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

ZÍSKÁVÁNÍ VSTUPŮ ZPRACOVÁNÍM OBRAZU PRO ŘÍZENÍ AUTONOMNÍHO VOZIDLA

ACQUISITION OF INPUTS BY IMAGE PROCESSING FOR CONTROLLING AN AUTONOMOUS VEHICLE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Daniel Midrla

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Pavel Kučera, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automobilního a dopravního inženýrství
Student:	**Bc. Daniel Midrla**
Studijní program:	Automobilní a dopravní inženýrství
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Pavel Kučera, Ph.D.**
Akademický rok:	2021/22

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:

Získávání vstupů zpracováním obrazu pro řízení autonomního vozidla

Stručná charakteristika problematiky úkolu:

Naprogramování řídicího algoritmu pro zpracování obrazu z kamer pro autonomní řízení vozidla využitím softwaru Simulink, nebo dalších programovacích jazyku C, C++, C#, atd. Jedná se o zpracování stereo obrazu pro určení vzdálenosti individuálních objektů před vozidlem a identifikace příslušného objektu. Experimentální ověření správné detekce vzdálenosti a typu objektu.

Cíle diplomové práce:

Rešerše o zpracování obrazů autonomních vozidel
Naprogramování základní komunikace se stereo kamerou
Naprogramování algoritmu pro určení vzdálenosti
Naprogramování algoritmu pro rozpoznání objektu
Experimentální ověření správné funkce zpracování obrazu

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 2021/22

V Brně, dne

L. S.

prof. Ing. Josef Štětina, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Předmětem této diplomové práce je sběr dat zpracováním obrazu pro řízení autonomního vozidla. Práce v první řadě nabízí shrnutí teoretických znalostí souvisejících s danou problematikou. Dále je popsán proces tvorby algoritmu, který s pomocí stereo kamery a neuronové sítě pro detekci objektů získává základní vstupy pro řízení autonomního vozidla. Těmi jsou informace o třídě rozpoznaného objektu a jeho vzdálenosti. Závěrem práce je provedeno experimentální ověření správné funkčnosti, přičemž je kladen důraz na optimalizaci přesnosti a rozsahu určení vzdálenosti. Také je vyhodnocena schopnost provozu algoritmu v reálném čase na kompaktním počítači s omezenou výpočetní kapacitou.

KLÍČOVÁ SLOVA

Detekce objektů, neuronová síť, stereo kamera, vnímání hloubky, počítačové vidění, autonomní vozidla, Python

ABSTRACT

This master's thesis deals with data acquisition by image processing in order to control an autonomous vehicle. Firstly, the thesis offers a summary of theoretical knowledge relevant to the given topic. Then follows a description of creating an algorithm, which acquires basic inputs for autonomous vehicle control with the use of a stereo camera and an object detection neural network. The inputs gained from this algorithm are the class of the detected object and its distance. Finally, an experimental evaluation of the correct functionality is performed with an emphasis on optimizing the accuracy and range of the distance computation. An assessment of the ability to deploy the created algorithm in real time on a compact computer with limited computing power is also performed.

KEYWORDS

Object detection, neural network, stereo camera, depth perception, computer vision, autonomous vehicles, Python

BIBLIOGRAFICKÁ CITACE

MIDRLA, Daniel. *Získávání vstupů zpracováním obrazu pro řízení autonomního vozidla.* Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/137095. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automobilního a dopravního inženýrství. 86 s. 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 20. května 2022 ……..……..……..

Daniel Midrla

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Pavlu Kučerovi, Ph.D., za ochotu a cenné připomínky při vedení mé práce a za zapůjčení vybavení potřebného k jejímu zpracování. Nesmírně si vážím významné podpory ze strany blízkých přátel a rodiny po dobu celého studia, i těm proto patří z mé strany velké poděkování.

OBSAH

Úvod **11**

1 Zpracování obrazu, neuronové sítě a autonomní řízení **13**
1.1 Zpracování obrazu 13
1.2 Umělá inteligence 13
1.2.1 Strojové učení a hluboké učení 13
1.2.2 Typy strojového učení 14
1.2.3 Aplikace řízeného učení 15
1.3 Neuronové sítě 16
1.3.1 Historie 16
1.3.2 Princip neuronové sítě 16
1.3.3 Učení neuronových sítí 18
1.4 Počítačové vidění 19
1.4.1 Úlohy počítačového vidění 19
1.4.2 Metriky pro vyhodnocení výkonnosti algoritmů detekce objektů 20
1.4.3 Konvoluční neuronové sítě 22
1.4.4 Rekurentní neuronové sítě 24
1.5 Příklady architektur neuronových sítí pro detekci objektů 24
1.5.1 R-CNN 24
1.5.2 SSD 25
1.5.3 YOLO 25
1.6 Určení vzdálenosti s využitím počítačového vidění 27
1.6.1 Epipolární geometrie 28
1.6.2 Rektifikace snímků 29
1.6.3 Stereo korespondence a disparita 29
1.7 Autonomní řízení 31
1.7.1 Úrovně automatizace řízení 31
1.7.2 Výhody a nevýhody autonomních vozidel 33
1.7.3 Současný stav a budoucnost 33

2 Využitý hardware a jeho konfigurace **35**
2.1 Stereo kamera Stereolabs ZED 2 35
2.2 Ovládání kamery prostřednictvím knihovny OpenCV 36
2.3 Počítač NVIDIA Jetson Nano 37

3 Algoritmus pro určení vzdálenosti **39**

4 Algoritmus pro detekci objektů **42**
4.1 Volba platformy pro detekci objektů 42
4.2 Dataset 43
4.3 Učení sítě 44
4.3.1 Instalace YOLOv5 a spuštění učení 44
4.3.2 Výsledky trénování 46
4.4 Spojení se stereo kamerou a určením vzdálenosti 48
4.5 Spuštění na počítači Jetson Nano 55

5 Ověření funkčnosti algoritmu 57
5.1 Přesnost a rozsah určení vzdálenosti 57
5.1.1 Tvorba dat pro vyhodnocení 57
5.1.2 Analýza vlivu parametrů disparity na přesnost určení vzdálenosti 59
5.1.3 Posouzení změny přesnosti při různých rozlišeních kamery 68
5.2 Zhodnocení přesnosti detekce objektů s určením vzdálenosti 69
5.2.1 Ověření varianty s architekturou YOLOv5s6 70
5.2.2 Ověření varianty s architekturou YOLOv5n6 71
5.2.3 Vyhodnocení spolehlivosti detekce objektů 72
5.3 Doba zpracování snímku dílčími částmi algoritmu 74
5.4 Rychlost zpracování na zařízení Jetson Nano 75

Závěr 77

Použité informační zdroje 79

Seznam použitých zkratek a symbolů 84

ÚVOD

Automatizace řízení v různých úrovních je tématem, které je v moderním prostředí automobilového průmyslu velmi aktuální. Jedná se o komplexní problematiku, která ve své nejjednodušší formě má za cíl zvýšit komfort řidičů ve městech či na dálnicích a v krajním případě má potenciál naprosto proměnit prostředí osobní i nákladní dopravy. S tím se pojí i široké interdisciplinární rozpětí, kdy je třeba se zabývat nejen samotným technickým zpracováním, ale i morálním a právním aspektem náhrady člověka v roli řidiče strojem.

Tato diplomová práce se věnuje získávání vstupů pro samotné řízení autonomního vozidla. Toho je docíleno vytvořením algoritmu, který zpracováním obrazu ze stereo kamery získává informace o objektech definovaných tříd v obrazu a jejich vzdálenost. Detekce objektů je realizována s využitím platformy pro trénování a provoz neuronových sítí pro detekci objektů YOLOv5 (You Only Look Once verze 5). K obdržení stereo obrazu je použita stereo kamera Stereolabs ZED 2.

V první kapitole je nabídnuto shrnutí potřebného teoretického základu k danému tématu. Zabývá se nejprve v krátkosti obecně zpracováním obrazu. Dále je zde představena problematika umělé inteligence, strojového učení a hlubokého učení. Také je zde obsaženo vysvětlení principu neuronových sítí, poté následuje teoretický základ problematiky počítačového vidění. Zde jsou popsány jednotlivé úlohy této disciplíny, základní typy používaných neuronových sítí a veličiny popisující jejich výkonnost. Dále jsou v kapitole představeny konkrétní příklady architektur neuronových sítí používaných v oblasti detekce objektů. V další části kapitoly je věnována pozornost zpracování stereo obrazu za účelem určení vzdálenosti objektů v tomto obrazu. Kapitolu uzavírá část zaměřená na autonomní řízení vozidel, kde jsou nejprve shrnuty úrovně autonomního řízení, výhody a nevýhody autonomních vozidel. Následuje představení současných technických řešení již provozovaných systémů pro automatizaci řízení a je nastíněn možný vývoj v budoucnosti.

Druhá kapitola je zaměřena na specifické hardwarové vybavení použité v rámci této práce. Nejprve je představena stereo kamera Stereolabs ZED 2 a je popsáno zprovoznění komunikace s touto kamerou připojenou k počítači s využitím programovacího jazyka Python ve spojení převážně s knihovnou OpenCV. Je představen proces načtení kalibračních parametrů kamery, získání obrazu z kamery a jeho následného zpracování, přesněji provedení rektifikace s využitím zmíněných kalibračních parametrů. Kapitola pokračuje popisem kompaktního počítače Jetson Nano a jeho základní uvedení do provozu. Jedná se o zařízení určené pro vývoj v různých oblastech umělé inteligence, na kterém je v dalších kapitolách spuštěn vytvořený algoritmus.

Třetí kapitola obsahuje popis vytvořeného Python kódu představujícího algoritmus pro získání informace o vzdálenosti zpracováním stereo obrazu. Tento algoritmus využívá postup obdržení rektifikovaného stereo obrazu představený v předchozí kapitole a rozšiřuje jej o výpočet disparity mezi levým a pravým obrazem kamery, což je jedna z hlavních veličin pro výpočet vzdálenosti v daném bodě obrazu. Vzdálenost dále závisí na rozestupu levé a pravé kamery a ohniskové vzdálenosti kamer.

Práce pokračuje kapitolou, která se zabývá řešením detekce objektů a následným spojením detekce objektů s určením vzdálenosti. Je zde popsána volba platformy pro detekci objektů a datového souboru pro trénování neuronové sítě. Následuje shrnutí procesu trénování dvou zvolených neuronových sítí v rámci platformy YOLOv5, a to s využitím datového souboru

BDD100K (Berkeley Deep Drive 100 000), včetně diskuze výsledků trénování. Dále je popsán proces implementace algoritmu pro určení vzdálenosti z předchozí kapitoly do Python kódu pro detekci objektů v rámci platformy YOLOv5. Výsledkem je doplnění informace o vzdálenosti detekovaných objektů. Kapitola je zakončena zprovozněním platformy YOLOv5 na zařízení Jetson Nano a kontrolním spuštěním vytvořeného algoritmu.

Poslední kapitola této diplomové práce je pak věnována ověření funkčnosti jednotlivých částí vytvořeného algoritmu. Nejprve je provedeno posouzení rozsahu a přesnosti samotného určení vzdálenosti. Následně je ověřena funkčnost ve spojení s detekcí objektů, kde je sledováno správné ohraničení objektů rámečky, přiřazení tříd a také výpis určené vzdálenosti ve zpracovaném obrazu. Také je vyhodnocena výpočetní náročnost jednotlivých kroků ve výsledném algoritmu a je posouzena schopnost provozu v reálném čase na zmíněném zařízení Jetson Nano.

1 ZPRACOVÁNÍ OBRAZU, NEURONOVÉ SÍTĚ A AUTONOMNÍ ŘÍZENÍ

Pro pochopení problematiky autonomního řízení je nutné se nejprve seznámit s koncepty a procesy, které funkci autonomních vozidel umožňují. To zahrnuje v první řadě obecně problematiku zpracování obrazu, dále pak počítačové vidění založené na neuronových sítích a následně určení vzdálenosti objektu zejména v návaznosti na zpracování obrazu ze stereo kamery.

1.1 ZPRACOVÁNÍ OBRAZU

S počítačovým viděním je velmi úzce spjata problematika zpracování obrazu. Obzvláště relevantní k tématu počítačového vidění je pojem konvoluce, a to v souvislosti s konvolučními neuronovými sítěmi, které budou objasněny v nadcházejících kapitolách. Konvoluce spočívá ve stanovení funkce *h* (konvoluční jádro – kernel) a následné postupné integraci součinu funkce *f* s otočenou hodnotou funkce *h*. Pro zpracování obrazu platí, že funkce *f* představuje zpracovávaný obrazový materiál a funkci konvolučního jádra *h* můžeme vnímat jako filtr, kterým je obrázek přetvářen. [1]

Další oblast zpracování obrazu, která nachází uplatnění v počítačovém vidění, je segmentace obrazu. Jedná se o proces, při kterém je pozorovaný obrázek rozdělen na dílčí oblasti potažmo objekty a na který navazuje další zpracování, jako je například extrakce příznaků (angl. Feature extraction) nebo klasifikace. Při segmentaci obrazu se uplatňují metody dvou základních typů. První z nich je založen na detekci hranic jednotlivých oblastí, jsou tedy vyhodnocovány výrazné změny mezi skupinami pixelů. Druhý typ metod segmentace naopak využívá k vytyčení segmentu vyhodnocování podobnosti jednotlivých pixelů. Podobnost či odlišnost pixelů může být vnímaná podle několika základních kritérií, kterými jsou barva a textura. U sekvencí složených z více snímků pak může být velmi nápomocný pohyb. Základní metodou segmentace je prahování intenzity, kdy je porovnávána intenzita jednotlivých pixelů vůči stanovené prahové hodnotě. Pokročilejší techniky pak zahrnují například Cannyho hranový detektor, metodu Watershed (vodní předěl) atd. [1]

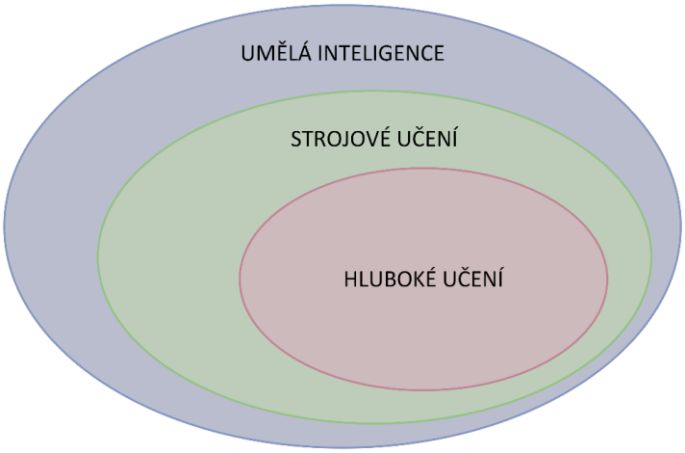
1.2 UMĚLÁ INTELIGENCE

Umělá inteligence je velmi obecný pojem, který zahrnuje množství rozmanitých úloh. Obecně lze definovat jako výsledek snahy o automatizaci procesů vykonávaných člověkem, které vyžadují kritické myšlení. Představy o tom, jak by měla umělá inteligence fungovat, se vyvíjejí už od 50. let minulého století, kdy se tento pojem zrodil. Na začátcích výzkumu v této oblasti stál koncept, který je označován jako symbolická umělá inteligence. Jednalo se o systémy, které činily rozhodnutí na základě pevně definovaných pravidel s využitím informací uložených v databázi. Vrcholem symbolické umělé inteligence byly tzv. expertní systémy, které se těšily velké popularitě v 80. letech minulého století. Tyto systémy byly poměrně snadno aplikovatelné na úlohy s přesně vytyčenými pravidly, jako je například hraní šachů. Oproti tomu s pomocí nich bylo takřka nemožné realizovat abstraktnější procesy, například klasifikaci obrázků. Tento nedostatek symbolické umělé inteligence byl motivací k širší aplikaci principu strojového učení. [2]

1.2.1 STROJOVÉ UČENÍ A HLUBOKÉ UČENÍ

Příchod myšlenky strojového učení přinesl zcela nový pohled na způsob, jakým pracují počítačové programy. Při klasickém programování, tedy i u dřívější symbolické AI (Artificial

Intelligence), jsou v kódu definovány pravidla, pomocí kterých jsou vstupní data převedena na patřičné odpovědi. Oproti tomu v případě strojového učení jsou systému k vstupním datům poskytnuty příklady očekávaného výstupu. Systém je schopen se učit, tedy definovat korelaci mezi vstupními daty a odpověďmi. Naučený vztah mezi vstupem a výstupem je pak možné aplikovat na nová data a získat na ně odpověď. Strojové učení je od 90. let minulého století dominantním směrem umělé inteligence, což je dáno především prudkým nárůstem výpočetní kapacity počítačů. [2]



Obr. 1 Vymezení podoblastí umělé inteligence

Základním mechanismem strojového učení je transformace vstupních dat na reprezentace, které se více podobají očekávanému výstupu. Jedná se o iterativní proces, kdy systém získává zpětnou vazbu ohledně přesnosti získaných reprezentací. Specifickou skupinou oblastí strojového učení je pak hluboké učení (obr. 1), které představuje spojení více postupných vrstev reprezentací. Tyto vrstvené reprezentace jsou trénovány s využitím neuronových sítí ve více postupných fázích (vrstvách) během jedné iterace. Neuronová síť každé vrstvy je reprezentována váhami, které popisují operace vykonávané danou vrstvou na datech. Cílem hlubokého učení je stanovit hodnoty vah tak, aby se výsledný výstup shodoval s definovaným očekáváním. K tomu je nutné sledovat míru odlišnosti skutečného výstupu od požadovaného, k čemuž slouží hodnota ztráty určená ztrátovou funkcí. Na základě toho získává systém zpětnou vazbu, je schopen s využitím optimalizátoru upravit hodnoty vah a celý cyklus se opakuje, dokud není dosaženo minimální hodnoty ztráty. [2]

1.2.2 TYPY STROJOVÉHO UČENÍ

Metod strojového učení existuje celá řada, všechny však lze rozdělit do čtyř základních kategorií: řízené (*supervised*), neřízené (*unsupervised*), samořízené (*self-supervised*) a zpětnovazební (*reinforcement learning*). [3]

ŘÍZENÉ UČENÍ

Převážná většina problémů strojového učení spadá do kategorie řízeného učení, též označovaného jako učení pod dohledem [4]. To spočívá v trénování přiřazení stanovených cílů (neboli anotací) na vstupní data podle příkladového souboru dat s přiřazenými anotacemi. [3] Učení je tedy v tomto případě korigováno přítomností tohoto příkladového datového souboru, kdy získané řešení je porovnáno se vzorem a pokud nastane neshoda, parametry jsou upraveny a proces se opakuje. [5] Typickou aplikací tohoto typu strojového učení jsou úlohy

klasifikace a regrese, dále například detekce objektů a segmentace obrazu [3], obecně je s výjimkou samořízeného učení uplatnitelná, pokud známe očekávanou podobu výstupu [4].

NEŘÍZENÉ UČENÍ

Tato oblast strojového učení (známá také pod pojmem učení bez dohledu [4]) si klade za cíl transformovat vstupní data bez dodání cílů tak, aby byla usnadněna další práce s těmito daty. Neřízené učení se uplatňuje například při vizualizaci dat, kompresi a odstraňování šumu. Je možné ho také použít pro přesnější interpretaci datového souboru před jeho použitím v řízeném učení. [3]

SAMOŘÍZENÉ UČENÍ

Ačkoliv je tento typ podoblastí řízeného učení, výrazně se odlišuje ve způsobu přípravy trénovacího datového souboru. Tvorba anotací vstupních dat je zde totiž automatizována většinou s využitím heuristického algoritmu. [3]

POSILOVANÉ UČENÍ

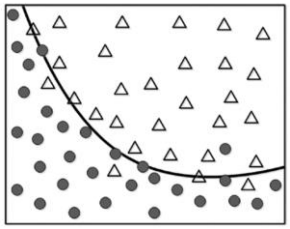
Posilované neboli zpětnovazební učení je založeno na vyhodnocování správnosti předchozích rozhodnutí, na základě toho algoritmus činí rozhodnutí v budoucnu. [4] Tento princip je realizován zavedením tzv. agenta, který sleduje své prostředí a zajišťuje svými rozhodnutími maximalizaci nějaké odměny. [3] Tato metoda byla zatím úspěšně aplikována na hraní her, v roce 2015 porazil počítačový program AlphaGo společnosti DeepMind profesionálního hráče deskové hry Go, který vyjádřil velké překvapení z kreativity tahů zahraných programem. Program AlphaGo byl následně dále rozvíjen a verze AlphaGo Zero byla trénována čistě zpětnovazebně tak, že systém hrál sám proti sobě. Tím byla prokázána použitelnost přístupu s čistě zpětnovazebním učením, přičemž bylo dosaženo lepších výsledků než u předchozích přístupů. Jediným negativem přitom byla o několik hodin delší doba trénování. [6] V reálném světě má zpětnovazební učení velký potenciál pro aplikaci u automatizovaných systémů, jako jsou například autonomní vozidla. [3]

1.2.3 APLIKACE ŘÍZENÉHO UČENÍ

Nejčastější úlohy, u kterých se aplikuje princip řízeného učení, jsou klasifikace a regrese. [5]

KLASIFIKACE

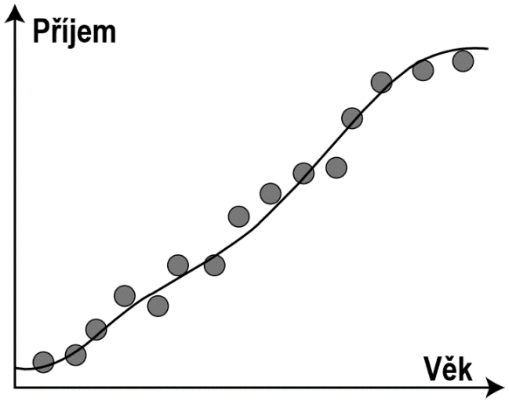
Klasifikace je jedna z nejrozšířenějších oblastí strojového učení vůbec. Spočívá v přiřazení třídy k vstupním datům, jak je pro případ dvou tříd znázorněno na obr. 2. [5] Klasifikace se dělí na binární a na klasifikaci do více tříd. V případě binární klasifikace je rozlišováno mezi dvěma stavy. U druhého typu je uvažováno více tříd a dále se rozlišuje na klasifikaci s jedním označením třídy a s více označeními tříd, u které může jeden datový bod náležet více třídám současně. [3]



Obr. 2 Ilustrace klasifikace dat do dvou tříd [5]

REGRESE

U regresních problémů není cílem stanovit třídu dat, ale odhadnout určitou hodnotu na základě známých dat. Například na obr. 3 je možné vidět vstupní datovou množinu věku a příjmu, na základě které model stanoví závislost příjmu na věku. [5] V případě regrese je tedy cílem stanovit spojitou hodnotu (resp. závislost), oproti klasifikaci, která spočívá v určení diskrétní hodnoty – třídy. Častou aplikací regresních úloh je predikce dalšího vývoje na základě doposud známých dat. [3]



Obr. 3 Příklad regrese [5]

1.3 NEURONOVÉ SÍTĚ

Neuronová síť je specifický model využívaný u strojového učení, jehož důležitost narostla v souvislosti s rozvojem hlubokého učení v poslední dekádě. [5] V této kapitole je nejprve popsána historie, následně je vysvětlen princip funkce a způsob učení neuronových sítí, na závěr jsou představeny konkrétní typy, tedy konvoluční a rekurentní neuronová síť.

1.3.1 HISTORIE

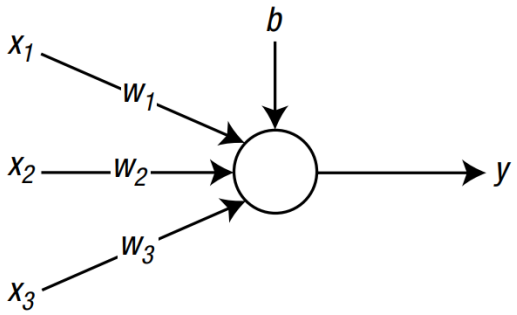
K širšímu uplatňování neuronových sítí došlo v 80. letech minulého století ve spojení s algoritmem zpětného šíření chyby, díky kterému bylo možné sítě efektivně trénovat. Za první praktickou aplikaci neuronové sítě je označováno využití sítě LeNet k strojovému rozpoznávání poštovních směrovacích čísel na zásilkách americkou poštovní službou v 90. letech 20. století. Úspěch nasazení neuronové sítě v praktické aplikaci vzbudil další zájem vědecké komunity o oblast strojového učení. To mělo paradoxně za následek odvedení pozornosti od neuronových sítí, důvodem bylo soustředění na další oblasti strojového učení, jako jsou jádrové metody, rozhodovací stromy a další. Návrat k neuronovým sítím přišel až v roce 2012, kdy došlo k výraznému průlomu díky využití moderních grafických procesorů k trénování hlubokých neuronových sítí, také bylo možné ukládat a zpracovávat větší množství dat, která jsou pro úspěšné natrénování neuronové sítě nezbytná. Od té doby je hluboké učení neuronových sítí konvolučního typu dominantním přístupem pro realizaci počítačového vidění, ale i dalších vjemových úloh. [2]

1.3.2 PRINCIP NEURONOVÉ SÍTĚ

Neuronová síť imituje funkci neuronů v mozku, které samy o sobě neuchovávají informaci, ale konkrétní informace je utvořena propojením určitých neuronů. U neuronových sítí jsou neurony reprezentovány uzly a spojení neuronů představují hodnoty váhy.

UZLY NEURONOVÉ SÍTĚ

Na obr. 4 je znázorněn příklad uzlu neuronové sítě se třemi vstupy x_1, x_2 a x_3. Tyto signály jsou před vstupem do uzlu ovlivněny odpovídajícími váhami w_1, w_2 a w_3. Dále uzel obsahuje hodnotu zkreslení b. [5]



Obr. 4 Uzel neuronové sítě se třemi vstupy [5]

Hodnoty vah a zkreslení jsou hlavními parametry, ve kterých je uložena informace o neuronové síti. S využitím těchto hodnot je získán vážený součet *vs* dle vztahu (1). [5]

$$vs = (w_1 x_1) + (w_2 x_2) + (w_3 x_3) + b \tag{1}$$

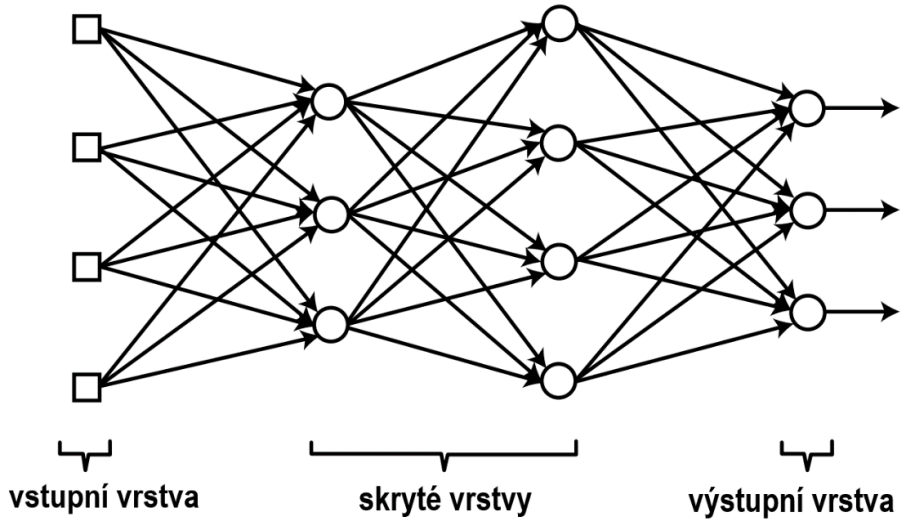
Ze vztahu plyne, že váhy určují, jakou mírou daný signál ovlivňuje výslednou hodnotu, přičemž může nabývat i hodnoty 0 a daný signál je pak od uzlu odpojen. [5]

Hodnota výstupního signálu y je následně určena jako funkční hodnota aktivační funkce φ dle vztahu (2). Aktivační funkce popisuje chování uzlu a existuje jich celá řada. [5]

$$y = \varphi(vs) \tag{2}$$

VRSTVY NEURONOVÉ SÍTĚ

Jak název napovídá, neuronová síť je uskupením velkého množství propojených uzlů, které jsou nejčastěji uspořádány do vrstev. [5]



Obr. 5 Uskupení uzlů uspořádané do vrstev [5]

Na obr. 5 je zobrazeno rozložení uzlů neuronové sítě v jednotlivých vrstvách. V počátcích byly používány tzv. jednovrstvé neuronové sítě, kde byla přímo napojena vstupní vrstva na výstupní. Později byla mezi vstupní a výstupní vrstvu umístěna skrytá vrstva, čímž byla vytvořena mělká vícevrstvá neuronová síť. Dnes se uplatňují převážně hluboké neuronové sítě, které obsahují dvě nebo více skrytých vrstev. [5]

1.3.3 UČENÍ NEURONOVÝCH SÍTÍ

Předpokladem pro úspěšné natrénování je rozdělení dat na trénovací, validační a testovací soubory. Trénovací soubor slouží k samotnému učení, tedy hledání optimálních hodnot vah sítě. Na základě validačních dat jsou upravovány hyperparametry, které představují vlastnosti modelu jako celku (např. počet vrstev a jejich velikost). Závěrečnou fází učení je ověření s využitím testovací množiny. Poměr pro rozdělení se může lišit v závislosti na velikosti celého souboru dat, většinou je ale vymezeno 70 % dat pro trénování a zbylých 30 % rozděleno pro validaci a testování. Díky tomuto rozdělení je model ověřován na jiných datech, než na kterých byl učen, a je tak minimalizováno riziko overfittingu, neboli přeučení. Trénovací soubor je při učení zpracován po předem stanovený počet iterací, které se označují jako epochy. [3]

GRADIENTNÍ SESTUP

Pod pojmem gradientní sestup se skrývají metody optimalizace vah sítě založené na iterativním postupu. Při každém kroku je opakovaně zjištěna derivace funkce chyby, čímž je nalezen směr dalšího kroku k co nejrychlejšímu dosažení optimální hodnoty váhy. Dále je však třeba uvážit velikost tohoto kroku, kterou udává rychlost učení (angl. learning rate). Rychlost učení se také řadí mezi výše zmíněné hyperparametry, jejím správným stanovením je možné výrazně urychlit učení sítě. Krok ale nesmí být příliš velký, aby při učení nedošlo k oscilaci kolem minima chyby. [7]

Nejčastěji uplatňovaným algoritmem optimalizace ve strojovém učení je stochastický gradientní sestup, označovaný zkráceně jako SGD (Stochastic Gradient Descent). Při této metodě je v každém kroku zvolen náhodný prvek z trénovacího souboru dat a pro ten je proveden gradientní sestup. Výsledkem je, že v každém kroku je optimalizována váha pro právě zvolený prvek, čímž jsou nalezena lokální minima daného souboru dat. Díky tomu je možné rychleji dosáhnout globálního minima chyby, které představuje nejmenší hodnota z lokálních minim. [7]

