VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

DETEKCE A KLASIFIKACE OBJEKTŮ ZÁJMU ZALÉVACÍHO ROBOTU ZPRACOVÁNÍM OBRAZU

DETECTION AND CLASSIFICATION OF OBJECTS OF INTEREST FOR WATERING MOBILE ROBOT USING IMAGE PROCESSING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Jiří Sladký

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Jiří Krejsa, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Jiří Sladký**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Jiří Krejsa, 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:

Detekce a klasifikace objektů zájmu zalévacího robotu zpracováním obrazu

Stručná charakteristika problematiky úkolu:

Jedním z projektů řešených na VUT FSI je projekt zalévacího autonomního mobilního robotu, jehož cílem je navrhnout zařízení, které se bude autonomně pohybovat ve vnitřním prostředí, a zde detekovat, klasifikovat a zalévat kytky. Podstatnou součástí takového robotu je subsytém, který zabezpečuje detekci a klasifikaci objektů zájmu, ať jsou to vlastní kytky, květináče, nebo například zdroj vody. Výhodné by bylo také určení vzdálenosti mezi robotem a těmito objekty. Návrh takového subsystému je podstatou zadání této práce.

Cíle diplomové práce:

1/ proveďte rešerši metod pro klasifikaci a detekci objektů zájmu zpracováním obrazu s přihlédnutím k typu hledaných objektů
2/ proveďte rešerši metod pro určení vzdálenosti detekovaných objektů na velkou i malou vzdálenost
3/ vyberte vhodné metody s uvažováním omezení výpočetního výkonu mobilního robotu
4/ metody implementujte a ověřte na reálných datech

Seznam doporučené literatury:

SEWAK M. et. al., Practical Convolutional Neural Networks: Implement advanced deep learning models using Python, Packt Publishing Ltd, 2018

BALLAR W., Hands-On Deep Learning for Images with TensorFlow: Build intelligent computer vision applications using TensorFlow and Keras, Packt Publishing Ltd, 2018"

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Tato práce se zabývá obrazovým zpracováním na autonomním zaléváním mobilním robotu s využitím embedded počítače NVIDIA Jetson Nano. Vybrána a aplikována byla metoda pro detekci objektů YOLOv5, která sloužila k detekci květin a květináčů. Pomocí metody pro monokulární predikci MiDaS byla predikována relativní hloubková mapa. Vytvořen byl algoritmus, který pomocí dat z LiDARu převedl tuto mapu na metrickou. Díky tomu mohla být určena vzdálenost detekovaných květin. Vytvořené nástroje byly implementovány v prostředí ROS a otestovány na reálných datech z vnitřního prostředí.

Summary

This thesis deals with image processing on autonomous watering mobile robot using embedded computer NVIDIA Jetson Nano. A method for object detection, YOLOv5, was chosen, which served for detection of flowers and flower pots. Using a method for monocular depth estimation, MiDaS, relative depth map was predicted. An algorithm was created, which converted this map to metric depth map using data from LiDAR. Thanks to that, distance of the detected flowers could be estimated. The created tools were implemented in ROS framework and tested on real data form indoor environment.

Klíčová slova

počítačové vidění, konvoluční neuronové sítě, detekce objektů, YOLO, monokulární predikce hloubky, MiDaS, ROS, indoor, detekce květin, ArUco

Keywords

computer vision, convolutional neural networks, object detection, monocular depth estimation, MiDaS, ROS, indoor, flower detection, ArUco

Bibliografická Citace

SLADKÝ, J. *Detekce a klasifikace objektů zájmu zalévacího robotu zpracováním obrazu.*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2022. 75 s., Vedoucí diplomové práce: doc. Ing. Jiří Krejsa, PhD..

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením svého vedoucího, a že jsem veškeré použité zdroje v práci citoval a uvedl v seznamu literatury.

Jiří Sladký

Brno . . . .

Tímto bych rád poděkoval svému vedoucímu doc. Ing. Jiřímu Krejsovi, PhD. za celkové vedení práce a firmě Bender Robotics za zapůjčený hardware. Děkuji Bc. Ondřeji Podolinskému za zorganizování projektu a jeho příkladné směřování, stejně tak dalším členům týmu za spolupráci. V neposlední řadě děkuji své rodině za podporu při studiu.

Jiří Sladký

Obsah

1 Úvod

Robotika ovlivňuje společnost již dlouhá desetiletí. V počátcích prováděly roboty spíše jednoduché, repetitivní úkony zejména ve výrobě. S rozvojem robotiky dokáží vykonávat stále komplexnější činnosti. Úlohy, jako např. odstraňování plevele nebo sběr plodin, jsou pro člověka samozřejmé, jejich korektní provedení je ale podmíněno smyslovými vjemy, znalostmi prostředí a precizní motorikou. Potřeba vnímat okolí výrazně narůstá v oblasti mobilní robotiky, od robotických vysavačů a sekaček až po autonomní vozidla. Pro člověka je dominantním smyslem zrak, obraz totiž obsahuje mnoho informací o okolním světě. Díky dostupnosti kamer a rozvoji algoritmů zpracování obrazu těží z obrazu i mobilní robotika. Za pokrok vděčí především umělým neuronovým sítím, které se na poli počítačového vidění staly standardem. Kamery tak dokáží doplnit nebo i nahradit jiné senzory.

Málo probádanou aplikací mobilní robotiky je zalévání pokojových květin. Zalévací robot by mohl najít uplatnění v rozsáhlejších prostorách, např. v akademických či komerčních budovách, kde bývají květiny rozmístěny na přístupných místech. Automatické pečování o zeleň by mohlo přispět k zazelenání vnitřních prostor a tím ke zvýšení spokojenosti.

Realizací prototypu autonomního zalévacího robotu se zabývá projekt, jehož součástí je i tato práce. Projekt zpracovává celkem pět studentů, náplň práce každého studenta shrnuje tabulka 1.1. Podstatou projektu je využití existujícího robotického podvozku Breach, jeho osazení zalévacím mechanismem a vytvoření funkcionalit, které zabezpečí pohyb prostorem a hledání zalévaných květin. Součástí je i mobilní aplikace, přes kterou může uživatel sledovat stav robotu a ovlivňovat jeho chování.

Tato práce se zaměřuje na zpracování obrazu s využitím konvolučních neuronových sítí, jejichž běh zprostředkovává embedded počítač NVIDIA Jetson Nano. Cílem práce je nejprve nalezení vhodné metody pro detekci květin a květináčů, určení jejich vzdálenosti od robotu a identifikaci květin pomocí referenčních štítků. Vybrané metody je potřeba implementovat do prostředí Robot Operating System (ROS) tak, aby byly použitelné v rámci projektu. Na závěr je nutné metody ověřit na reálných datech z chodu robotu.

Tabulka 1.1: Rozdělení projektu

Student	Náplň práce
Bc. Ondřej Podolinský	Idea projektu, jeho definice a dekompozice na jednotlivé práce, návrh pro konkrétní robotickou platformu, simulační prostředí, mechanismus kamery, integrace modulů a testování celku. [1]
Bc. Jan Bajer	Komunikační rozhraní, mobilní aplikace, řídicí část programu. [2]
Bc. Miroslav Doseděl	Pohyb robotu – globální v prostoru a lokální v okolí květin. [3]
Bc. Jiří Sladký	Obrazová detekce a identifikace květin, určení jejich vzdálenosti.
Bc. Peter Vizváry	Návrh, konstrukce a řízení mechanismu zalévání a čerpání. [4]

2 Rešeršní část

Rešeršní část se zabývá metodami detekce objektů s využitím konvolučních neuronových sítí. Ty se v poslední dekádě využívají na většinu problémů počítačového vidění. Klasické metody jsou vhodnější pro objekty s jasně definovanou geometrií, mezi něž květiny a květináče díky své různorodosti nepatří. Proto budou nejprve přiblíženy základní principy konvolučních neuronových sítí, poté rozebrány relevantní metody detekce objektů pro použití na robotu. Poté proběhne průzkum způsobů a metod, jak lze z obrazu a případně s využitím dalších senzorů určit vzdálenost detekovaných květin. Nastíněny budou principy identifikačních štítků (referenčních značek), které mají být připevněny na květináče a jejichž použití se na robotu předpokládá z důvodu, aby bylo možno identifikovat konkrétní květiny a určit jejich orientaci vůči robotu pro potřeby zalévacího mechanismu.

2.1 Konvoluční neuronové sítě

Problémy z oblasti počítačového vidění bývají čím dál častěji řešeny pomocí konvolučních neuronových sítí. Architektura sítě se skládá z vrstev, ukázkovou síť ilustruje obrázek 2.1. První vrstva se nazývá vstupní, její rozměr je $w \times h \times 3$, kde w a h jsou šířka a výška obrazu. Třetí rozměr udává počet kanálů, v případě RGB obrazu tedy 3. Dalšími vrstvami ukázkové sítě jsou konvoluční (Conv), max-poolingové (MP) a plně propojené (FC) vrstvy. Poslední plně propojená vrstva je výstupní, na výstupu jsou tedy 2 čísla. [5, 6]

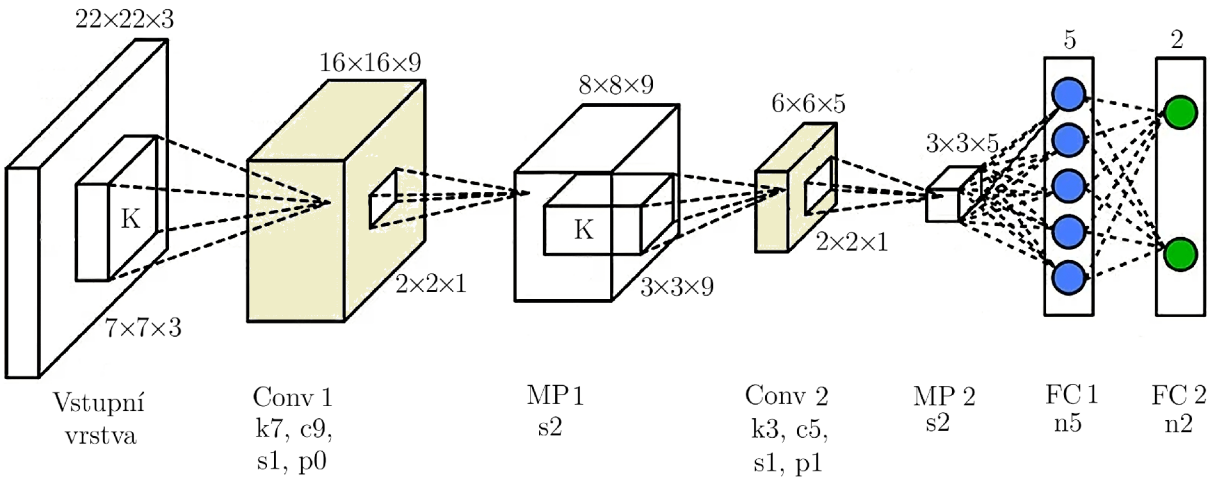
2.1.1 Vrstvy neuronových sítí

Plně propojená vrstva

Parametrem plně propojené vrstvy je počet neuronů n. Každý neuron je spojen se všemi neurony předchozí vrstvy. Výstup jednoho neuronu se určí na základě hodnot vstupu a naučitelných parametrů – vah (*weights*) a prahu (*bias*) – a to jako vážený součet vstupů, povýšen o práh. Výstup poté prochází aktivační funkcí, což vnáší do modelu nelinearitu. Plně propojená síť obsahuje $(i + 1)n$ naučitelných parametrů, kde i je počet vstupních neuronů. Plně propojené vrstvy bývají výpočetně náročné z důvodu vysokého počtu spojení, nahrazují se tak efektivnějšími, konvolučními vrstvami. [5, 6]

Konvoluční vrstva

Základem konvoluční vrstvy je posuvné okno K zvané jádro (*kernel*). Hlavními parametry konvoluční vrstvy jsou velikost jádra k, počet výstupních kanálů c, *stride* s a *padding* p. Konvoluční vrstva obsahuje $(k^2z + 1)c$ naučitelných parametrů, kde z je počet vstupních kanálů. Jeden kanál výstupní vrstvy se určí jako diskrétní konvoluce vstupní vrstvy a jádra. *Stride* s udává velikost posuvu okna při konvoluci. *Padding* p značí šířku rámečku samých nul okolo vstupu. Lze díky ní docílit stejného rozměru výstupu jako vstupu i v případě velikosti jádra větší než 1 (tzv. *zero padding*). Výstup konvoluční vrstvy opět prochází aktivační funkcí. [5, 6]



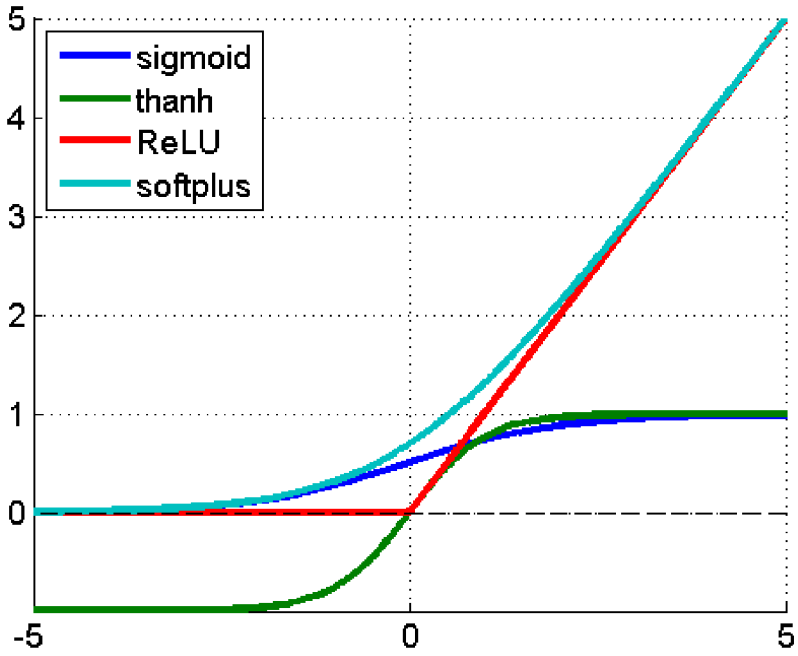
Obrázek 2.1: Jednoduchá konvoluční neuronová síť. Vychází z [7].

