memory |
| *R-CNN* | | Region-based Convolutional Neural Networks |
| *ReLU* | | Rectified Linear Unit |
| *RESA* | | REcurrent Feature-Shift Aggregator |
| *RGB* | | Red Green Blue |
| *RNN* | | Rekurentní neuronová síť |
| *ROI* | | Oblast zájmu |
| *SAE* | | Society of Automotive Engineers |
| *SCNN* | | Sequential convolutional neural network |
| *ST-RNN* | | Spatial-temporal recurrent neural network |
| *ToF* | | Time of flight |
| *W&B* | | Weights and Biases |
| *Win* | | Windows |
| $\alpha$ | [°] | Vertikální zorné pole kamery |
| $\beta C$ | [°] | Úhel roviny yz kamery se spojnicí počátku souřadného systému kamery |
| $\theta c$ | [°] | úhel sklonu kamery vůči ose z vozovky |
| $c$ | [m·s-1] | Rychlost světla signálu |
| $\tau$ | [s] | Doba letu paprsku |

# SEZNAM PŘÍLOH

I. Skript pro komunikaci s kamerou
II. Video pro ověření funkčnosti algoritmů
III. Algoritmus detekce, určování trajektorie a polohy vozidla
IV. Jupyter Notebook se skripty pro trénink modelu
V. Model U-Net