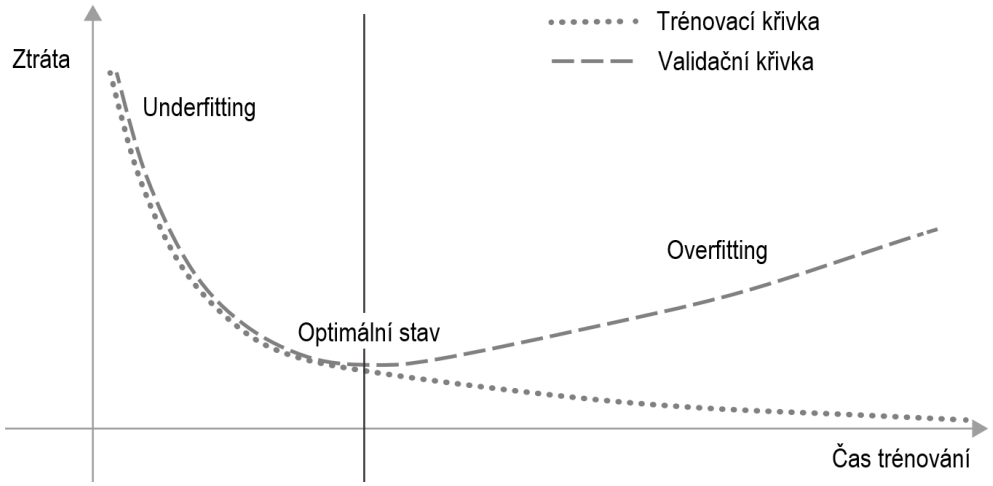
ALGORITMUS ZPĚTNÉHO ŠÍŘENÍ CHYBY

Tato metoda trénování vícevrstvých neuronových sítí byla popsána už v roce 1986 a poprvé stanovila způsob, jakým určit chybu uzlů ve skrytých vrstvách. Tato chyba je určena při zpětném průchodu sítí jako vážený součet hodnot sousední vrstvy směrem k výstupní vrstvě (tj. na obr. 5 vrstva více vpravo). [5][8]

UNDERFITTING A OVERFITTING

V průběhu učení jsou zaznamenávány hodnoty ztráty, tj. odlišnosti skutečného výstupu od očekávaného, pro trénovací a validační data. Snahou je docílit minimální hodnoty pro ztrátové funkce obou souborů dat, nicméně v případě například nevhodného rozdělení těchto souborů může dojít k overfittingu neboli přeučení sítě. Znamená to, že model je příliš specializovaný na data obsažená v trénovacím souboru a není schopen fungovat na neznámých datech. Tento

jev, znázorněn na obr. 6, se projevuje opětovným nárůstem ztráty validačního souboru, zatímco u trénovacího souboru ztráta dále klesá. [2]



Obr. 6 Příklad přeučení modelu [2]

1.4 POČÍTAČOVÉ VIDĚNÍ

Oblast počítačového vidění je považována za první velký úspěch v aplikaci hlubokého učení. Je také jednou z hlavních motivací pro výzkum hlubokého učení obecně, jelikož počítačové vidění představuje významný potenciál pro rozvoj v mnoha odvětvích, jako například zdravotnictví, robotika nebo autonomní vozidla. [2]

1.4.1 ÚLOHY POČÍTAČOVÉHO VIDĚNÍ

V rámci počítačového vidění se realizují nejčastěji tři typy základní typy úloh rozdělené podle podoby výstupu. Jsou to klasifikace, detekce objektů a segmentace obrázků. [2]

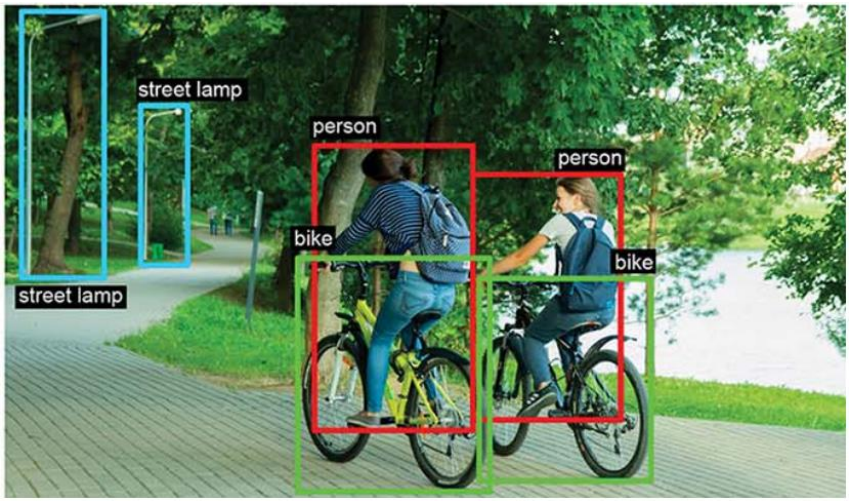
KLASIFIKACE

Samotný snímek v tomto případě zůstává v nezměněné podobě a cílem je klasifikovat jej podle obsahu, tj. přiřadit mu klíčové slovo nebo kategorii. Klasifikace se dále rozlišuje na případ, kdy je obrázek přiřazen z možných variant pouze do jedné kategorie, která obecně shrnuje obsah celého obrázku (obr. 7 vlevo). Druhou variantou je označení více klíčovými slovy, kterými jsou například jednotlivé předměty na obrázku (obr. 7 vpravo). Příkladem může být vyhledávání fotografií pomocí klíčových slov ve službě Google Photos. [2]

Obr. 7 Klasifikace obrázku do jedné (vlevo) nebo do více kategorií (vpravo) [2]

DETEKCE OBJEKTŮ

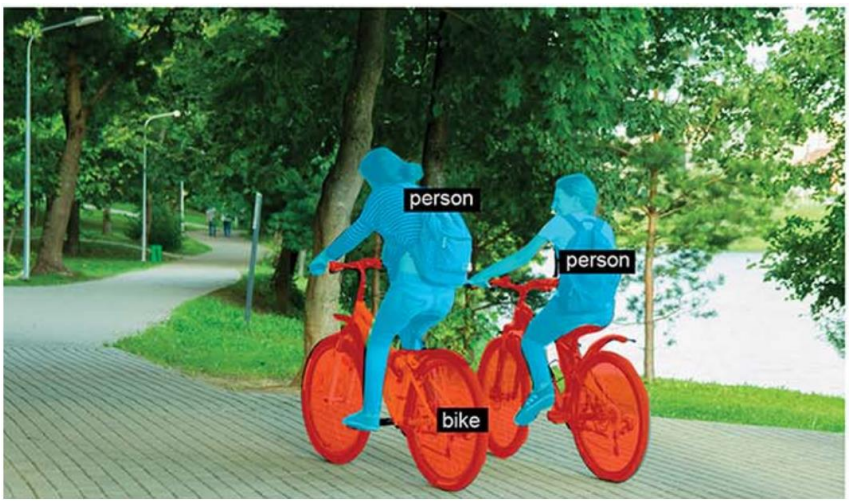
Tento typ je podobný klasifikaci s označováním více klíčovými slovy, přičemž je doplněn o ohraničující rámeček pro každý rozpoznaný objekt. Kromě informace o obsahu obrázku je tedy navíc známo i přesné umístění objektů, jak je zobrazeno na obr. 8. Detekce objektů může být využita u autonomních vozidel k rozpoznání dalších vozidel, chodců, značek apod. [2]



Obr. 8 Detekce objektů na obrázku [2]

SEGMENTACE OBRÁZKU

V případě segmentace jsou v obrázku rozpoznány objekty podobně jako v předchozím případě, ohraničující rámeček je ale nahrazen přesným vyznačením daného objektu na úrovni jednotlivých pixelů, jak je vidno na obr. 9. Tento typ počítačového vidění je aplikován například u funkcí rozostření nebo změny pozadí ve videokonferencích služeb jako Zoom. [2]



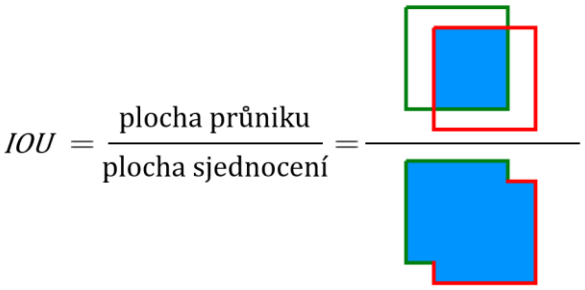
Obr. 9 Segmentace obrázku [2]

1.4.2 METRIKY PRO VYHODNOCENÍ VÝKONNOSTI ALGORITMŮ DETEKCE OBJEKTŮ

Aby bylo možné vyhodnocovat výkonnost jednotlivých algoritmů pro detekci objektů a porovnávat je mezi sebou, byly zavedeny různé veličiny. Ty nejčastěji uplatňované jsou popsány v této kapitole.

Základem pro srovnání algoritmů detekce objektů je poměr označovaný jako *IOU* (Intersection Over Union). Jedná se o poměr plochy průniku ohraničujícího rámečku určeného algoritmem B_p a ideálního rámečku B_{gt} ku ploše sjednocení těchto dvou rámečků, jak je popsáno ve vztahu (3) a znázorněno na obr. 10. [9]

$$IOU = \frac{plocha(B_p \cap B_{gt})}{plocha(B_p \cup B_{gt})} \tag{3}$$



Obr. 10 Poměr *IOU* [9]

Porovnáním hodnoty *IOU* pro konkrétní rozpoznaný objekt se stanovenou hraniční hodnotou *t* pak je detekce vyhodnocena jako správná ($IOU \geq t$) nebo nesprávná ($IOU < t$). [9]

Informace o počtu správných detekcí (*TP*, true positive) spolu s počtem nesprávných detekcí (*FP*, false positive) a počtem nerozpoznaných objektů (*FN*, false negative) jsou vstupem pro stanovení metrik precision (*P*), definované ve vztahu (4) a recall (*R*), popsané vztahem (5). [9]

$$P = \frac{TP}{TP + FP} = \frac{TP}{všechny\ detekce} \tag{4}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{všechny\ objekty} \tag{5}$$

Z výše uvedených vztahů lze odvodit, že hodnota *P* charakterizuje schopnost algoritmu detekovat pouze relevantní objekty, tj. čím blíže je hodnota $P = 1$, tím správněji model vykonává rozpoznávání objektů. Hodnota *R* pak vyjadřuje náchylnost algoritmu k „přehlédnutí“ objektu – čím blíže je hodnota $R = 1$, tím je tato náchylnost nižší. [9]

Závislost dvou výše zmíněných veličin, označovaná jako PR křivka, je výchozí metrikou pro stanovení hodnoty průměrné správnosti AP (average precision) modelu, kdy hodnota AP je rovna ploše pod PR křivkou. Je třeba doplnit, že PR křivka i AP hodnota se stanovuje jednotlivě pro všechny třídy. Proto je určen průměr hodnoty AP přes všechny třídy daného modelu. Tím je získána průměrná hodnota mAP (mean average precision), která je nejčastěji používána pro porovnávání výkonnosti modelů pro detekci objektů. I zde ale nastávají odlišnosti, jelikož dosažená hodnota mAP se liší v závislosti na určené hraniční hodnotě správné detekce *t*. Hodnoty AP, resp. mAP, tedy musí být doplněny o informaci, při jaké hodnotě *t* byly stanoveny, označení je pak například *AP@50:5:95* (průměr hodnot AP pro *t = 50; 55; ...; 95*). [9]

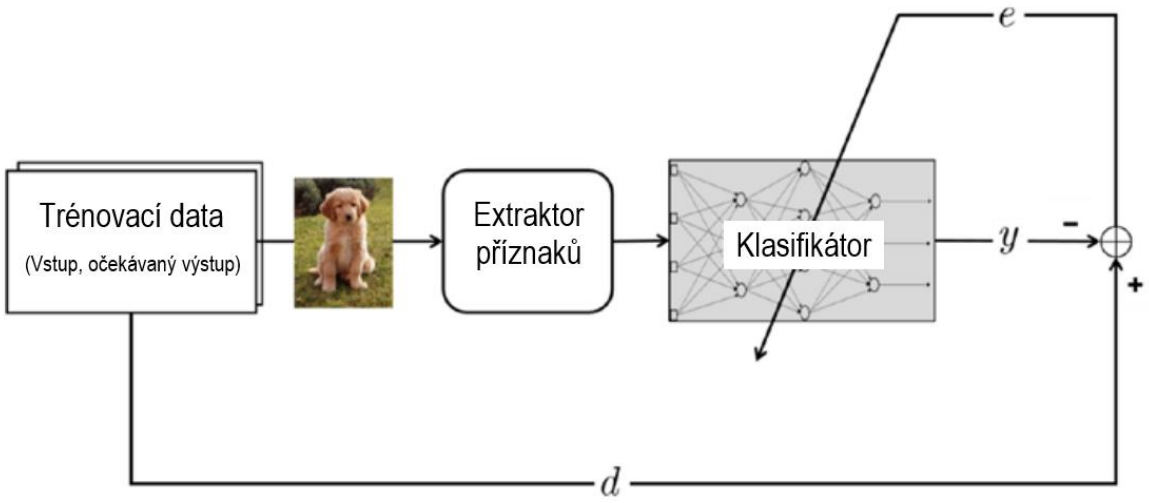
Pro charakterizaci přesnosti detekce daného modelu se tedy používá hodnota mAP, dále je však nutno brát v potaz i rychlost zpracování snímků. Ta bývá nejčastěji uváděna v podobě počtu zpracovaných snímků za sekundu FPS (frames per second). [7]

Výsledky vyhodnocení modelů pro detekci objektů jsou závislé na datasetu, na kterém byl daný model učen. V praxi se proto k jejich porovnávání používají standardizované metody vycházející ze soutěží, které byly vytvořeny k motivaci vývoje detekce objektů. Nejčastěji jsou modely vyhodnocovány na soutěžích PASCAL VOC (Visual Object Classes) nebo COCO (Common Objects in Context). První z uvedených v poslední verzi z roku 2012 zahrnuje 20 tříd objektů. Druhá jich obsahuje 80 a klade tak na testované modely vyšší nároky, proto je dnes používána častěji právě soutěž COCO. [9]

1.4.3 KONVOLUČNÍ NEURONOVÉ SÍTĚ

Tento typ neuronových sítí v současnosti dominuje při řešení úloh počítačového vidění. Obecně u úloh počítačového vidění je nutné snímky nejprve zpracovat, aby došlo k extrakci příznaků. Pokud by snímky byly použity bez tohoto zpracování, byly by výsledky detekce obrazu a dalších úloh velmi nepřesné.

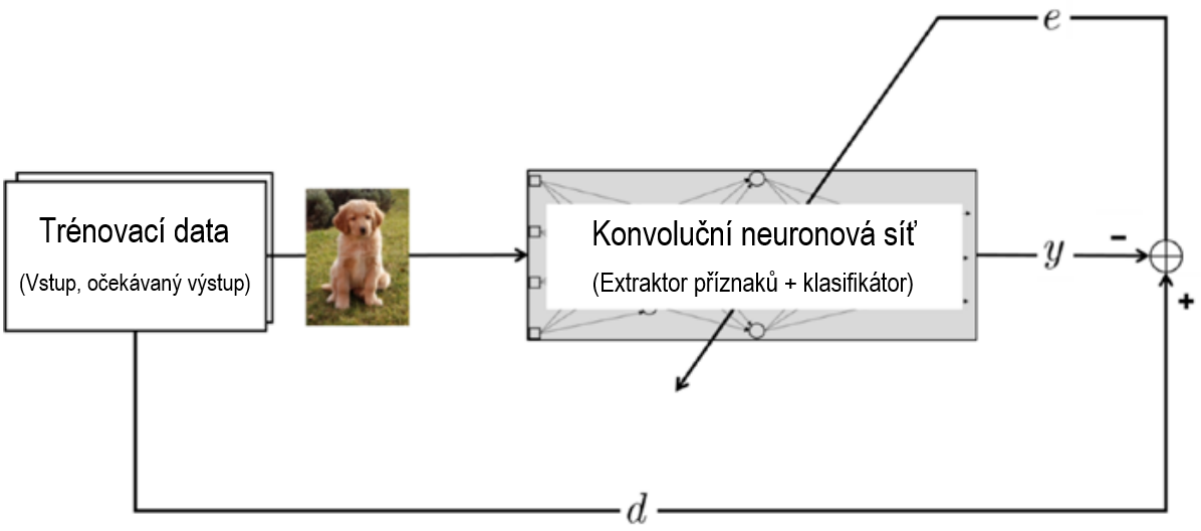
Dříve byly extraktory příznaků vytvářeny manuálně odborníky daných oblastí, což vedlo k vysoké časové náročnosti a nestálým výsledkům. Snímek byl nejprve zpracován tímto extraktorem příznaků, následně vstoupil již zpracovaný do neuronové sítě klasifikátoru, jak ilustruje obr. 11.



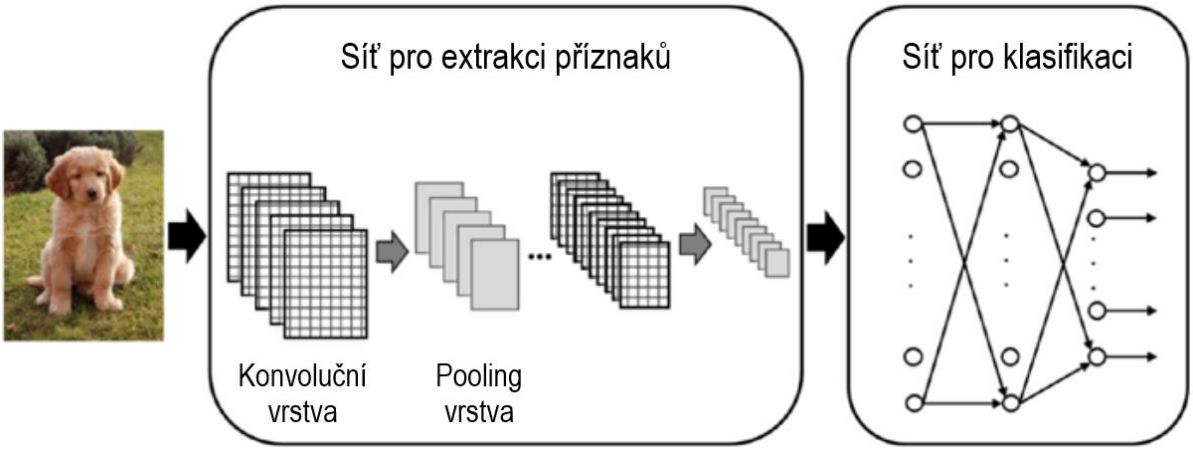
Obr. 11 Ilustrace procesu klasifikace obrázků v minulosti [5]

Moderní přístup tento proces automatizuje právě s využitím konvolučních neuronových sítí. Extrakce příznaků je tak již zahrnuta spolu s klasifikací v neuronové síti (obr. 12) a hodnoty pro zpracování obrázků jsou získávány učením stejně jako hodnoty pro klasifikaci. [5]

Typicky je pak architektura sítě rozdělena na síť pro extrakci příznaků a na síť klasifikátoru, jak popisuje obr. 13. [5]



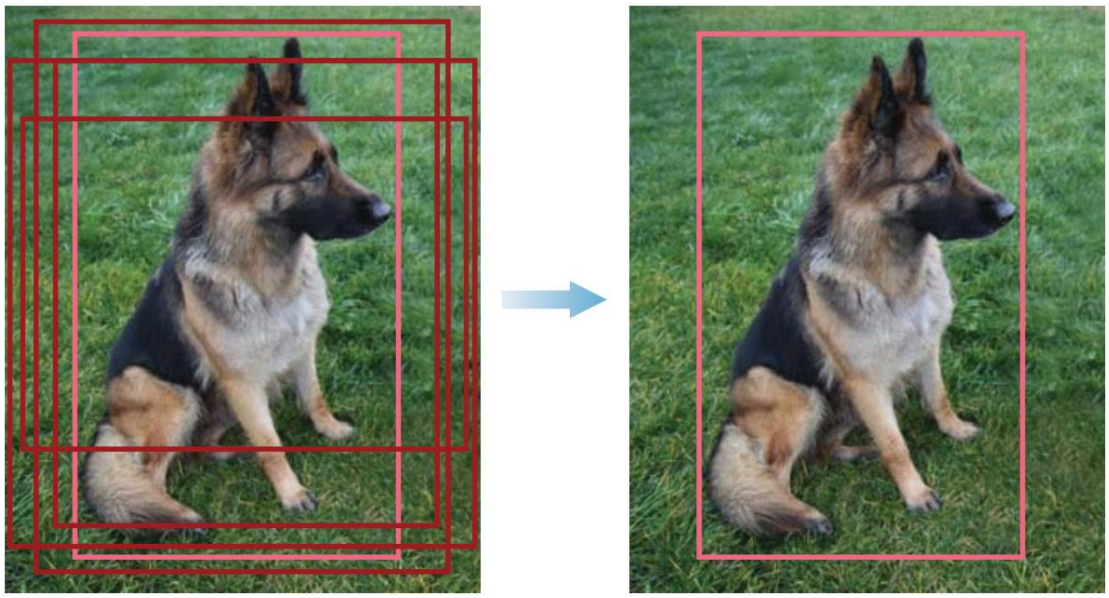
Obr. 12 Ilustrace procesu klasifikace obrázků; v současnosti [5]



Obr. 13 Typická architektura konvoluční neuronové sítě [5]

V extraktoru příznaků jsou zahrnuty konvoluční a pooling vrstvy, které jsou na sebe napojeny střídavě. První typ vrstvy provádí operaci konvoluce, definovanou v kapitole 1.1. V tomto případě si lze konvoluční vrstvu představit jako filtr, který vstupní snímek přetvoří a vytvoří tak mapu příznaků. Na tu je pak aplikována aktivační funkce, kdy nejčastěji je používán typ označovaný jako ReLU (Rectified Linear Unit). Tato funkce zpracuje mapu příznaků jako vstup do pooling vrstvy. Cílem pooling vrstvy je pak zmenšit velikost snímku spojením sousedících pixelů. Nejčastěji je uplatňován způsob max pooling, kdy je snímek rozdělen na podoblasti o dané velikosti a z každé této podoblasti je vybrána maximální hodnota. Snímek zpracovaný posloupností konvolučních a pooling vrstev pak může být zpracován sítí pro klasifikaci. [5]

Výstupem ze sítě pro klasifikaci je pak informace o ohraničujícím rámečku a třídě objektu pro každý zpracovaný snímek. Je však pravděpodobné, že pro stejný objekt došlo k umístění více ohraničujících rámečků. K eliminaci tohoto jevu slouží technika NMS (non-max suppression), která vybere rámeček s nejvyšší pravděpodobností predikce a odstraní ty zbývající. Tento proces je ilustrován na obr. 14. [7]



Obr. 14 Aplikace techniky NMS [7]

Zásadní dosud nezmíněnou charakteristikou konvolučních neuronových sítí je skutečnost, že neuchovávají žádné informace v paměti a zpracovávají každý snímek nezávisle, neberou tedy v potaz možný kontext mezi více snímky. Jedinou cestou je spojení souvisejících datových bodů do jednoho a následně jeho zpracování. Alternativou je použití rekurentní neuronové sítě. [2]

1.4.4 REKURENTNÍ NEURONOVÉ SÍTĚ

Tento typ neuronové sítě umožňuje zpracovat určitou sekvenci dat cyklicky, kdy je do architektury sítě zavedena zpětná vazba z jejího výstupu. Není tak třeba spojovat tato data do jediného bodu, jsou zpracovávána v rámci smyčky a je mezi nimi zachován kontext. Takový přístup je vhodný pro aplikaci na datech, u kterých záleží na jejich pořadí, jako jsou například závislosti veličin v čase. [2]

1.5 PŘÍKLADY ARCHITEKTUR NEURONOVÝCH SÍTÍ PRO DETEKCI OBJEKTŮ

Tato kapitola poskytuje shrnutí často používaných architektur neuronových sítí konkrétně v oblasti detekce objektů. Ve všech případech se jedná o celé softwarové balíky, které v sobě zahrnují všechny nutné části procesu detekce objektů, od vytipování oblastí zájmu na snímcích přes predikci objektů a aplikaci NMS až po vyhodnocení výkonnosti pomocí výše zmíněných metrik. [7]

1.5.1 R-CNN

Architektura R-CNN (region-based convolutional neural network) se roku 2014 stala jedním z prvních případů úspěšné aplikace konvolučních neuronových sítí na detekci objektů. Princip spočívá ve vytipování oblastí na snímku, které by mohly představovat určitý objekt, před jeho vstupem do konvoluční sítě pro extrakci příznaků. Konvoluce tak při extrakci příznaků není prováděna na celém obrázku, ale pouze na těchto vybraných oblastech zájmu. Po extrakci příznaků následuje klasifikace a stanovení ohraničujícího rámečku. V podstatě je v této metodě úloha detekce objektů převedena na úlohu klasifikace jednotlivých podoblastí zpracovávaného obrázku. [7]

Tato metoda je velmi výpočetně náročná, protože v průměru je na každý snímek vybráno zhruba 2000 podoblastí, z nichž každá následně musí být zpracována výše popsaným řetězcem. Současně je komplikované i trénování sítě. Architektura R-CNN je totiž složena ze samostatných modulů pro extrakci příznaků, klasifikaci a stanovení ohraničujících rámečků. Ve snaze zrychlit detekci objektů vznikly modifikace Fast R-CNN a Faster R-CNN. [7] Stále však architektura Faster R-CNN dosahovala maximální rychlosti 5 FPS při přesnosti 73,2 % mAP (PASCAL VOC 2007) [10] a nebylo tak možné docílit detekce objektů v reálném čase. Stěžejní překážkou byla základní myšlenka rozdělení detekce do více fází – navržení oblastí potenciálních ohraničujících rámečků, kterým je přiřazeno skóre *objectness* (tj. pravděpodobnost, že se v této oblasti nachází nějaký objekt) a následná detekce objektů, při které je vyhodnocena pravděpodobnost konkrétní třídy. Obecně se takto koncipované architektury řadí do kategorie multi-stage detektorů. Oproti nim se rozlišují single-stage detektory, které se vyznačují mnohem vyšší rychlostí, avšak za cenu nižší přesnosti. [7]

1.5.2 SSD

Neuronová síť SSD (single-shot detector) byla představena v roce 2016, kdy pokořila rekordy jak v rychlosti, tak i přesnosti (až 59 FPS při 74,3 % mAP pro PASCAL VOC 2007 [11]). Oproti výše zmíněným architekturám typu R-CNN se jedná o single-stage detektor. To znamená, že při jednom průchodu sítí je současně vyhodnocována hodnota *objectness* i pravděpodobnost třídy objektu. Architektura je složena ze základní sítě, která je doplněna o navazující konvoluční vrstvy určené pro detekci objektů různých velikostí. Pro vyfiltrování optimálních rámečků je poté podobně jako u R-CNN použita technika NMS. [7]

1.5.3 YOLO

Architektura YOLO (You Only Look Once) byla představena ve verzi YOLOv1 roku 2015 a znamenala zásadní posun v rychlosti zpracování snímků. Bylo toho docíleno sjednocením predikce více ohraničujících rámečků a pravděpodobností tříd pro tyto rámečky do jedné konvoluční sítě. Zásadní rozdíl oproti předchozím pokusům je v tom, že detekce objektů, tedy určení souřadnic ohraničujících rámečků i stanovení pravděpodobnosti třídy, je zde vnímána jako jeden regresní problém. Podobně jako u SSD se tak jedná o single-stage detektor, který umožňuje realizovat detekci objektů v reálném čase. První verze dosahovala 45 FPS při přesnosti 63,4 % mAP pro PASCAL VOC 2007, což ve srovnání s SSD nejsou nikterak převratné hodnoty. [12]

Koncem roku 2016 však byla vydána vylepšená verze YOLOv2, která již s rychlostí 67 FPS při 76,8 % mAP na stejném datasetu VOC 2007 překonala i architekturu SSD. [13] Další pokrok přinesla verze YOLOv3 z roku 2018, která je zároveň posledním vydáním původních tvůrců systému YOLO. Výkonnost byla pro tuto variantu vyhodnocena již na datasetu COCO, proto je hodnota mAP 57,9 % zdánlivě nižší, avšak ve srovnání s mAP 50,4 % u architektury SSD na stejném datasetu je vidět významný pokrok. [14]

Obecně je princip architektur typu YOLO založený na rozdělení zpracovávaného obrázku na buňky a následném navržení ohraničujících boxů pro každou z buněk (obr. 15). Výstupem každé této buňky jsou souřadnice ohraničujících rámečků, skóre *objectness* a hodnoty pravděpodobností všech definovaných tříd. Za celý tento proces je zodpovědná jedna ucelená neuronová síť nazvaná DarkNet, jejíž architektura byla navržena přímo tvůrci YOLO. [7] Současně je Darknet také názvem open source frameworku pro práci s neuronovými sítěmi, který byl rovněž vytvořen tvůrci YOLO. [15]

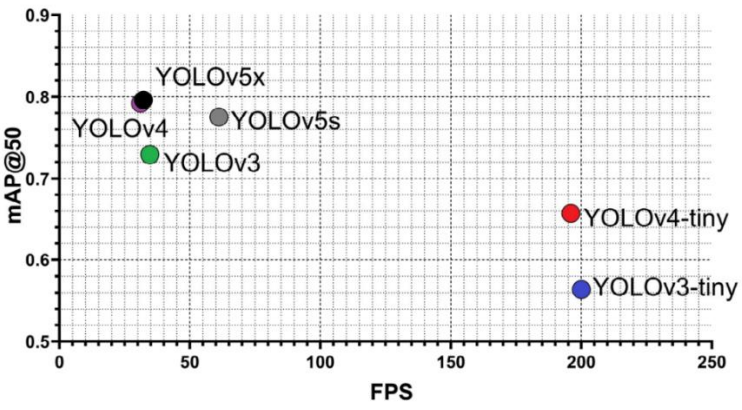
Obr. 15 Princip funkce detektoru YOLO [7]

Významným přínosem verze YOLOv3 bylo rozdělení obrázku na buňky třemi různými měřítky. Tím byla umožněna detekce objektů různých velikostí, což bylo hlavní slabinou předchozích verzí. [7]

Na úspěchy YOLOv3 navázaly další úpravy, které už nebyly prováděny původními tvůrci systému YOLO. Z množství vzniklých variací lze usoudit, že detektory založené na architektuře YOLO se dodnes těší větší popularitě a nacházejí v praxi větší uplatnění než SSD. Novější varianty zahrnují YOLOv4 představené roku 2020, ve které bylo další optimalizací docíleno dalšího zrychlení stále na frameworku Darknet [16], dále pak YOLOv5 vydané téhož roku, které mimo další úpravy těžilo také z přesunu na rozšířenější a více podporovaný framework PyTorch [17].