Normalizační vrstva

Normalizační vrstva normalizuje vstupní hodnoty tak, že je vynásobí škálou γ a přičte posunutí β. Oba parametry jsou naučitelné. Normalizační vrstva se přidává mezi konvoluční či plně propojenou vrstvu a aktivační funkci. Trénování se tak stává rychlejším a stabilnějším. [6]

Poolingová vrstva

Poolingová vrstva je tvořena posuvným oknem o velikost k. Výstupem pro každou polohu okna je aritmetická operace aplikovaná na vstupy, nejčastěji průměr nebo maximum. V poolingových vrstvách se běžně používá *stride* rovný 2, tím pádem snižují rozměr obrazu 2×, neboli podvzorkovávají. [6]



Obrázek 2.2: Přehled aktivačních funkcí. Převzato z [8].

Aktivační funkce

Aktivační funkce vnáší do modelu nelinearitu. Vhodnou vlastnosti jsou hladkost a monotónnost. Tradičními aktivačními funkcemi jsou sigmoida (viz identitu 2.1) a hyperbolický tangens. Jejich derivace lze vyjádřit pomocí funkčních hodnot, avšak v oblasti saturace se derivace blíží nule a zpomaluje trénování sítě. Proto se, převážně v konvolučních neuronových sítích, používá spíše ReLU (viz obrázek 2.2), které trénování výrazně zrychluje [6]. Kvůli nulové derivaci pro záporné vstupy může ReLU zastavit trénování některých neuronů. Problém řeší Leaky-ReLU, případně hladké varianty SiLU a Mish, které se nyní standardně používají v konvolučních blocích, skládajících se z konvoluční a normalizační vrstvy zakončených aktivační funkcí. Ve výstupní vrstvě se aplikuje taková aktivační funkce, jejíž obor hodnot vyhovuje charakteru úlohy. [6, 8]

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

2.2 Detekce objektů

Detekce objektů v obraze představuje klasickou úlohu počítačového vidění. Podstatou úlohy je nalezení rámečků ohraničujících detekované objekty a klasifikování těchto objektů do příslušných tříd. Metody hlubokého učení vynikají v detekci objektů vyznačujících se vysokou variabilitou. Těmi mohou být např. vozidlo, pes, člověk, okno, ale také pokojová rostlina a květináč, kterými se zabývá tato práce. Proto budou v následujícím průzkumu uvažovány právě metody na bázi umělých neuronových sítí. [9]

Podle počtu fází detekce se metody dělí na dva základní typy:

- **Dvoustupňové** – V první fázi dochází k návrhu oblastí, v kterých by se potenciálně mohl nacházet objekt. V druhé fázi se jednotlivé oblasti analyzují a v případě potvrzení jeho přítomnosti je objekt zařazen do patřičné třídy. Dvoustupňové metody bývají typicky velmi přesné, avšak výpočetně náročné. Patří sem např. R-CNN a jeho varianty, Mask R-CNN, R-FCN. [9]
- **Jednostupňové** – Detekci zajišťuje jediná neuronová síť – z barevného obrazu předpovídá přímo údaje o poloze rámečků a třídě objektu. Díky vyšší rychlosti a nižším paměťovým nárokům jsou jednostupňové detektory pro embedded zařízení vhodnější [10]. Celý proces trénování se rovněž usnadňuje. Další výhodou je, že klasifikují objekt na základě větší části obrázku (dané velikostí tzv. receptivního pole), čímž zohledňují i kontext snímané scény. Jednostupňové metody ale stále zaostávají v detekci malých objektů. Řadí se sem např. YOLO, SSD, RetinaNet. [9]

2.2.1 Metriky pro detekci objektů

Metody detekce objektů je potřebné kvantitativně srovnávat. Typicky se k tomu používá metrika *mean average precision* (mAP), která je definovaná pomocí následujících pojmů.

Intersection over Union (IoU) udává poměr plochy průniku skutečného a předpovězeného rámečku podělený plochou jejich sjednocení. Je-li definována prahová hodnota IoU, lze o každém předpovězeném rámečku rozhodnout, zda je skutečně pozitivní (TP), skutečně negativní (FN), falešně pozitivní (FP) nebo falešně negativní (FN). Pro dataset (množinu dat) lze poté určit bezrozměrné metriky *precision* (přesnost) a *recall* (výtěžnost). *Precision* udává počet skutečně pozitivních rámečků (TP) ku všem předpovězeným

rámečkům (TP+FP). *Recall* určuje počet skutečně pozitivních rámečků (TP) ku všem skutečným rámečkům (TP+FN). [11]

Z obou metrik lze sestrojit *precision-recall* křivka [11]. Plocha pod touto křivkou se nazývá *average precision* (AP), nabývá hodnot od 0 do 1, hodnota 1 značí ideální případ. Kýžená metrika mAP se určí tak, že se stanoví AP pro každou třídu objektů zvlášť a vypočte se z nich průměr. Metrika mAP obsahuje několik podtypů v závislosti na tom, jakou prahovou hodnotu IoU používá k posouzení správnosti predikce. Nejčastěji se vyhodnocují mAP(0.50), kde práh IoU je 0.50, a mAP(0.5:0.05:0.95), kde práh nabývá hodnot od 0,5 do 0,95 s krokem 0,05 a vypočten je pak průměr [12]. [11]

2.2.2 Přehled metod detekce objektů

Metody detekce objektů se nejčastěji porovnávají na základě metriky mAP dosažené trénováním na datasetu COCO [9]. Posouzení metod v následujícím přehledu vychází z tohoto porovnání, které autoři metod v článcích prezentují.

Region based metody

Dvoustupňová metoda R-CNN (z roku 2013) funguje tak, že navrhuje algoritmem selektivního výběru oblasti zájmu, pro každou oblast extrahuje konvoluční neuronovou sítí příznaky, z příznaků metodou *support vector machine* určuje třídu objektů a regresorem zpřesňuje polohu rámečku [13]. Ve vylepšené, rychlejší metodě Fast R-CNN se extrahují příznaky pouze jednou a sdílí se mezi navrženými oblastmi [14]. Faster R-CNN (z roku 2015) navrhuje oblasti zájmu pomocí neuronové sítě a to až po extrakci příznaků, což celý proces ještě více zrychluje [15]. Architektura Faster R-CNN je v průběhu let obohacována o nové poznatky v oblasti počítačového vidění. Díky tomu patří stále mezi velmi přesné metody, avšak výpočetně stále relativně náročné [10]. Vznikají i rychlejší varianty (R-FCN, Light-head R-CNN), které jsou ale překonány v poměru přesnost-rychlost např. novějšími verzemi YOLO [10, 16].

SSD

Metoda SSD (Single Shot Detector), vydaná v roce 2015, svou architekturou jednostupňového detektoru navazuje na první generaci YOLO (YOLOv1). Jedná se o plně konvoluční síť, která oproti YOLOv1 predikuje rámečky i na nižších úrovních podvzorkování. To pomáhá s detekcí malých objektů, avšak chybí zde kombinace příznaků z vyšších úrovní. Při detekci aplikuje kotevní rámečky (*anchor boxes*) představené ve Faster R-CNN. Jsou to rámečky předdefinovaných rozměrů, které reflektují typické rozměry rámečků v trénovacím datasetu. Velikost predikovaných rámečků se poté určuje vůči nim, což urychluje trénování. SSD a jeho varianty (např. DSSD [17]) byly na svou dobu rychlé a přesné, nyní je překonává YOLO počínaje 3. verzí (3-5× rychlejší při stejné přesnosti) [16, 18]. [19]

RetinaNet

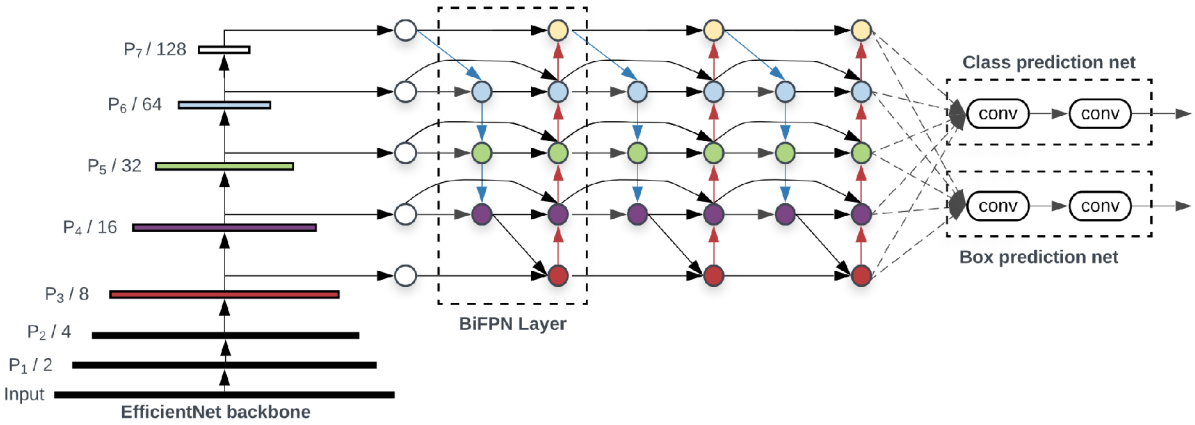
Jednostupňový detektor RetinaNet (vydaný v roce 2017) obohacuje rychlost jednostupňových detektorů o přesnost dvoustupňových. Hlavní přínos spočívá v rozšíření klasifikační chybové funkce křížové entropie o tzv. *focal loss*. Skládá se ze dvou parametrů. Vyvažující parametr α řeší nevyváženost zastoupení jednotlivých tříd objektů, případně tříd objektů a pozadí. Zaměřující parametr γ způsobuje, že mají obtížně klasifikované objekty větší vliv na hodnotu chyby. Metoda dále využívá poznatky jiných metod, např. kotevní rámečky a pyramidovou strukturu (FPN). Rychlejší varianta této metody je přesností a rychlostí

srovnatelná s YOLOv3 [18], novější verze YOLO [16] ji překonávají. Myšlenka chyby *focal loss* je však užitečná a stále využívaná. [20]

EfficientDet

Metoda EfficientDet (vydaná v roce 2019) staví na typické struktuře moderních jednostupňových detektorů (podrobněji v kapitole 2.2.3). Obsahuje síť EfficientNet pro extrakci příznaků, zavádí strukturu BiFPN pro kombinaci příznaků, detekci a klasifikaci provádí ve dvou oddělených větvích sítě o 3 – 5 konvolučních blocích (viz obrázek 2.3). Celá architektura je škálovatelná, čímž produkuje 8 modelů o různé komplexnosti. Škálují se počet vrstev, počet parametrů ve vrstvách a rozlišení vstupu v rozsahu od 512 px do 1536 px. [21]

Komplexnější modely jsou velice přesné, avšak výpočetně náročné. Méně komplexní modely běží v reálném čase (přes 30 snímků za sekundu) pouze na výkonných GPU (např. Tesla V100) [21]. Přesnějších a rychlejších výsledků na méně výkonných zařízeních dosahuje YOLOv4 [16]. Autoři však vyvinuli i odlehčené modely EfficientDet-lite vhodné pro mobilní zařízení, které vykazují poměrem přesnosti a rychlosti velký potenciál [22].



Obrázek 2.3: Architektura EfficientDet. Převzato z [21].

Další metody

Mezi aktuálně nejpřesnější metody patří varianty metod Detection Transformer (DETR) [23], vydané v roce 2020, a Swin Transformer [24], vydané v roce 2021. Kromě detekce objektů jsou využitelné i na jiné úlohy počítačového vidění. Jsou ale výpočetně velmi náročné, převážně při trénování.

Existují jednostupňové detektory, které nevyužívají kotevní rámečky. Příkladem jsou CornerNet [25], FCOS (Fully Convolutional One-Stage) [26], CenterNet [27] a jeho nejnovější varianta vydaná v roce 2022 CenterNet++ [28]. Metody v přesnosti výrazně nezaostávají za výše jmenovanými, rychlostí se pohybují kolem hranice reálného času, ale opět jen na výkonných GPU.

Některé metody upřednostňují použití v reálném čase na mobilních zařízeních. Mezi nejnovější metody se řadí PP-PicoDet [29] (z roku 2021) s výborným poměrem rychlosti a přesnosti. Článek srovnává mnoho metod a ukazuje také, že odlehčenější varianty YOLOv5 dosahují pro mobilní zařízení stále dobrých výsledků. [29]

2.2.3 Metoda YOLO

YOLO (You Only Look Once) přišlo poprvé s myšlenkou jednostupňových detektorů. Původní autoři, Joseph Redmon a Ali Farhadi, publikovali celkem 3 verze metody – YOLOv1 [30] v červnu 2015, YOLOv2 [31] v prosinci 2016 a YOLOv3 [18] v dubnu 2018. Metodu implementovali ve vlastním frameworku Darknet v jazycích C a CUDA. Ve vývoji Daknetu a YOLO pokračovali Alexey Bochkovskiy a kolektiv, kteří vydali YOLOv4 [16] v dubnu 2020, Scaled-YOLOv4 [32] v listopadu 2020 a YOLOR [33] v květnu 2021. V červnu 2020 zveřejnil Glenn Jocher v rámci GitHub repozitáře rozšíření metody s kontroverzním označením YOLOv5 [34]. Navzdory malému počtu revolučních změn oproti předchozím verzím, YOLOv5 využívá místo Darknetu populární framework PyTorch v jazyce Python, přínos tak spočívá mimo jiné ve snadnější implementaci metody v praxi.

Z uvedených verzí se pro detekci pokojových květin v této práci nabízí některá z novějších verzí - YOLOv4, YOLOv5 nebo YOLOR. Verze v4 a v5 bývají v současných odborných publikacích často používány na různých typech úloh a vzájemně srovnávány [35, 36, 37, 38]. Obě verze jsou si poměrem přesnosti a rychlosti velmi podobné, mírně lepší se jeví v5. Ta navíc nabízí větší škálu předtrénovaných modelů s různou výpočetní náročností a je snadno použitelná v jazyce Python. Princip metody YOLO bude proto vysvětlen na v5, konkrétně vydání **YOLOv5 6.0** [34] (říjen 2021). Následně budou rozepsány podrobnosti vývoje a přínos každé verze.

Architektura sítě

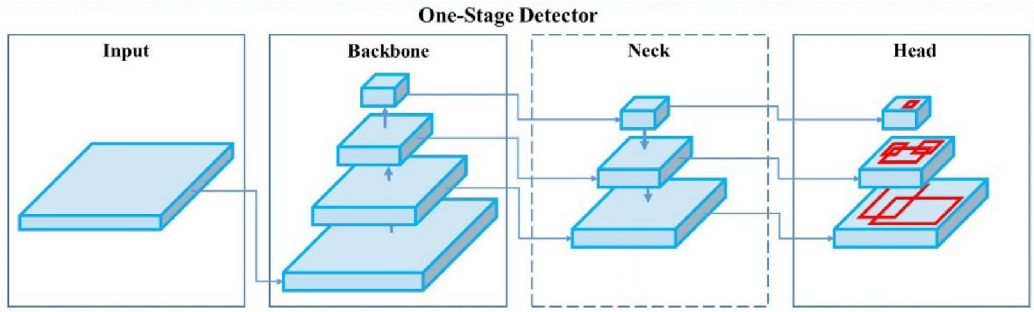
Architekturu detektoru YOLO znázorňuje obr. 2.4. Vstupní obraz prochází třemi celky sítě: [9, 16]

- **Backbone** – Páteř tvoří neuronová síť pro extrakci příznaků. Průchodem vstupního obrazu sítí dochází k jeho podvzorkování. Podvzorkování dané vrstvy je vždy rovno mocnině 2. Zajišťují to *stride* rovný 2 a *zero padding* v daných konvolučních vrstvách. Páteř se často trénuje nejprve na klasifikační úloze na větším datasetu, YOLOv5 toho však nevyužívá.
- **Neck** – Krk kombinuje nízkoúrovňové příznaky z nižších vrstev páteře (např. hrany), které upřesňují lokalizaci, se sémantickými příznaky z vyšších vrstev, které charakterizují daný objekt. YOLOv5 využívá neck metody PANet [39], která mapy příznaků opakovaně zmenšuje, zvětšuje a vzájemně řetězí (*concatenation*).
- **Head** – Detektor provádí detekce na několika úrovních podvzorkování (8×, 16× a 32×). Nižší úrovně pomáhají detekovat malé objekty, vyšší úrovně velké objekty. Hlava je tvořena konvolučními vrstvami s jádrem o velikosti 1×1, jejich smysl by se dal interpretovat jako vážený součet hodnot buňky napříč jednotlivými kanály. V jejichž výstupu jsou zakódovány detekce.

Interpretace výstupu

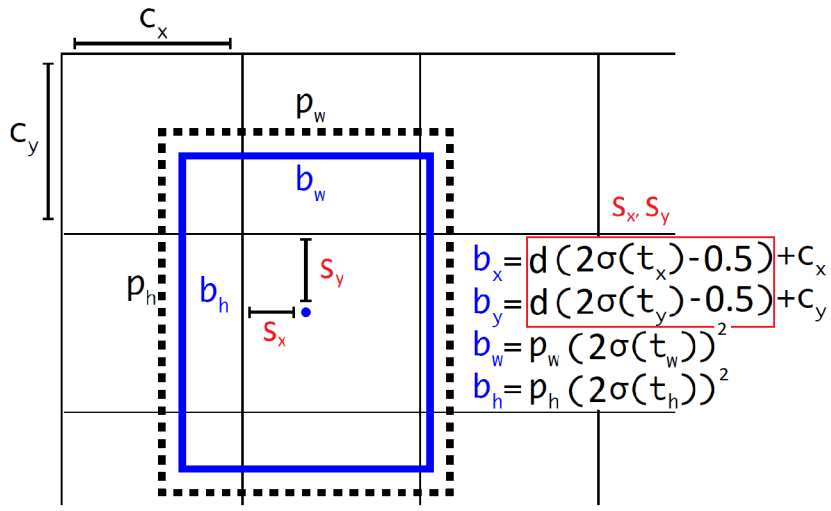
Pro vstupní obraz o velikosti 640×640 px vznikají na třech úrovních části *head* mřížky o velikosti 80×80, 40×40 a 20×20 buněk. Každá buňka predikuje vektor o $A \cdot (5 + c)$ prvcích dle vztahu 2.2, kde A je počet kotevních rámečků (obvykle 3) a c je počet tříd objektů. [31]

$$A \times [t_x, t_y, t_w, t_h, t_o, c_1, c_2, ..., c_c], \tag{2.2}$$



Obrázek 2.4: Struktura jednostupňového detektoru. Upraveno z [16].

Význam prvních 4 parametrů je zřejmý z obr. 2.5. Parametry t_x a t_y slouží k vyjádření souřadnic středu předpovězeného rámečku b (modře), kde d je šířka buňky v pixelech (rovno podvzorkování dané úrovně), c_x a c_y jsou souřadnice levého horního rohu buňky. Parametry t_w a t_h značí šířku a výšku rámečku b vůči šířce a výšce kotevního rámečku p (čárkovaně). Převodní vztahy využívají sigmoidu $\sigma(x)$ definovanou dle identity 2.1. Parametry c_i vyjadřují příslušnost objektu k dané třídě za podmínky, že rámeček nějaký objekt obsahuje. Z parametru t_0 se přes vztah $\sigma(t_0)$ určuje tzv. objektovost (*objectness*), konkrétně jde o predikci IoU (průniku ku sjednocení) mezi předpovězeným rámečkem a skutečným rámečkem. Z objektovosti se dále určuje pro výsledný rámeček skóre jistoty (*confidence score*) pro nejpravděpodobnější třídu c_{max} podle vztahu 2.3. [31, 34]



Obrázek 2.5: Výstupní parametry YOLOv5. Upraveno z [31] dle [34].

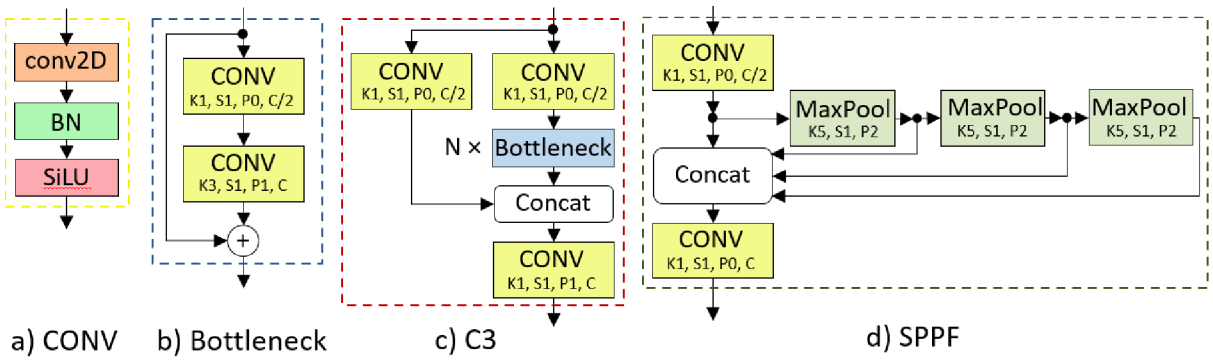
$$confidencescore = \sigma(t_0) \cdot \sigma(c_{max}) \tag{2.3}$$

Rámečky jsou filtrovány algoritmem Non Maximum Suppression, který funguje následovně. Nejprve jsou ignorovány všechny rámečky, které mají skóre jistoty menší než definovaný práh. Ponechávány jsou rámečky počínaje tím s nejvyšším skóre jistoty, přičemž je vyhodnoceno jejich IoU s rámečky s menším skóre jistoty. Rámečky, které dosáhnou hodnoty IoU větší, než je prahová hodnota, jsou ignorovány. Potenciální duplicitní rámečky jsou tak odstraněny. [40]

Funkční bloky

Architektura YOLOv5 se skládá z několika typů funkčních bloků (viz obrázek 2.6): [34]

- **CONV** – Konvoluční blok představuje základní stavební kámen konvolučních neuronových sítí. Tvořen je sériovou kombinací konvoluční vrstvy (conv2D), normalizační vrstvy (BN – *batch normalisation*) a aktivační funkce (konkrétně SiLU, nazývané také Swish). Základními parametry CONV bloku jsou parametry konvoluční vrstvy, tedy velikost jádra K, *stride* S, *padding* P a počet kanálů C.
- **Bottleneck** – Zúžení se skládá ze dvou konvolučních bloků. První blok snižuje počet kanálů na polovinu jádrem o velikosti 1×1, druhý ho opět navyšuje širším jádrem 3×3 (odtud název *bottleneck*). Po vzoru architektury ResNet [41] je vstup hrdla přičten k jeho výstupu, tzv. *skip-connection*.
- **C3** – Tento funkční blok dostal svůj název podle tří konvolučních bloků. Tvořen je dle struktury CSP [42]. Vstup prochází dvěma větvemi, v každé z nich konvolučním blokem redukujícím počet kanálů. V hlavní větvi je obsaženo několik po sobě jdoucích bottleneck bloků. Obě větve jsou zřetězeny a prostupují konvolučním blokem.
- **SPPF** – Na konec části *backbone* je zařazen blok SPPF – rychlejší verze SPP [43] upravená pro YOLOv5, funkčně však stejná. Smyslem je zvýšení receptivního pole díky max-poolingovým vrstvám s většími jádry a kombinace víceúrovňových příznaků v rámci jedné vrstvy [16]. Receptivní pole udává, jak velká část vstupního obrazu se podílí na výstupu daného neuronu. Větší receptivní pole umožňuje zpracování většího kontextu v obraze.



Obrázek 2.6: Funkční bloky YOLOv5. Vychází z [44].

Celek *backbone* je tvořen kombinací C3 bloků a konvolučních bloků způsobujících podvzorkování map příznaků. Zakončen je SPPF blokem. Celek *neck* se skládá z C3 bloků, jejichž *bottleneck* bloky neobsahují *skip-connection*, konvolučních bloků a převzorkovacích vrstev. Celek *head* obsahuje na třech úrovních podvzorkování konvoluční vrstvy, které generují výstupní mapy příznaků. [34]

YOLOv5 nabízí 5 modelů, které oproti výchozímu modelu v5l škálují hloubku a šířku modelu. Počet *bottleneck* bloků uvnitř C3 bloků je dán hloubkou modelu, počet kanálů v konvolučních vrstvách je dán šířkou modelu. K dispozici jsou také natrénované modely pracující s dvojnásobným vstupním rozlišením, které jsou komplexnější a obsahují jednu úroveň podvzorkování a detekce navíc. [34]

Chybová funkce

Hodnota chybová funkce metody YOLO je rovna váženému součtu 3 částí. Pro dataset COCO autoři zvolili váhy 0,05 pro lokalizaci, 1 pro objektovost a 0,5 pro klasifikaci. [30, 34]

- **Chyba lokalizace** – Zahrnuje první 4 parametry výstupního vektoru – polohu a velikost rámečku. Využívá sofistikovaný Complete-IoU Loss (CIoU) [45], který porovnává predikovaný a skutečný rámeček s využitím metriky IoU. Zohledňuje překryv, vzdálenost středů rámečků a poměr jejich stran. Chyba lokalizace je ovlivněna pomocí vah tak, aby byla větší na úrovních s nižší mírou podvzorkování, což klade důraz na přesnější lokalizaci malých objektů.
- **Chyba objektovosti** – Zahrnuje 5. parametr výstupního vektoru – objektovost. Využívá binární křížovou entropii (BCE) se sigmoidou, společně s *Focal loss* [20].
- **Chyba klasifikace** – Zahrnuje parametry jednotlivých tříd, v případě správné třídy má být $\sigma(c_i)$ rovno 1, jinak 0. Využívá stejný druh chyby jako v případě objektovosti, proto díky BCE podporuje příslušnost objektu k více třídám.