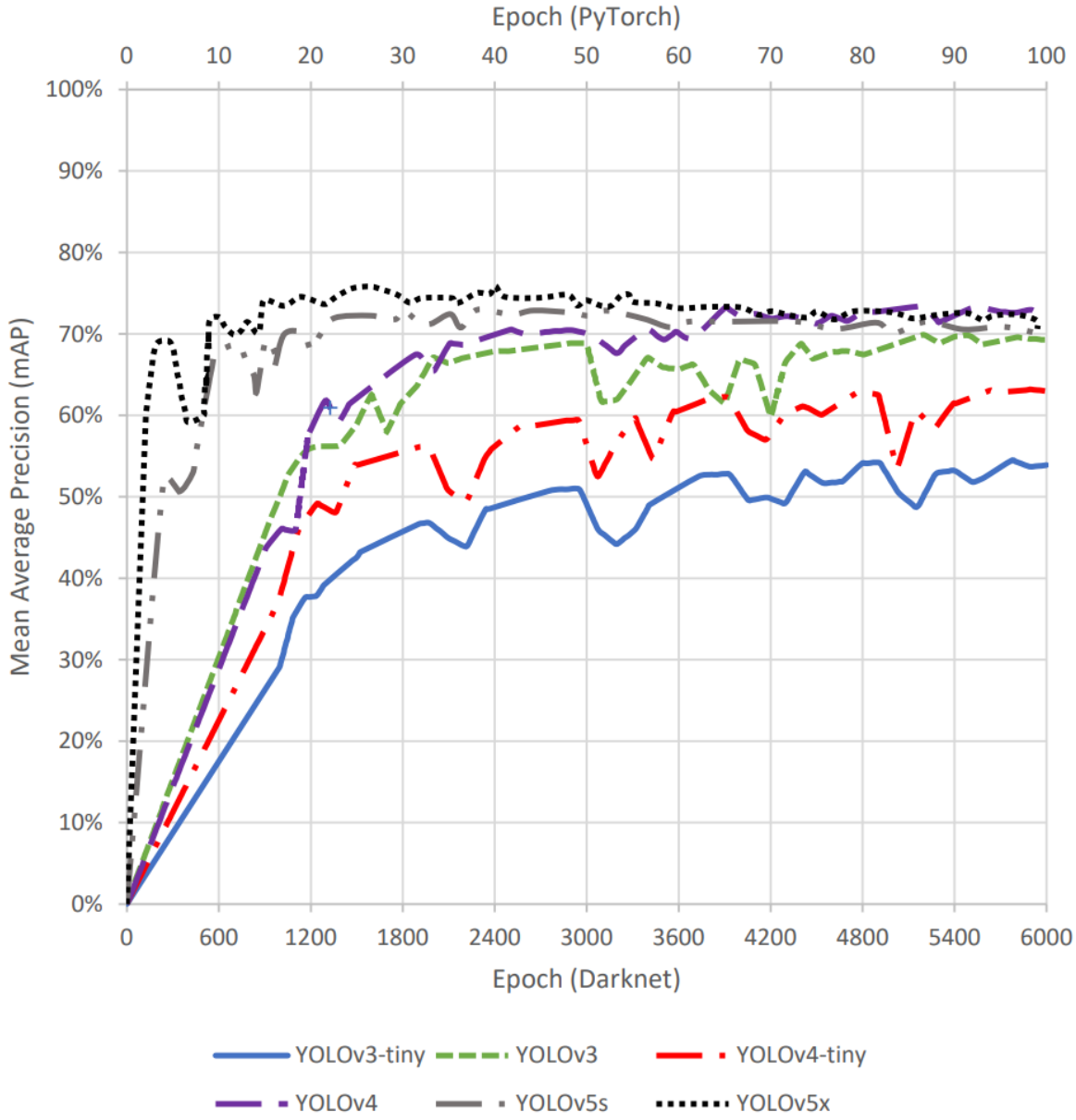
SROVNÁNÍ YOLOV3, YOLOV4 A YOLOV5

Vzhledem k vzniku většího množství variant v poměrně krátkém časovém období se nabízí jejich přímé srovnání k posouzení přesnosti a rychlosti detekce. Na obr. 16 můžeme pozorovat závislost dosažené přesnosti mAP na rychlosti v FPS pro různé varianty jednotlivých verzí detektoru objektů YOLO. Pro každou z verzí YOLOv3, YOLOv4 a YOLOv5 je do srovnání zařazena úplná architektura a její odlehčená varianta určená pro provoz na méně výkonném hardwaru (tzn. YOLOv3-tiny, YOLOv4-tiny a YOLOv5s). Z té je zřejmý dobrý poměr přesnosti a rychlosti detekce obzvláště u YOLOv4-tiny. U YOLOv5s je oproti menším variantám jiných verzí patrný menší pokles přesnosti, avšak současně nedochází k tak výraznému nárůstu FPS. [18]



Obr. 16 Srovnání verzí YOLO v závislosti mAP na FPS [18]

Na obr. 17 je dále zobrazen průběh trénování jednotlivých variant na datasetu zaměřeném na detekci zralých hroznů ve vinici. Ze srovnání je patrné, že modely YOLOv5 dosahují stabilní hodnoty mAP podstatně dříve než modely předchozích verzí a přitom se jedná o nejvyšší dosažené hodnoty. [18]



Obr. 17 Srovnání průběhu trénování tří verzí detektoru YOLO [18]

Podobné výsledky dokládají i další provedené analýzy v jiných oblastech aplikace. Srovnání stejných tří verzí YOLO, avšak pouze jejich plnohodnotných variant, prokázalo nejvyšší dosahovanou přesnost mAP verzí YOLOv5 i při trénování na rozpoznání zóny pro autonomní přistání bezpilotního letounu [19] nebo pro detekci znakové řeči [20].

1.6 Určení vzdálenosti s využitím počítačového vidění

V rámci úlohy snímání vzdálenosti jsou rozlišovány aktivní a pasivní metody. Mezi aktivní metody se řadí měření s využitím snímačů, které vysílají určitý signál a snímají dobu

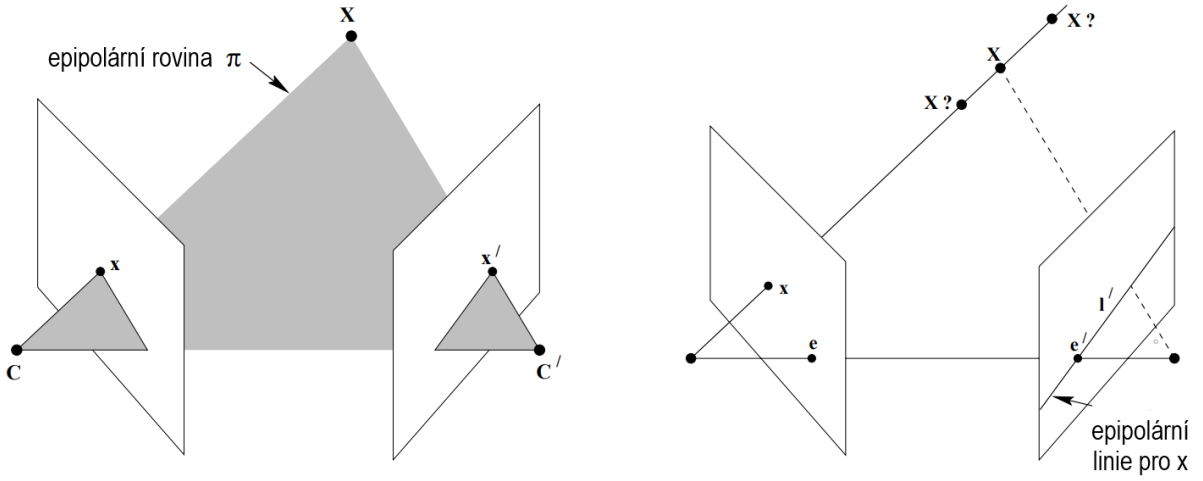
navrácení odrazu tohoto signálu od objektu, jehož vzdálenost je měřena. Typickým příkladem jsou ultrazvukové nebo laserové snímače. Komplikací je u aktivních snímačů správné přiřazení odrazů jednotlivých pulzů, kdy může dojít k záměně odrazu s předchozím nebo následujícím pulzem signálu. Navíc je poměrně omezen rozsah vzdáleností, ve kterém tyto snímače pracují s vysokou přesností, a to zhruba na vzdálenost 1 až 4 metry. [21]

Alternativním přístupem jsou pasivní metody založené na kamerách s využitím počítačového vidění, které vyhodnocují vzdálenost na základě polohy měřeného objektu. V rámci pasivního měření vzdálenosti existují i techniky založené na monoskopickém vidění, tedy s využitím pouze jedné kamery. Například lze využít detekce objektů a následného přiřazení rozměrů daného objektu. Takový systém tak musí obsahovat databázi konkrétních objektů (např. specifických modelů osobních automobilů) a jejich rozměrů. Je zřejmé, že tento přístup vyžaduje ke správné funkci velké množství dat, a přitom je správná funkce možná jen pro objekty v těchto datech obsažené. Tento systém tak může nalézt smysluplné použití pouze v případech, kdy je měřena vzdálenost malého počtu typů objektů s přesně definovanými rozměry. [21]

V rámci pasivních metod nachází širší uplatnění určení vzdálenosti s využitím stereoskopického vidění, tedy s dvěma kamerami. Základním problémem tohoto přístupu je nalezení korespondujících pixelů v obrazech obou kamer. [21]

1.6.1 EPIPOLÁRNÍ GEOMETRIE

Stereoskopické počítačové vidění je založeno na epipolární geometrii, kterou je určen vztah mezi obrazy jednotlivých kamer. Na obr. 18 vlevo je zobrazeno promítnutí bodu zájmu *X* na pixelu *x* v obrazu první kamery a na pixelu *x'* druhé kamery. Epipolární linie (obr. 18 vpravo) představuje projekci spojnice středu kamery *C* a bodu zájmu *X* do druhé kamery a naopak. Korespondující bod je tak v druhém obrazu hledán pouze na této epipolární přímce. [22]



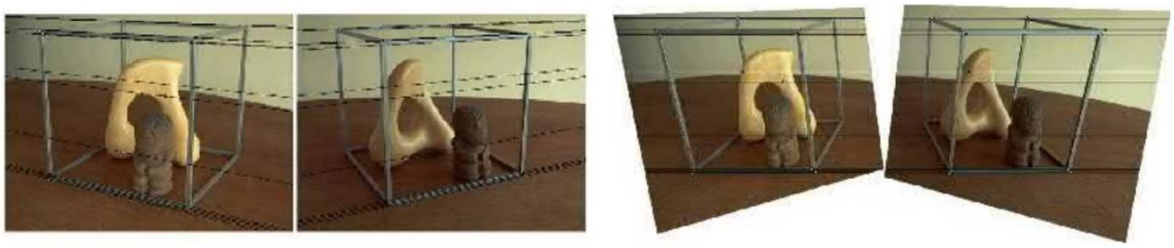
Obr. 18 Epipolární geometrie [22]

Matematickým vyjádřením epipolární geometrie je fundamentální matice *F*. Ze znalosti této matice o rozměru 3x3 je tak možné získat vztah mezi projekcí bodu v jedné kameře *x* a projekcí v druhé kameře *x'* s využitím vztahu (6). [22]

$$x'^{T}Fx = 0 \tag{6}$$

1.6.2 REKTIFIKACE SNÍMKŮ

Rektifikace je proces, při kterém jsou obrazy kamer přetvořeny tak, aby byly epipolární linie obou obrazů rovnoběžné a ve stejné vertikální poloze (obr. 19). Díky tomu je hledání odpovídajících pixelů redukováno na jeden rozměr, kdy jsou porovnávány pouze odpovídající řádky. [23]



Obr. 19 Stereoskopická dvojice snímků před (vlevo) a po rektifikaci (vpravo) [23]

Před samotnou rektifikací je třeba znát kalibrační parametry kamer, a to jak každé kamery samostatně, tak i v jejich vzájemném vztahu. Parametry jednotlivých kamer se označují jako vnitřní (*intrinsic*) parametry a zahrnují ohniskovou vzdálenost kamery *f*, pixelovou polohu počátku souřadného systému v obrazu kamery a popis zkreslení obrazu čočkou kamery. [24]

Vnější parametry (*extrinsic*) pak určují vzájemnou polohu kamer, a to v podobě matice rotace R_k a vektoru translace *T*. [24]

1.6.3 STEREO KORESPONDENCE A DISPARITA

Základní výzvou při zpracování stereo obrazu je stanovení korespondencí pixelů jednoho obrazu v obrazu druhém. Tento proces, označovaný také jako stereo matching, spočívá za zjednodušujících předpokladů v určení horizontálního posunu mezi jednotlivými pixely v obou obrazech. Tento posun, známý pod pojmem disparita, je pak nepřímo úměrný vzdálenosti objektu od dvojice kamer a je tak hlavním parametrem, který je nutné stanovit pro určení vzdálenosti v daném bodě obrazu. Zmíněné zjednodušující předpoklady zahrnují umístění dvojice kamer ve stejné výšce, jejich vzájemnou rovnoběžnost a provedení rektifikace dvojice snímků.

Za dodržení výše uvedených předpokladů je pak vztah mezi disparitou *d* a vzdáleností bodu *Z* umístěného na pixelu (x, y) popsán vztahem (7), kde dále vystupuje ohnisková vzdálenost kamery *f* (v pixelech) a vzdálenost mezi kamerami *B*. [23]

$$d(x,y) = f\frac{B}{Z(x,y)} \tag{7}$$

Z tohoto vztahu už lze za předpokladu znalosti dalších veličin odvodit výraz (8) pro určení vzdálenosti *Z* na konkrétním pixelu (x, y) obrazu.

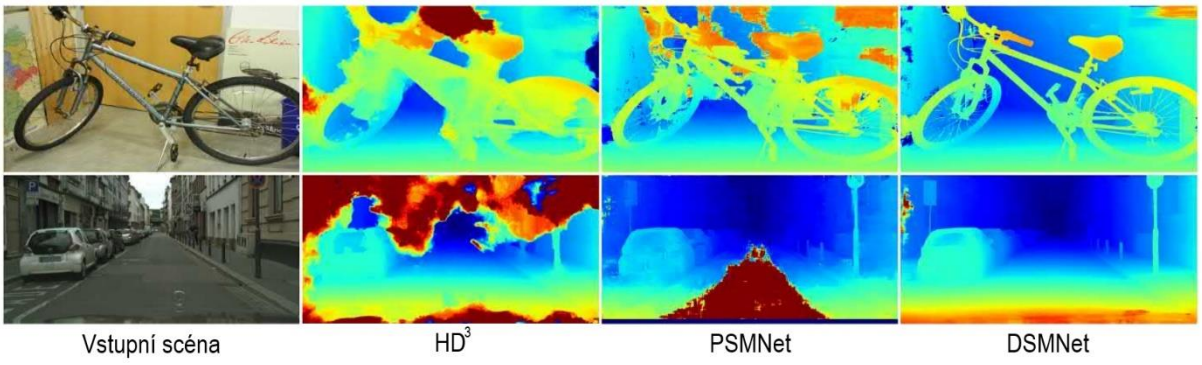
$$Z(x,y) = f\frac{B}{d(x,y)} \tag{8}$$

Samotné určení vzdálenosti je tak redukováno na stanovení disparity mezi dvěma obrazy, přičemž k tomuto účelu byla vytvořena celá řada algoritmů, jejichž základní rozdělení je na řídké a husté hledání korespondencí. Řídké algoritmy slouží ke stanovení disparity pouze na některých místech, dnes jsou většinou uplatňovány algoritmy hustého typu. Ty spočívají ve stanovení disparitní mapy, která obsahuje hodnoty disparity pro každý pixel obrazu (obr. 20). [23]

Obr. 20 Stanovení disparitní mapy ze stereo dvojice snímků [23]

Jednou z nejpoužívanějších metod stanovení disparity je metoda SGM (Semiglobal Matching). Tato metoda vzájemně porovnává pixely obou obrazů, přičemž je brán v potaz i vzájemný rozdíl mezi sousedícími pixely pro eliminaci skokových změn a tím získání hladší disparitní mapy. [25]

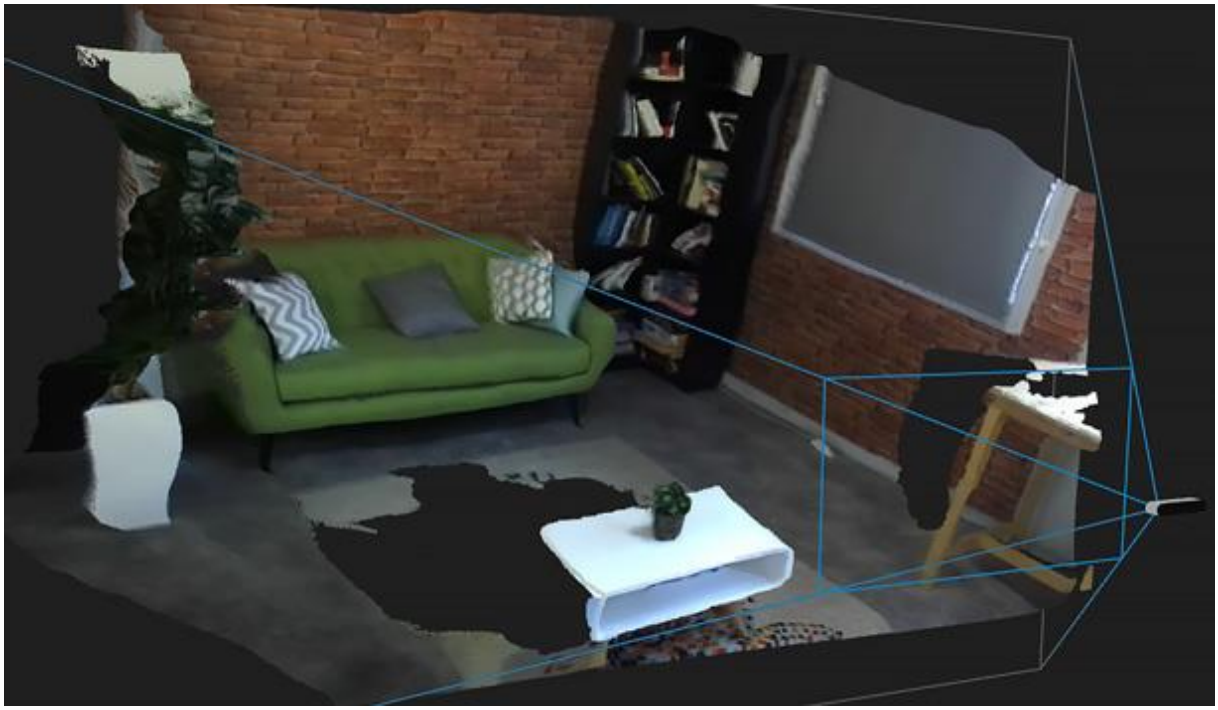
S rozšířením hlubokých neuronových sítí přišla možnost nahrazení klasických hustých algoritmů pro hledání korespondencí end-to-end architekturami neuronových sítí. Ty je možné za předpokladu velkého množství dat natrénovat na odhad disparity, dosud však bylo velmi obtížné dosáhnout dostatečného zobecnění a požadované funkce na odlišných datech než těch, na kterých byla síť trénována. V tomto ohledu bylo dosaženo dostatečného výkonu až v roce 2020 architekturou DSMNet (Domain-invariant Stereo Matching Networks). (obr. 21) [23]



Obr. 21 Srovnání odhadů disparitních map hlubokými neuronovými sítěmi [23]

Alternativním přístupem k odhadu vzdálenosti určitého bodu v obrazu je využití kamer k rekonstrukci trojrozměrného mračna bodů (obr. 22). V tom je každý bod reprezentován souřadnicemi, z nichž je následně pomocí jednoduchého vztahu (9) vycházejícího z teorie Euklidovského prostoru určena vzdálenost bodu od kamery. [26]

$$Z(x,y,z) = \sqrt{x^2 + y^2 + z^2} \tag{9}$$



Obr. 22 Rekonstruované 3D mračno bodů z obrazu stereo kamery [27]

1.7 AUTONOMNÍ ŘÍZENÍ

Prudký rozvoj počítačové techniky v posledních letech se nevyhnutelně promítl i do výbavy moderních vozidel v podobě pokročilých infotainment systémů. Ty jsou čím dál více interaktivní a tím pádem výrazněji zahlcují smyslovou soustavu řidičů a odvádějí tak jejich pozornost. V důsledku tohoto negativního dopadu, ale i v rámci úsilí obecně zvýšit bezpečnost v silničním provozu, se ve výbavě silničních dopravních prostředků začaly uplatňovat aktivní bezpečnostní systémy. Mnohým z těchto systému dalo za vznik uplatnění technologie počítačového vidění, díky čemuž je možné monitorovat jak vnitřní prostor vozidla, tak jeho okolí. Vyvrcholením tohoto úsilí je nástup technologií automatizovaného řízení. [28]

Ačkoliv myšlenka samořídících vozidel se zrodila už v 30. letech 20. století, její realizaci umožnil až technologický pokrok zejména v oblastech bezdrátové komunikace, přesné senzoriky a umělé inteligence. Zpočátku 21. století vzbudily zájem o tuto oblast mimo jiné soutěže Grand Challenge americké agentury DARPA (Defense Advanced Research Projects Agency). Další posun lze přikládat velkým společnostem (Tesla Motors, Google, Waymo, Uber a jiné), které v minulé dekádě začaly nasazovat vozidla s různými úrovněmi autonomie do reálného provozu. Nejen tyto společnosti očekávají v následujících letech další rozvoj podpořený zlepšením v oblasti zpracování obrazu v reálném čase a také příchodem nové generace mobilních sítí 5G a výhledově i 6G. Větší kapacita mobilních sítí má totiž potenciál umožnit hlubší propojení prvků v dopravě, ať už v komunikaci vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), nebo vehicle-to-everything (V2X). [29]

1.7.1 ÚROVNĚ AUTOMATIZACE ŘÍZENÍ

Organizace SAE (Society of Automotive Engineers) stanovila normou J3016 šest úrovní automatizace řízení v závislosti na roli uživatele v průběhu funkce systému. [30]

ÚROVEŇ 0

Do systémů této úrovně lze zařadit aktivní bezpečnostní prvky, které do jisté míry usnadňují, ale nepřebírají zcela žádnou z úloh souvisejících s řízením vozidla. Řízení je tedy plně vykonáváno řidičem. Příkladem mohou být systémy varování před opuštěním jízdního pruhu (Lane Departure Warning – LDW) nebo hlídání mrtvého úhlu (Blind Spot Warning – BSW). [30]

ÚROVEŇ 1

Jedná se o základní úroveň automatizace řízení, kdy systém realizuje činnosti související s pohybem vozidla buď jen v podélném, nebo pouze v příčném směru. Funkce systému je dále možná pouze ve specifických situacích. Řidič pak vykonává zbývající část úloh a je také zodpovědný za dozor činnosti automatizovaného systému. Do této úrovně spadají asistenční systémy, jako je adaptivní tempomat nebo parkovací asistent přebírající pouze řízení. [30]

ÚROVEŇ 2

Systémy této úrovně oproti té předchozí přebírají řízení jak v podélném, tak v příčném směru. Samotná činnost řízení je tedy v určitých situacích plně automatizována, systém však realizuje pouze omezenou detekci objektů a vyhodnocování okolí. Od řidiče je proto stále vyžadována plná pozornost a v případě potřeby i zákrok. Jedná se například o adaptivní tempomat doplněný o udržování jízdního pruhu nebo plně samostatný parkovací asistent. [30] Je třeba zmínit, že do této kategorie spadají i systémy často prezentované jako plně autonomní řízení. Například systémy Autopilot, ale také pokročilejší Full Self Driving (FSD) společnosti Tesla stále vyžadují nepřetržitou pozornost řidiče, řadí se tedy do této kategorie stejně jako systém Super Cruise společnosti Cadillac. [31][32]

ÚROVEŇ 3

Od této kategorie jsou systémy doplněny o pokročilou detekci objektů a vnímání okolí. Jsou tak schopny vykonávat řízení vozidla v širším spektru situací. Od uživatele se tak nepožaduje dohled nad správnou funkcí aktivního systému, musí ale být připraven na vyžádání systému zakročit a převzít řízení. Automatizovaný systém třetí úrovně je totiž schopen rozpoznat svoje funkční limity a uživateli není mimo jejich rozsah umožněna aktivace systému. [30] Systémy na této úrovni se dosud u vozidel dostupných na trhu nevyskytují, širší adopce je očekávána prozatím zejména u luxusních automobilů. V roce 2022 plánují nabízet automobily vybavené touto technologií například společnost BMW [33] a Hyundai pro značku Genesis [34]. Prvním systémem třetí úrovně mezinárodně schváleným pro použití v provozu se stal DRIVE PILOT společnosti Mercedes-Benz. Použití je však u tohoto systému stále omezeno pouze na vybrané úseky německých dálnic a při rychlostech pod 60 km/h. [35]

ÚROVEŇ 4

Z pohledu podpory uživatelem představuje tato úroveň významný skok. Neočekává se totiž žádná nutnost zásahu a v případě aktivovaného systému je oproti nižším úrovním uživatel považován pouze za pasažéra a ne řidiče. Znamená to, že v případě jakékoliv nestandartní situace je systém schopen sám zasáhnout. Pokud je tedy například systém vyhrazen pro automatizaci řízení na dálnicích a ocitne se mimo tuto vyhrazenou oblast funkce a řidič není schopen převzít řízení, systém sám učiní patřičná rozhodnutí k dosažení stavu s minimálním rizikem. Takový stav může představovat například vhodné a bezpečné odstavení vozidla. [30]

ÚROVEŇ 5

Nejpokročilejší úroveň je oproti čtvrté úrovni povýšena o nezávislost na situaci a systém tak není vyhrazený například jen pro jízdu po dálnici nebo po městě. Naopak je schopen provozu za všech podmínek, ve kterých by byl schopen řídit vozidlo běžně zkušený lidský řidič. [30] Vozidla v této kategorii provozuje například společnost Waymo. [36]

1.7.2 VÝHODY A NEVÝHODY AUTONOMNÍCH VOZIDEL

Zmíněný pokrok zejména v oblastech sítí a konektivity má velký dopad na oblast inteligentních systému v dopravě a jejich širší nasazení. Důsledkem toho by mohla být především zvýšená efektivita dopravy a tím snížené emise, dále by odstranění lidského faktoru ze silniční dopravy nepochybně snížilo počet vážných nehod a úmrtí.

Nelze ale přehlížet i nevýhody, které by zahrnovaly především snížení počtu pracovních míst. Dále není možné vyloučit neočekávané chybné chování systému autonomního řízení nebo například náchylnost vůči kybernetickým útokům. [29]

1.7.3 SOUČASNÝ STAV A BUDOUCNOST

Pro realizaci autonomního řízení vozidla je stěžejní správné vnímaní okolí, přičemž se uplatňují dva rozdílné přístupy. První je založený na počítačovém vidění, druhý využívá technologie light detection and ranging (LiDAR) ve spojení s další senzorikou. [36][37]

Přístup počítačového vidění je založený na soustavě kamer, které snímají okolí vozidla a s využitím hlubokého učení je systém schopen vnímat danou situaci v 2D doméně (obr. 23). [36] Tento systém má řadu výhod. Informace jsou získány s vysokým rozlišením, kamery jsou cenově dostupné. Vnímání založené na zraku je výhodné, neboť jsou tomu už silnice uzpůsobeny, systém je tak více univerzální.

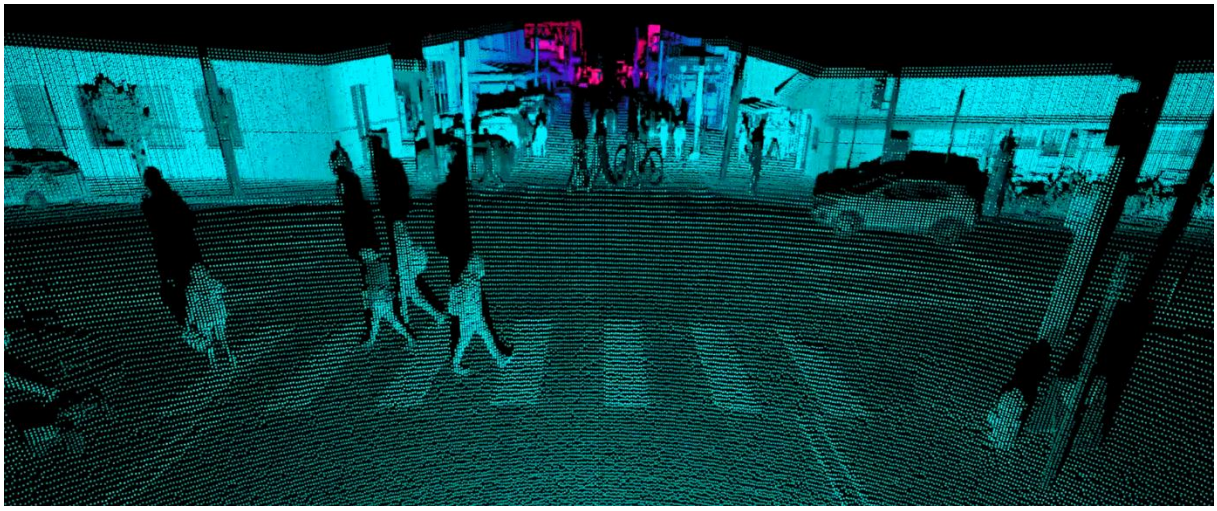
Obr. 23 Vnímání okolí založené na kamerách u vozidla společnosti Tesla [38]

Na druhou stranu může být problematická diagnostika systému založeného na hlubokém učení, které není jednoduché analyticky popsat a porozumět vnitřnímu dění. Hluboké učení také vyžaduje soubor velkého množství dat a pokud dat není dostatek, systém je nepřesný. [37] Rozšiřování datového souboru je však s využitím moderní konektivity poměrně snadné. Například společnost Tesla, která pro technologii AutoPilot používá systém založený primárně na kamerách, využívá pro další učení sběr dat od zákazníků z reálného provozu. [36]

Druhým přístupem je aplikace technologie LiDAR, s jejíž pomocí je okolí vozidla vnímáno prostorově. Výsledkem je mračno bodů v 3D prostoru, jak znázorňuje obr. 24. [36] Tento přístup je aplikován ve spojení s velmi přesnými mapovými podklady a dalšími senzory pro přesnou lokalizaci na těchto podkladech.