2.2.4 Vývoj metody YOLO

YOLOv1

První verze metody YOLO přišla s myšlenkou jednostupňových detektorů a vyniká svou jednoduchostí. Architektura využívá kombinaci konvolučních a max-pooling vrstev zakončených plně propojenými vrstvami. Ty podmiňují pevnou velikost výstupního vektoru (velikost mřížky) a tím pádem i vstupního obrazu. V1 neobsahuje *neck*, takže příznaky z více úrovní nejsou nijak kombinovány, navíc je výstupní mřížka příliš hrubá (podvzorkování je rovno 64). To způsobuje špatnou přesnost jemné lokalizace. Interpretace výstupní vrstvy má jisté nedostatky. Jedna buňka může predikovat 2 rámečky, ale pouze jednu třídu. Velikost rámečku je předpovídána vůči vstupnímu obrazu, což komplikuje trénování. [30]

YOLOv2

Druhá verze nahrazuje plně propojené vrstvy konvolučními vrstvami, výstupní mřížka tak vzniká přirozeně a její velikost je dána pouze velikostí vstupního obrazu a podvzorkováním. Vstupní obraz může mít variabilní rozlišení, čehož je využito pří trénování na různě velkých obrazech. Přetrénování (*overfitting*) sítě je omezeno normalizačními vrstvami místo dropout vrstev. Architektura využívá jednoduchý *neck* – řetězí příznaky z poslední a předposlední úrovně podvorzkování, což zpřesňuje lokalizaci. Metoda však stále zaostává v přesnosti detekce malých objektů. Výstupní vrstva využívá kotevní rámečky, kterých predikuje 5 na jednu buňku mřížky, navíc predikuje pro každý rámeček samostatnou třídu. [31]

YOLOv3

Třetí verze navyšuje komplexnost části *backbone* o residuální bloky [41] (*bottleneck*). Výrazné vylepšení zaznamenal *neck*, který vychází z metody FPN [46]. Vrstvy nejvyšší úrovně jsou převzorkovány a jsou k nim přičteny mapy příznaků z nižších vrstev, jak ilustruje obrázek 2.4. Část *head* provádí detekce na 3 úrovních, čímž zlepšuje detekci malých objektů. Stejnou *head* aplikují i verze v4 a v5. Chybová funkce objektovosti a klasifikace se určuje na základě logistické regrese místo střední kvadratické chyby. Autoři zkoušeli rozšířit chybovou funkci o *focal loss*, ke zlepšení to ale nevedlo. YOLOv3 dosahuje skvělých výsledků v metrice COCO AP50, horších v AP50:5:95 přísnější na lokalizaci [47]. [18]

YOLOv4

Autoři 4. verze provádějí empirickou studii vlivu technik na přesnost a rychlost modelu. Tyto techniky dělí na dvě skupiny: *bag of specials*, které ovlivňují výpočetní náročnost modelu, a *bag of freebies*, které ji neovlivňují. Nejvhodnější techniky zahrnují do výsledného modelu. Mezi nejpodstatnější techniky patří: [16]

- **Bag of specials** – V části *backbone* zařazují BottleneckCSP bloky vytvořené po vzoru CSP struktury místo residuálních bloků z v3, což redukuje komplexitu modelu při zvýšení přesnosti. Za *backbone* zařazují upravený originální SPP blok pro zvýšení receptivního pole a kombinaci příznaků. V části *neck* nahrazují FPN strukturou PANet. Ta přidává k sestupné větvi z FPN metody ještě vzestupnou větev, autoři v4 navíc místo sčítání příznaků používají jejich řetězení. Do architektury zařazují DropBlock, lepší verzi dropout vrstvy. Za aktivační funkci volí Mish.
- **Bag of freebies** – V části chybové funkce zodpovědné za lokalizaci nahrazují střední kvadratickou chybu chybou CIoU. Na základě poznatků z vývoje v5 upravují kódování lokalizačních výstupních parametrů a aplikují mozaikovou augmentaci dat.

YOLOv5

Pátá verze vznikala paralelně se čtvrtou verzí, proto neobsahuje mnoho výrazných změn. Metoda se stále aktualizuje, v současnosti existuje vydání 6.1. Mezi hlavní přínosy patří mozaiková augmentace – síť je trénována na obrazech sestavených z 4 trénovacích obrázků. To mimo jiné umožňuje použít nižší minibatch při trénování, tedy trénovat na GPU s nižší pamětí [16]. Dále mění již zmíněné kódování lokalizačních výstupních parametrů. YOLOv5 nově definuje kotevní rámečky automaticky na základě použitého datasetu pomocí algoritmu k-means a genetického algoritmu. BottleneckCSP blok autoři mírně upravují, nový blok nazývají C3. Místo aktivační funkce Mish používají SiLU (Swish), přestože v testech ve v4 vykazuje mnohem horší výsledky [16]. Celou metodu autoři implementují v PyTorch a zaměřují se na použití na široké škále platforem od mobilních zařízení po výkoné cloudové platformy. [34]

2.3 Určení vzdálenosti objektů

Jedním z cílů této práce je určit vzdálenost detekovaných pokojových rostlin. Z detekce budou k dispozici pro každou rostlinu rámečky ohraničující danou rostlinu a její květináč. Zeleň rostliny může mít rozličný tvar, v rámečku mohou být zahrnuty jak pixely rostliny, tak pozadí či popředí. Určovat vzdálenost na základě zeleně by bylo obtížné a málo spolehlivé. Naopak květináče mívají v obraze nejčastěji obdélníkový tvar a i z různých úhlů tvoří jednolitou plochu, z které se dá vzdálenost určit spolehlivěji. Navíc robot potřebuje pro správný chod určit právě polohu květináče, ke kterému pojede nebo ho bude objíždět, proto budou při průzkumu metod uvažovány právě květináče.

2.3.1 Průzkum potenciálních metod

Na robotické platformě jsou umístěny webkamera a 360° LiDAR měřící v jedné rovině, k dispozici jsou i data z odometrie. Vhodná metoda by měla vyžadovat co nejméně dalších senzorů, které by navíc měly zabírat minimum místa (ideálně ne o moc více než použitá webkamera). Užitečné také bude, pokud nebude muset robot kvůli určení vzdálenosti vykonávat dodatečný pohyb. Metoda by také měla určovat vzdálenost v rozsahu přibližně

(0,5 – 3) m. Samotné vzdálenosti jednotlivých květináčů postačují, nicméně pro budoucí vývoj je užitečná i hloubková mapa. Ta by našla uplatnění např. při detekci překážek nebo lokalizaci.

Snímače vzdálenosti

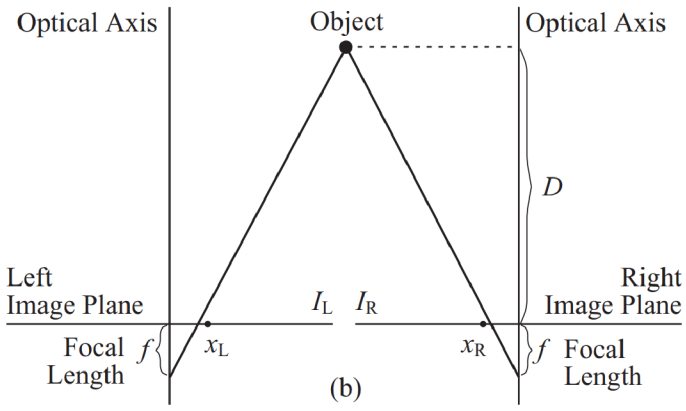
Relevantní snímač vzdálenosti přestavuje LiDAR (Light Detection And Ranging). Existují i v podobách, kdy snímají kruhové okolí v několika řadách paprsků. Takové LiDARy ale bývají drahé a rozměrné, nepřicházejí proto v úvahu.

Robot již obsahuje LiDAR, který snímá v jedné vodorovné rovině. Ten je umístěn 24 cm nad zemí, tím pádem není schopen zachytit nízké květináče, případně květináče ve výšce. Takové květináče by mohl zaměřit dálkoměr, který by byl umístěn na dvouosém mechanismu. Příkladem může být Lidar TF Luna [48], vyznačující se rozsahem (0,2 - 8) m s přesností 6 cm do 3 m nebo 2 % od 3 m, malými rozměry a přijatelnou cenou. Květináč o rozměru 10 cm × 10 cm by měl změřit až na 3 m. Dálkoměry však vyžadují precizní natočení mechanizmu, který by se musel natáčet ve dvou osách (*pitch* a *yaw*).

Stereovize

Stereovize využívá dvou kamer umístěných vedle sebe na přímce kolmé k optickým osám kamer (viz obrázek 2.7). Z obrazů I_L a I_R jsou extrahovány příznaky a ty následně párovány. Určena je mapa disparit $d = x_L - x_R$ [px], kde x_L a x_R jsou polohy bodu v obrazech I_L a I_R. Hloubková mapa se získá na základě rovnice 2.4, kde D je hloubka [m], f je ohnisková vzdálenost kamer [px] a l je jejich vzájemná vzdálenost [m]. [49, 50]

$$D = \frac{fl}{d} \tag{2.4}$$



Obrázek 2.7: Princip stereovize. Převzato z [50].

V posledních letech bývá ze vstupních obrazů určována mapa disparit také pomocí neuronových sítí. Metody postupně dosahují přesnějších výsledků s lepšími detaily než v případě klasických algoritmů. [51]

Na trhu existují také hotová zařízení s integrovaným hardwarem i softwarem. Příkladem je řada hloubkových kamer Intel RealSense. Cena těchto zařízení se na oficiálních stránkách pohybuje v rozsahu (259 – 399) amerických dolarů, na českém trhu více. Šířka zařízení činí (90 – 124) mm. Ideální pracovní rozsah se pohybuje od cca 0.5 m do 3 m, u dražšího

modelu do 6 m. Levnější modely disponují přesností a prostorovým šumem do 2 % při vzdálenosti do 2 m. [52]

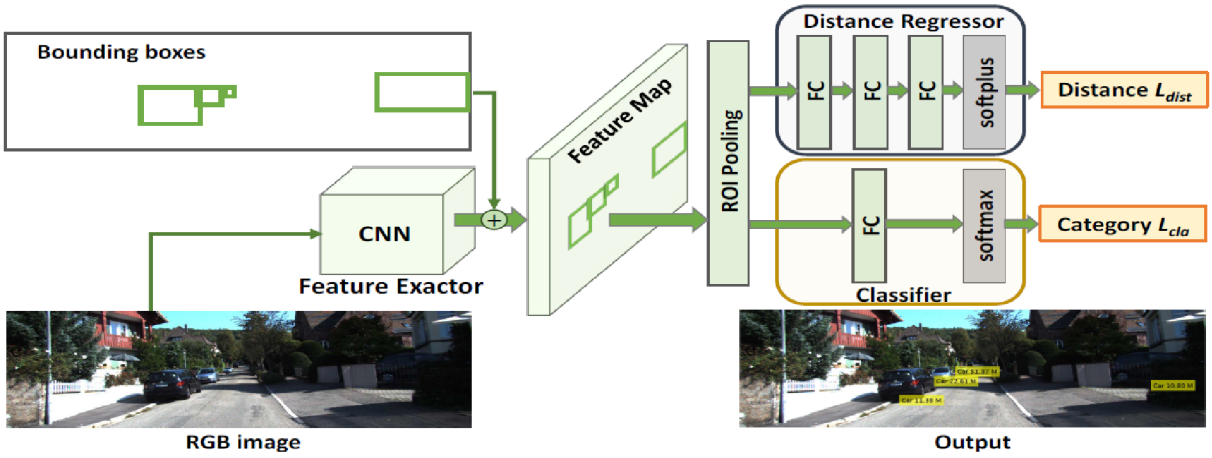
Specializované stereokamery poskytují dobrou přesnost, vyznačují se ale příliš vysokou cenou a velkými rozměry. Vlastní sestava by také zabírala na šířku místo, jelikož nemohou být z důvodu přesnosti kamery přímo vedle sebe.

Monokulární metody na bázi neuronových sítí

Díky pokroku v oblasti neuronových sítí existují metody, kterými lze určit vzdálenost květináčů pouze se senzory, které jsou na robotické platformě k dispozici (monokulární kamera, rovinný LiDAR a odometrie).

S pohybem kamery – Metoda z článku [53] určuje vzdálenost objektů na základě detekovaných rámečků a pohybu kamery. Autoři v článku volí k detekci novější variantu Faster R-CNN, lze ale použít libovolnou jinou metodu. Pro každý rámeček získávají 4 parametry – souřadnice středu, šířku a výšku. Z odometrie měří pohyb kamery ve 3 osách totožných se souřadnicovým systémem kamery, získávají tedy souřadnice x, y, z vůči počáteční poloze. Během pohybu robotu vytvářejí sekvence $\bar{X}$ alespoň 2 pozorování. Každé pozorování $\bar{x}_i$ je tvořeno těmito 7 parametry. Parametry rámečků autoři normují vůči rozměrům obrazu. Z parametrů pohybu určují posunutí mezi dvěma pozorováními, to poté normují vůči Euklidovské vzdálenosti posunutí z celé sekvence. Jednotlivá pozorování tvoří vstup Long Short-Term Memory (LSTM) rekurentní neuronové sítě. Výstup sítě společně se sekvencí vstupují do plně propojené vrstvy, výstupem je kýžená vzdálenost objektu. [53]

Autoři metody navíc prezentují nástroj, který vytvoří syntetický dataset sekvencí s realisticky nepřesnou podobou rámečků a pohybů kamery. To značně zvyšuje přesnost metody, navíc trénovací data lze vygenerovat vhodným nastavením nástroje. Metoda je také vůči detekci objektů výpočetně nenáročná. Metoda vyžaduje detekce o přijatelné přesnosti, kterou by YOLOv5 mělo zajistit. Nicméně není příliš praktické, že musí být robot s kamerou stále v pohybu a květináč během pohybu stále detekovat. [53]



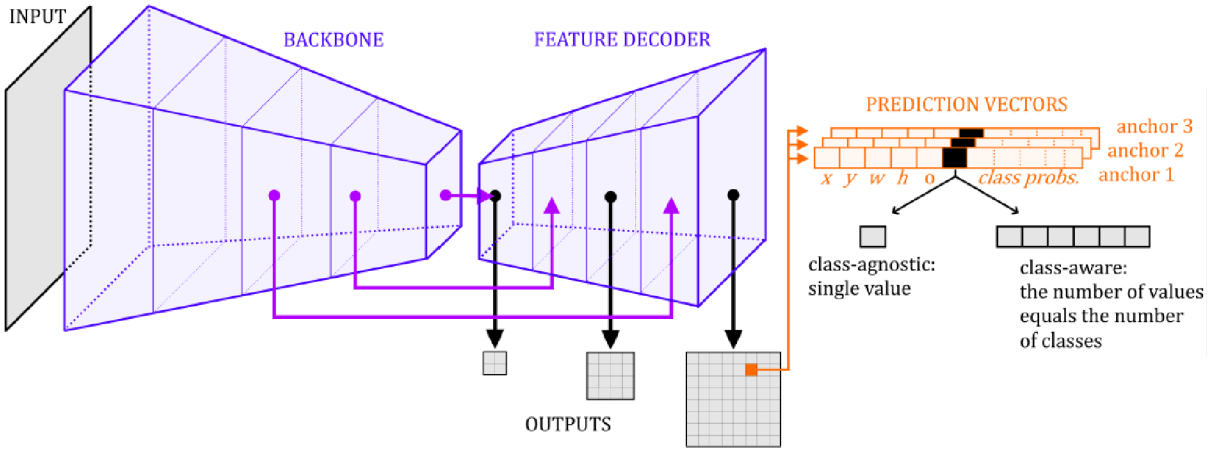
Obrázek 2.8: Predikce vzdálenosti pomocí R-CNN. Převzato z [55]

Bez pohybu kamery – Jednoduchou metodu prezentují autoři článku [54]. Určují vzdálenost objektů na železnici, např. aut a lidí. Pro každý rámeček, získaný z YOLOv3, sestavují vektor 6 parametrů – převrácenou hodnotu parametrů rámečku (šířky, výšky, diagonály) a průměrné dimenze objektů z dané třídy (šířky, výšky a hloubky). Vektor

je vstupem jednoduché plně propojené sítě, výstupem je metrická vzdálenost. Metoda funguje dobře na třídách s typickými rozměry, pro květináče není vhodná. [54]

Pokročilejší metoda je představena v článku [55]. Vychází z metody R-CNN, obohacená je o několik plně propojených vrstev predikujících vzdálenost detekovaných objektů (viz obrázek 2.8). Metoda využívá k určení vzdálenosti příznaky extrahované z původního obrazu. Chybovou funkci rozšiřuje o část *softplus*, která při trénování vyžaduje skutečné vzdálenosti objektů. Přesnost vylepšuje díky trénování s pomocnou větví, která předpovídá souřadnice x a y bodu objektu v souřadnicovém systému kamery. [55]

Článek [56] prezentuje podobnou metodu, Dist-YOLO, rozšiřující YOLOv3. Aplikuje enkodér-dekodér architekturu, pro každou buňku tří výstupních vrstev je předpovězena navíc vzdálenost (viz obrázek 2.9). K chybové funkci je přičítána střední kvadratická chyba mezi předpovězenou vzdálenosti a skutečnou vzdáleností. Díky využití YOLO je metoda velmi rychlá a predikce vzdálenosti téměř nezvyšuje výpočetní náročnost. [56]



Obrázek 2.9: Princip Dist-YOLO. Převzato z [56].

Metody předpovídající vzdálenost v rámci sítě vykazují veliký potenciál. Vyznačují se jednoduchostí architektury a rozumnými výpočetními nároky. Rešeršované metody jsou testovány na datech z dopravního prostředí, kde mají objekty jasně definovanou velikost. Zahrnutí sémantických informací by mělo přispět k spolehlivosti predikce vzdálenosti květináčů, bylo by to však nutno ověřit. Nevýhodou navíc je, že musí trénovací data obsahovat skutečné vzdálenosti objektů, což ztěžuje jejich sběr.

2.3.2 Monokulární predikce hloubky

V posledních letech rapidně vzrostl zájem o monokulární predikci hloubky (MPH), dominantně na bázi neuronových sítí. Jedná se o špatně podmíněnou úlohu, jíž cílem je z jediného obrazu předpovědět hloubkovou mapu. Ta mívá typicky stejné rozlišení jako vstupní obraz. [51, 57]

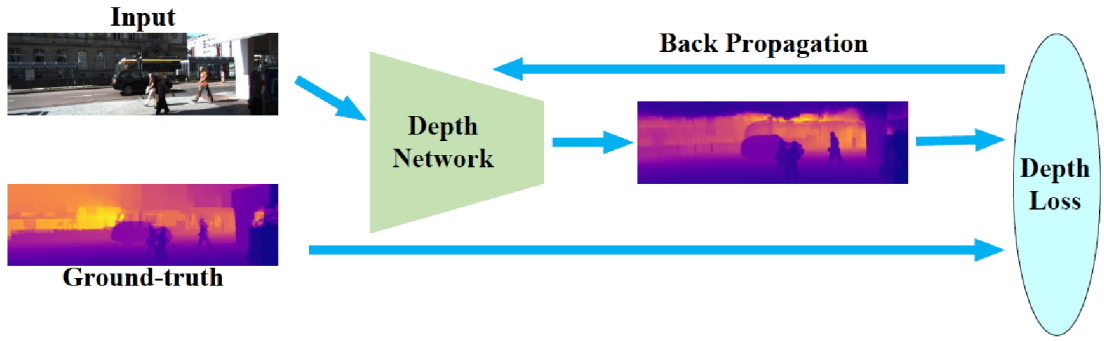
Interpretace výstupu sítě, tedy hodnot hloubkové mapy, se liší od použité metody. Nejužitečnější bývá absolutní (metrická) hloubka v jednotkách délky. Další možností je relativní hloubka, kterou lze převést na metrickou pomocí škálovací konstanty α [58]. Častým případem je disparita, což je inverzní relativní hloubka, kterou lze na metrickou přepočíst opět pomocí α [59]. Relativní hloubka a disparita mohou být normovány vzhledem k rozměru trénovacího obrazu, v takovém případě je nutno α vynásobit tímto rozměrem

[58]. [51, 57]

Rozdělení metod

Metody MPH se rozdělují podle způsobu trénování na *supervised*, *unsupervised* a *semi-supervised*, což úzce souvisí s charakterem použitých dat a s výstupem sítě. [51, 57]

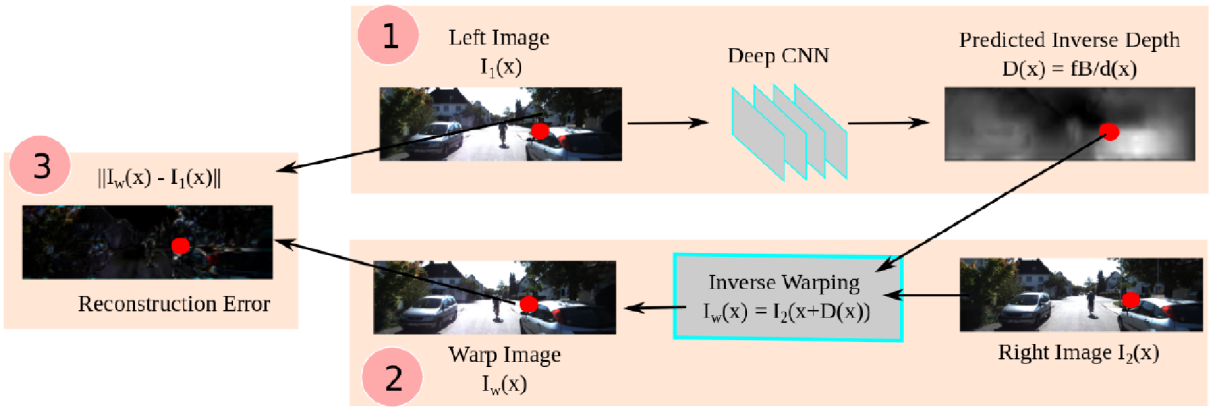
Supervised – Nejstarší supervised metoda pro MPH je prezentována v [60]. Obecný princip ilustruje obrázek 2.10. Supervised metody jsou trénovány na obrazech o známé metrické hloubce, díky tomu predikují metrické hloubkové mapy. Vzorové hloubkové mapy (ground-truth) se pořizují kvalitními RGB-D či LiDAR senzory, sběr datasetu je tedy komplikovaný a drahý. Tento typ metod patří v současnosti mezi nejpřesnější. [51, 61, 62]



Obrázek 2.10: Princip supervised metody. Převzato z [51]

Unsupervised (Self-supervised) – Nároky na tvorbu datasetu snižují unsupervised metody, které těží z geometrických závislostí mezi obrazy pořízenými z mírně odlišných míst. Dělí se dále na dvě větve: jedna využívá stereo obrazy, druhá monokulární video. [51]

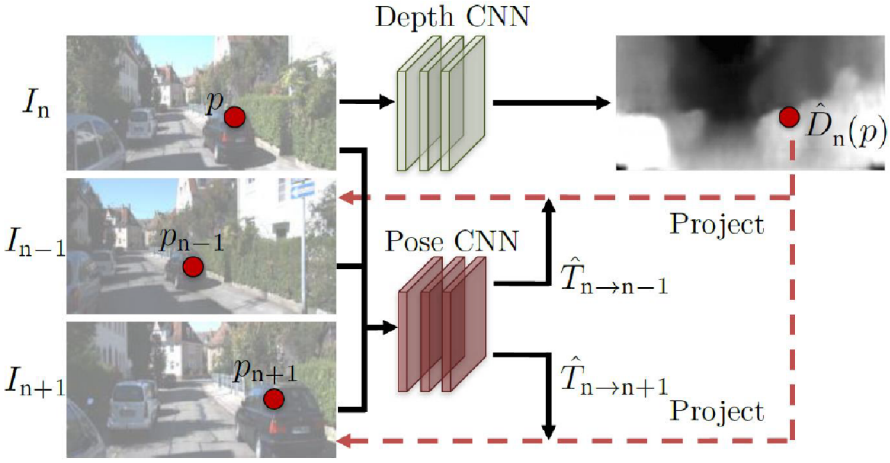
Trénování na neoznačených stereo párech započal článek [59]. Princip trénování zachycuje obrázek 2.11. Z levého obrazu I_1 je predikována mapa disparit. Z pravého obrazu I_2 je na základě disparity rekonstruován levý obraz I_w (na obrázku *warping*). Obrazy I_1 a I_w jsou porovnány v rámci chybové funkce. Pokud je stereokamera pro sběr datasetu kalibrovaná, metrickou hloubku lze vypočíst jako u stereovize dle vztahu 2.4. [51, 59]



Obrázek 2.11: Princip unsupervised metody s využitím stereo párů. Převzato z [59].

Trénování na monokulárním videu prezentovala poprvé metoda SfMLearner [63]. Její princip se podobá stereu, přibližuje ho obrázek 2.12. K trénování slouží sekvence 3 po sobě

jdoucích obrazů. Z prostředního obrazu I_n je predikována mapa disparit sítí pro MPH, zároveň jsou predikovány translace a rotace kamery mezi obrazy dané sekvence pomocí neuronové sítě *pose estimation network* (PE síť). Z predikovaných údajů jsou rekonstruovány zbylé obrazy sekvence I_{n-1} a I_{n+1}. Ty jsou v chybové funkci opět porovnány s původními obrazy. Obě sítě jsou trénovány současně. Nevýhodou tohoto typu metod je, že předpovídají pouze relativní hloubku, protože PE síť nedokáže predikovat translaci v metrech. K převodu na metrickou hloubku slouží α, která se ale musí určit pro každý snímek zvlášť (např. LiDARem), jelikož predikovaná relativní hloubka je z důvodu nedokonalostí PE sítě nekonzistentní. Nekonzistentnost se pokoušejí vyřešit metody [58, 64], ale i přesto se směrodatná odchylka škály α pohybuje okolo 9 % [58]. Článek [65] navrhuje, jak určit α pouze z obrazu, jejich aplikace je však použitelná na dopravním prostředí, hůře na vnitřním prostředí. Článek [66] upozorňuje, že přesnost těchto metod negativně ovlivňují vibrace v trénovacím videu, což lze minimalizovat korekcí datasetu. [51, 57, 63]



Obrázek 2.12: Princip unsupervised metody s využitím monokulárního videa. Převzato z [63].

Semi-supervised – Modely těchto metod se často trénují na malém označeném datasetu a velkém neoznačeném datasetu. Nejprve se natrénuje síť na označených datech, poté jsou sítí vygenerovány pseudo ground-truth pro neoznačená data a trénováno je na všech datech dohromady [51]. Jiné metody mohou aplikovat sporadická ground-truth, např. z LiDARu [57]. Někdy se do semi-supervised metod řadí i metody využívající kalibrované stereo páry [57]. Přesnost semi-supervised metod se pohybuje mezi přesností supervised a unsupervised metod [51, 57].

Přehled metod

Charakteristiky podstatných metod MPH jsou popsány v této části. Vybrané metody jsou poté experimentálně ověřeny a srovnány v kapitole 4.2. Mezi nejznámější datasety, na kterých se metody srovnávají, patří širokoúhlý KITTI [67] z dopravního prostřední a NYUv2 [68] z vnitřního prostředí. Obecně nejvýraznějších vylepšení dosahují metody díky aplikaci vhodnějších chybových funkcí [51].

DenseDepth – Tato supervised metoda z roku 2018 staví na jednoduché enkodér-dekodér architektuře. Enkodér je část sítě, která zmenšuje rozměr obrazu a extrahuje příznaky. Část dekodér postupně rozměr zvětšuje na původní rozměr obrazu, konkrétně na hloubkovou mapu. Metoda predikuje metrickou hloubku. K dispozici jsou natrénované

modely na KITTI a NYUv2. Jak ukazuje článek [51], metoda dosahuje přijatelných 3 FPS na výkonném embedded počítači Jetson AGX Xavier. [61]

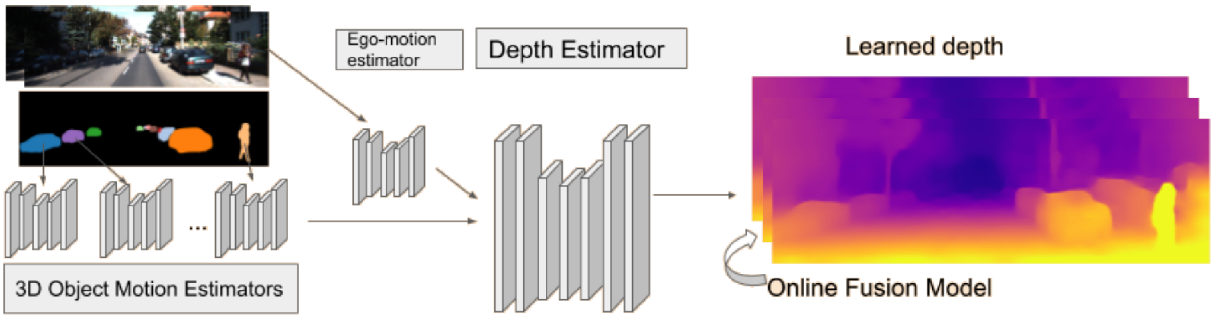
AdaBins – Supervised metoda AdaBins (2021), navazuje na DenseDepth. Enkodér-dekodér architekturu rozšiřuje o komplexní strukturu inspirovanou strukturou Vision Transformer (ViT) [69]. Predikuje metrickou hloubku o vysoké kvalitě [51]. [62]

FastDepth – Aplikací MPH v reálném čase na slabším hardware se zabývá FastDepth (2019). Na nejvýkonnějším embedded zařízení rodiny NVIDIA Jetson (TX2) dosahuje až 178 FPS. Z porovnání metriky δ_1 v článcích [70] a [51] je vidět, že ji v přesnosti výrazně překonávají téměř všechny ostatní významné metody. [70]

DepthNet Nano – S rychlostí 14 FPS na Jetson AGX Xavier, supervised metoda DepthNet Nano (2020) dosahuje vyšší rychlosti než DenseDepth při stejné přesnosti, nabízí tak výrazně přesnější, avšak stále rychlou, alternativu k Fast Depth [51, 70, 71].