Vysoká přesnost je tedy jednou z hlavních výhod, systém je také více předvídatelný a pevně vymezený, naopak ale není možné rozšířit schopnosti trénováním na nových datech. Oproti kamerám jsou LiDAR senzory také podstatně dražší. [37] Nejrozšířenějším příkladem jsou vozidla společnosti Waymo, která s využitím této technologie provozuje plně autonomní vozidla na 5. úrovni. [36]

Širšímu rozšíření LiDAR senzorů brání jejich vysoká cena, proto je snaha využít pro prostorové vnímání stereo kamery. Dále je třeba zmínit, že v praxi se vždy uplatňují v jisté míře obě výše zmíněné metody a jejich syntézou je tak možné částečně vzájemně eliminovat jejich nedostatky. [36]



Obr. 24 Vizualizace vnímání okolí technologií LiDAR [39]

V budoucnu lze očekávat výraznější nástup zpětnovazebního, resp. posilovaného učení, které je lépe uplatnitelné na náhlé situace, kde je nutné rychlé rozhodnutí. Zpětnovazební učení v kombinaci s rekurentními neuronovými sítěmi totiž lépe imituje proces řízení vozidla člověkem. Ten nevychází čistě z analyzování okolí vozidla, ale je také nutné předvídat chování ostatních účastníků provozu a neustále vyhodnocovat situace, které mohou nastat. Je tak možné, že skutečně univerzálně uplatnitelné systémy autonomního řízení 5. úrovně budou založeny právě na tomto principu oproti tomu současnému, který vychází z počítačového vidění. [40]

2 VYUŽITÝ HARDWARE A JEHO KONFIGURACE

2.1 STEREO KAMERA STEREOLABS ZED 2

Jako vstupní zařízení pro získávání obrazu k určení vzdálenosti byla použita stereoskopická kamera ZED 2 od společnosti Stereolabs (obr. 25). Jedná se o zařízení obsahující dva 4MP (megapixel) senzory s roztečí 120 mm, přičemž každý je schopen produkovat obraz o rozlišení až 2208x1242 pixelů při 15 snímcích za vteřinu. Výrobce pro tuto kameru uvádí možnost měření vzdálenosti v rozsahu od 0,3 m do 20 m, kdy do vzdálenosti 3 m zaručuje nepřesnost měření 1 % a do 15 m pak chyba činí nejvýše 5 %. [41]

Obr. 25 Kamera Stereolabs ZED 2 [41]

Výhodou této kamery pro zamýšlenou aplikaci je komunikace prostřednictvím USB (Universal Serial Bus) rozhraní, pomocí něhož je kamera i napájena. Spojení kamery s počítačem tak spočívá v připojení jediného kabelu, načež se zařízení chová jako standartní webkamera, kdy jsou obrazy z levé i pravé kamery promítány v jednom přenosu vedle sebe (obr. 26).

Obr. 26 Nezpracovaný snímek pořízený ze stereo kamery

Pro využití kamery ke zpracování stereo obrazu je přínosná podpora od výrobce, který nabízí ke stažení tovární kalibrační soubor konkrétní kamery podle jejího sériového čísla. Z něj je možné vyčíst přesné hodnoty vnitřních i vnějších parametrů kamery a není tak nutné tyto hodnoty zjišťovat vlastní kalibrací, kdy není zaručena stejná přesnost kalibrace. Kalibrační soubor je ke stažení na adrese http://calib.stereolabs.com/?SN=xxxx, kde za xxxx je třeba dosadit sériové číslo kamery.

Další výhodou může být přítomnost balíku pro vývoj softwaru ZED SDK (Software Development Kit), s jehož pomocí je umožněno zpracování stereo obrazu a získání

požadovaných výstupů, jako jsou bodová mračna a informace o vzdálenosti. Tento softwarový nástroj však vyžaduje pro správnou funkci grafický procesor NVIDIA a nebude v zájmu vyšší kompatibility v rámci této práce využit. [41]

2.2 OVLÁDÁNÍ KAMERY PROSTŘEDNICTVÍM KNIHOVNY OPENCV

Komunikace s kamerou byla realizována v programovacím jazyce Python s využitím knihovny OpenCV zaměřené na zpracování obrazu a počítačové vidění. Pro demonstraci získávání obrazu a základního nastavení kamery byl vytvořen Python skript po vzoru příkladu poskytnutého výrobcem kamery. [42]

Nejprve jsou importovány potřebné knihovny OpenCV, NumPy a configparser.

```
import cv2
import numpy as np
import configparser
```

Dále následuje definice metody *init_calibration*, která zajišťuje načtení kalibračních parametrů kamery z kalibračního souboru s příponou *.conf*. Ten byl pro konkrétní kameru stažen z odkazu uvedeného v předchozí kapitole. Metoda *init_calibration* byla využita z příkladu [42]. Vstupují do ní proměnné *calibration_file*, která obsahuje string s cestou ke kalibračnímu souboru a *image_size*, která je instancí třídy *Resolution* a obsahuje definici šířky a výšky obrazu v pixelech. Metoda vrací proměnné *cameraMatrix_left* a *cameraMatrix_right*, které obsahují kalibrační parametry levé a pravé kamery v podobě matice. Dále jsou vráceny proměnné *map_left_x*, *map_left_y*, *map_right_x* a *map_right_y*. Ty představují mapy pro přetvoření snímků k jejich rektifikaci získané z kalibračních parametrů.

Následuje definice zmíněné třídy *Resolution*.

```
class Resolution:
    width = 1280
    height = 720
```

Poté kód pokračuje definicí hlavní funkce *main*, která obsahuje nejprve vytvoření instance třídy *VideoCapture* z knihovny OpenCV s názvem *cap*. Jestliže je kamera ZED připojena k notebooku s integrovanou webkamerou, může být třeba změnit číslo kamery z 0 na 1. Třídě *VideoCapture* je také možné místo čísla kamery předat jako parametr cestu k video souboru, což lze v tomto programu použít při zpracování předem vytvořeného záznamu z kamery. Následuje ověření úspěšného vytvoření přenosu kamery, kdy podmínkou if je kontrolován stav *isOpened*. Pokud je zjištěna neúspěšná inicializace kamery, program je ukončen.

```
def main():
    cap = cv2.VideoCapture(0)
    if cap.isOpened() == 0:
        exit(-1)
```

Následně je vytvořena proměnná *image_size* dříve definované třídy *Resolution* a jsou jí přiřazeny hodnoty šířky a výšky rozlišení kamery. Tyto hodnoty jsou pak nastaveny vytvořenému přenosu kamery *cap*, čímž je nastaveno rozlišení kamer v tomto případě na 1280x720.

```
    image_size = Resolution()
    image_size.width = 1280
    image_size.height = 720

    cap.set(cv2.CAP_PROP_FRAME_WIDTH, image_size.width*2)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, image_size.height)
```

Dále je do proměnné *calibration_file* načtena cesta ke staženému kalibračnímu souboru kamery a je ověřeno nalezení souboru, načež je volána metoda *init_calibration* pro zjištění veličin nutných k rektifikaci obrazu kamer.

```
    calibration_file = "SNxxxx.conf"
    if calibration_file == "":
        exit(1)
    camera_matrix_left, camera_matrix_right, map_left_x, map_left_y,\
        map_right_x, map_right_y = init_calibration(calibration_file,\
        image_size)
```

Funkce *main* pokračuje nekonečnou smyčkou while, ve které je při každém průchodu uložen snímek do proměnné *frame*, který je následně funkcí *split* rozdělen na levý a pravý obraz. Dále je zobrazen snímek pouze z levé kamery a je provedena rektifikace obrazů kamer pomocí funkce *remap* z knihovny OpenCV. Poté jsou zobrazeny rektifikované snímky z obou kamer a funkcí *waitKey* je vyčkáno po dobu 30 ms. Přitom je sledován stisk libovolné klávesy, kterým dojde k přerušení smyčky a následnému ukončení programu. Kód je zakončen voláním funkce *main*.

```
    while True:
        retval, frame = cap.read()
        left_right_image = np.split(frame, 2, axis=1)
        cv2.imshow("left RAW", left_right_image[0])
        left_rect = cv2.remap(left_right_image[0], map_left_x,
                    map_left_y, interpolation=cv2.INTER_LINEAR)
        right_rect = cv2.remap(left_right_image[1], map_right_x,
                     map_right_y, interpolation=cv2.INTER_LINEAR)

        cv2.imshow("left RECT", left_rect)
        cv2.imshow("right RECT", right_rect)
        if cv2.waitKey(30) >= 0:
            break
    exit(0)

if __name__ == "__main__":
    main()
```

2.3 POČÍTAČ NVIDIA JETSON NANO

Vytvořený algoritmus bude aplikován na zařízení NVIDIA Jetson Nano, které lze vidět na obr. 27. Jedná se o malý počítač vyvinutý společností NVIDIA specificky pro práci s neuronovými sítěmi. Z výbavy je v návaznosti na neuronové sítě stěžejní především 128jádrový grafický procesor s architekturou Maxwell. Dále počítač obsahuje 64bitový procesor se čtyřmi jádry ARM Cortex-A57 a 4 GB operační RAM paměti. Zařízení dále poskytuje rozsáhlé možnosti z hlediska konektivity – nabízí čtveřici USB portů specifikace

3.0, port Ethernet a také je možná komunikace přes sběrnice I^2C, SPI, UART apod. Napájení je možné dvěma způsoby, a to přes micro USB konektor typu B nebo s využitím napájecího DC konektoru. [43] Je třeba zmínit, že počítač Jetson Nano je nabízen i ve variantě s 2 GB RAM paměti [44], pro účely této práce byla použita výše zmíněná 4GB verze.

Obr. 27 Zařízení NVIDIA Jetson Nano [43]

Základní zprovoznění tohoto zařízení představuje v první řadě stažení souboru operačního systému Jetpack určeného pro zařízení Jetson a jeho nahrání na microSD kartu dle příručky [45]. Dále je třeba připravenou kartu vložit do zařízení Jetson Nano, připojit klávesnici, myš, monitor a nejlépe i síťový kabel pro připojení k internetu. Při prvním spuštění je nutné provést prvotní konfiguraci, která zahrnuje nastavení jazyka, rozložení klávesnice, připojení k síti apod.

3 ALGORITMUS PRO URČENÍ VZDÁLENOSTI

Základ zpracování stereo obrazu z kamery vytvořený v předchozí kapitole byl využit k vytvoření programu pro určení vzdálenosti na základě disparity. Vytvořený algoritmus využívá stejné knihovny – OpenCV, NumPy a configparser. Dále následuje definice metody *init_calibration*, kde jedinou odlišností od předchozího programu je doplnění vrácení proměnné *left_cam_fx*, která představuje hodnotu ohniskové vzdálenosti kamery v pixelech potřebnou k určení vzdálenosti.

Bez odlišností je také definice třídy *Resolution* a počátek funkce *main* po načtení kalibračního souboru. Následuje volání metody *init_calibration* doplněné o uložení ohniskové vzdálenosti do proměnné *foc_length*. Dále je do proměnné *baseline* uložena vzdálenost mezi levou a pravou kamerou 120 mm, která je známá ze specifikace kamery ZED 2 [41].

```
def main():
    ...
    camera_matrix_left, camera_matrix_right, map_left_x, map_left_y,\
        map_right_x, map_right_y, foc_length =\
        init_calibration(calibration_file, image_size)

    baseline = 0.120  # m
```

Následuje vytvoření objektu *StereoSGBM* z knihovny OpenCV pod názvem *stereoProcessor*, který představuje výpočet disparity mezi levým a pravým obrazem metodou SGBM (Semiglobal Block Matching). Jedná se o implementaci SGM algoritmu doplněnou o možnost párování bloků snímků, nikoliv jen jednotlivých pixelů. [46] Před vytvořením jsou definovány parametry minimální hodnoty disparity *min_disp*, která je vzhledem k rovnoběžnému uspořádání kamer rovna nule, rozsah disparity *num_disp*, který dle dokumentace [46] musí být dělitelný 16, a velikost bloků pro porovnávání *window_size*, která je stanovena na 9. Dále jsou stanoveny proměnné *P1mult* a *P2mult*, pomocí kterých lze regulovat míru vyhlazování disparitní mapy danou parametry *P1* a *P2*. Všechny tyto parametry ovlivňují výslednou disparitní mapu a tím následně i přesnost určení vzdálenosti, jejich stanovení proto není konečné a budou ještě v dalších kapitolách upraveny. Zmíněné parametry jsou předány funkci *StereoSGBM_create*, kterou je vytvořen objekt *stereoProcessor*. Při vytváření jsou dále upřesněny parametry *speckleWindowSize* a *speckleRange*, které slouží k vyfiltrování lokálních malých oblastí šumu a jejich hodnoty jsou stanoveny podle doporučení v dokumentaci třídy [46]. S využitím funkce *createRightMatcher* byla pak vytvořena kopie objektu pro obraz pravé kamery, která bude využita později k filtrování disparitní mapy pomocí WLS (Weighted Least Squares) filtru.

```
    min_disp = 0
    num_disp = 64

    window_size = 9

    P1mult = 8
    P2mult = 32

    stereoProcessor = cv2.StereoSGBM_create(minDisparity = min_disp,
        numDisparities = num_disp,
        blockSize = window_size,
```

```
        P1 = int(P1mult*3*window_size**2),
        P2 = int(P2mult*3*window_size**2),
        speckleWindowSize = 100,
        speckleRange = 2
         )
    stereoProcessor_R = cv2.ximgproc.createRightMatcher(stereoProcessor)
```

Dále je vytvořen objekt *wls_filter* zmíněného WLS filtru založeného na metodě nejmenších čtverců, který pomáhá vyhladit disparitní mapu a zpřesnit hodnoty disparity v oblastech hran. Funkci filtru je možné ladit parametry lambda a sigma. Zvýšením prvního ze zmíněných parametrů je možné docílit přesnějšího kopírování hran původního snímku, doporučená hodnota je 8000. Druhý parametr pak ovlivňuje citlivost filtrování v oblastech hran, běžné hodnoty jsou mezi 0,8 a 2,0. [47] Tyto parametry jsou definovány v samostatných proměnných *lmbda* a *sigma* a podobně jako v případě parametrů objektu *stereoProcessor* mají také vliv na přesnost určení vzdálenosti a rovněž budou dále upraveny.

```
    lmbda = 8000
    sigma = 1.5

    wls_filter =\
       cv2.ximgproc.createDisparityWLSFilter(matcher_left=stereoProcessor)

    wls_filter.setLambda(lmbda)
    wls_filter.setSigmaColor(sigma)
```

Tím je ukončena inicializace a dále je spuštěna nekonečná while smyčka, ve které je nejprve získán snímek z kamery, načež je rozdělen na levý a pravý obraz a tyto snímky jsou rektifikovány. Kód pro tuto část je shodný s popisem v předchozí kapitole. Dále jsou snímky po rektifikaci přetvořeny do stupňů šedi, načež je metodou *compute* proveden výpočet disparit pravého obrazu vůči levému (*disparity_L*) a levého vůči pravému (*disparity_R*). Hodnoty výsledných disparit jsou uloženy jako 32bitový float. Hned při výpočtu jsou hodnoty poděleny 16, čímž je dosaženo skutečné hodnoty disparity. Obě tyto disparity jsou ve spojení s původním levým snímkem využity při aplikaci WLS filtru, který je využit na základě příručky pro zvýšení kvality vypočtené disparitní mapy [48].

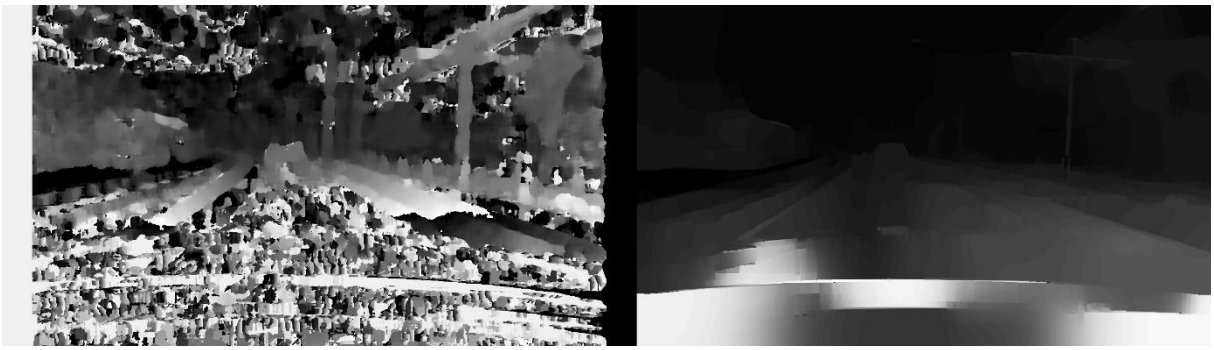
```
    while True:
        ...
        grayL = cv2.cvtColor(left_rect, cv2.COLOR_BGR2GRAY)
        grayR = cv2.cvtColor(right_rect, cv2.COLOR_BGR2GRAY)

        disparity_L = stereoProcessor.compute(grayL,
                          grayR).astype(np.float32)/16
        disparity_R = stereoProcessor_R.compute(grayR,
                          grayL).astype(np.float32)/16

        disparity_filtered = wls_filter.filter(disparity_L, grayL, None,
                             disparity_R)
```

Ze srovnání na obr. 28 je patrné, že disparitní mapa získaná z výchozího snímku na obr. 26 pouze metodou *compute* je oproti té po aplikování filtru velmi zkreslená. Využití

nezpracované disparitní mapy pro výpočet vzdálenosti by kvůli výskytu skokových změn disparity a nejasných hran objektů nebylo vhodné.



Obr. 28 Srovnání disparitní mapy před (vlevo) a po aplikaci WLS filtru (vpravo)

Disparitní mapa zpracovaná WLS filtrem je již využita k výpočtu vzdálenosti na souřadnicích (*x_disp*, *y_disp*), které jsou pro ilustraci stanoveny do středu obrazu. Pro lepší orientaci je do rektifikovaného snímku z levé kamery vykreslen metodou *circle* kroužek ve vyhodnocovaném bodě. Obrázek s vyznačeným bodem je pak metodou *imshow* vykreslen. Výpočet vzdálenosti je za dodržení podmínky nenulové hodnoty disparity proveden aplikací vztahu (8) z kapitoly 1.6.3. Jelikož jednotka ohniskové vzdálenosti i disparity je pixel, jednotka stanovené vzdálenosti odpovídá jednotce vzdálenosti kamer. Ta byla v proměnné *baseline* uložena v m. Hodnota zjištěné vzdálenosti je následně pomocí příkazu *print* vypsána do příkazové řádky. Následně je sledováno stisknutí klávesy pro přerušení smyčky a ukončení programu. Kód je ukončen voláním funkce *main*.

```
        x_disp = int(image_size.width/2)
        y_disp = int(image_size.height/2)
        cv2.circle(left_rect, (x_disp, y_disp), radius=5,
                   color=(0, 0, 255), thickness=2)
        cv2.imshow("left", left_rect)

        if disparity_filtered[y_disp, x_disp] != 0:
            dist_disparity = baseline *\
                             foclen / disparity_filtered[y_disp, x_disp]
            print(f"distance: {dist_disparity:.1f} m")

        if cv2.waitKey(30) >= 0:
            break

    exit(0)


if __name__ == "__main__":
    main()
```

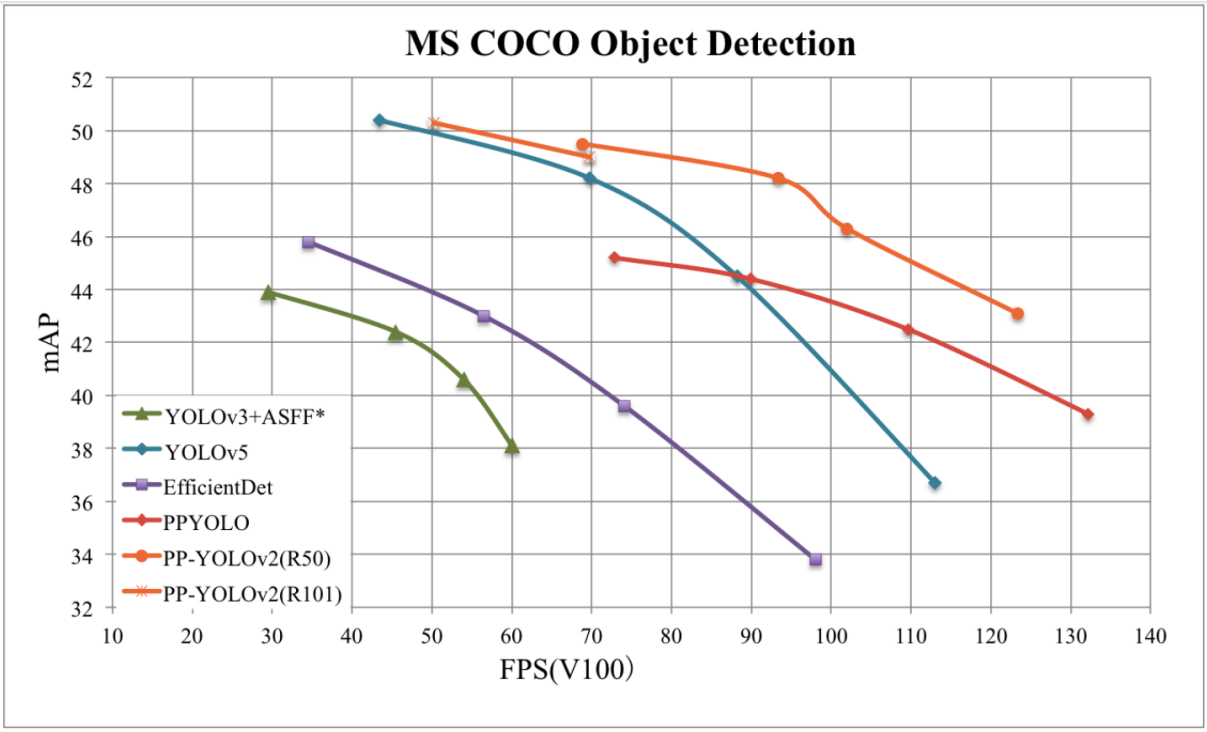
4 ALGORITMUS PRO DETEKCI OBJEKTŮ

Cílem této části práce je vytvořit funkční algoritmus, který realizuje zpracování obrazu pro řízení autonomního vozidla. Je zde vypracována detekce objektů s ohledem na zamýšlenou aplikaci a následně je spojena s vytvořeným algoritmem pro určení vzdálenosti, který byl popsán v předchozí kapitole.

4.1 VOLBA PLATFORMY PRO DETEKCI OBJEKTŮ

Pro aplikaci v detekci objektů již byla vytvořena celá řada architektur neuronových sítí, z nichž některé byly představeny v kapitole 1.5. Z uvedených se pro využití v rámci této práce jeví jako vhodné architektury typu YOLO, a to zejména z důvodu množství dostupných variant. Uvažované varianty zahrnovaly YOLOv4 a YOLOv5, dále PP-YOLO ve verzi 2.

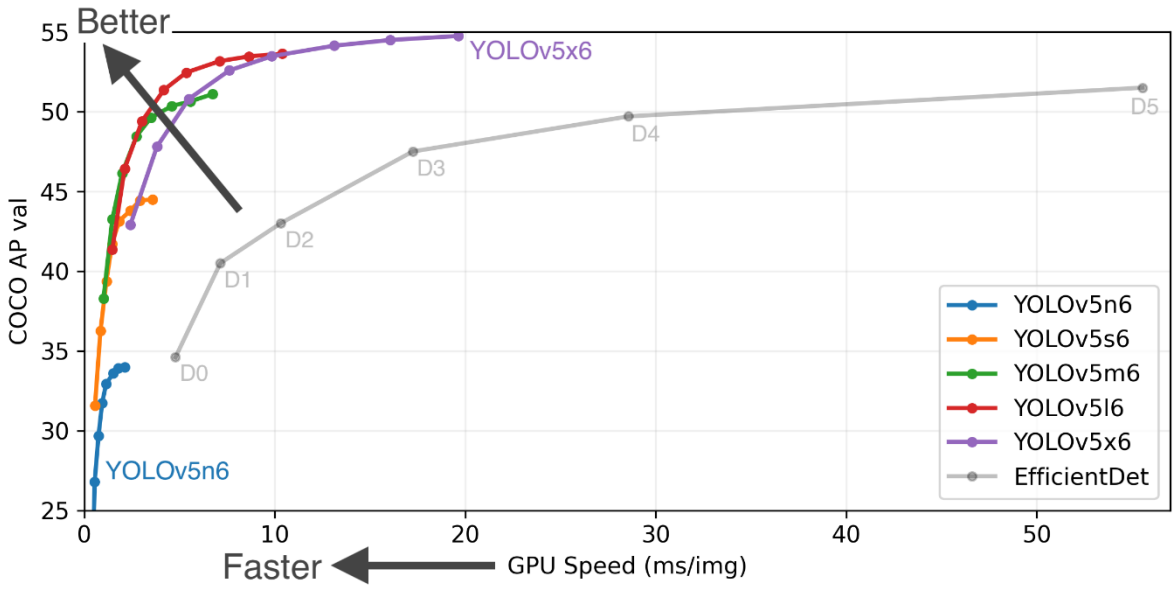
Při výběru konkrétní platformy byl brán v potaz výkon jejich dílčích modelů neuronových sítí, tedy poměr přesnosti a rychlosti zpracování obrázku. Dále bylo třeba zohlednit také dostupnost podkladů a použitý framework s ohledem na uživatelskou přívětivost pro spojení s algoritmem určení vzdálenosti. Z pohledu dokumentace a obtížnosti implementace se jevil nejhůře model PP-YOLO v2. Je založen na frameworku PaddlePaddle, který byl vytvořen čínskými vývojáři a velká část dokumentace v angličtině je buď neaktuální nebo zcela chybí [49]. Ačkoliv se tedy zdál tento model z výkonnostního hlediska jako nejlepší (obr. 29), byl z výběru vyřazen.



Obr. 29 Srovnání modelu PP-YOLOv2 s modelem YOLOv5 a dalšími [50]

Ze srovnání platforem YOLOv4 a YOLOv5 byla zvolena druhá zmíněná, a to z důvodu přechodu z frameworku Darknet na rozšířenější PyTorch, Dalším argumentem bylo vydání verze 6.0, která oproti předchozím vydáním obsahuje nový model Nano. Ten má menší počet parametrů, zpracování snímků je tak rychlejší, a je proto vhodný pro aplikaci na méně

výkonných zařízeních. Srovnání přesnosti a rychlosti jednotlivých modelů YOLOv5 je zobrazeno na obr. 30. Rychlost je zde místo počtu snímků za vteřinu (FPS) vyjádřena v době zpracování jednoho snímku v milisekundách. U nejmenší architektury tak uváděná rychlost až 0,6 ms na snímek odpovídá přibližně 1600 FPS, ovšem při velmi nízké hodnotě mAP 25 %. [17]

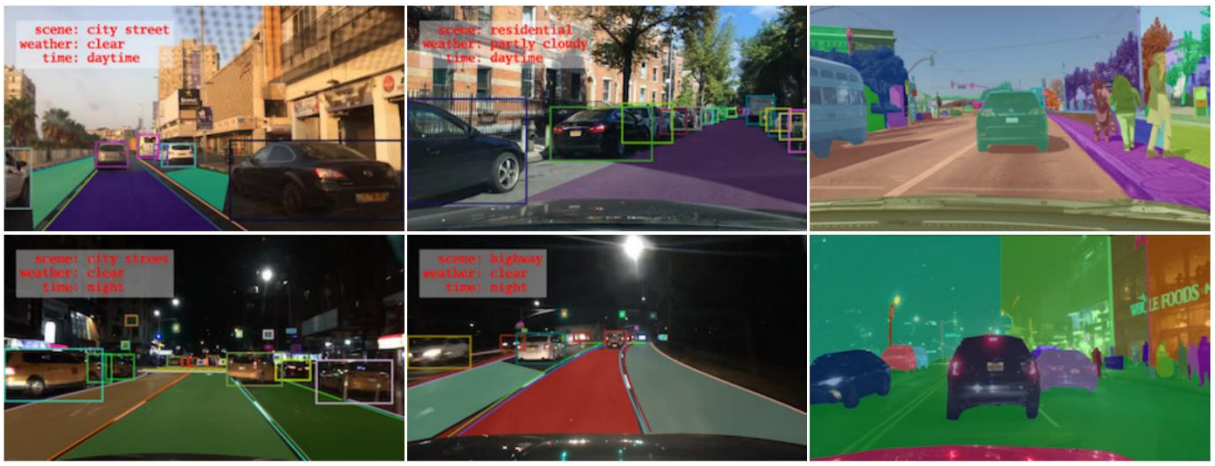


Obr. 30 Srovnání jednotlivých modelů YOLOv5 ve verzi 6.0 [17]

4.2 DATASET

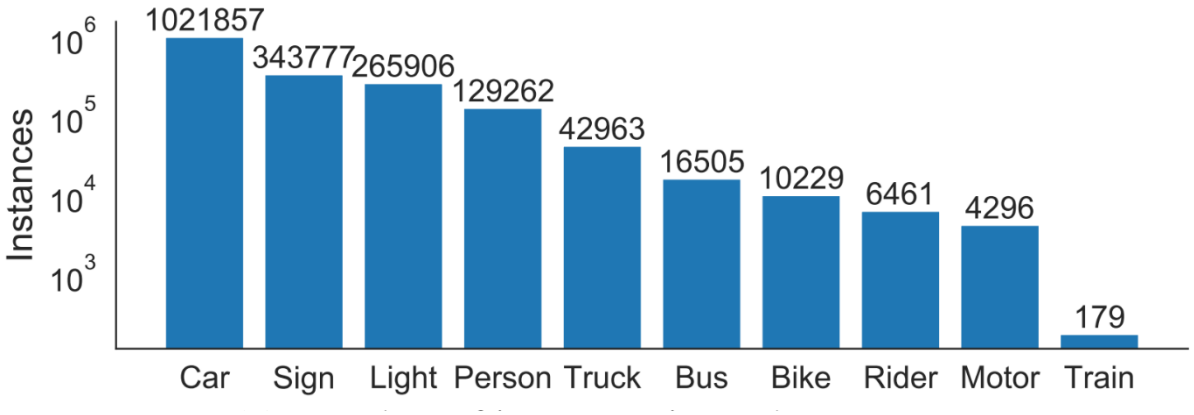
Dále bylo nutné zvolit relevantní datový soubor pro zamýšlenou aplikaci, tedy rozpoznávání objektů v souvislosti s autonomním řízením. Dataset by tedy měl obsahovat převážně třídy související se silničním provozem. Nepochybně nejvíce zastoupenou třídou by měly být automobily, dále pak chodci, značky, semafory atd.

Pro trénování sítě byl zvolen dataset BDD100K. Jedná se o sadu 100 000 videoklipů určenou pro různé aplikace v autonomním řízení. Soubor dat je opatřen informacemi o prostředí a počasí, vyznačením jízdních pruhů a ohraničením objektů pro detekci objektů i segmentaci snímků, jak je zřejmé z obr. 31. [51]



Obr. 31 Ukázka datasetu BDD100K [51]

Specificky pro detekci objektů je k dispozici sada snímků opatřených ohraničujícími rámečky a přiřazením třídy, získaná z 10. sekundy každého videa. Celkem tak soubor obsahuje 100 000 snímků a je rozdělen na 70 000 snímků pro trénování, 10 000 pro validaci a 20 000 pro testování. Soubor dat obsahuje označení 10 tříd: automobil, značka, semafor, chodec, nákladní vozidlo, autobus, kolo, jezdec, motocykl, vlak. Počet výskytů objektů jednotlivých tříd je zobrazen na obr. 32. [51]



Obr. 32 Třídy a počty jejich výskytů v datasetu BDD100K [51]

4.3 Učení sítě

Po zvolení datasetu a modelu detekce objektů bylo třeba natrénovat neuronovou síť na vybraný dataset, přičemž bylo třeba sledovat patřičné metriky a zamezit stavu přetrénování sítě. Trénování bylo provedeno nejprve pro architekturu YOLOv5s6, následně i pro menší model YOLOv5n6. Ten byl doplněn, jelikož nebylo možné předem usoudit, jak výrazně se na rychlosti detekce projeví zpracování obrazu ze stereo kamery a určení vzdálenosti objektu. Primárně tedy bude využit větší ze zvolených modelů a pouze v případě příliš vysoké výpočetní náročnosti bude aplikován menší model ve snaze zvýšit rychlost zpracování.