MonoDepth2 – Monodepth (2017) [72] je self-supervised metoda trénující na kalibrovaných stereo obrazech, její výstup lze převést na metrickou hloubku. Nová self-supervised verze, MonoDepth2 (2019) [58], trénuje na kalibrovaných stereo obrazech, monokulárním videu nebo kombinaci obou. Výborných výsledků dosahuje převážně díky několika navrženým chybovým funkcím. K dispozici jsou modely natrénované na KITTI. [58]

SC-SfMLearner – Tato unsupervised metoda z roku 2019 trénuje na monokulárním videu. Zaměřuje se na problém nekonzistentnosti α. Díky chybové funkci, která porovnává predikované hloubkové mapy dvou po sobě jdoucích obrazů, vylepšují predikci PE sítě a tím i konzistentnost α v rámci sekvence obrazů. [64]



Obrázek 2.13: Princip metody Struct2Depth. Převzato z [74].

Struct2Depth – Trojice článků [73, 74, 75], publikovaných v letech 2019 a 2020, představuje a rozšiřuje unsupervised metodu Struct2Depth trénovanou na monokulárním videu. Její princip ilustruje obrázek 2.13. Tvořena je enkodér-dekodér sítí pro MPH a PE sítí. Oproti podobným metodám vyžaduje masky objektů, které by se mohly pohybovat, získané pomocí instanční segmentace. Obsahuje síť, která se učí předpovědět translaci a rotaci těchto objektů mezi obrazy trénovacích sekvencí. To vede k přesnější rekonstrukci obrazu a kvalitnějším výsledkům ve vysoce dynamických scénách, typických pro dopravní prostředí nebo vnitřní prostředí s lidmi. Poslední z článků zlepšuje predikci pohybu objektů. Masky objektů nejsou vyžadovány. Síť pro predikci pohybu objektů je nahrazena sítí pro predikci pohybových map, které predikují pohyb každého pixelu v rámci dvou obrazů. Vylepšená metoda se z videa dokáže naučit parametry matice kamery, která je nutná pro rekonstrukci obrazu a bývá jinak určena kalibrací kamery. [73, 74, 75]

MiDaS – Metoda MiDaS [76] z roku 2020 se zaměřuje na robustní MPH v libovolné scéně. Architektura je založena na konvoluční enkodér-dekodér síti. Na rozdíl od ostatních

metod, trénování probíhá na kombinaci datasetů, včetně 19 3D filmů. Některé datasety obsahují vzory o metrické hloubce, jiné disparitu o známé či neznámé škále α, u 3D filmů může být disparita ještě posunutá o β a může nabývat i záporných hodnot. Proto jsou datasety sjednoceny tak, aby byly vzory jednotlivých obrazů v podobě disparity. Navržené chybové funkce jsou vůči α a β invariantní, což zapříčiňuje nekonzistentní α a β predikovaných map disparit napříč jednotlivými obrazy. Síť ale díky tomu může být trénována obdobně, jako v případě supervised metod. Metrickou hloubku D lze z predikované disparity d určit podle rovnice 2.5.

$$\frac{1}{D} = \alpha d + \beta \tag{2.5}$$

Rozšíření metody MiDaS, Dense Prediction Transformer (DPT) [77], z roku 2021 nahrazuje konvoluční enkodér moderní ViT architekturou [69]. Navrženy jsou dva modely – DPT-Hybrid a DPT-Large. Hybridní model rozděluje obraz na oblasti, pro každou extrahuje konvoluční sítí vektor příznaků určený pro transformer. Způsob trénování je totožný s metodou MiDaS, zdvojnásobuje se množství datasetů. Trénováno je na bezmála 1,5 milionu obrazů, potenciál ViT je tak naplno využit. DPT díky inovacím predikuje kvalitnější hloubkové mapy s jemnějšími detaily než MiDaS při stejné výpočetní náročnosti. [76, 77]

Boosting Monocular Depth (BMD) – Tato metoda z roku 2021 dokáže predikovat hloubkové mapy na obrazech o téměř libovolném rozlišení (např. desítky MPx). Na pozadí využívá MiDaS pro iterativní odhad hloubky na více úrovních rozlišení a výsledky vhodně slučuje. Výpočetní čas metody je obrovský. [78]

Zhodnocení monokulární predikce hloubky

Metody MPH dosahují v posledních letech velkých pokroků. Jejich výhodou pro tuto práci je, že není nutno na robota přidávat dodatečné senzory – postačují kamera, případně LiDAR. Také není k určování vzdálenosti vyžadován pohyb robotu. Navíc se jedná o novou oblast a pro další využití je užitečné její potenciál prozkoumat. Nevýhoda spočívá ve vyšší výpočetní náročnosti nebo naopak horší kvalitě výstupu – je nutno najít přijatelnou rovnováhu. V blízkých letech lze však očekávat zlepšení obou charakteristik.

2.4 Identifikační štítky

Identifikační štítky (častěji referenční značky, *fiducial markers*) jsou kontrastní rovinné značky obvykle čtvercového tvaru, složené z jednotlivých černých a bílých čtverečků (bitů). V této práci mají sloužit ke stanovení identifikátoru rostliny a k určení polohy a natočení kamery vůči štítku připevněnému na květináči. K určení polohy a natočení je nutno znát parametry kamery, které lze získat její kalibrací. [49]

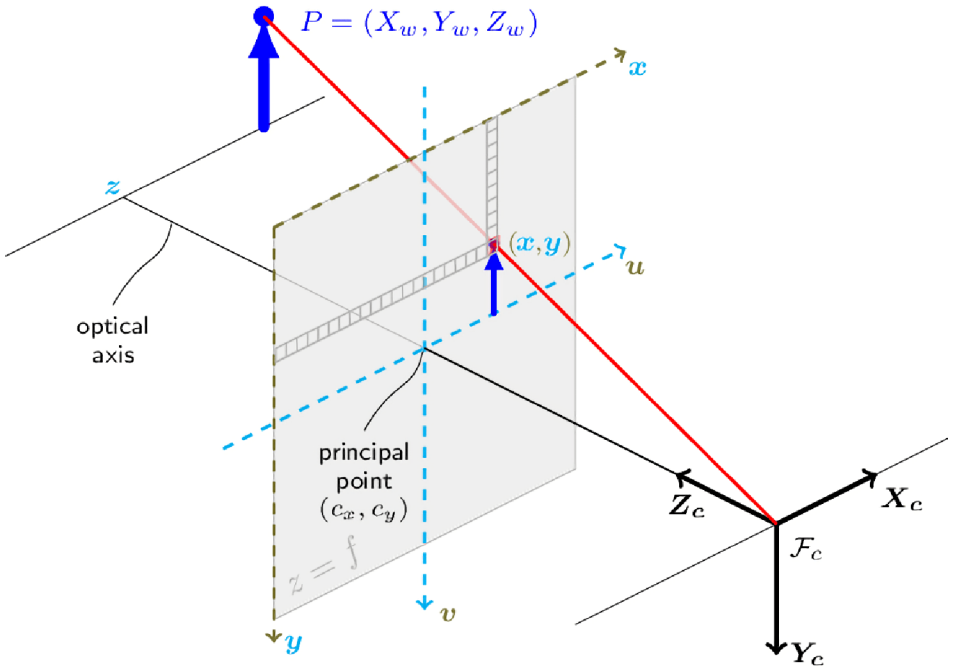
2.4.1 Model kamery

Běžným modelem kamery je dírkový model (*pinhole model*), znázorněný na obrázku 2.14. Počátek globálního souřadnicového systému je zvolen v optickém středu $\mathcal{F}_c$. V ohniskové vzdálenosti f od $\mathcal{F}_c$ se nachází hlavní bod obrazové roviny (*principal point*). Hlavním bodem prochází rovina obrazu, která má počátek svého souřadnicového systému v levém horním rohu obrazu. Bod P v prostoru se promítne do obrazové roviny dle rovnic 2.6. Model lze rozšířit o zkreslení objektivu. [49, 79]

V modelu figurují vnitřní a vnější parametry. Mezi vnitřní parametry patří ohniskové

vzdálenosti f_x, f_y [px] a souřadnice hlavního bodu c_x, c_y [px]. Společně tvoří matici kamery. Další vnitřní parametry definují radiální a tangenciální zkreslení. Vnitřní parametry jsou charakteristickým rysem každé kamery a určují se její kalibrací. Vnější parametry určují translaci a rotaci souřadnicového systému kamery vůči globálnímu.

$$\begin{aligned} x &= f_x \frac{X_\omega}{Z_\omega} + c_x \\ y &= f_y \frac{Y_\omega}{Z_\omega} + c_y \end{aligned} \tag{2.6}$$



Obrázek 2.14: Model dírkové kamery. Převzato z [49] a upraveno.

2.4.2 Typy identifikačních štítků

Mezi běžné štítky využívané v robotice a rozšířené realitě patří:

- **QR kód** – Jedná se o 2D čárový kód, do kterého je možné zakódovat přes 7000 znaků. Lze ho využít i pro určení orientace kamery. Z důvodu vysokého počtu bitů musí při snímání vyplňovat velkou část zorného pole. [80]
- **AprilTags** – Tento typ štítků obsahuje předdefinované sady čtvercových či kruhových štítků o malém počtu bitů (nejčastěji 6×6). Každá sada disponuje desítkami až tisíci identifikátory. V závislosti na Hammingově vzdálenosti sady je možno opravit několik chybně detekovaných bitů. Identifikace funguje i při zakrytí malé části štítku. [81]
- **ArUco** – ArUco jsou čtvercové štítky velmi podobné AprilTags. Liší se převážně generováním bitů a identifikačním algoritmem. Ve srovnání s AprilTags nabízejí sady s větším množstvím identifikátorů při stejném počtu bitů za cenu snížení Hammingova okna [49]. Lze tak využít i štítky s menším počtem bitů (4×4) a detekovat je na větší vzdálenost. Nevýhodou je, že identifikace selhává při překryvu štítku. [82]

3 Cíle práce

Během rešeršní studie docházelo ke konkretizaci celého projektu a tím pádem i požadavků na výstup této práce. Náplň práce lze rozdělit na tři větší celky. Pro každý celek je v této kapitole vysvětleno, jaký je jeho význam v rámci projektu. Specifikovány jsou požadavky na celek a jeho výstupy. Na základě rešeršní části je odůvodněn výběr vhodných metod a nastíněn další postup, jak budou zvolené metody implementovány.

Detekce květin a květináčů

Detekce květin slouží k nalezení květin v prostoru i v případě, že se nenacházejí na očekávaném místě nebo když není viditelný identifikační štítek. Cílem je ve vnitřním prostředí detekovat pokojové rostliny různého druhu a velikostí. Jak bylo zmíněno v sekci 2.3, pro potřeby určení vzdálenosti květin jsou vyžadovány i detekce květináčů. V projektu je uvažováno umístění květin na podlaze, pro snadnou rozšiřitelnost je však nutno počítat i s umístěním ve výšce (např. na nábytku a parapetech). Detekce květin bude probíhat za pomalého pohybu robotu, proto je žádoucí rychlost detekce aspoň 5 FPS.

Uvedené požadavky na rychlost a dobrou přesnost by měly splňovat menší modely metody YOLOv5. Dále je potřeba shromáždit a anotovat reprezentativní dataset, na kterém budou modely natrénovány. Proběhne také třídění detekcí na páry květina-květináč.

Určení polohy a orientace identifikačního štítku na malou vzdálenost

Identifikační štítek, nalepený na květináči, slouží k jednoznačné identifikaci dané květiny a odkazu na databázi. Dalším smyslem je určení polohy a orientace robotu vůči štítku s dobrou přesností tak, aby robot dojel do plánované polohy pro zalévání. Štítky by mohly také sloužit pro navádění robotu ke zdroji vody, který by byl nejspíše součástí dokovací stanice a v rámci projektu nebyl uvažován. Pracovní vzdálenost štítků od kamery se předpokládá přibližně do 1 m.

Zvoleny byly štítky ArUco, které nabízejí velký počet identifikátorů (až 1000) i pro malý počet bitů. Algoritmy detekce ArUco štítků jsou dostupné v knihovně OpenCV. Dokáží pracovat i s AprilTag, které mohou být lepší při překryvu štítku částmi květin.

Určení polohy květináčů na velkou vzdálenost

Znalost polohy květináčů bez použití štítků je potřebná v případě, že není štítek přímo viditelný a je nutno květináč objíždět, nebo při hledání neznámých květin. Určuje se poloha okraje květináče na předpokládanou vzdálenost (1 – 5) m. Robot při tom bude stát, není tudíž vyžadován krátký výpočetní čas.