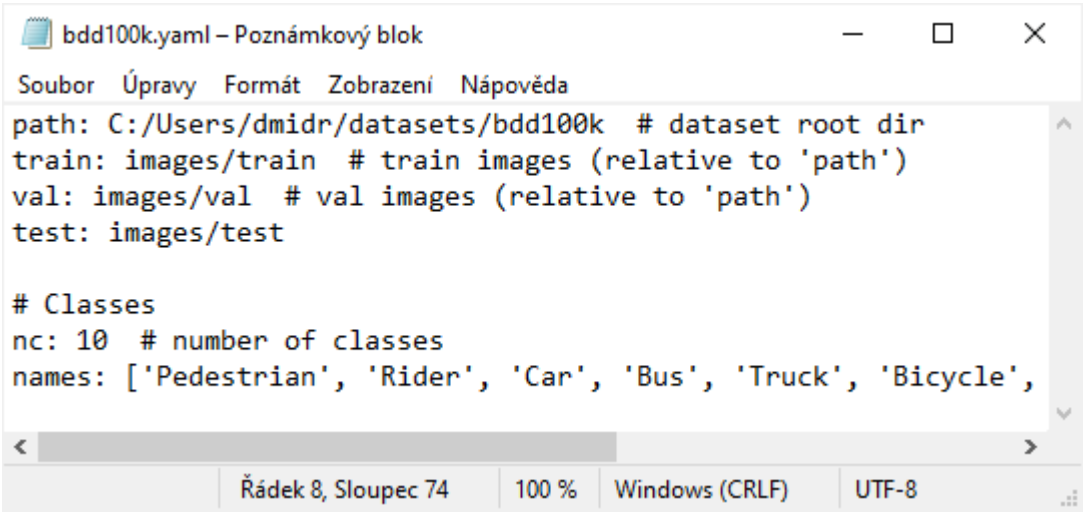
4.3.1 Instalace YOLOv5 a spuštění učení

Učení neuronové sítě bylo prováděno na stolním počítači osazeném grafickou kartou NVIDIA GeForce GTX 1080, procesorem AMD Ryzen 1600 a 16 GB RAM paměti. Na tomto stroji s operačním systémem Windows 10 bylo nejprve nutné nainstalovat platformu YOLOv5, která je vytvořena v jazyce Python. K tomu bylo třeba stáhnout obsah repozitáře pro verzi v6.0 [17] a rozbalit jej do vhodné složky. Následným krokem bylo nainstalování všech požadovaných balíčků v souboru *requirements.txt* pomocí příkazu `pip install -r requirements.txt` v příkazovém řádku otevřeném v adresáři rozbaleného archivu. Dále bylo nutné nainstalovat platformu CUDA [52] a knihovnu cuDNN [53] od společnosti NVIDIA. S pomocí těchto dvou balíků bylo možné využít k učení sítě maximální výpočetní kapacitu grafického procesoru.

K trénování bylo třeba dodat datový soubor se soubory definujícími souřadnice ohraničujících rámečků ve správném formátu. Platforma YOLOv5 využívá formát YOLO, ve kterém je pro každý snímek přiřazen textový soubor se souřadnicemi a třídami objektů v něm obsažených. [54] Zvolená datová sada však obsahuje tyto informace ve formátu nástroje Scalabel, kdy všechny tyto anotace, tedy souřadnice a třídy pro všechny snímky, jsou uloženy v jednom souboru s příponou *.json*. [55] Bylo tedy třeba zajistit správné převedení do potřebného

formátu. K tomu bylo využito nejprve Python skriptu poskytovaného tvůrci datasetu [56] pro převedení anotací do formátu COCO. Následně byly anotace z tohoto formátu převedeny do formátu YOLO nástrojem pro konverzi z formátu COCO. [57]

Aby platforma YOLOv5 měla přístup k souboru dat pro učení, bylo dále nutné vytvořit ve složce *data* adresáře YOLOv5 konfigurační soubor *bdd100k.yaml* obsahující informace o zvoleném datasetu BDD100K. Jedná se o textový soubor, ve kterém je definována absolutní cesta k datasetu a relativní cesty ke složkám souborů pro trénování, validaci a testování. Dále je zde uveden počet tříd v daném datasetu a výpis jejich názvů. Podoba souboru je představena na obr. 33.



Obr. 33 Podoba konfiguračního souboru datasetu

Nyní již bylo možné spustit Python skript *train.py* a tím i samotné učení pomocí následujícího příkazu zadaného do příkazového řádku otevřeného v adresáři platformy YOLOv5.

```
python train.py --img 640 --cfg yolov5s.yaml --hyp hyp.scratch.yaml
--batch -1 --epochs 100 --data bdd100k.yaml --weights yolov5s.pt
--device 0 --name bdd
```

V rámci tohoto příkazu byly specifikovány doplňující parametry trénování. V první řadě pomocí parametru `--img 640` byla upřesněna velikost ve čtvercovém formátu, na kterou mají být obrázky v trénovacím a validačním souboru přetvořeny. Parametr `--cfg yolov5s.yaml` pak upřesňuje umístění konfiguračního souboru, který obsahuje vlastnosti architektury, jako např. počet vrstev sítě. Souborem *yolov5s.yaml* umístěným v adresáři *YOLOv5/models* tak bylo určeno použití architektury YOLOv5s.

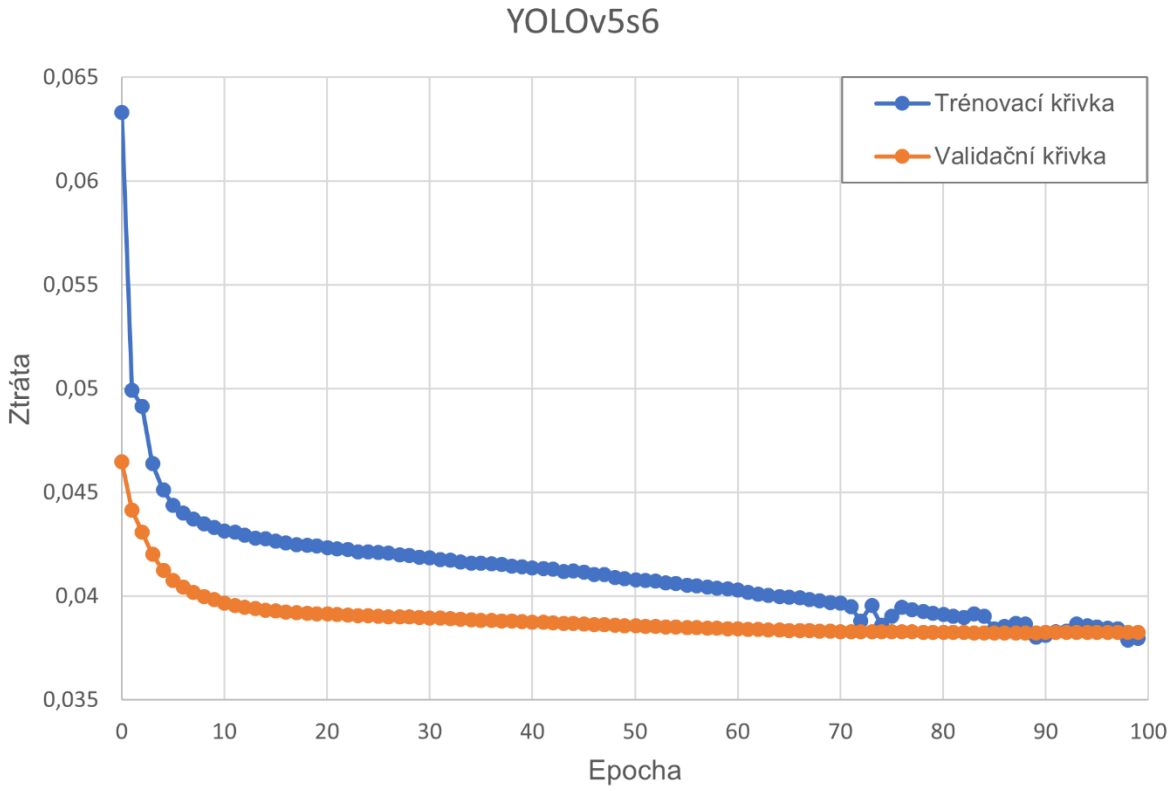
Následující parametr `--hyp` specifikuje použití souboru pro nastavení hyperparametrů sítě, kdy byl použit výchozí soubor *hyp.scratch.yaml* umístěný v adresáři *YOLOv5/data/hyps*. Konfigurace `--batch -1` dále upřesňuje počet snímků uložených v paměti grafického procesoru při jedné iteraci trénování, hodnotou -1 je počet zjištěn automaticky dle dostupné paměti. Následuje upřesnění trénování po dobu 100 epoch parametrem `--epochs 100` a použitého souboru dat odkazem na výše vytvořený soubor *bdd100k.yaml*. Dodáním parametru `--weights` je určeno použití předem natrénovaných vah *yolov5s.pt*. Jedná se

o hodnoty vah, které jsou stanoveny jako výchozí pro vlastní trénování a které byly vytvořeny trénováním na datasetu COCO.

Parametr `--device` specifikuje zařízení použité pro trénování, kdy hodnota 0 představuje grafickou kartu (podporováno je i použití většího počtu grafických karet, pak jsou hodnoty 0, 1, 2, …). Závěrem je určen název, pod kterým je vytvořen adresář ve složce *YOLOv5/runs/train* a do kterého jsou ukládány hodnoty vah v souboru s příponou .pt a další informace o průběhu učení. Při spouštění daného skriptu lze specifikovat i další možnosti, jejichž výčet lze zobrazit příkazem `python train.py -h`. [17]

4.3.2 VÝSLEDKY TRÉNOVÁNÍ

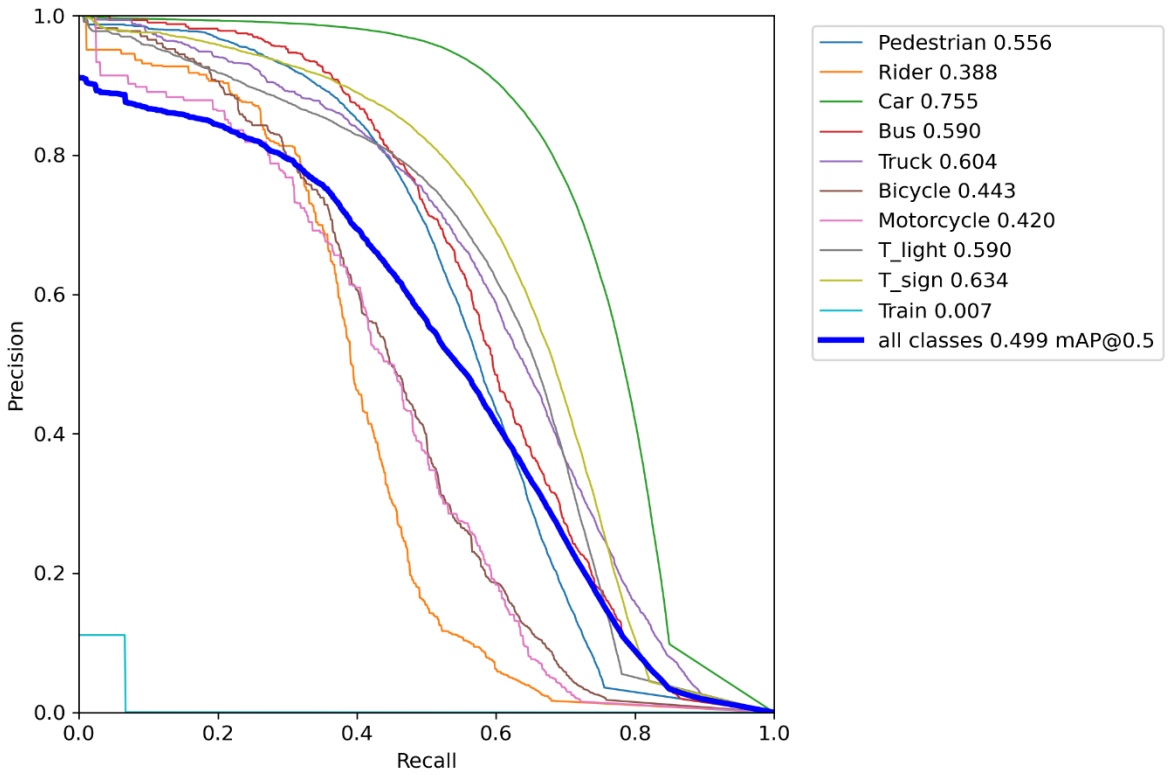
Trénování většího modelu YOLOv5s6 zabralo přibližně 7 dní. Z výsledného srovnání průběhů ztrát trénovacích a validačních dat (obr. 34) lze usoudit, že nedošlo k přeučení modelu a lze tak očekávat správnou funkci na nových datech.



Obr. 34 Průběhy ztrát při učení modelu YOLOv5s6

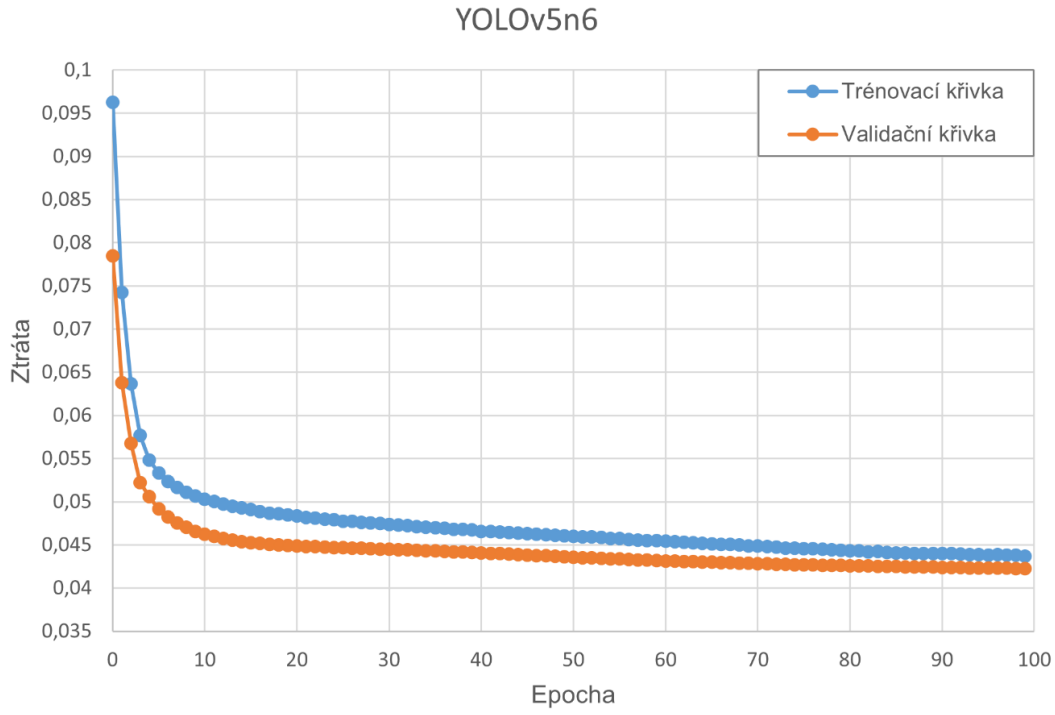
Na obr. 35 je zobrazena PR křivka s hodnotami AP pro jednotlivé třídy a průměrem všech tříd mAP při hraniční hodnotě *IOU t=0,5*. Z průběhu pro třídu vlak je patrné, že datový soubor neobsahuje dostatek výskytů této třídy a nelze tedy očekávat, že natrénovaný model bude správně rozpoznávat objekty této třídy. U všech ostatních tříd je průběh v souladu s očekáváním s ohledem na počty výskytů a unikátnost objektů dané třídy.

Výsledná hodnota mAP@0.5 = 0,499 je v porovnání s hodnotou 0,560 na datasetu COCO prezentovanou tvůrci platformy YOLO [17] taktéž obstojná. S výjimkou výše zmíněné třídy vlak tak lze u tohoto modelu očekávat správnou detekci objektů.



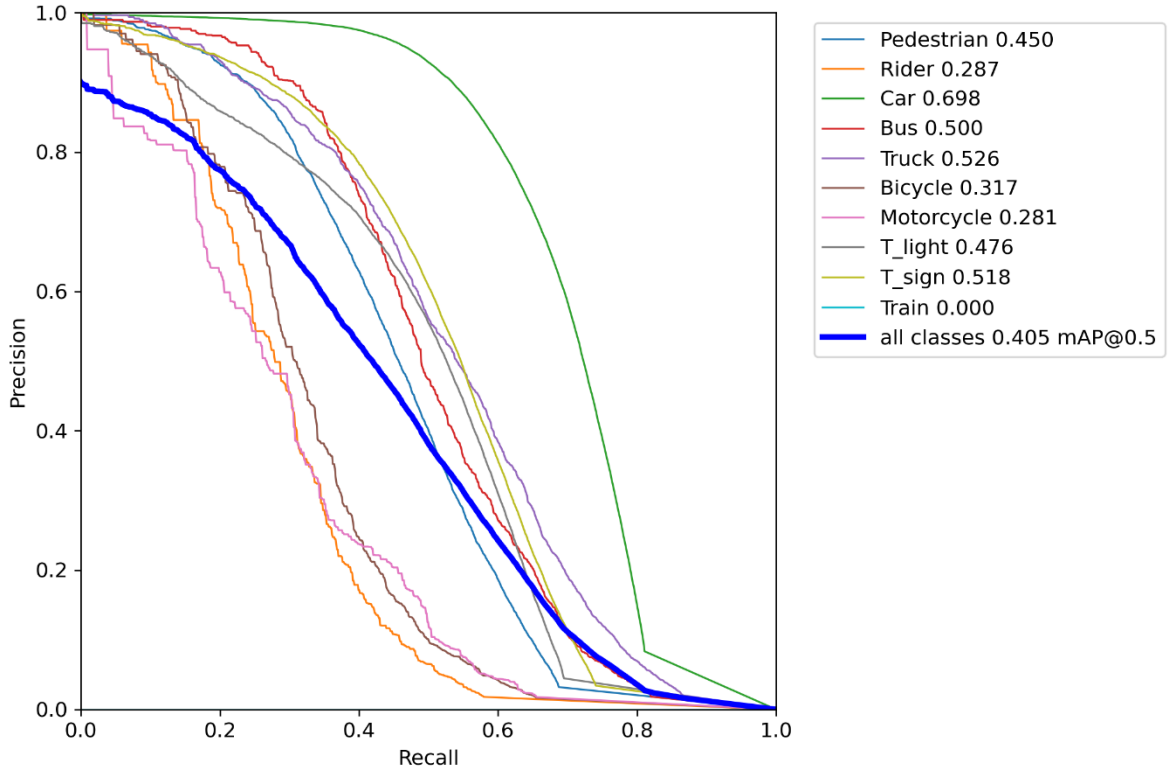
Obr. 35 PR křivka a hodnoty AP a mAP pro model YOLOv5s6 učený po dobu 100 epoch

V případě architektury YOLOv5n6 s menším počtem parametrů byla doba učení výrazně kratší. Průběh trénovacích a validačních ztrát opět napovídá, že nedošlo k přeučení modelu, jak je vidno na obr. 36.



Obr. 36 Průběhy ztrát při učení modelu YOLOv5n6

Z PR křivky na obr. 37 je opět patrné selhání v trénování třídy vlak, průběhy se obecně podobají těm při trénování modelu YOLOv5s6 na obr. 35. Hodnoty AP jednotlivých tříd jsou v souladu s očekáváním vzhledem k menšímu počtu parametrů modelu nižší než u varianty YOLOv5s6. Výsledná hodnota mAP@0.5 = 0,405 se opět blíží hodnotě 0,460 pro model naučený jeho tvůrci na datasetu COCO [17]. Ani u tohoto modelu s ohledem na průběh učení nejsou očekávány potíže v detekci objektů, byť s výjimkou zmiňované třídy vlak.



Obr. 37 PR křivka a hodnoty AP a mAP pro model YOLOv5n6 učený po dobu 100 epoch

Výstupem učení obou modelů je soubor s příponou .pt, který obsahuje hodnoty vah ve formátu frameworku PyTorch. S využitím tohoto souboru se v platformě YOLOv5 realizuje pomocí Python skriptu *detect.py* samotná detekce objektů, a to na libovolném zařízení, které je schopno YOLOv5 provozovat.

Dále je možné pomocí Python skriptu *export.py* konvertovat .pt soubor do formátu pro spuštění v jiné platformě pro detekci objektů. V tomto případě byly naučené váhy využity pouze v rámci YOLOv5, převedení formátu tudíž nebylo nutné.

4.4 SPOJENÍ SE STEREO KAMEROU A URČENÍM VZDÁLENOSTI

Pro zobrazení vzdálenosti stanovené stereo kameru současně při detekci objektů byl vytvořen Python skript *detectdist.py* využívající skriptu pro detekci objektů *detect.py* obsaženého v YOLOv5. Ke zpracování stereo obrazu a výpočtu vzdálenosti je implementován algoritmus vytvořený v kapitole 3.

V první řadě bylo doplněno importování balíků *NumPy* a *configparser*. Dále byla z algoritmu pro určení vzdálenosti přenesena třída *Resolution* obsahující proměnné šířky a výšky obrazu.

```
class Resolution:
    width = 1280
    height = 720
```

Následně byla vytvořena třída *Stereo*, v rámci které jsou nejprve v metodě *__init__* inicializovány objekty související se zpracováním stereo obrazu pro určení vzdálenosti. To zahrnuje kalibrační parametry kamery a jejich vzájemnou vzdálenost, dále pak objekty pro výpočet disparity levého vůči pravému obrazu i opačné, která je využita u následně vytvořeného WLS filtru.

```
class Stereo:
    def __init__(self):
        calibration_file = "SNxxxx.conf")
        if calibration_file == "":
            exit(1)
        print("Calibration file found. Loading...")

        image_size = Resolution()
        image_size.width = 1280
        image_size.height = 720
        self.camera_matrix_left, self.camera_matrix_right,\
          self.map_left_x, self.map_left_y,\
          self.map_right_x, self.map_right_y,\
          self.foclen = self.init_calibration(
                        calibration_file, image_size)
        self.baseline = 0.120  # m

        min_disp = 0
        num_disp = 64
        window_size = 9

        P1mult = 8
        P2mult = 32

        self.stereoProcessor = cv2.StereoSGBM_create(
            minDisparity = min_disp,
            numDisparities = num_disp,
            blockSize = window_size,
            P1 = int(P1mult*3*window_size**2),
            P2 = int(P2mult*3*window_size**2),
            speckleWindowSize = 100,
            speckleRange = 2
             )

        self.stereoProcessor_R =\
          cv2.ximgproc.createRightMatcher(self.stereoProcessor)

        lmbda = 8000
        sigma = 1.5

        self.wls_filter = cv2.ximgproc.createDisparityWLSFilter\
                          (matcher_left=self.stereoProcessor)
```

```
        self.wls_filter.setLambda(lmbda)
        self.wls_filter.setSigmaColor(sigma)
```

Dále je v rámci této třídy vytvořena metoda *distance*, která bude použita pro určení vzdálenosti objektů detekovaných ve snímku. Mezi vstupní proměnné patří uložený snímek stereo obrazu *img_stereo*, pole souřadnic středových bodů jednotlivých objektů v tomto snímku *coords* a inicializované pole vzdáleností jednotlivých objektů *dist*. V rámci této metody je pak provedena rektifikace obrazu z kamery, určení disparity a aplikace WLS filtru.

```
def distance(self, img_stereo, coords, dist):
    left_right_image = np.split(img_stereo, 2, axis=1)
    left_rect = cv2.remap(left_right_image[0], self.map_left_x,\
                self.map_left_y, interpolation=cv2.INTER_LINEAR)
    right_rect = cv2.remap(left_right_image[1], self.map_right_x,\
                 self.map_right_y, interpolation=cv2.INTER_LINEAR)
    grayL = cv2.cvtColor(left_rect, cv2.COLOR_BGR2GRAY)
    grayR = cv2.cvtColor(right_rect, cv2.COLOR_BGR2GRAY)

    disparity_L = self.stereoProcessor.compute(grayL, grayR)
    disparity_R = self.stereoProcessor_R.compute(grayR, grayL)
    disparity_filtered = self.wls_filter.filter(disparity_L, grayL,
        None, disparity_R)
```

Následuje for cyklus, který pomocí funkce *enumerate* prochází polem *coords* souřadnic pro určení vzdálenosti všech detekovaných objektů v rámci daného snímku. V rámci něj je proveden výpočet vzdálenosti s využitím vzdálenosti kamer, ohniskové vzdálenosti a souřadnic aktuálního objektu *det*. Výsledek je uložen do pole *dist* na pozici indexu aktuální iterace *i*. Metoda *distance* je po for cyklu ukončena vrácením proměnné *dist*.

```
    for i, det in enumerate(coords):
        if disparity_filtered[int(det[1]), int(det[0])] != 0:
            dist[i] = self.baseline * self.foclen\
                      / disparity_filtered[int(det[1]), int(det[0])]

    return dist
```

Oproti samostatnému algoritmu pro určení vzdálenosti v kapitole 3 je zde tento proces přizpůsoben zpracování jednotlivých snímků postupně, kód tak neběží ve while smyčce, ale je spuštěn vždy voláním metody po detekci objektů, což bude popsáno později.

Třída *Stereo* závěrem zahrnuje definici metody *init_calibration*, která oproti popisu v kapitole 3 zůstává nezměněna.

V rámci metody *run* jsou snímky pro provedení detekce objektů načteny do proměnné *dataset*. K tomu je v případě sběru snímků z kamery (tzn. při zadání parametru `--source 0` při spuštění skriptu) použita metoda *LoadStreams*. Při zpracování video souboru nebo jednotlivých obrázků (tedy předání cesty k danému souboru do parametru `--source`) je pak použita metoda *LoadImages*.

```
if webcam:
    view_img = check_imshow()
    cudnn.benchmark = True
```

```
    dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt
              and not jit)
    bs = len(dataset)  # batch_size
else:
    cudnn.benchmark = True
    dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt
              and not jit)
    bs = 1  # batch_size
```

Obě zmíněné metody jsou obsaženy v modulu *datasets.py* uloženém ve složce *utils* a do skriptu *detectdist.py* jsou naimportovány. V těchto metodách jsou požadované snímky načteny a také přetvořeny do velikosti stanovené parametrem `--imgsz` při spouštění skriptu *detect.py* (výchozí hodnota je 640x640). Při detekci objektů jsou pak pro zrychlení zpracování využity snímky s takto upravenou velikostí.

V souvislosti se zpracováním stereo obrazu, který do algoritmu vstupuje z kamery jako jeden přenos s obrazy levé a pravé kamery vedle sebe, je třeba zajistit provedení detekce objektů pouze na obrazu jedné kamery. To znamená, že bylo třeba upravit i zmíněné metody *LoadStreams a LoadImages* tak, aby byl k upravení velikosti předán pouze snímek z levé kamery.

Přitom bylo nutné vzít v potaz odlišný princip načítání jednotlivých obrázků u těchto dvou tříd. V případě třídy *LoadImages* je do proměnné *dataset* vrácen vždy jen jeden samostatný obrázek s upravenou velikostí pro detekci objektů v proměnné *img* a obrázek v původní podobě pro vykreslení rámečků objektů v proměnné *img0*. Obrázek *img* je v metodě *__next__* této třídy před upravením velikosti rozdělen, přičemž je využito NumPy funkce *split*. Obrázek *img0* je ponechán v původní stereo podobě.

```
img = np.split(img0, 2, axis=1)[0]
```

U třídy *LoadStreams* bylo nejprve třeba nastavit režim kamery, tedy správné rozlišení a počet snímků za vteřinu. Tento krok je proveden za ověřením správného načtení přenosu z kamery, rozlišení je pevně nastaveno na HD (2560x720 pixelů) při 30 snímcích za vteřinu.

```
            assert cap.isOpened(), f'Failed to open {s}'
            w = 2560
            h = 720
            cap.set(cv2.CAP_PROP_FRAME_WIDTH,w)
            cap.set(cv2.CAP_PROP_FRAME_HEIGHT,h)
            self.fps[i] = 30
```

V případě této třídy je dále situace odlišná o ukládání jednotlivých snímků do pole. Díky tomu obsahují proměnné *img* a *img0* pole obrázků, což je třeba reflektovat při rozdělení stereo obrazu. Rozdělení snímku a získání levého obrazu je pak provedeno v metodě *__next__* ve for cyklu procházejícím všemi prvky *x* pole *img0*.

```
img0 = self.imgs.copy()
img = [np.split(x, 2, axis=1)[0] for x in img0]
```

Tím jsou ukončeny změny provedené v souboru *utils/datasets.py*. Po načtení dat v hlavním skriptu je dále vytvořena instance *stereo* výše popsané třídy *Stereo*.

```
stereo = Stereo()
```

Následuje for cyklus procházející jednotlivými snímky v proměnné *dataset*, ve kterém probíhá samotná detekce objektů. V něm je nejprve do proměnné *t1* pomocí funkce *time_sync* uložen čas začátku iterace. Proměnná *im0s* v tomto cyklu koresponduje s výše zmíněnou proměnnou *img0*, která obsahuje neupravený obrázek později potřebný k vykreslení výsledků detekce. Ten však při tvorbě proměnné *dataset* nebyl upraven a obsahuje tak stále obraz z obou kamer. Pro určení vzdálenosti je tedy obsah proměnné *im0s* uložen do nové proměnné *imstereo*. Obsah proměnné *im0s* je pak rozdělen opět s využitím metody *split*. Přitom je třeba opět zohlednit rozdílnou povahu proměnné v závislosti na použitém zdroji, tedy provést rozdělení ve for cyklu pro každý prvek v poli v případě vstupu z kamery (podmínka `if webcam`) a při zpracování videa či obrázku rozdělit samostatný snímek.

```
for path, im, im0s, vid_cap in dataset:
    t1 = time_sync()
    imstereo = im0s
    if webcam:
        im0s = [np.split(x, 2, axis=1)[0] for x in im0s]
    else:
        im0s = np.split(im0s, 2, axis=1)[0]
```

V rámci tohoto for cyklu je dále provedena detekce objektů a vyfiltrování detekcí pomocí NMS. Doby trvání jednotlivých operací jsou uloženy v proměnné *dt* a jsou určeny jako rozdíl časů získaných funkcí *time_sync* na začátku a konci dané operace. Aby bylo možné porovnat časovou náročnost detekce objektů a zpracování stereo obrazu, byla proměnná *dt* rozšířena o dobu trvání určení vzdálenosti. Za tímto účelem byla oproti původnímu YOLOv5 skriptu *detect.py* doplněna po provedení operace NMS proměnná *t4* obsahující aktuální čas a doba trvání NMS byla uložena do třetího prvku proměnné *dt*.

```
t4 = time_sync()
dt[2] += t4 - t3
```

Za tento krok bylo doplněno určení vzdálenosti. Protože vzdálenost pro každý detekovaný objekt bude vyhodnocována ve středu jeho ohraničujícího rámečku, bylo nutné zjistit souřadnice tohoto bodu ze souřadnic rohových bodů rámečků. Ty jsou vyjmuty z proměnné *pred*, která obsahuje PyTorch tenzor s výsledky detekce objektů. Obsah této proměnné je pomocí sekvence PyTorch funkcí *clone, detach, cpu* a *numpy* nejprve zkopírován, dále je odstraněna jeho vazba na původní proměnnou, poté je v případě použití grafické karty přesunut do operační paměti procesoru a na závěr je převeden do formátu NumPy pole. Výsledek těchto operací je uložen do proměnné *bbox_coords* a má následující podobu:

```
[[     282.61      156.6     322.95     190.16    0.93519          2]
 [     111.22     127.69     203.06     189.54    0.92067          3]
 [     257.58     161.06     271.57     171.68    0.81622          2]
 [     199.15     145.45     220.07     176.02    0.79162          3]
 [     236.89     159.16     246.86     167.07    0.51926          2]
 [     325.84     156.99     332.31     165.12    0.46713          8]
 [     326.44     145.04     332.39     153.81    0.36996          8]
 [     467.25     171.72     485.52     180.42     0.2572          2]]
```

První dva sloupce obsahují (x, y) souřadnice počátečního bodu rámečku, další dva sloupce pak souřadnice bodu protějšího rohu. Následuje sloupec obsahující pravděpodobnost třídy, poslední sloupec obsahuje stanovení čísla třídy podle pořadí určeného při trénování sítě v konfiguračním souboru datasetu (obr. 33). Počet řádků pak odpovídá počtu detekovaných objektů v rámci daného snímku.

Následně jsou ze získaného pole odstraněny poslední dva sloupce, aby proměnná *bbox_coords* obsahovala pouze souřadnice rámečků. Detekce objektů byla provedena na obrázku se změněnými rozměry, avšak pro určení vzdálenosti je potřeba získat souřadnice odpovídající rozměrům původního snímku. K tomu je využita YOLOv5 funkce *scale_coords*, obsažená v modulu *utils/general.py*. Výsledkem jsou souřadnice odpovídající původnímu obrazu, které jsou funkcí *round* zaokrouhleny na celá čísla.

```
[[       565         289         646         356]
 [       222         231         406         355]
 [       515         298         543         319]
 [       398         267         440         328]
 [       474         294         494         310]
 [       652         290         665         306]
 [       653         266         665         284]
 [       934         319         971         337]]
```

Dále je funkcí *zeros* do proměnné *ctr_coords* vytvořeno NumPy pole o počtu řádků odpovídajícím počtu detekovaných objektů a o dvou sloupcích. Do této proměnné jsou pak uloženy souřadnice středového bodu rámečku pro každý objekt, které jsou získány výpočtem aritmetického průměru souřadnic rohových bodů rámečků. Následně je vytvořena proměnná *dist* s polem o stejném počtu řádků a jednom sloupci, do kterého je voláním výše popsané metody *distance* třídy *Stereo* uložen výsledek určení vzdálenosti pro dané souřadnice a daný stereo snímek *imstereo*. Následně je z důvodu výpisu do obrázku pole metodou *flip* otočeno.

Pro výše popsané operace a případ zpracování přenosu z kamery je určení vzdálenosti provedeno v rámci for cyklu pro výsledky detekce všech obrázků v proměnné *pred*. V tomto případě má proměnná *dist* obsahující výsledné vzdálenosti podobu Python seznamu obsahujícího NumPy pole odpovídající jednotlivým snímkům.

```
if webcam:
    dist = [None] * len(pred)
    for i, x in enumerate(pred):
        bbox_coords = x.clone().detach().cpu().numpy()
        bbox_coords = bbox_coords[:, 0:4]
        bbox_coords = scale_coords(im.shape[2:], bbox_coords,
                      im0s[0].shape).round()
        ctr_coords = np.zeros((bbox_coords.shape[0], 2))
        ctr_coords[:, 0] = (bbox_coords[:, 0] + bbox_coords[:, 2]) * 0.5
        ctr_coords[:, 1] = (bbox_coords[:, 1] + bbox_coords[:, 3]) * 0.5
        dist[i] = np.zeros((len(bbox_coords), 1))
        dist[i] = stereo.distance(imstereo[i], ctr_coords, dist[i])
        dist[i] = np.flip(dist[i])
```

V případě vstupu v podobě video souboru či obrázku je situace jednodušší, výsledná proměnná *dist* obsahuje pouze jediné NumPy pole pro právě zpracovávaný obrázek či snímek z videa.

```
else:
    bbox_coords = pred[0].clone().detach().cpu().numpy()
    bbox_coords = bbox_coords[:, 0:4]
    bbox_coords = scale_coords(im.shape[2:], bbox_coords,
im0s.shape).round()
    ctr_coords = np.zeros((bbox_coords.shape[0], 2))
    ctr_coords[:, 0] = (bbox_coords[:, 0] + bbox_coords[:, 2]) * 0.5
    ctr_coords[:, 1] = (bbox_coords[:, 1] + bbox_coords[:, 3]) * 0.5
    dist = np.zeros((len(bbox_coords), 1))
    dist = stereo.distance(imstereo, ctr_coords, dist)
    dist = np.flip(dist)
```

Po provedení určení vzdálenosti je v proměnné *t5* uchován aktuální čas a do proměnné *dt* je uložena doba trvání operací k určení vzdálenosti. Ta je určena jako rozdíl aktuálního času *t5* od času *t4* uloženého v okamžiku počátku zpracování stereo obrazu.

```
t5 = time_sync()
dt[3] += t5 - t4
```

V kódu dále následuje vnořený for cyklus, v rámci kterého je provedeno zpracování výsledků pro jednotlivé obrázky. Do tohoto for cyklu je pak vnořen další for cyklus, který v rámci daného obrázku prochází jednotlivými detekovanými objekty v opačném pořadí a je v něm provedeno vykreslení rámečků objektů s přiřazením třídy a její pravděpodobnosti do snímku. V rámci tohoto for cyklu byl doplněn výpis určené vzdálenosti, což je důvodem pro dřívější obrácení pořadí proměnné *dist*. Pořadí vzdáleností je určeno počítadlem *index_dist*, které je před for cyklem nastaveno na hodnotu 0 a při každé iteraci je hodnota zvýšena o 1.

```
index_dist = 0
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        ...
        if webcam:
            line = (cls, *xywh, conf, dist[i][index_dist, 0]) if save_conf
                else (cls, *xywh, dist[i][index_dist, 0])
        else:
            line = (cls, *xywh, conf, dist[index_dist, 0]) if save_conf
                else (cls, *xywh, dist[index_dist, 0])

        ...


    if save_img or save_crop or view_img:  # Add bbox to image
        c = int(cls)  # integer class
        if webcam:
            label = None if hide_labels else (
               f'{names[c]} {(dist[i][index_dist, 0]):.2f}' if hide_conf
               else
               f'{names[c]} {conf:.2f} {(dist[i][index_dist, 0]):.2f}')
```

```
        else:
            label = None if hide_labels else (
               f'{names[c]} {(dist[index_dist, 0]):.2f}' if hide_conf else
               f'{names[c]} {conf:.2f} {(dist[index_dist, 0]):.2f}')
        ...
    index_dist += 1
```

Poslední odlišností oproti kódu pouze pro detekci objektů v rámci platformy YOLOv5 je doplnění doby trvání určení vzdálenosti do výpisu v příkazovém řádku po zpracování daného snímku. Příkaz pro výpis této informace se nachází bezprostředně po ukončení for cyklu v předchozí ukázce.

```
LOGGER.info(f'{s}Done. (det: {t3 - t2:.3f} s, dist: {t5-t4:.3f} s)')
```

4.5 SPUŠTĚNÍ NA POČÍTAČI JETSON NANO

Algoritmus pro detekci objektů a měření vzdálenosti byl aplikován na malém počítači Jetson Nano. Cílem bylo vyhodnocení provozuschopnosti vytvořeného algoritmu na zařízení s omezeným výpočetním výkonem při zachování schopnosti zpracování vstupu v reálném čase.

Aby bylo možné detekovat objekty na počítači Jetson Nano s využitím natrénovaných vah v .pt souboru, bylo nejprve nutné před spuštěním zařízení připojit do jeho USB portu kameru Stereolabs ZED 2 a provést správnou konfiguraci tohoto zařízení dle popisu v kapitole 2.3. Poté již bylo možné přistoupit k instalaci YOLOv5 a potřebných prerekvizit, podobně jako v kapitole 4.3.1. Kvůli odlišné architektuře procesoru (ARM Aarch64 namísto x86-64) a nedostupnosti předkompilovaných balíků však bylo nutné většinu nezbytných softwarových nástrojů zkompilovat ze zdrojového kódu.

Knihovna OpenCV, která je standartně v operačním systému Jetpack již zahrnuta, musela být přeinstalována, jelikož předinstalovaná verze nezahrnuje podporu knihoven CUDA a cuDNN. S touto standartní verzí tak není možné využít grafický akcelerátor CUDA, který je hlavní výhodou počítače Jetson Nano v porovnání s jinými zařízeními podobného formátu (např. Raspberry Pi) při využití pro aplikace hlubokého učení. Při reinstalaci knihovny OpenCV bylo postupováno podle návodu [58] s využitím dodaného instalačního skriptu.

Dále bylo dle postupu [59] provedeno stažení a instalace knihoven PyTorch a torchvision. Další požadované Python balíčky byly nainstalovány pomocí nástroje pip dle souboru *requirements.txt* v repozitáři YOLOv5, a to pomocí příkazu `pip3 install (název balíčku)`.

Adresář instalace YOLOv5 byl poté doplněn o vytvořené Python skripty zahrnující určení vzdálenosti. Do kořenového adresáře byl umístěn skript vytvořený skript *detectdist.py* a do složky *utils* upravený soubor *datasets.py*. Skript pro detekci objektů s určením vzdálenosti byl pak z příkazové řádky spuštěn pomocí následujícího příkazu

```
python3 detectdist.py --weights best.pt --source 0,
```

kde parametrem `--weights` je určen soubor *best.pt* s natrénovanými váhami pro model YOLOv5s6 a parametrem `--source 0` je specifikováno použití připojené kamery. Při spouštění tohoto skriptu je možné upřesnit další parametry, všechny možnosti lze zobrazit

pomocí příkazu `python3 detectdist.py -h`. Příklad snímku z kamery zpracovaného algoritmem po spuštění uvedeným příkazem lze vidět na obr. 38. Bližší zkoumání správné funkce vytvořeného algoritmu je obsaženo v následující kapitole.

Obr. 38 Výstupní snímek zpracovaný vytvořeným algoritmem

5 OVĚŘENÍ FUNKČNOSTI ALGORITMU

Následně bylo nutné potvrdit správnou funkci všech částí vytvořeného algoritmu. V rámci této kapitoly je nejprve posouzena přesnost a rozsah určení vzdálenosti. Dále je provedeno vyhodnocení přesnosti detekce objektů, kdy je nejprve ze srovnání dvou různých architektur neuronové sítě zvolena vhodná varianta, u které je pak provedena podrobnější analýza spolehlivosti rozpoznání objektů.

5.1 PŘESNOST A ROZSAH URČENÍ VZDÁLENOSTI

5.1.1 TVORBA DAT PRO VYHODNOCENÍ

Pro tuto část práce byl vytvořen soubor snímků osobního automobilu jakožto měřeného objektu pořízených kamerou Stereolabs ZED 2 v různých vzdálenostech, jehož ukázka je zobrazena na obr. 39.

Obr. 39 Ukázka ze souboru snímků pro vyhodnocení určení vzdálenosti

Tyto snímky byly pořízeny s kamerou připojenou k notebooku, ve kterém se pak zařízení jevilo jako standardní webkamera. Jednotlivé snímky tak bylo možné pořídit s využitím vestavěné aplikace Kamera na operačním systému Windows 10, přičemž tato aplikace umožnila také přepínání mezi dostupnými rozlišeními kamery. Kamera byla umístěna na

stativu a její umístění do příslušných vzdáleností bylo prováděno s využitím laserového měřiče vzdálenosti Bosch GLM 30. Konfigurace měření je znázorněna na obr. 40.

Obr. 40 Konfigurace měření pro ověření přesnosti určení vzdálenosti kamerou

Použité měřidlo Bosch GLM 30 disponuje rozsahem měření 0,15 m až 30 m. Přesnost samotného zařízení uváděná výrobcem je ±2 mm. [60] Další nepřesnost je do měření vnesena ručním použitím tohoto měřidla, kdy nebylo zaručeno zachování pevné vazby mezi kamerou a laserovým dálkoměrem při jejich přesunu mezi jednotlivými vzdálenostmi. Bylo však dbáno na dodržení počátku kamery a měřidla ve stejné rovině (obr. 41). Pro každou vzdálenost bylo zaznamenáno pět hodnot měření, z nichž byla následně stanovena průměrná hodnota a směrodatná odchylka.

Obr. 41 Měření vzdálenosti laserovým měřičem

Vzdálenost byla dálkoměrem měřena vzhledem k přední registrační značce vozidla, jejíž reflektivní povrch zvýšil využitelnost použitého laserového měřidla navzdory nepříznivým podmínkám způsobeným zejména dobou měření a počasím (slunečno v dopoledních hodinách). S ohledem na velikost měřeného objektu byly snímky pořízeny v rozsahu od 2,5 m do 40 m. V rozmezí 2,5 m až 10 m byla kamera posouvána o 2,5 m, poté byl interval zvýšen na 5 m, přičemž v každé vzdálenosti bylo z kamery uloženo pět různých snímků. Vzhledem k rozsahu použitého měřidla bylo ve vzdálenostech 35 m a 40 m měření provedeno k překážce umístěné ve vzdálenosti 30 m. I v těchto vzdálenostech bylo provedeno pět různých měření, kdy celková vzdálenost k měřenému objektu byla získána přičtením průměrné hodnoty měření ve vzdálenosti 30 m.

Snímky z kamery byly ve všech vzdálenostech pořízeny v HD rozlišení, tzn. 2560x720 pixelů pro spojený obraz obou kamer. Za účelem možnosti posouzení vlivu rozlišení na přesnost a rozsah určení vzdálenosti, byly každých 10 m pořízeny snímky ve všech dostupných rozlišeních kamery [41], tedy kromě zmíněného HD také v rozlišení 2K (4416x1242), Full HD (3840x1080) a VGA (1344x376).

5.1.2 ANALÝZA VLIVU PARAMETRŮ DISPARITY NA PŘESNOST URČENÍ VZDÁLENOSTI

Za účelem posouzení vlivu parametrů výpočtu disparity na přesnost disparitní mapy a tím i určení vzdálenosti byl vytvořen upravený Python skript vycházející z algoritmu pro určení vzdálenosti představeného v kapitole 3. Úprava spočívala v zajištění zpracování postupně všech snímků, tzn. pět snímků v HD rozlišení pro každou vzdálenost, a v následném uložení nastavených parametrů a hodnot vzdáleností do textového souboru. Dále bylo nutné zjistit souřadnice registrační značky vozidla v každém z uložených snímků tak, aby určení vzdálenosti probíhalo ve stejném místě jako při měření laserovým dálkoměrem. Souřadnice byly zjištěny pomocí volně dostupného grafického editoru GIMP, který v otevřeném obrázku vypisuje aktuální polohu kurzoru v pixelech.

Zmíněnými parametry jsou zde myšleny zejména hodnoty rozsahu disparity (*num_disp*), velikost bloků pro porovnávání levého a pravého obrazu (*window_size*), parametry *P1mult, P2mult* určující míru vyhlazování disparitní mapy a dále pak parametry WLS filtru *lambda* a *sigma* mající vliv na ostrost přechodu hodnot disparity na hranách objektů.

Pro různé kombinace těchto parametrů byla vyhodnocována především relativní chyba aritmetického průměru hodnot vzdálenosti určených z pěti různých snímků pro danou vzdálenost vzhledem k aritmetickému průměru pěti hodnot vzdálenosti naměřených laserovým dálkoměrem.

Následuje příklad statistického zpracování stanovených hodnot vzdálenosti pro 2,5 m, 10 m a 20 m za použití výchozích hodnot parametrů tak, jak byly stanoveny v kódu v kapitole 3. Jedná se o následující hodnoty:

```
num_disp = 64, window_size = 9, P1mult = 8, P2mult = 32,
lambda = 8000, sigma = 1.5.
```

S touto konfigurací byly zjištěny hodnoty vzdáleností Z, které jsou spolu s hodnotami vzdáleností změřenými laserovým dálkoměrem Z_0 zaznamenány v tab. 1.

Tab. 1 Hodnoty vzdálenosti naměřené laserovým měřidlem a určené ze stereo obrazu

číslo měření		1	2	3	4	5
2,5 m	Z_0 [m]	2,502	2,503	2,505	2,504	2,504
	Z [m]	2,481	2,478	2,480	2,480	2,481
10 m	Z_0 [m]	10,003	10,003	10,005	10,006	10,004
	Z [m]	13,136	13,272	13,32	13,245	13,172
20 m	Z_0 [m]	20,003	20,002	20,003	20,004	20,005
	Z [m]	33,519	33,765	34,044	33,918	33,727

Následně byly pro měření laserem i určení algoritmem v jednotlivých vzdálenostech z naměřených hodnot stanoveny hodnoty aritmetického průměru $\overline{Z_0}$ a $\bar{Z}$ podle vztahu (10), kde *n* je počet provedených měření a je ve všech případech roven 5. S využitím aritmetických průměrů byly stanoveny výběrové směrodatné odchylky s_0 a s podle vztahu (11) a variační koeficienty v_0 a v podle vztahu (12). Výběrová směrodatná odchylka vyjadřuje odlišnost naměřených hodnot od jejich průměru a poskytuje tak informaci o ovlivnění měření náhodnými chybami. [61] Variační koeficient je relativním vyjádřením výběrové směrodatné odchylky vůči aritmetickému průměru vzdálenosti určené z kamery.

$$\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i \tag{10}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Z_i - \bar{Z})^2} \tag{11}$$

$$v = \frac{s}{\bar{Z}}100\ [\%] \tag{12}$$

Vypočtené hodnoty těchto statistických veličin jsou shrnuty v tab. 2.

Tab. 2 Statistické vyhodnocení provedeného měření

	2,5 m	10 m	20 m
$\overline{Z_0}$ [m]	2,5036	10,0042	20,0034
$\bar{Z}$ [m]	2,4800	13,2290	33,7946
s_0 [m]	0,0011	0,0013	0,0011
s [m]	0,0012	0,0746	0,1993
v_0 [-]	0,0455 %	0,0130 %	0,0057 %
v [-]	0,0494 %	0,5642 %	0,5896 %

Poté následovalo stanovení chyby vlastního určení vzdálenosti vytvořeným algoritmem vůči provedenému měření laserovým dálkoměrem. K tomu byla nejprve podle (13) stanovena velikost absolutní chyby ε_{abs} jako absolutní hodnota rozdílu aritmetických průměrů určení vzdálenosti z kamery $\bar{Z}$ a z měření dálkoměrem $\overline{Z_0}$. Z absolutní chyby pak byla podle (14)

stanovena relativní chyba δ jako poměr absolutní chyby ε_{abs} vůči aritmetickému průměru vzdáleností naměřených laserem $\overline{Z_0}$.

$$\varepsilon_{abs} = |\bar{Z} - \overline{Z_0}| \tag{13}$$

$$\delta = \frac{\varepsilon_{abs}}{\overline{Z_0}} 100\ [\%] \tag{14}$$

Výsledky těchto výpočtů jsou obsaženy v tab. 3.

Tab. 3 Vyhodnocení chyby určení vzdálenosti pro danou konfiguraci parametrů

	2,5 m	10 m	20 m
ε_{abs} [m]	0,0236	3,2248	13,7912
δ [-]	0,9426 %	32,2345 %	68,9443 %

Z výsledků analýzy pro tuto konkrétní konfiguraci vyplývá, že relativní chyba určení vzdálenosti dosahuje příznivé hodnoty nepřesahující 1 % pouze pro vzdálenost 2,5 m. S rostoucí vzdáleností drasticky narůstá i chyba měření, tato kombinace parametrů tak není pro rozsah nutný k aplikaci v dané oblasti použitelná.

Za účelem nalezení optimálních parametrů pro dosažení nejnižší možné chyby bylo kromě výchozího nastavení *konfig0* vytvořeno čtrnáct dalších konfigurací, které jsou shrnuty v tab. 4. Konfigurace byly navrženy tak, aby bylo možné co nejlépe samostatně porovnat vliv jednotlivých uvedených parametrů. Výše popsaná analýza byla provedena pro uvedené konfigurace ve všech vzdálenostech.

Tab. 4 Shrnutí vyhodnocovaných konfigurací parametrů pro výpočet disparity

číslo konfigurace	*num_disp*	*window_size*	*P1mult*	*P2mult*	*lambda*	*sigma*
konfig0	64	9	8	32	8000	1,5
konfig1	64	3	8	32	8000	1,5
konfig2	32	3	8	32	8000	1,5
konfig3	16	3	8	32	8000	1,5
konfig4	32	3	0	0	8000	1,5
konfig5	32	3	0	1	8000	1,5
konfig6	32	3	1	1	8000	1,5
konfig7	32	3	0	1	8000	2,0
konfig8	32	3	0	1	10000	2,0
konfig9	32	3	0	1	8000	1,6
konfig10	32	3	0	1	10000	1,5
konfig11	32	3	0	1	8000	2,2
konfig12	32	3	0	1	8000	2,1
konfig13	32	3	0	1	9000	1,9
konfig14	32	3	8	32	*bez filtru*	
konfig15	32	3	0	0	*bez filtru*	

Hodnoty všech vzdáleností změřených laserovým měřidlem jsou shrnuty v tab. 5. Tabulky tab. 6 až tab. 15 pak obsahují všechny vzdálenosti určené jednotlivými konfiguracemi.

Tab. 5 Vzdálenosti naměřené laserovým dálkoměrem

číslo měření	1	2	3	4	5
Z_0 [m]	2,502	2,503	2,505	2,504	2,504
	5,005	4,999	5,002	5,004	5,003
	7,500	7,498	7,497	7,503	7,499
	10,003	10,003	10,005	10,006	10,004
	15,005	15,001	15,005	15,004	15,005
	20,003	20,002	20,003	20,004	20,005
	24,997	24,998	25,000	25,005	24,998
	30,001	30,002	30,004	29,997	30,001
	35,038	35,039	35,037	35,037	35,037
	40,034	40,033	40,035	40,033	40,032

Tab. 6 Vzdálenosti určené z kamery ve vzdálenosti 2,5 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	2,481	2,482	2,482	4,369	2,504	2,499	2,499	2,544
	2,478	2,478	2,478	4,371	2,450	2,450	2,447	2,509
	2,480	2,477	2,477	4,357	2,443	2,460	2,428	2,522
	2,480	2,481	2,481	4,381	2,467	2,460	2,462	2,522
	2,481	2,477	2,477	4,415	2,498	2,480	2,481	2,529
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	2,562	2,506	2,506	2,574	2,558	2,540	2,515	2,632
	2,529	2,459	2,458	2,545	2,526	2,503	2,490	2,527
	2,543	2,469	2,468	2,560	2,540	2,515	2,466	2,688
	2,542	2,470	2,469	2,558	2,539	2,515	2,466	2,454
	2,547	2,488	2,487	2,559	2,543	2,524	2,472	2,478

Tab. 7 Vzdálenosti určené z kamery ve vzdálenosti 5 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	5,705	5,687	5,687	5,687	4,845	5,176	5,145	5,355
	5,686	5,673	5,672	5,673	4,882	5,056	5,066	5,169
	5,766	5,770	5,770	5,770	4,991	5,296	5,152	5,253
	5,419	5,391	5,391	5,391	4,834	4,909	4,938	5,027
	5,533	5,504	5,504	5,504	4,498	4,830	4,741	4,884
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	5,414	5,223	5,237	5,394	5,376	5,363	5,292	4,815
	5,230	5,081	5,112	5,204	5,187	5,182	5,292	4,861
	5,291	5,294	5,335	5,244	5,247	5,282	5,264	4,908
	5,089	4,915	4,927	5,092	5,061	5,022	5,292	4,908
	4,936	4,833	4,860	4,910	4,897	4,897	5,264	4,793

Tab. 8 Vzdálenosti určené z kamery ve vzdálenosti 7,5 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	9,497	9,408	9,408	9,410	6,491	7,291	7,217	7,336
	9,559	9,413	9,414	9,419	6,483	7,491	7,144	7,311
	9,493	9,369	9,370	9,374	6,021	7,106	6,735	7,209
	9,447	9,322	9,322	9,325	6,399	7,642	7,135	7,606
	9,526	9,392	9,392	9,394	6,404	7,411	7,099	7,500
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	7,388	7,298	7,350	7,357	7,346	7,356	8,610	9,072
	7,352	7,431	7,513	7,301	7,304	7,347	8,684	9,153
	7,296	7,123	7,199	7,249	7,230	7,236	8,128	8,912
	7,652	7,640	7,708	7,590	7,598	7,644	8,194	9,072
	7,570	7,428	7,494	7,526	7,514	7,526	8,538	9,072

Tab. 9 Vzdálenosti určené z kamery ve vzdálenosti 10 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	13,136	12,924	12,925	12,925	7,719	8,991	8,543	9,365
	13,272	13,017	13,017	13,017	7,672	8,985	8,292	9,282
	13,320	13,162	13,162	13,162	8,294	10,402	9,208	10,545
	13,245	13,056	13,056	13,057	8,228	9,551	8,963	9,692
	13,172	12,968	12,968	12,969	8,511	9,997	9,055	10,066
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	9,513	9,037	9,089	9,526	9,447	9,358	11,416	10,583
	9,408	8,992	9,055	9,439	9,363	9,266	11,546	10,583
	10,615	10,435	10,478	10,577	10,562	10,565	11,678	10,583
	9,805	9,548	9,623	9,788	9,739	9,706	11,678	10,583
	10,182	9,959	10,050	10,169	10,117	10,079	11,416	10,583

Tab. 10 Vzdálenosti určené z kamery ve vzdálenosti 15 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	22,310	21,848	21,848	21,848	13,301	16,692	16,484	15,822
	22,761	22,431	22,431	22,432	14,530	17,544	17,562	16,394
	22,495	22,227	22,227	22,227	13,757	16,751	16,306	15,981
	22,391	22,034	22,034	22,034	13,704	16,334	15,956	15,472
	22,625	22,306	22,306	22,306	14,990	17,886	17,672	16,905
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	15,511	16,559	16,562	15,448	15,632	15,871	19,922	15,631
	16,062	17,348	17,394	15,943	16,162	16,477	20,735	19,922
	15,673	16,644	16,626	15,616	15,797	16,021	20,735	19,170
	15,174	16,218	16,215	15,118	15,289	15,522	19,922	18,473
	16,553	17,766	17,772	16,432	16,666	16,983	19,922	16,127

Tab. 11 Vzdálenosti určené z kamery ve vzdálenosti 20 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
	33,519	32,625	32,657	32,663	18,977	23,100	22,823	18,331
	33,765	32,648	32,661	32,683	17,830	21,201	20,375	17,191
Z [m]	34,044	33,029	33,041	33,053	18,840	21,921	21,339	17,442
	33,918	33,492	33,502	33,526	17,545	21,434	20,206	16,903
	33,727	32,913	32,921	32,945	17,436	22,363	21,175	17,755
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
	17,431	22,059	22,394	17,055	17,652	18,625	31,750	22,087
	16,404	20,186	20,420	16,178	16,658	17,367	31,750	21,167
Z [m]	16,514	20,971	21,169	16,206	16,783	17,692	30,788	26,737
	16,071	20,366	20,693	15,812	16,317	17,135	31,750	29,029
	16,857	21,316	21,560	16,548	17,113	18,000	29,029	22,087

Tab. 12 Vzdálenosti určené z kamery ve vzdálenosti 25 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
	49,444	48,220	48,220	48,226	29,482	33,506	32,938	26,808
	48,222	46,121	46,121	46,131	30,632	34,932	33,813	27,393
Z [m]	47,480	45,394	45,394	45,411	26,957	30,337	28,933	24,077
	46,819	45,617	45,617	45,620	28,566	32,745	31,684	25,807
	47,403	45,895	45,895	45,896	27,051	30,973	29,757	25,175
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
	25,368	32,151	32,389	24,712	25,707	27,255	50,801	31,750
	25,837	33,361	33,726	25,180	26,224	27,863	46,183	35,035
Z [m]	22,842	28,862	29,110	22,418	23,197	24,397	48,382	32,775
	24,456	31,021	31,344	24,070	24,887	26,118	48,382	92,365
	23,867	29,664	29,789	23,494	24,293	25,449	53,475	92,365

Tab. 13 Vzdálenosti určené z kamery ve vzdálenosti 30 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
	60,304	60,672	60,674	60,670	27,448	33,305	33,434	28,759
	60,272	59,762	59,762	59,745	29,364	34,094	33,879	29,523
Z [m]	61,441	60,913	60,914	60,912	27,293	36,236	32,405	30,342
	63,602	63,681	63,682	63,678	31,920	38,391	39,960	33,044
	64,038	66,689	66,691	66,685	30,681	39,082	40,184	31,847
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
	27,723	32,270	32,472	27,323	28,021	29,022	63,501	78,155
	28,527	32,973	33,173	28,174	28,829	29,761	63,501	inf
Z [m]	29,213	34,695	35,028	28,713	29,501	30,666	67,734	inf
	31,702	37,213	37,298	31,193	32,102	33,345	63,501	78,155
	30,485	37,334	37,749	29,757	30,761	32,318	72,573	101,602

Tab. 14 Vzdálenosti určené z kamery ve vzdálenosti 35 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	63,677	64,006	64,006	64,005	71,553	78,215	54,564	41,040
	65,206	70,177	70,174	70,173	37,316	42,836	41,892	31,731
	63,778	66,530	66,529	66,527	44,410	48,549	43,901	32,338
	62,737	65,147	65,144	65,144	36,349	45,378	37,075	33,573
	63,844	66,291	66,291	66,290	133,358	141,067	69,834	65,331
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	38,795	66,183	73,484	35,112	37,826	43,662	63,501	72,573
	30,209	39,672	40,685	29,204	30,412	32,375	67,734	92,365
	30,894	43,221	45,758	29,538	30,840	33,300	63,501	inf
	32,305	41,893	43,436	31,014	32,233	34,398	63,501	92,365
	60,484	123,452	136,271	50,033	56,855	73,068	63,501	92,365

Tab. 15 Vzdálenosti určené z kamery ve vzdálenosti 40 m pro všechny konfigurace

konfigurace	*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
Z [m]	78,836	136,340	136,335	136,335	76,756	98,494	103,406	35,814
	75,546	84,976	84,974	84,977	52,087	56,081	57,489	33,234
	71,069	73,317	73,314	73,319	51,296	55,250	55,727	30,388
	75,272	79,315	79,313	79,313	45,323	48,650	49,914	28,999
	73,992	84,842	84,841	84,841	80,531	84,321	83,348	36,877
konfigurace	*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
Z [m]	31,948	78,917	90,496	27,811	31,208	39,579	84,668	53,475
	30,137	51,307	53,842	27,581	30,121	35,262	78,155	53,475
	27,427	49,736	52,582	25,577	27,660	32,159	72,573	63,501
	26,707	44,022	46,493	24,457	26,487	30,746	72,573	56,445
	32,929	74,790	81,048	27,969	31,772	41,119	78,155	59,766

Následující dvě tabulky tab. 16 a tab. 17 obsahují výsledné variační koeficienty pro všechny konfigurace v jednotlivých vzdálenostech i jako průměrné hodnoty napříč všemi vzdálenostmi. Podmíněné formátování je aplikováno vždy pro každý řádek samostatně, aby bylo názorněji vystiženo srovnání konfigurací v jednotlivých vzdálenostech.