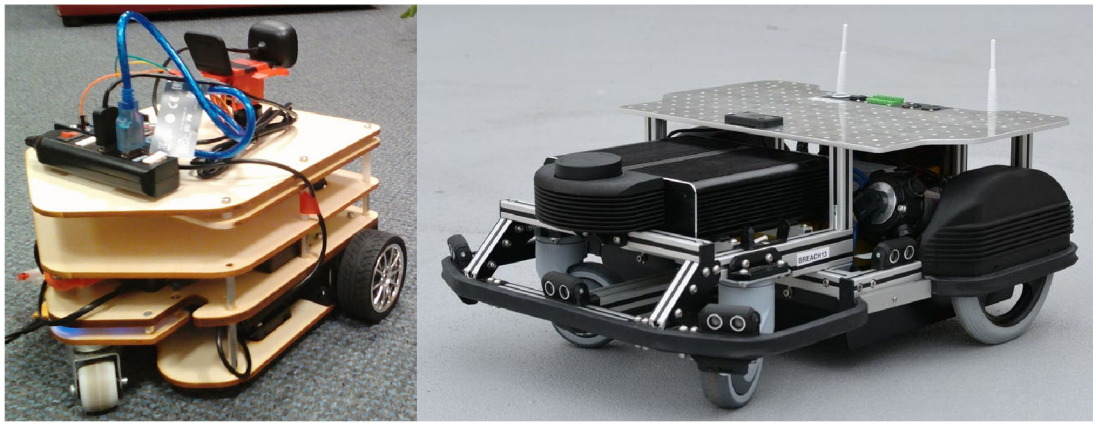
K určení vzdálenosti květináče byly vybrány metody monokulární predikce hloubky. Nevyžadují dodatečné senzory ani pohyb robotu, navíc se jedná o novou oblast s širším potenciálem. Pro výběr konkrétní metody je nutno metody nejprve otestovat na vlastních datech. Vybraná metoda buď dokáže určit přímo metrickou vzdálenost, nebo bude sloučena s LiDARem. Z detekovaných rámečků a predikované vzdálenosti bude určena poloha.

4 Postup řešení a výsledky

4.1 Použitý hardware a software

Práce na projektu probíhaly na dvou robotických podvozcích, které jsou zobrazeny na obrázku 4.1. Leela sloužila k testování algoritmů, Breach byl cílovou platformou. Mezi nejvýznamnější hardware, který byl na robotech k dispozici a byl v této práci aktivně využíván, patří:

- **Microsoft LifeCam HD 3000** – Jedná se o běžnou webkameru s pevným zaostřením na 0,7 m s hloubkou ostrosti (0,3 – 1,5) m. Webkamera disponuje maximálním rozlišením (1280 × 800) px při 10 FPS nebo (1280 × 720) px při 30 FPS. Dalšími užitečnými rozlišeními jsou (640 × 360) px a (424 × 240) px. Maximální úhel záběru činí přibližně 59° horizontálně a 39° vertikálně.
- **RPLiDAR A1M8** – Tento laserový skener funguje na bázi triangulace. Měří v jedné rovině v kruhovém rozsahu 360° s krokem 1° s frekvencí kruhových skenů až 7 Hz. Na robotu měří vzdálenost v rozsahu (0,31 – 8,00) m.
- **NVIDIA Jetson Nano 4GB** – Jedná se o finančně dostupný embedded počítač vhodný pro běh algoritmů umělé inteligence na robotických zařízeních. Nano disponuje GPU Tegra X1 s Compute Capability 5.3, která sdílí 4 GB paměti s CPU.



Obrázek 4.1: Používané robotické platformy: vlevo Leela, vpravo Breach. [83]

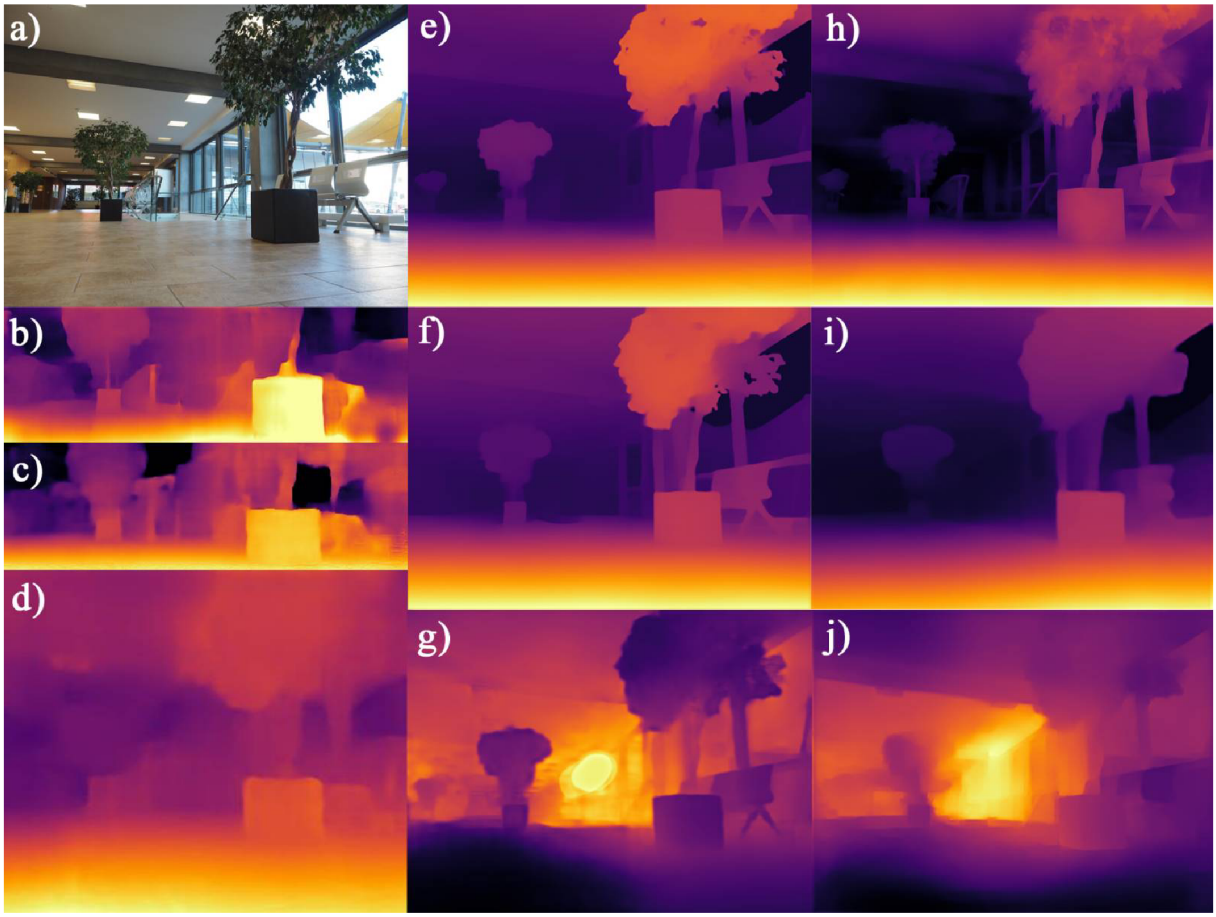
Vývoj algoritmů probíhal na PC s operačním systémem Ubuntu 20.04, veškerý zdrojový kód byl psán v jazyce Python verze 3.8.10. Na robotických platformách běžely algoritmy na Jetson Nano. Instalace software na Jetson byla provedena pomocí připraveného obrazu Micro SD karty z [84]. Nejnovější verze Ubuntu totiž nebyla v době instalace podporovaná v oficiálních balíčcích JetPack. Mezi hlavní software připravený a doinstalovaný na Jetson patří Ubuntu 20.04, JetPack 4.6-b197, Python 3.8.10, ROS Noetic, PyTorch 1.9.0, OpenCV 4.5.3.

4.2 Testování metod MPH

Všechny metody monokulární predikce hloubky prezentují v článcích kvalitativní a kvantitativní výsledky dosažené na běžných trénovacích datasetech. Mnohdy ale nemusejí dosahovat stejných výsledků na obrazech mírně odlišného charakteru. Bylo proto přistoupeno k rozsáhlému srovnání metod na obrazech z veřejných datasetů a především vlastních obrazech z vnitřního prostředí s květinami. Na základě tohoto srovnání bylo možné vybrat nejvhodnější metodu pro účely určení vzdálenosti detekovaných květináčů.

Metody byly testovány s využitím oficiálních zdrojových kódů metod v jazyce Python zveřejněných v příslušných GitHub repositářích. Obvykle obsahovaly skripty pro běh nebo dema v prostředí Google Colab. V kódech byly učiněny drobné úpravy zejména pro ukládání samostatných výstupních hloubkových map a sjednocené vykreslování škálovaných hloubkových map v barevné mapě *inferno*.

Testování probíhalo lokálně na PC s Windows 10 v prostředí Anaconda. Pro každou metodu bylo vytvořeno samostatné virtuální prostředí a nainstalovány balíčky dle instrukcí každé metody. Z důvodu stáří použité grafické karty (NVIDIA 920M) nebylo možné nainstalovat novější verze frameworků PyTorch (1.3.1+) a TensorFlow (2.4+), proto byly některé metody testovány také v online prostředí Google Colab. To nabízí bezplatný přístup ke GPU (nejčastěji byla přidělena NVIDIA K80 11GB) a předinstalované balíčky.



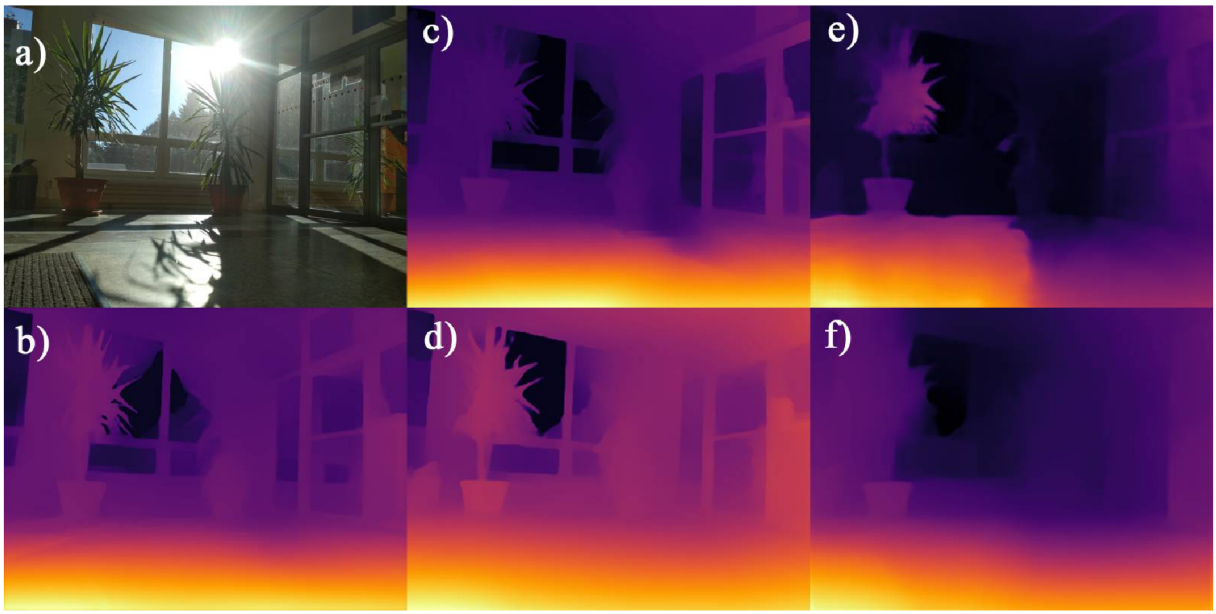
Obrázek 4.2: Metody MPH ve vnitřním prostředí s květinami: a) původní obraz, b) Monodepth2, c) Struct2Depth, d) SC-SfMLearner, e) DPT-hybrid, f) DPT-large, g) AdaBins NYUv2, h) BMD, i) MiDaSv2.1 small, j) DenseDepth.

Zprovozněno a vyzkoušeno bylo celkem 8 metod rozebíraných v podkapitole 2.3.2. Přehledné srovnání testovaných metod bude uvedeno na konci této podkapitoly. Metoda MiDaS rozšířená o DPT poskytovala 4 natrénované modely – MiDaSv2.1, MiDaSv2.1small, DPT-hybrid a DPT-large. DPT-hybrid obsahoval i variantu dotrénovanou na datasetu NYUv2. Podobně metoda AdaBins nabízela modely natrénované na KITTI nebo NYUv2.

Metody byly testovány a vizuálně posuzovány na obrazech z 5 kategorií:

- testovací obrazy datasetu KITTI z dopravního prostředí
- vlastní fotografie z dopravního prostředí
- testovací obrazy datasetu NYUv2 z vnitřního prostředí
- vlastní fotografie z vnitřního prostředí s květinami
- vlastní fotografie z různého prostředí – krajina, příroda, město, architektura

V této podkapitole jsou metody srovnány na ukázkové vlastní fotografii z vnitřního prostředí, které je pro účely zalévacího robotu nejpodstatnější. Srovnání na zbylých 4 kategoriích je uvedeno v příloze B. Výsledky na dalších 52 fotografiích jsou k nahlédnutí v elektronické příloze. Predikované hloubkové mapy na zmíněné fotografii jsou na obrázku 4.2. Metody AdaBins a DenseDepth predikují metrickou hloubku, ostatní metody predikují inverzní hloubku, proto je barevná škála opačná.



Obrázek 4.3: Metoda MiDaS ve vniřním prostředí s květinami: a) původní obraz, b) DPT-large, c) DPT-hybrid, d) MiDaSv2.1, e) DPT-hybrid NYUv2, f) MiDaSv2.1 small.

Metody Monodepth2, SC-SfMLearner a Struct2Depth jsou trénovány na KITTI, pro vnitřní prostředí se natrénované modely nehodí. Na ukázkovém obrázku dosahují přijatelných výsledků, na ostatních obrazech z vnitřního prostředí obsažených v elektronické příloze selhávají. Modely jsou však rychlé a po natrénování na vlastním datasetu by mohly dosahovat slibných výsledků. U metody Struct2Depth se ale podařilo připravit trénování pouze na CPU, jelikož 11 GB paměti nejčastěji přidělené GPU v Google Colab nestačilo. Tyto tři metody proto dále uvažovány nebyly.

Metoda SfMLearner byla zprovozněna pouze v Colabu. Využívá python2, který přestal být v Colabu podporován, proto je uveden její výstup pouze z prvotního testování na vlastním obraze z dopravního prostředí v příloze B. Metoda je staršího data a predikuje hloubkové mapy ve velmi nízkém rozlišení.