Tab. 16 Variační koeficienty stanovení vzdálenosti pro konfigurace *konfig0* až *konfig7*

konfigurace		*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
v [-]	2,5 m	0,049 %	0,095 %	0,095 %	0,504 %	1,117 %	0,795 %	1,130 %	0,505 %
	5 m	2,529 %	2,743 %	2,741 %	2,743 %	3,849 %	3,763 %	3,445 %	3,617 %
	7,5 m	0,438 %	0,395 %	0,397 %	0,398 %	3,052 %	2,746 %	2,688 %	2,146 %
	10 m	0,564 %	0,700 %	0,698 %	0,697 %	4,587 %	6,497 %	4,326 %	5,343 %
	15 m	0,802 %	1,039 %	1,039 %	1,040 %	4,878 %	3,795 %	4,609 %	3,426 %
	20 m	0,590 %	1,071 %	1,054 %	1,065 %	4,031 %	3,451 %	4,907 %	3,136 %
	25 m	2,109 %	2,455 %	2,455 %	2,452 %	5,537 %	5,756 %	6,570 %	5,080 %
	30 m	2,896 %	4,549 %	4,550 %	4,554 %	6,864 %	7,028 %	10,511 %	5,663 %
	35 m	1,382 %	3,497 %	3,496 %	3,496 %	63,481 %	58,389 %	26,418 %	34,831 %
	40 m	3,747 %	27,658 %	27,658 %	27,656 %	26,460 %	31,569 %	32,413 %	10,236 %
	průměr	1,511 %	4,420 %	4,418 %	4,460 %	12,385 %	12,379 %	9,702 %	7,398 %

Tab. 17 Variační koeficienty stanovení vzdálenosti pro konfigurace *konfig8* až *konfig15*

konfigurace		*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
v [-]	2,5 m	0,465 %	0,752 %	0,767 %	0,402 %	0,450 %	0,545 %	0,846 %	3,938 %
	5 m	3,560 %	3,869 %	3,944 %	3,494 %	3,548 %	3,687 %	0,290 %	1,084 %
	7,5 m	2,039 %	2,580 %	2,558 %	1,984 %	2,063 %	2,180 %	2,998 %	0,971 %
	10 m	5,027 %	6,415 %	6,371 %	4,780 %	5,053 %	5,480 %	1,135 %	0,000 %
	15 m	3,361 %	3,738 %	3,811 %	3,188 %	3,312 %	3,506 %	2,199 %	10,590 %
	20 m	3,101 %	3,602 %	3,654 %	2,860 %	2,991 %	3,277 %	3,821 %	14,284 %
	25 m	4,873 %	5,870 %	6,014 %	4,502 %	4,787 %	5,292 %	5,629 %	57,046 %
	30 m	5,355 %	6,712 %	6,743 %	5,154 %	5,406 %	5,765 %	6,084 %	55,893 %
	35 m	33,036 %	56,478 %	59,506 %	24,976 %	29,619 %	39,692 %	2,942 %	45,783 %
	40 m	9,137 %	26,626 %	30,126 %	5,900 %	7,765 %	12,630 %	6,488 %	7,528 %
	průměr	6,995 %	11,664 %	12,349 %	5,724 %	6,499 %	8,205 %	3,243 %	19,712 %

Dále pak byly uvedeným postupem stanoveny relativní chyby určení vzdálenosti opět pro všechny případy i průměrně napříč vzdálenostmi. Výsledky jsou shrnuty v tab. 18 a tab. 19.

Tab. 18 Relativní chyby stanovení vzdálenosti pro konfigurace *konfig0* až *konfig7*

konfigurace		*konfig0*	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
δ [-]	2,5 m	0,943 %	0,983 %	0,983 %	74,892 %	1,246 %	1,350 %	1,606 %	0,863 %
	5 m	12,378 %	12,042 %	12,038 %	12,042 %	3,850 %	1,015 %	0,116 %	2,699 %
	7,5 m	26,735 %	25,087 %	25,093 %	25,135 %	15,199 %	1,483 %	5,779 %	1,427 %
	10 m	32,234 %	30,199 %	30,201 %	30,205 %	19,186 %	4,188 %	11,915 %	2,141 %
	15 m	50,069 %	47,755 %	47,755 %	47,757 %	6,316 %	13,579 %	11,943 %	7,403 %
	20 m	68,944 %	64,679 %	64,754 %	64,842 %	9,387 %	10,000 %	5,900 %	12,393 %
	25 m	91,497 %	85,001 %	85,001 %	85,030 %	14,152 %	29,996 %	25,702 %	3,410 %
	30 m	106,431 %	107,804 %	107,808 %	107,786 %	2,199 %	20,735 %	19,904 %	2,340 %
	35 m	82,228 %	89,597 %	89,593 %	89,590 %	84,365 %	103,236 %	41,143 %	16,454 %
	40 m	87,201 %	129,204 %	129,197 %	129,201 %	52,869 %	71,255 %	74,796 %	17,413 %
	průměr	55,866 %	59,235 %	59,242 %	66,648 %	20,877 %	25,684 %	19,880 %	6,654 %

Tab. 19 Relativní chyby stanovení vzdálenosti pro konfigurace *konfig8* až *konfig15*

konfigurace		*konfig8*	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
δ [-]	2,5 m	1,638 %	1,007 %	1,039 %	2,221 %	1,502 %	0,631 %	0,871 %	2,085 %
	5 m	3,786 %	1,331 %	1,831 %	3,322 %	3,018 %	2,930 %	5,561 %	2,910 %
	7,5 m	0,637 %	1,539 %	0,621 %	1,264 %	1,347 %	1,035 %	12,420 %	20,759 %
	10 m	0,996 %	4,098 %	3,451 %	1,044 %	1,585 %	2,093 %	15,420 %	5,786 %
	15 m	5,269 %	12,683 %	12,729 %	4,715 %	6,033 %	7,803 %	34,945 %	19,066 %
	20 m	16,737 %	4,880 %	6,218 %	18,215 %	15,491 %	11,196 %	55,041 %	21,086 %
	25 m	2,102 %	24,049 %	25,088 %	4,099 %	0,552 %	4,867 %	97,782 %	127,436 %
	30 m	1,570 %	16,319 %	17,143 %	3,230 %	0,527 %	3,405 %	120,533 %	186,559 %
	35 m	9,989 %	79,476 %	93,868 %	0,164 %	7,408 %	23,754 %	83,653 %	149,495 %
	40 m	25,488 %	49,261 %	62,095 %	33,358 %	26,437 %	10,642 %	92,901 %	43,211 %
	průměr	6,821 %	19,464 %	22,408 %	7,163 %	6,390 %	6,836 %	51,913 %	57,839 %

Z výše uvedených tabulek je zřejmé, že využití filtrace pomocí WLS filtru je pro stanovení přesné disparitní mapy stěžejní. Konfigurace *konfig14* a *konfig15*, ve kterých není provedena filtrace, vykazují špatné hodnoty vyhodnocovaných charakteristik napříč takřka všemi vzdálenostmi.

Stejně důležitá jako užití filtrace ve vhodném rozsahu je aplikace vyhlazování v adekvátní míře určením parametrů *P1mult* a *P2mult*. Ve variantách *konfig0* až *konfig3* je oproti dalším konfiguracím těmito parametry určena příliš vysoká míra vyhlazování, což zapříčiňuje podobně nežádoucí výsledky jako nevhodně nastavený filtr.

Jako nejlepší přístup se ukázalo použití velmi malé míry vyhlazování s parametry *P1mult* = 0 a *P2mult* = 1 ve spojení s hodnotami parametrů WLS filtru blízkými hodnotám *lambda* = 8000 a *sigma* = (0,8 ÷ 2,0) doporučeným v dokumentaci OpenCV třídy tohoto filtru [47]. Konfigurace *konfig5* a *konfig7* až *konfig13* se odlišují jen změnou parametrů *lambda* a *sigma* a z výsledků je zřejmé, že změna těchto hodnot má na výslednou chybu významný vliv. To dokládá snížení hodnoty *sigma* v konfiguracích *konfig5, konfig9* a *konfig10*, které u nich zapříčinilo pokles chyby ve vzdálenosti 20 m. Určení vzdálenosti v této vzdálenosti je naopak u konfigurací s vyšší hodnotou *sigma* zatíženo vyšší chybou, avšak zejména ve vyšších vzdálenostech je vzdálenost určena přesněji. Nastavení jednotlivých parametrů tak má v různých vzdálenostech protichůdný účinek a je třeba nalézt konfiguraci s nejoptimálnějšími výsledky ve všech zkoumaných vzdálenostech.

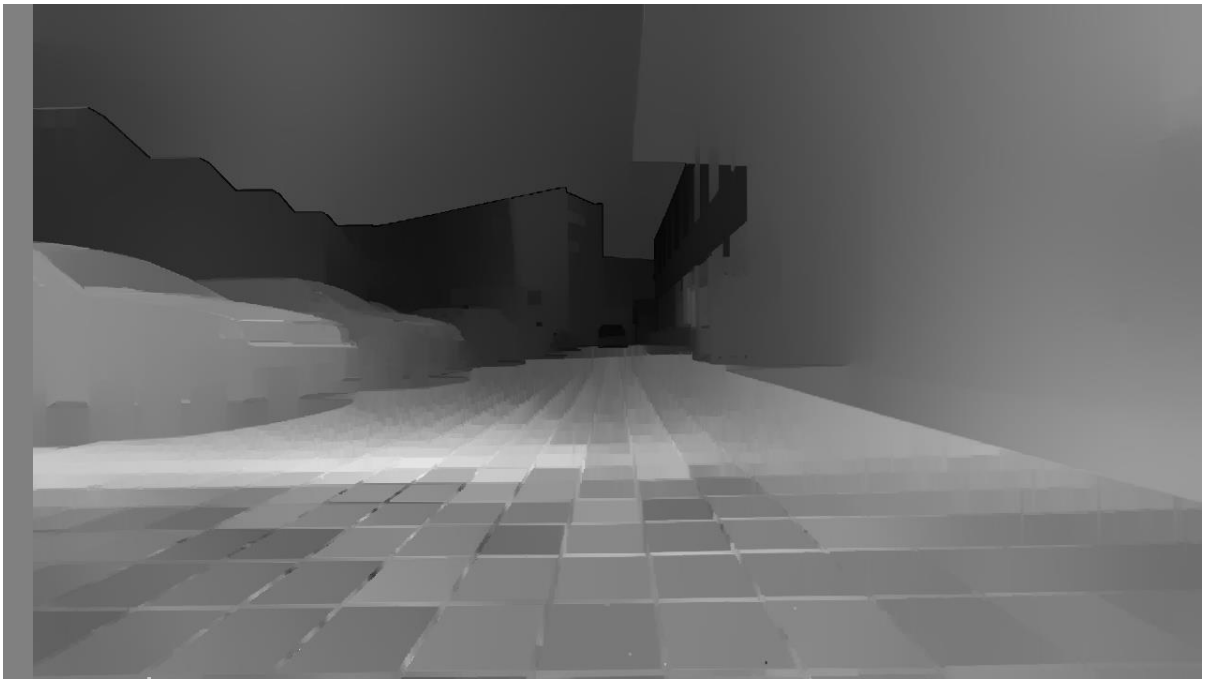
U parametru *win_size* pak bylo při jeho snížení mezi konfiguracemi *konfig0* a *konfig1* zjištěno zvýšení přesnosti. V případě rozsahu disparity *num_disp* nebyla při snížení z hodnoty 64 na 32 zjištěna výrazná změna vyhodnocovaných charakteristik. Při nastavení hodnoty 16 v konfiguraci *konfig3* však došlo k zásadnímu nárůstu relativní chyby v nejbližší vzdálenosti 2,5 m. To je dáno nepřímou závislostí vzdálenosti na hodnotě disparity. Čím je hodnota disparity vyšší, tím je menší vzdálenost. Pokud je však parametrem *num_disp* maximální hodnota disparity omezena, není možné stanovit vzdálenost bližší než odpovídající této maximální disparitě. Jelikož parametr *num_disp* je třeba udávat v násobcích 16 [46], hodnota 32 byla stanovena jako nejvhodnější.

Za účelem zvolení konfigurace s nejoptimálnějším poměrem variačního koeficientu a relativní chyby byl vypočten průměr z průměrných hodnot obou těchto parametrů pro všechny konfigurace. Tyto hodnoty jsou obsaženy v tab. 20. Vyhodnocením těchto hodnot byla jako nejvhodnější stanovena varianta *konfig12*. Ačkoliv zkoumaná metrika této varianty dosahuje mírně nepatrně vyšší hodnoty než v případě konfigurace *konfig11*, bylo přihlédnuto k zamýšlené aplikaci. Konfigurace *konfig12* oproti druhé zmíněné menší hodnoty relativní chyby zejména ve vzdálenostech nad 20 m, což je pro rozšíření funkčního rozsahu výhodnější. Dosažené hodnoty relativní chyby dle výsledků v tab. 19 s jistými odchylkami odpovídají hodnotám uváděným výrobcem kamery, tedy méně než 1 % do vzdálenosti 3 m a méně než 5 % do vzdálenosti 15 m. Ve vzdálenostech 2,5 m a 15 m je u této konfigurace chyba vyšší, avšak ve vzdálenostech nad 20 m dosahuje relativní chyba nízkých hodnot.

Tab. 20 Průměrné hodnoty průměrů variačního koeficientu a relativní chyby

konfig0	*konfig1*	*konfig2*	*konfig3*	*konfig4*	*konfig5*	*konfig6*	*konfig7*
28,688 %	31,828 %	31,830 %	35,554 %	16,631 %	19,031 %	14,791 %	7,026 %
konfig8	*konfig9*	*konfig10*	*konfig11*	*konfig12*	*konfig13*	*konfig14*	*konfig15*
6,908 %	15,564 %	17,379 %	6,444 %	6,445 %	7,521 %	27,578 %	38,775 %

Jak lze pozorovat na obr. 42, disparitní mapa vypočtená s využitím parametrů zvolené konfigurace pro vzdálenost 30 m velmi dobře kopíruje hrany jednotlivých objektů, zároveň však díky vyhlazení není citlivá na detaily v rámci těchto objektů, jako jsou například okna budov.



Obr. 42 Výsledná disparitní mapa ve vzdálenosti 30 m pro konfiguraci *konfig12*

5.1.3 POSOUZENÍ ZMĚNY PŘESNOSTI PŘI RŮZNÝCH ROZLIŠENÍCH KAMERY

Dále byla s využitím zvolených optimálních parametrů z předchozí kapitoly provedena analýza vlivu rozlišení kamery na přesnost určení vzdálenosti. Snímky ve všech rozlišeních kamery byly pořízeny ve vzdálenostech 10 m, 20 m, 30 m a 40 m. Hodnoty vzdáleností určené algoritmem v těchto vzdálenostech pro jednotlivá rozlišení jsou shrnuty v tab. 21.

Tab. 21 Hodnoty vzdáleností pro různá rozlišení snímků z kamery

	10 m				20 m			
rozlišení	HD	2K	FHD	VGA	HD	2K	FHD	VGA
Z [m]	9,447	11,395	11,754	9,628	17,652	17,264	23,483	5,864
	9,363	11,492	12,177	9,660	16,658	17,265	24,273	6,976
	10,562	11,478	11,472	9,632	16,783	18,609	25,473	7,479
	9,739	11,280	11,924	9,312	16,317	19,680	23,239	7,416
	10,117	11,226	11,936	9,921	17,113	18,560	24,177	11,982
	30 m				40 m			
rozlišení	HD	2K	FHD	VGA	HD	2K	FHD	VGA
Z [m]	28,021	36,035	39,072	11,387	31,208	47,855	40,097	9,406
	28,829	32,960	39,021	9,872	30,121	36,364	41,737	9,367
	29,501	33,756	37,143	10,030	27,660	34,37	44,106	8,79
	32,102	34,535	36,268	9,698	26,487	35,075	40,903	10,072
	30,761	34,609	35,842	10,425	31,772	33,81	42,308	9,291

Na základě těchto hodnot bylo provedeno statistické vyhodnocení shodným postupem jako v předešlé kapitole. Výsledné variační koeficienty v obsahuje tab. 22, shrnutí relativních chyb δ pak nabízí tab. 23.

Tab. 22 Variační koeficienty při různých rozlišeních kamery

rozlišení		HD	2K	FHD	VGA
v [-]	10 m	5,053 %	1,040 %	2,200 %	2,243 %
	20 m	2,991 %	5,613 %	3,612 %	29,567 %
	30 m	5,406 %	3,323 %	4,042 %	6,550 %
	40 m	7,765 %	15,655 %	3,641 %	4,870 %
	průměr	5,304 %	6,408 %	3,374 %	10,807 %

Tab. 23 Relativní chyby určení vzdálenosti při různých rozlišeních kamery

rozlišení		HD	2K	FHD	VGA
δ [-]	10 m	1,585 %	13,694 %	18,476 %	3,734 %
	20 m	15,491 %	8,638 %	20,624 %	60,290 %
	30 m	0,527 %	14,593 %	24,893 %	65,726 %
	40 m	26,437 %	6,341 %	4,488 %	76,557 %
	průměr	11,010 %	10,816 %	17,121 %	51,577 %

Při nastavení rozlišení 2K, které je nejvyšší dostupné, lze pozorovat snížení relativní chyby ve vzdálenosti 20 m a 40 m, avšak ve vzdálenostech 10 m a 30 m naopak u relativní chyby došlo k nárůstu. V nejbližší zkoumané vzdálenosti 10 m pak u obou vyšších rozlišení dochází k výraznějšímu vzrůstu relativní chyby než u nejnižšího rozlišení VGA. To může vypovídat o nastavení příliš nízkého rozsahu disparity, kdy v důsledku většího rozlišení je dosahováno vyšších hodnot disparity, které se v menších vzdálenostech blíží definované maximální hodnotě. U rozlišení VGA naopak výrazněji roste variační koeficient i relativní chyba ve vzdálenostech 20 m a 30 m, což je nejspíš způsobeno nedostatečnou rozlišovací schopností, kdy měřený objekt už je v obrazu na dané rozlišení příliš malý.

Z provedené analýzy vyplývá, že nastavení parametrů výpočtu disparity je specifické pro dané rozlišení kamery. Při měření vzdálenosti pomocí stereo kamery je tak třeba nejprve zvolit vhodné rozlišení s ohledem na požadovanou rozlišovací schopnost.

5.2 ZHODNOCENÍ PŘESNOSTI DETEKCE OBJEKTŮ S URČENÍM VZDÁLENOSTI

Přesnost detekce objektů, správné určení jejich vzdálenosti a výpis vzdálenosti ve zpracovaném obrazu byl vyhodnocen na 10 minut dlouhém záznamu jízdy, který byl pořízen stereo kamerou umístěnou ve vozidle. Kamera byla napájena z připojeného notebooku, pomocí kterého byl i prostřednictvím vestavěné aplikace Kamera operačního systému Windows 10 uložen samotný záznam. Nahrávka byla vytvořena při nastaveném rozlišení HD (1280x720 pixelů pro levou i pravou kameru).

Z tohoto záznamu pak byly vybrány klíčové snímky tak, aby byly postihnuty všechny třídy objektů zahrnuté v testovacím souboru dat. Těmi jsou chodec, jezdec, osobní automobil, autobus, nákladní automobil, kolo, motocykl, semafor, dopravní značka a vlak. Tyto snímky jsou využity ke srovnání jednotlivých variant zkoumaných architektur neuronových sítí, kdy je sledováno správné rozpoznání objektů všech tříd.

Je však třeba připomenout závěry získané zhodnocením výsledků trénování neuronové sítě v kapitole 4.3.2, kde bylo vyhodnoceno nedostatečné natrénování na detekci objektů třídy vlak u obou architektur. To bylo způsobeno příliš nízkým počtem výskytů objektů této třídy v trénovacím souboru dat. Nelze tedy předpokládat správné rozpoznání objektů této třídy, není však očekáván negativní vliv na detekci objektů jiných tříd.

Ze srovnání architektur YOLOv5s6 a YOLOv5n6 je následně posouzením rozpoznaných objektů jednotlivých tříd zvolena spolehlivější varianta. Pro vybranou variantu je pak ze zmíněného záznamu vytvořen širší testovací soubor obsahující každý 300. snímek z videa. V každém snímku je zaznamenán počet vyskytujících se objektů jednotlivých tříd a po zpracování těchto snímků vytvořeným algoritmem je vyhodnocena úspěšnost rozpoznání objektů těchto tříd.

5.2.1 Ověření varianty s architekturou YOLOv5s6

Nejprve byla přesnost detekce zkoumána na variantě, u které byla použita architektura neuronové sítě YOLOv5s6. Vytvořený Python skript *detectdist.py* pro detekci objektů s určením vzdálenosti byl spuštěn příkazem

```
python detectdist.py --source data/images/drive --weights best.pt
--conf-thres 0.4 --save-txt --save-conf,
```

kde je volbou `--source` specifikován adresář obsahující zvolené snímky, které mají být zpracovány a parametrem `--weights` je upřesněn soubor *best.pt* s natrénovanými vahami neuronové sítě YOLOv5s6. Možností `--conf-thres` je nastavena minimální hranice pravděpodobnosti třídy detekovaného objektu.

Zbylé dvě volby `--save-txt` a `--save-conf` definují uložení výstupu v textové podobě, kdy pro každý zpracovaný snímek je vytvořen textový soubor obsahující třídu, souřadnice a pravděpodobnost každého detekovaného objektu na daném snímku.

Na obr. 43 jsou zobrazeny výstupní snímky pro tuto konfiguraci. Z výstupu v podobě zpracovaných snímků je patrná správná detekce objektů všech tříd s výjimkou třídy vlak, jejíž nedostatečné natrénování již bylo zmíněno. Za objekt třídy vlak by bylo možné považovat tramvaj v pravé části prvního snímku, zde byla však zařazena nesprávně do třídy autobus.

Přesnost určení vzdálenosti při zamýšlené aplikaci v reálném provozu nelze přesně vyčíslit, lze však konstatovat, že hodnoty uváděné u detekovaných objektů ve většině případů přibližně odpovídají odhadu a logickému vnímání. To znamená, že v případě zjevně bližších objektů je určena nižší hodnota vzdálenosti než u objektů vzdálenějších od kamery.

Obr. 43 Snímky zpracované algoritmem při použití architektury YOLOv5s6

5.2.2 OVĚŘENÍ VARIANTY S ARCHITEKTUROU YOLOV5N6

Stejné zpracování bylo provedeno i s využitím vah natrénované architektury YOLOv5n6, a to pomocí následujícího příkazu, kde použití zmíněné architektury bylo upřesněno odkázáním na soubor *best_nano.pt*.

```
python detectdist.py --source data/images/drive --weights best_nano.pt
--conf-thres 0.4 --save-txt --save-conf
```

Výstup v podobě obrázků zpracovaných touto konfigurací je k vidění na obr. 44. Za využití architektury neuronové sítě s menším počtem parametrů je při detekci objektů možné pozorovat vetší množství nedostatků, a to zejména zcela chybějící rozpoznání objektů tříd chodec, jezdec a kolo. Objekty by patrně byly rozpoznány nastavením nižší hranice pravděpodobnosti, což však vypovídá o menší spolehlivosti detekce.

Obr. 44 Snímky zpracované algoritmem při použití architektury YOLOv5n6

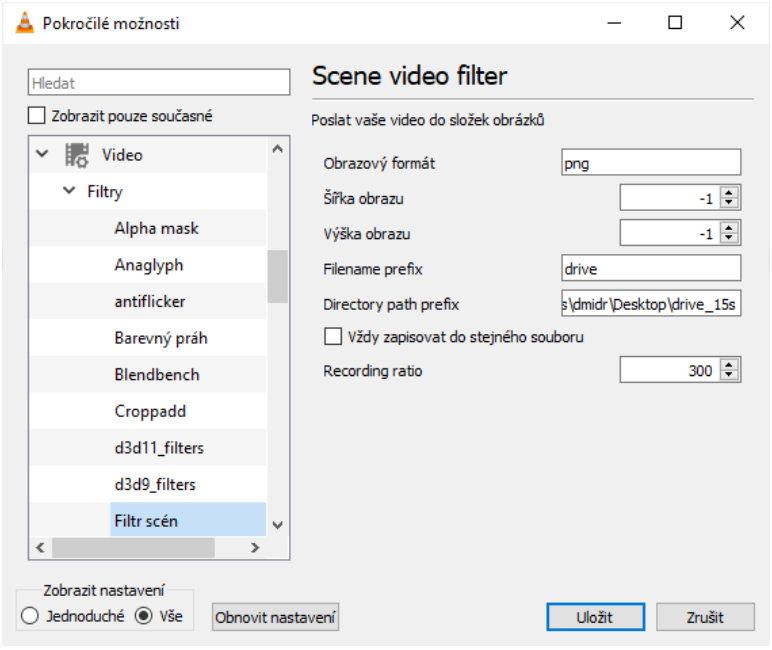
Ze srovnání dvou výše uvedených architektur pro detekci objektů plyne, že natrénované váhy neuronové sítě YOLOv5n6 obsažené v souboru *best_nano.pt* nejsou pro spolehlivou detekci objektů dostačující. Dále byl v této části potvrzen očekávaný nedostatek v podobě nesprávné detekce objektů třídy vlak.

5.2.3 VYHODNOCENÍ SPOLEHLIVOSTI DETEKCE OBJEKTŮ

Spolehlivost rozpoznání objektů byla vyhodnocena s využitím souboru snímků pořízených ze stejného záznamu. V záznamu byl uložen každý 300. snímek, což při vzorkovací frekvenci 30 FPS a délce záznamu 600 s znamená uložení 60 snímků v intervalu 10 s.

K tomu byl využit nástroj Filtr scén zahrnutý v multimediálním přehrávači VLC [62], který zajišťuje uložení snímků do souboru obrázku v definovaném intervalu. Tento filtr se nachází v okně Nástroje-Nastavení-Zobrazit nastavení-Vše-Filtry-Filtr scén. Použité nastavení filtru je znázorněno na obr. 45.

Pořízené snímky pak byly ručně zpracovány a byl zaznamenán počet objektů jednotlivých tříd vyskytující se v každém snímku.



Obr. 45 Nastavení přehrávače VLC pro uložení snímků z videa

Následně byly tyto snímky zpracovány vytvořeným algoritmem pomocí následujícího příkazu.

```
python detectdist.py --source data/images/drive_10s --weights best.pt
--conf-thres 0.4 --save-txt --save-conf
```

Po dokončení zpracování každého snímku je do příkazového řádku vypsán počet rozpoznaných objektů jednotlivých tříd, čehož bylo v této části práce využito. Pro první snímek byl výpis této části následující.

```
image 1/60 drive00001.png: 384x640 2 Pedestrians, 3 Cars, 2 T_signs
```

Následně byl vyhodnocen počet nerozpoznaných objektů *FN* jako rozdíl počtu objektů daných tříd určeného ručním zpracováním a počtu stanoveného neuronovou sítí, avšak pouze v případech, kdy počet objektů určených neuronovou sítí nepřevyšoval počet objektů zjištěný ručním zpracováním snímků. Dále byl určen počet falešně rozpoznaných objektů *FP*. Ten je určen jako rozdíl počtu objektů určených detekcí a počtu zjištěných objektů ručním zpracováním, což je podmíněno opačnou podmínkou než v předchozím případě.

S využitím těchto hodnot byla následně stanovena úspěšnost detekce jako poměr správně rozpoznaných snímků *TP*, který je roven rozdílu ručně zjištěných objektů a počtu *FN*, ku počtu ručně zjištěných objektů. Jedná se o formu metriky *R* (recall) ze vztahu (5) v kapitole 1.4.2 zjednodušenou o vyhodnocení správné detekce, které zde není provedeno na základě překrytí rámečků *IOU*, ale manuální vizuální kontrolou.

Dále bylo vyjádřen poměr chybně rozpoznaných objektů jako podíl počtu *FP* ku počtu objektů zjištěnému manuální analýzou. Výsledek pak názorně ukazuje, jak moc jsou jednotlivé třídy náchylné k detekci falešného objektu, který se však ve skutečnosti v obrazu nevyskytuje.

Výsledné průměrné hodnoty napříč všemi 60 snímky obou těchto statistik jsou pro jednotlivé třídy vyčísleny v tab. 24 spolu s celkovým počtem objektů skutečně se vyskytujících ve všech snímcích. V tabulce nejsou zobrazeny třídy jezdce, kola a motocyklu, jelikož pro tyto třídy nebyl v žádném snímku nalezen při ruční analýze žádný objekt a ani nebyl v žádném případě chybně rozpoznán objekt těchto tříd neuronovou sítí.

Tab. 24 Úspěšnost a chybnost detekce objektů pro jednotlivé třídy

třída	chodec	osobní automobil	autobus	nákladní vozidlo	semafor	dopravní značka	vlak
výskyt	19	187	6	23	21	90	3
podíl správně rozpoznaných objektů	52 %	81 %	38 %	91 %	63 %	81 %	0 %
podíl chybných detekcí	0 %	10 %	0 %	7 %	1 %	15 %	0 %

Z těchto výsledků plyne, že nejlépe si z hlediska procenta správně detekovaných objektů neuronová síť vede v případě třídy nákladní vozidlo. Nejhorší výsledek v tomto ohledu vykazuje třída vlak, což je na základě dříve uvedených závěrů v souladu s očekáváním. Nízká úspěšnost byla zjištěna i u třídy autobus, kde je však výsledek zkreslen nízkým počtem výskytů objektů této třídy, podobný případ nastává u třídy chodec.

Ačkoliv třídy chodec a autobus vykazují nejnižší podíl správně rozpoznaných objektů, nedošlo u nich k mylnému výskytu. Hodnoty podílu rozpoznání falešných objektů jsou u obou těchto tříd 0 %. Naopak největší podíl chybných výskytů mají objekty třídy dopravní značka, což lze vysvětlit záměnou reklamních billboardů za objekty této třídy. Nejvíce reprezentativním vzorkem jsou objekty třídy osobní automobil, kdy ze 187 skutečně zjištěných objektů rozpoznala neuronová síť správně 81 % a 10 % detekcí bylo falešných. Celkově tyto výsledky však mají vzhledem k nízkému počtu výskytů objektů některých tříd pouze nastínit provoz detekce objektů v reálném provozu. Je proto třeba zohlednit i výsledky získané validací trénování neuronové sítě, které jsou shrnuty v kapitole 4.3.2.

5.3 DOBA ZPRACOVÁNÍ SNÍMKU DÍLČÍMI ČÁSTMI ALGORITMU

Díky snímání a následnému výpisu doby zpracování stereo obrazu do příkazového řádku bylo možné vyhodnotit poměr výpočetní náročnosti dílčích částí algoritmu, tj. detekce objektů a určení vzdálenosti. Tyto údaje byly zaznamenány při provedení předchozí analýzy, kdy byl algoritmus spuštěn na stolním počítači vybaveném procesorem AMD Ryzen 5, 16 GB operační paměti RAM a grafickou kartou NVIDIA GeForce GTX 1080.

Po dokončení zpracování všech snímků definovaných příkazem v předchozí kapitole je do příkazové řádky vypsána průměrná doba zpracování jednotlivých částí na jeden snímek. Mezi tyto části patří v první řadě předzpracování snímků, následně samotná detekce objektů, uplatnění techniky NMS, a nakonec zpracování stereo obrazu a určení vzdálenosti objektů.

V případě použití objemnější architektury YOLOv5s6 měl výsledný výpis následující podobu.

```
Speed: 1.5ms pre-process, 128.9ms inference, 11.5ms NMS, 396.6ms depth per
image at shape (1, 3, 640, 640).
```

Pro menší architekturu YOLOv5n6 byl pak výsledek tento:

```
Speed: 0.8ms pre-process, 59.1ms inference, 2.2ms NMS, 390.3ms depth per
image at shape (1, 3, 640, 640).
```

Doba provedení detekce objektů je v prvním případě 128,9 ms, ve druhém 59,1 ms. Příslušné doby zpracování stereo obrazu jsou pak 396,6 ms a 390,3 ms.

Závěr z uvedených výsledků je takový, že při použití menší architektury je detekce objektů rychlejší o přibližně 54 %. V obou případech však podstatně delší dobu trvá zpracování stereo obrazu. To je zapříčiněno nevyužitím grafického procesoru pro zpracování stereo obrazu. V této oblasti je tak velký potenciál pro optimalizaci výpočetní doby. S využitím GPU modulu knihovny OpenCV využívajícího platformu CUDA by bylo možné docílit přibližně sedminásobného zkrácení doby zpracování stereo obrazu. [63]

5.4 RYCHLOST ZPRACOVÁNÍ NA ZAŘÍZENÍ JETSON NANO

Poslední provedenou zkouškou bylo spuštění vytvořeného algoritmu na zařízení Jetson Nano. Toto zařízení bylo předem nakonfigurováno podle popisu v kapitole 4.5. Cílem bylo vyhodnocení rychlosti zpracování a ověření schopnosti provozu algoritmu v reálném čase.

K tomuto účelu bylo na počítači Jetson Nano nakonfigurovaném pro použití platformy YOLOv5 provedeno zpracování stejných snímků jako v předchozích kapitolách. V případě použití vah neuronové sítě natrénovaných na větší architektuře YOLOv5s6 byl výstup po zpracování snímků následující:

```
Speed: 60.5ms pre-process, 928.7ms inference, 212.3ms NMS, 1489.5ms depth
per image at shape (1, 3, 640, 640).
```

Pro variantu s menší architekturou neuronové sítě měl pak výpis výsledků do příkazového řádku tuto podobu:

```
Speed: 137.3ms pre-process, 772.7ms inference, 307.5ms NMS, 1587.2ms depth
per image at shape (1, 3, 640, 640).
```

Výsledné doby zpracování snímků pořízených ze záznamu videa jsou dle předpokladu podstatně delší než při spuštění na stolním počítači v předchozí kapitole. Doba rozpoznání objektů je pro větší architekturu YOLOv5s6 928,7 ms, pro menší variantu YOLOv5n6 s hodnotou 772,7 ms. Celková doba zpracování snímku je však u menší varianty dokonce delší.

Následně bylo ověřeno zpracování přenosu přímo z připojené kamery ZED 2 změnou parametru `--source` na hodnotu 0.

```
python detectdist.py --source 0 --weights best.pt --conf-thres 0.4
--save-txt --save-conf
```

Tím byla ověřena rychlost zpracování na větším počtu snímků a výsledné průměrné doby trvání jednotlivých částí jsou tak určeny z více hodnot. V tomto případě byl výsledný výpis průměrných dob zpracování následující.

```
Speed: 22.5ms pre-process, 242.2ms inference, 27.7ms NMS, 1563.7ms depth
per image at shape (1, 3, 640, 640).
```

Z hodnot je zřejmá nižší doba výpočtu pro detekci objektů, doba určení vzdálenosti však zůstává takřka nezměněna, ve všech variantách se pohybovala kolem hodnoty 1500 ms. Díky využití grafických jader počítače s využitím optimalizace v platformě CUDA je detekce objektů podstatně rychlejší oproti zpracování stereo obrazu, které je prováděno na procesoru.

Celková doba zpracování přímého přenosu z kamery činí průměrně 1856,1 ms na jeden snímek. Určením převrácené hodnoty byl pak zjištěn počet snímků zpracovaných za sekundu, který v tomto případě je 0,54 FPS.

Aby mohl být vytvořený algoritmus využit pro řízení autonomního vozidla při provozu na tomto zařízení, musel by být nastaven pevný časový interval jedné iterace smyčky. S ohledem na výsledný průměr doby zpracování jednoho snímku a s přidáním času navíc jakožto rezervy by tato doba musela být stanovena na 2000 ms. Maximální prodleva vyžadovaná pro řízení funkcí automobilu v reálném čase činí 100 ms, pro kritické funkce, kterou řízení vozidla nepochybně je, pak spíše 10 ms. [64] Vzhledem k tomu je vytvořený algoritmus ve stávající podobě pro řízení vozidla počítačem s omezeným výkonem, jako je Jetson Nano, nevhodný. Pro zrychlení algoritmu na tomto hardwaru by byla nutná jeho další optimalizace, jako například využití grafického procesoru počítače ke zpracování stereo obrazu.

ZÁVĚR

Cílem práce bylo vytvoření funkčního algoritmu pro zpracování stereo obrazu v souvislosti s řízením autonomního vozidla. Zpracováním obrazu je zde myšleno konkrétně rozpoznání definovaných objektů v obrazu a následné určení vzdálenosti těchto objektů s využitím vztahu mezi obrazy obou kamer.

Po poskytnutí nezbytného teoretického základu souvisejícího s daným tématem bylo přistoupeno k realizaci komunikace se stereo kamerou Stereolabs ZED 2. Za tímto účelem byl vytvořen kód v jazyce Python s využitím převážně knihovny OpenCV, která poskytuje nástroje pro řešení problémů v oblasti počítačového vidění. Na základě nabytých teoretických znalostí byl pak tento kód rozšířen o stanovení míry disparity mezi obrazy levé a pravé kamery. Informace o disparitě mezi obrazy pak byla využita ke stanovení vzdálenosti daného bodu v obrazu.

Následně byla zpracována část týkající se rozpoznání objektů. K tomu byly zvoleny dvě různé architektury neuronových sítí typu YOLO v rámci platformy YOLOv5. Tyto neuronové sítě byly natrénovány na rozsáhlém datovém souboru BDD100K, který je zaměřený na oblast silniční dopravy a problematiku autonomních vozidel. Analýzou průběhů trénování byla zjištěna vhodnost výsledků pro další využití. Do stávajícího kódu pro detekci objektů v rámci platformy YOLOv5 byl pak implementován vytvořený algoritmus pro stanovení vzdálenosti ze stereo obrazu.

Ověření funkčnosti bylo provedeno nejprve pro samotné určení vzdálenosti, v rámci čehož byly s využitím vytvořeného souboru dat optimalizovány parametry výpočtu disparity za účelem zvýšení přesnosti na zkoumaném rozsahu vzdáleností od 2,5 m do 40 m. Na tomto rozsahu dosahuje výsledná konfigurace průměrné relativní chyby měření 6,4 %, přičemž minimální hodnoty relativní chyby je dosaženo ve vzdálenosti 30 m. Ve srovnání s parametry uváděnými výrobcem kamery (chyba menší než 5 % do vzdálenosti 15 m a maximální vzdálenost 20 m) se jedná o uspokojivé hodnoty. Těchto hodnot bylo dosaženo při rozlišení 1280x720 pixelů pro levou i pravou kameru. Z dále provedeného zhodnocení vlivu rozlišení kamery na přesnost určení vzdálenosti byla vyvozena nutnost přizpůsobení parametrů výpočtu disparity konkrétnímu rozlišení kamery.

Dále byla provedena kontrola správné detekce objektů, kdy byl ověřen očekávaný nedostatek sledovaný již při analýze průběhu trénování neuronových sítí, a to nesprávná detekce objektů třídy vlak v důsledku nedostatečného obsažení této třídy v trénovacím datovém souboru. U ostatních tříd objektů pak v případě větší architektury YOLOv5s6 byla zjištěna správná detekce s dostatečnou pravděpodobností detekovaných objektů. Varianta s menší architekturou YOLOv5n6 vykazovala větší počet nedostatků, kdy objekty některých tříd nebyly spolehlivě rozpoznány. Pro rozsáhlejší architekturu byla provedena podrobnější analýza na vlastním datovém souboru. Při ní bylo vyhodnoceno poměrné množství úspěšně rozpoznaných objektů a poměr mylných detekcí.

Při spuštění vytvořeného algoritmu na stolním počítači byla vyhodnocena doba zpracování snímků pro jeho dílčí části, tedy detekci objektů a určení vzdálenosti zpracováním stereo obrazu. V případě detekce objektů byla vyhodnocena varianta s menší architekturou jako výrazně rychlejší, a to o více než 50 %. Zpracování stereo obrazu a určení vzdálenosti se však ukázalo jako podstatně časově náročnější. Důvodem je, že detekce objektů probíhala na

grafickém procesoru počítače, který umožňuje mnohem vyšší míru paralelizace probíhajících operací.

Podobné výsledky byly vyvozeny i při spuštění algoritmu na méně výkonném kompaktním počítači Jetson Nano. Zde byla zkoumána možnost provozu algoritmu v reálném čase, dosažená hodnota počtu zpracovaných snímků za sekundu přibližně 0,5 FPS, resp. doba 1856 ms pro zpracování jednoho snímku, však tuto možnost vyvrací.

Výše zmíněná skutečnost může být vnímána jako hlavní nedostatek práce. S ohledem na dlouhou dobu zpracování jednoho snímku by na zařízení Jetson Nano nebylo možné využít vytvořený algoritmus jako základ pro řízení autonomního vozidla. Optimalizace části algoritmu zajišťující určení vzdálenosti a využití grafického procesoru k jejímu provozu by výrazně přispělo možnostem provozu tohoto algoritmu v reálném čase. Ke zvýšení výkonnosti by dále mohlo přispět uplatnění metod zpracování stereo obrazu založených na hlubokém učení.

POUŽITÉ INFORMAČNÍ ZDROJE

[1] 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.

[2] CHOLLET, François. *Deep Learning with Python, Second Edition*. 2. Shelter Island, NY: Manning, 2021. ISBN 9781617296864.

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

[4] *Algoritmy strojového učení* [online]. Microsoft, 2022 [cit. 2022-02-11]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/machine-learning-algorithms/#uses

[5] KIM, Phil. *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence.* 1. Berkeley, CA: Apress, 2017. ISBN 978-1-4842-2845-6.

[6] SILVER, David, Julian SCHRITTWIESER, Karen SIMONYAN, et al. Mastering the game of Go without human knowledge. *Nature* [online]. 2017, **550**(7676), 354-359 [cit. 2022-02-11]. ISSN 0028-0836. Dostupné z: doi:10.1038/nature24270

[7] ELGENDY, Mohamed. *Deep Learning for Vision Systems*. 1. Shelter Island, NY: Manning, 2020. ISBN 9781617296192.

[8] RUMELHART, David E., Geoffrey E. HINTON a Ronald J. WILLIAMS. Learning representations by back-propagating errors. *Nature* [online]. 1986, **323**(6088), 533-536 [cit. 2022-02-14]. ISSN 0028-0836. Dostupné z: doi:10.1038/323533a0

[9] PADILLA, Rafael, Sergio L. NETTO a Eduardo A. B. DA SILVA. A Survey on Performance Metrics for Object-Detection Algorithms. *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* [online]. IEEE, 2020, 237-242 [cit. 2022-03-21]. ISBN 978-1-7281-7539-3. Dostupné z: doi:10.1109/IWSSIP48289.2020.9145130

[10] REN, Shaoqing, Kaiming HE, Ross GIRSHICK a Jian SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. 2016 [cit. 2022-04-08]. Dostupné z: https://arxiv.org/pdf/1506.01497.pdf

[11] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU a Alexander C. BERG. SSD: Single Shot MultiBox Detector [online]. 2016 [cit. 2022-04-08]. Dostupné z: https://arxiv.org/pdf/1512.02325.pdf

[12] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. You Only Look Once: Unified, Real-Time Object Detection [online]. 2015 [cit. 2022-02-05]. Dostupné z: https://arxiv.org/abs/1506.02640v5

[13] REDMON, Joseph a Ali FARHADI. YOLO9000: Better, Faster, Stronger [online]. 2016 [cit. 2022-04-08]. Dostupné z: https://arxiv.org/abs/1612.08242

[14] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement [online]. 2018 [cit. 2022-04-08]. Dostupné z: https://arxiv.org/pdf/1804.02767.pdf

[15] REDMON, Joseph. Darknet: Open Source Neural Networks in C [online]. 2016 [cit. 2022-04-09]. Dostupné z: https://pjreddie.com/darknet/

[16] BOCHOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection [online]. 2020 [cit. 2022-04-09]. Dostupné z: https://arxiv.org/abs/2004.10934

[17] JOCHER, Glenn. Release v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support. *GitHub* [online]. 12 Oct 2021 [cit. 2022-02-05]. Dostupné z: https://github.com/ultralytics/yolov5/releases/tag/v6.0

[18] SOZZI, Marco, Silvia CANTALAMESSA, Alessia COGATO, Ahmed KAYAD a Francesco MARINELLO. Automatic Bunch Detection in White Grape Varieties Using YOLOv3, YOLOv4, and YOLOv5 Deep Learning Algorithms. *Agronomy* [online]. 2022, **12**(2) [cit. 2022-04-09]. ISSN 2073-4395. Dostupné z: doi:10.3390/agronomy12020319

[19] NEPAL, Upesh a Hossein ESLAMIAT. Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *Sensors* [online]. 2022, **22**(2) [cit. 2022-04-09]. ISSN 1424-8220. Dostupné z: doi:10.3390/s22020464

[20] ALI, Sahla Muhammed. Comparative Analysis of YOLOv3, YOLOv4 and YOLOv5 for Sign Language Detection. *International Journal of Advance Research and Innovative Ideas in Education* [online]. Bangalore, India: Informatics Limited, 2021, **7**(4), 2393-2398 [cit. 2022-04-09]. ISSN 2395-4396. Dostupné z: https://ijariie.com/AdminUploadPdf/Comparative_Analysis_of_YOLOv3__YOLOv4_and_YOLOv5_for_Sign_Language_Detection_ijariie15253.pdf

[21] ZAARANE, Abdelmoghit, Ibtissam SLIMANI, Wahban AL OKAISHI, Issam ATOUF a Abdellatif HAMDOUN. Distance measurement system for autonomous vehicles using stereo camera. *Array* [online]. 2020, **5** [cit. 2022-04-09]. ISSN 25900056. Dostupné z: doi:10.1016/j.array.2020.100016

[22] HARTLEY, Richard a Andrew ZISSERMAN. *Multiple view geometry in computer vision.* 2nd ed. Cambridge: Cambridge University Press, 2004. ISBN 978-052-1540-513.

[23] SZELISKI, Richard. *Computer Vision: Algorithms and Applications*. Second Edition. Cham, Switzerland: Springer Nature Switzerland, 2022. ISBN 978-3-030-34371-2.

[24] TSAI, R. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation* [online]. 1987, **3**(4), 323-344 [cit. 2022-04-26]. ISSN 0882-4967. Dostupné z: doi:10.1109/JRA.1987.1087109

[25] HIRSCHMULLER, H. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2008, **30**(2), 328-341 [cit. 2022-05-01]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2007.1166

[26] Using the Depth Sensing API. *Stereolabs* [online]. San Francisco: Stereolabs, © 2021 [cit. 2022-04-18]. Dostupné z: https://www.stereolabs.com/docs/depth-sensing/using-depth/

[27] Depth Sensing Overview. *Stereolabs* [online]. San Francisco: Stereolabs, © 2021 [cit. 2022-04-18]. Dostupné z: https://www.stereolabs.com/docs/depth-sensing/

[28] HANSEN, John H. L., Pinar BOYRAZ, Kazuya TAKEDA a Hüseyin ABUT. *Digital Signal Processing for In-Vehicle Systems and Safety*. 1. London: Springer, 2012. ISBN 978-1441996060.

[29] GUPTA, Abhishek, Alagan ANPALAGAN, Ling GUAN a Ahmed Shaharyar KHWAJA. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array* [online]. 2021, **10** [cit. 2022-02-11]. ISSN 25900056. Dostupné z: doi:10.1016/j.array.2021.100057

[30] J3016. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2021-04. USA: SAE, 2021.

[31] MORRIS, James. Why Is Tesla's Full Self-Driving Only Level 2 Autonomous? *Forbes* [online]. New York: Forbes Media, ©2022, Mar 13, 2021 [cit. 2022-02-20]. Dostupné z: https://www.forbes.com/sites/jamesmorris/2021/03/13/why-is-teslas-full-self-driving-only-level-2-autonomous/?sh=3588b4406a32

[32] GARSTEN, Ed. First Drive: Cadillac Enhanced Super Cruise Adds Hands-Free Lane Changes. *Forbes* [online]. New York: Forbes Media, ©2022, Feb 8, 2021 [cit. 2022-02-20]. Dostupné z: https://www.forbes.com/wheels/features/first-drive-cadillac-enhanced-super-cruise-adds-hands-free-lane-changes/

[33] TAYLOR, Michael a Carly SCHAFFNER. BMW 7 Series To Reach Level 3 Autonomy Next Year. *Forbes* [online]. New York: Forbes Media, ©2022, Nov 4, 2021 [cit. 2022-02-20]. Dostupné z: https://www.forbes.com/wheels/features/bmw-7-series-level-3-autonomy/

[34] KIM, Il-Gue a Ji-Hoon LEE. Hyundai to launch Level 3 self-driving Genesis G90 in H2. *KED Global* [online]. Seoul: KED Global News Network, ©2020, Jan 02, 2022 [cit. 2022-02-20]. Dostupné z: https://www.kedglobal.com/newsView/ked202201020001

[35] *First internationally valid system approval for conditionally automated driving* [online]. Stuttgart: Mercedes-Benz Group, December 09, 2021 [cit. 2022-02-20]. Dostupné z: https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/system-approval-for-conditionally-automated-driving.html

[36] GRIGORESCU, Sorin, Bogdan TRASNEA, Tiberiu COCIAS a Gigel MACESANU. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* [online]. 2020, **37**(3), 362-386 [cit. 2022-02-11]. ISSN 1556-4959. Dostupné z: doi:10.1002/rob.21918

[37] FRIDMAN, Lex. Self-Driving Cars: State of the Art. [přednáška]. In: YouTube [online]. Cambridge: Massachusetts Institute of Technology, 1. února 2019. [vid. 16. 2. 2022]. Záznam dostupný z: https://youtu.be/sRxaMDDMWQQ

[38] Autopilot. *Tesla* [online]. Tesla, 2022 [cit. 2022-02-16]. Dostupné z: https://www.tesla.com/autopilot

[39] See how it works. *Argo AI* [online]. Argo AI, 2022 [cit. 2022-02-16]. Dostupné z: https://www.argo.ai/products/#argo-lidar

[40] SALLAB, Ahmad EL, Mohammed ABDOU, Etienne PEROT a Senthil YOGAMANI. Deep Reinforcement Learning framework for Autonomous Driving. *Electronic Imaging* [online]. 2017, **2017**(19), 70-76 [cit. 2022-02-11]. ISSN 2470-1173. Dostupné z: doi:10.2352/ISSN.2470-1173.2017.19.AVM-023

[41] ZED 2 Camera Datasheet. *Stereolabs* [online]. San Francisco: Stereolabs, © 2021, November 2019 [cit. 2022-04-18]. Dostupné z: https://cdn.stereolabs.com/assets/datasheets/zed2-camera-datasheet.pdf

[42] *Stereolabs ZED - OpenCV Native Capture* [online]. Stereolabs, 2019 [cit. 2022-05-01]. Dostupné z: https://github.com/stereolabs/zed-opencv-native/tree/master/python

[43] NVIDIA Jetson Nano Developer Kit. *NVIDIA Developer* [online]. Santa Clara: NVIDIA Corporation, ©2022 [cit. 2022-02-05]. Dostupné z: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[44] Jetson Developer Kits. *NVIDIA Developer* [online]. Santa Clara: NVIDIA Corporation, ©2022 [cit. 2022-03-20]. Dostupné z: https://developer.nvidia.com/embedded/jetson-developer-kits

[45] Getting Started with Jetson Nano Developer Kit. *NVIDIA Developer* [online]. Santa Clara: NVIDIA Corporation, © 2022 [cit. 2022-03-23]. Dostupné z: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

[46] *Cv::StereoSGBM Class Reference* [online]. OpenCV, 2022 [cit. 2022-05-01]. Dostupné z: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html

[47] *Cv::ximgproc::DisparityWLSFilter Class Reference* [online]. OpenCV, 2022 [cit. 2022-05-01]. Dostupné z: https://docs.opencv.org/3.4/d9/d51/classcv_1_1ximgproc_1_1DisparityWLSFilter.html#details

[48] Disparity map post-filtering. [online]. OpenCV, 2022 [cit. 2022-05-11]. Dostupné z: https://docs.opencv.org/4.x/d3/d14/tutorial_ximgproc_disparity_filtering.html

[49] PaddleDetection. *GitHub* [online]. GitHub, © 2022 [cit. 2022-05-16]. Dostupné z: https://github.com/PaddlePaddle/PaddleDetection/blob/release/2.4/README_en.md

[50] HUANG, Xin, Xinxin WANG, Wenyu LV, et al. PP-YOLOv2: A Practical Object Detector [online]. 2021 [cit. 2022-02-05]. Dostupné z: https://arxiv.org/abs/2104.10419

[51] YU, Fisher, Haofeng CHEN, Xin WANG, Wenqi XIAN, Yingying CHEN, Fangchen LIU, Vashisht MADHAVAN a Trevor DARRELL. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning* [online]. 2018 [cit. 2022-02-05]. Dostupné z: https://arxiv.org/abs/1805.04687

[52] CUDA Installation Guide for Microsoft Windows. *NVIDIA Documentation Center* [online]. Santa Clara: NVIDIA Corporation, © 2009-2022 [cit. 2022-03-23]. Dostupné z: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

[53] Installing cuDNN On Windows. *NVIDIA Documentation Center* [online]. Santa Clara: NVIDIA Corporation, © 2017-2022 [cit. 2022-03-23]. Dostupné z: https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#install-windows

[54] JOCHER, Glenn. Train Custom Data. *GitHub* [online]. 2022 [cit. 2022-05-16]. Dostupné z: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

[55] YU, Fisher. Label Format. *BDD100K documentation* [online]. © 2022 [cit. 2022-05-16]. Dostupné z: https://doc.bdd100k.com/format.html#categories

[56] *Bdd2coco.py* [online]. 2020 [cit. 2022-05-16]. Dostupné z: https://github.com/ucbdrive/bdd100k/blob/master/bdd100k/bdd2coco.py

[57] JOCHER, Glenn. JSON2YOLO. *GitHub* [online]. 2022 [cit. 2022-05-16]. Dostupné z: https://github.com/ultralytics/JSON2YOLO

[58] Install OpenCV 4.5 on Jetson Nano. *Q-engineering* [online]. Hoogezand: Q-engineering, December 31, 2021 [cit. 2022-03-23]. Dostupné z: https://qengineering.eu/install-opencv-4.5-on-jetson-nano.html

[59] FRANKLIN, Dustin [dusty_nv]. PyTorch for Jetson - version 1.10 now available. *NVIDIA Developer Forums* [online]. Santa Clara: NVIDIA Corporation, © 2021 [cit. 2022-03-23]. Dostupné z: https://forums.developer.nvidia.com/t/pytorch-for-jetson-version-1-10-now-available/72048

[60] GLM 30 Laserový měřič vzdálenosti. *Elektrické nářadí Bosch* [online]. Gerlingen: Robert Bosch Power Tools, © 2021 [cit. 2022-05-11]. Dostupné z: https://www.bosch-professional.com/cz/cs/products/glm-30-0601072500

[61] VYBÍRAL, Bohumil. ZPRACOVÁNÍ DAT FYZIKÁLNÍCH MĚŘENÍ: Studijní text pro řešitele FO, studující fyziku na UHK a ostatní zájemce o fyziku. *Fyzikální olympiáda* [online]. Fyzikální olympiáda, © 2002–2022 [cit. 2022-05-13]. Dostupné z: http://fyzikalniolympiada.cz/texty/mereni.pdf

[62] *Documentation:Modules/scene* [online]. VideoLAN, 2019 [cit. 2022-05-18]. Dostupné z: https://wiki.videolan.org/Documentation:Modules/scene/

[63] CUDA. *OpenCV* [online]. OpenCV, © 2022 [cit. 2022-05-15]. Dostupné z: https://opencv.org/platforms/cuda/

[64] REIF, Konrad, ed. *Automotive Mechatronics: Automotive Networking, Driving Stability Systems, Electronics*. Wiesbaden: Springer Fachmedien, 2015. ISBN 978-3-658-03974-5.

SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

AI		Artificial Intelligence
AP		Average Precision
b	[-]	Zkreslení uzlu neuronové sítě
B	[m]	Vzdálenost mezi kamerami
baseline	[m]	Proměnná – vzdálenost mezi kamerami
bbox_coords		Proměnná – souřadnice rohových bodů rámečků
BDD100K		Berkeley Deep Drive 100 000
B_{gt}	[-]	Plocha ohraničujícího rámečku odpovídající skutečnosti
B_p	[-]	Plocha ohraničujícího rámečku stanoveného neuronovou sítí
C, C‘		Střed kamery
calibration_file		Proměnná – umístění souboru s kalib. soubory kamery
cap		Proměnná – přenos z kamery
COCO		Common Objects in Context
ctr_coords		Proměnná – souřadnice středů rámečků
d		Disparita
DARPA		Defense Advanced Research Projects Agency
disparity_filtered		Proměnná – disparitní mapa zpracovaná WLS filtrem
disparity_L		Proměnná – disparita pravého vůči levému obrazu
disparity_R		Proměnná – disparita levého vůči pravému obrazu
dist	[m]	Proměnná – vzdálenost objektu stanovená algoritmem
DSMNet		Domain-invariant Stereo Matching Networks
F		Fundamentální matice kamery
f		Ohnisková vzdálenost kamery
FN		False Negative
foc_length		Proměnná – ohnisková vzdálenost kamery
FP		False Positive
FPS		Frames per Second
frame		Proměnná – aktuální snímek
FSD		Full Self Driving
GB		Gigabajt
grayL		Proměnná – snímek z levé kamery v stupních šedi
grayR		Proměnná – snímek z pravé kamery v stupních šedi

h	[-]	Konvoluční jádro
height		Proměnná – výška obrazu jedné kamery
image_size		Proměnná – zahrnuje proměnné *width* a *height*
imstereo		Proměnná – snímek stereo obrazu pro určení vzdálenosti
IOU		Intersection Over Union
left_rect		Proměnná – rektifikovaný snímek z levé kamery
left_right_image		Proměnná – pole obsahující levý a pravý obraz
LiDAR		Light Detection and Ranging
lmbda		Proměnná – parametr *lambda* WLS filtru
mAP	[-]	Mean Average Precision
min_disp		Proměnná – minimální hodnota disparity
MP		Megapixel
NMS		Non-max Suppression
num_disp		Proměnná – maximální hodnota disparity
P		Precision
P1mult		Proměnná – násobitel parametru P1 pro vyhlazení disparity
P2mult		Proměnná – násobitel parametru P2 pro vyhlazení disparity
PR		Precision-Recall křivka
R		Recall
RAM		Random Access Memory
R-CNN		Region-based Convolutional Neural Network
ReLU		Rectified Linear Unit
right_rect		Proměnná – rektifikovaný snímek z pravé kamery
R_k		Matice rotace kamery
s	[m]	Výběrová směrodatná odchylka
SAE		Society of Automotive Engineers
SDK		Software Development Kit
SGD		Stochastic Gradient Descent
SGM		Semiglobal Matching
sigma		Proměnná – parametr *sigma* WLS filtru
SSD		Single Shot Detector
t	[-]	Hraniční hodnota správné detekce
T		Vektor translace kamery

t4		Proměnná – čas počátku zpracování stereo obrazu
t5		Proměnná – čas ukončení zpracování stereo obrazu
TP		True Positive
USB		Universal Serial Bus
v	[-]	Variační koeficient
V2I		Vehicle-to-infrastructure
V2V		Vehicle-to-vehicle
V2X		Vehicle-to-everything
VOC		Visual Object Challenge
vs	[-]	Vážený součet uzlu neuronové sítě
w_i	[-]	Hodnoty vah uzlu neuronové sítě
width		Proměnná – šířka obrazu jedné kamery
window_size		Proměnná – velikost bloku pixelů pro nalezení disparity
WLS		Weighted Least Squares
wls_filter		Proměnná – WLS filtr
X		Bod zájmu v prostoru
x, x‘		Projekce bodu zájmu v obrazových rovinách kamer
x_disp		Proměnná – souřadnice *x* bodu pro určení vzdálenosti
x_i	[-]	Vstupy uzlu neuronové sítě
y	[-]	Výstupní signál uzlu neuronové sítě
y_disp		Proměnná – souřadnice *y* bodu pro určení vzdálenosti
YOLO		You Only Look Once
YOLOv5		You Only Look Once version 5
Z	[m]	Vzdálenost určená ze stereo obrazu
Z_0	[m]	Vzdálenost změřená laserovým měřičem vzdálenosti
δ	[-]	Relativní chyba určení vzdálenosti ze stereo obrazu
ε_{abs}	[m]	Velikost absolutní chyby určení vzdálenosti ze stereo obrazu
φ	[-]	Aktivační funkce uzlu neuronové sítě