Tabulka 4.1: Srovnání metod MPH

Metoda	**SfMLearner**	**DenseDepth**	**Monodepth2**	**SC-SfMLearner**
Rok vydání	2017 [63]	2018 [61]	2019 [58]	2019 [64]
Způsob trénování	unsupervised stereo	supervised	unsupervised monovideo, stereo	unsupervised monovideo
Hloubka	relativní (α)	metrická	relativní (α)	relativní (α)
Framework	TensorFlow1	TensorFlow1,2	PyTorch	PyTorch
Vstupní obraz	širokoúhlý (416×128)	širokoúhlý nebo 4:3 (640×480)	širokoúhlý (1024×320, 640×192)	širokoúhlý nebo 4:3 (320x256)
Dataset	KITTI	NYUv2, (KITTI)	KITTI	NYUv2, (KITTI)
Složitost	nízká	nízká	nízká	nízká
Testování	Colab	Colab	PC	PC
Kvalita výstupu	nízká	střední	střední	nízká
Výpočetní náročnost	nízká	střední	nízká	nízká
Metoda	**Struct2Depth**	**MiDaS+DPT**	**AdaBins**	**BMD**
Rok vydání	2019 [75]	2020+2021 [76, 77]	2021 [62]	2021 [78]
Způsob trénování	unsupervised monovideo	kombinovaný	supervised	/
Hloubka	relativní (α)	relativní (α, β)	metrická	relativní (α, β)
Framework	TensorFlow1	PyTorch, TF2	PyTorch	PyTorch
Vstupní obraz	širokoúhlý (416×128)	kratší strana 384 px, nejlépe 1:1	kratší strana do 500 px	libovolný, megapixely
Dataset	KITTI	různé, 3D filmy	KITTI, NYUv2	/
Složitost	nízká	vysoká	vysoká	vysoká
Testování	Colab a PC	Colab a PC	Colab	Colab
Kvalita výstupu	střední	střední, vysoká	vysoká	velmi vysoká
Výpočetní náročnost	nízká	vysoká	nízká, vysoká	velmi vysoká

Metody AdaBins a DenseDepth představují výhodu v přímé predikci metrické hloubky. Při využití jiné kamery než při trénování by mělo stačit nalézt škálovací konstantu pouze jednou a tím model kalibrovat. DesnseDepth dosahuje skvělých výsledků na NYUv2, na kterém byl trénován. Na vlastních snímcích z vnitřního prostředí dosahuje obecně dobrých

výsledků, často ale vykazuje výraznější chyby. AdaBins vykresluje mnohem kvalitněji detaily, hloubkové mapy jsou vizuálně také přesnější. Nicméně v jistých částech obrazů také chybuje. Z obrazů s měřítkem metrické hloubky (součástí elektronické přílohy) je patrné, že škálovací konstanta není pro všechny obrazy totožná. Metoda BMD, která využívá v základu DPT-hybrid, dosahuje jednoznačně nejlepších výsledků na obrazech o rozlišení několika megapixelů. Její výpočetní čas je ale obrovský (v řádu desítek sekund až minut i na moderních GPU). Pro tuto práci není tato metoda použitelná.

Skvělých výsledků dosahují modely metody MiDaS a jeho rozšíření DPT, které jsou robustní v jakémkoliv prostředí. Výborně vykreslují detaily, takže dokáží spolehlivě zachytit i malé květináče na vzdálenost několika metrů, což je pro tuto práci klíčové. Nejvhodnější se zdá být DPT-hybrid, který je ve srovnání s DPT-large 2,5× menší (470 MB oproti 1200 MB) a kvalitou výrazně nezaostává. Stejně tak je výpočetně méně náročný než AdaBins (900 MB), dle testů v [85] je AdaBins o 30 % pomalejší. Model DPT-hybrid se při prvotních testech podařilo zprovoznit na Jetson Nano s využitím GPU. Zajímavou alternativu k DPT-hybrid představuej MiDaSv2.1small. Ten sice nevykresluje zdaleka tolik detailů, ale hloubková mapa poměrově téměř vždy odpovídá. Navíc je asi 10× rychlejší než DPT-hybrid. Srovnání jednotlivých modelů metody MiDaS ilustruje také obrázek 4.3.

Dosažené poznatky jsou shrnuty v tabulce 4.1. Na základě prezentovaného porovnání byl pro další postup vybrán model **DPT-hybrid**, pro experimenty s rychlejším modelem také **MiDaSv2.1small**. Jelikož zvolené modely predikují relativní hloubku s proměnnou škálou α a posunutím β, bude nutné vytvořit algoritmus, který tyto parametry určí pomocí LiDARu, čímž se získá metrická hloubka.

4.3 Detekce květin

Tato podkapitola se zabývá tvorbou detektoru květin a květináčů a jeho zasazením do projektu ve frameworku ROS. Pro vytvoření samotného detektoru bylo potřeba nejdříve vytvořit dataset sběrem obrazů a jejich anotací, poté zvolit vhodnou architekturu a naladit hyperparametry trénování. Dále bylo nutno detekce zpracovat, především přiřadit každé květině správný květináč pro pozdější určování jeho vzdálenosti. Závěrem byl detektor implementován v ROSu a zařazen do celého projektu.

4.3.1 Tvorba datasetu

Na základě požadavku na rychlost detekce alespoň 5 FPS na Jetson Nano bylo předpokládáno, že zvolený detektor bude moci pracovat s rozlišením delší strany maximálně 640 px. Datová sada by se proto měla skládat z obrazů o minimálně takovém rozlišení. Ideální je vyšší rozlišení dat jednak pro možné rozšíření detektoru, jednak pro přesnější anotování jednotlivých objektů.

Veřejně dostupné datové sady

Nejprve byl proveden průzkum veřejných datasetů, které by mohly obsahovat květiny ve vnitřním prostřední. Z užitečných datasetů byly nalezeny Indoor Training Set (ITS) [86] a MIT Indoor Scenes (MIT) [87], které v části obrazů obsahovaly květiny. Oba datasety nebyly navrženy pro detekci objektů, nedisponovaly proto anotacemi květin ani květináčů. S využitím nejkomplexnějšího modelu metody YOLOv5 byla separována naprostá většina obrazů s květinami nebo vázami spolu s částí falešně pozitivních obrazů. Dataset ITS se skládal z obrazů o rozlišení (620 × 460) px o přijatelné kvalitě. Ze 1400 obrazů bylo

vytříděno 311. Dataset MIT obsahoval přes 15000 obrazů, vytříděno bylo 2800. Dostatečné obrazové kvality pro potřeby detekce dosahovala asi šestina obrazů. Nejčastěji měly rozlišení pouze (256 × 256) px, protože dataset sloužil primárně pro klasifikaci druhu místností.

Vlastní datová sada

Z důvodu absence většího množství kvalitních dat bylo přistoupeno také ke sběru vlastních obrazů. Obrazy byly pořízeny 13Mpx kamerou mobilního telefonu a 16Mpx bezzrcadlovkou, lze z nich získat i kvalitní výřezy. Testovací data byla získána webkamerou z robotu. Data byla sbírána především v domácnostech a v areálu VUT Brno ve vnitřních prostorách. Bylo dbáno na různorodé světelné podmínky a zachycení jak světelně vyvážených scén, tak kontrastních např. proti oknům. Většina fotografií byla pořízena z úrovně několika desítek centimetrů nad zemí, což odpovídalo poloze kamery na robotu. Květiny byly umístěny ve všech běžných polohách, např. na zemi, na nábytku, na parapetu. Foceny byly z různých vzdáleností v závislosti na jejich velikosti – přibližně od 20 cm po 10 m. Ukázková trénovací data společně s anotacemi tříd květina (*flw*) a květináč (*pot*) jsou na obrázku 4.4. Ukázka trénovacích a validačních dat se shluky květin je v elektronické příloze.

Obrázek 4.4: Vlastní data s anotacemi

Anotace a příprava datasetu

V obrazech byly objekty daných tříd anotovány hraničními rámečky podle těchto kritérií.

- **Květina** – veškerá část samotné květiny a květináče. Patří sem i umělé květiny, jelikož z fotografií téměř nejdou rozeznat od živých a jejich neoznačení by komplikovalo trénování detektoru. Jednoznačné kvítí ve váze není považováno za květinu.
- **Květináč** – pouze květináč, ve kterém se nachází květina a v obrazu je alespoň částečně viditelná. Patří k němu i miska pod květináč.

Jedním z argumentů trénovacího skriptu je cesta k yaml souboru, kde jsou definovány třídy objektů a struktura datasetu. Takový soubor může vypadat takto:

```
path: datasets/flowers_640px    # relative to 'train.py'
train: images/train             # train images (relative to 'path')
val: images/valid               # val images (relative to 'path')
test: images/test               # test images (relative to 'path')
nc: 2                           # number of classes
names: ['flw', 'pot']           # class names
```

Trénovací skript automaticky hledá anotační soubory tak, že poslední slovo `images` v cestách nahradí slovem `labels`. Struktura datasetu a anotací pro YOLOv5 pak vypadá například takto:

```
flowers_640px
├── images
│   ├── train
│   │   ├── IMG_20211112_132616.jpg
│   │   └── ...
│   ├── valid
│   │   └── ...
│   └── test
│       └── ...
└── labels
    ├── train
    │   ├── IMG_20211112_132616.txt
    │   └── ...
    ├── valid
    │   └── ...
    └── test
        └── ...
```

Pro každý obraz existuje samostatný textový soubor s anotacemi. V případě, že v obraze není žádný objekt, je soubor prázdný. Každý řádek přísluší jednomu objektu a obsahuje 5 parametrů: číslo třídy, x a y středu rámečku, šířku a výšku rámečku. Hodnoty jsou normovány velikostí obrazu, nabývají tedy hodnot $< 0; 1 >$. Ukázka souboru pro obraz s jednou květinou a jedním květináčem je níže.

```
0   0.521226    0.241667    0.155660    0.225000
1   0.525948    0.317803    0.044132    0.071057
```

K anotacím byl využit online nástroj MakeSense [88]. Podporuje export anotací přímo v textových souborech ve formátu, který vyžaduje YOLOv5.

Hotový dataset

Konečný dataset se skládal převážně z vlastních dat. Z veřejného datasetu ITS bylo použito všech 200 obrazů s květinami. Dataset MIT neměl z důvodu horší kvality obrazů pro trénování přínos, po anotaci 114 obrazů od něho bylo upuštěno. Konečné počty anotovaných obrazů a jednotlivých instancí tříd jsou zaznamenány v tabulce 4.2.

Tabulka 4.2: Počet obrazů a instancí tříd získaného datasetu

	vlastní			veřejné	celkem
	trénovací	validační	testovací	trénovací	
fotografie	692	125	36	314	1167
květiny	1804	306	99	626	2835
květináče	1598	295	91	461	2445

4.3.2 Příprava modelu

Pro trénink detekčního modelu a jeho implementaci na robotu ve frameworku ROS bylo využíváno oficiálního zdrojového kódu YOLOv5 vydání 6.0 z repozitáře [34]. Ten nabízel modely různých velikostí předtrénované na rozsáhlém datasetu COCO. Experimentováno bylo s trojicí menších modelů YOLOv5 a s modelem YOLOv3-tiny, který lze trénovat pomocí [89] a je s kódem YOLOv5 kompatibilní. Podstatné parametry jednotlivých modelů jsou uvedeny v tabulce 4.3. GFLOPs udává v miliardách počet operací v pohyblivé řádové čárce při dopředném průchodu sítí. Všechny modely byly trénovány v PyTorch a šlo je převést i do jiných frameworků.

Tabulka 4.3: Relevantní modely YOLO

model	YOLOv5n	YOLOv5s	YOLOv5m	YOLOv3-tiny
počet vrstev	270	270	369	48
milionů parametrů	1,8	7,0	20,1	8,7
velikost [MB]	4	14	41	17
GFLOPs	4,2	15,9	48,0	12,9

Pro trénování bylo využito techniky *transfer learning* – předtrénované modely byly dotrénovány na vlastním datasetu s využitím slabšího trénování většiny vrstev nebo jejich zmrazení (nastavení nulového kroku učení). Při pokusech s trénováním bylo experimentováno s nastavením hyperparametrů, které se definovaly v .yaml souboru a ovlivňovaly např. augmentaci dat. Průběh ladění parametrů blíže popisuje podkapitola 4.6.1. Zdrojový kód disponoval i hledáním nejvhodnějších parametrů pomocí genetického algoritmu. To ale vyžadovalo až stovky samostatných tréninků a v rámci bezplatného účtu Google Colab s omezenou délkou běhu to nebylo jednoduše realizovatelné.

4.3.3 Zpracování detekcí

V rámci originálního zdrojového kódu metody YOLOv5 je zajištěna úprava velikosti obrazu na vstupní velikost modelu, dopředný průchod modelem, běžná filtrace detekcí algoritmem Non-Maximum Suppression na základě prahu skóre jistoty a přepočet detekcí na původní obraz. Výstupem této posloupnosti je pole (ve formátu *numpy array*) s počtem řádků rovným počtu detekovaných objektů. Sloupce jsou tvořeny celočíselnými souřadnicemi levého horního a pravého spodního rohu ohraničujícího rámečku, skórem jistoty (od 0 do 1) a indexem třídy objektu. Detekce byly dále rozděleny na instance květin a květináčů. Podle detekovaných tříd a jejich skóre jistoty byly nastaveny příznaky řídící další zpracování.

Párování květin a květináčů

V případě, že měly být určovány vzdálenosti květin, bylo nutno detekované rámečky významově roztřídit na květiny a jejich květináče. Níže uvedený pseudoalgoritmus metody zachycuje nejpodstatnější kroky. Postupně se prochází všechny rámečky květin od největšího skóre jistoty. Pro všechny rámečky nespárovaných květináčů se s využitím vektorizace vypočte metrika M charakterizující míru jeho příslušnosti k dané květině. Ta byla stanovena jako průnik rámečků květiny A a i-tého květináče B_i, normovaný plochou rámečku květináče B_i (viz vztah 4.1). Pokud květináč s nejvyšší metrikou přesahuje definovaný práh (stanoveno 0,5), je přiřazen dané květině. Proces končí po průchodu všech květin nebo po přidělení všech květináčů.

$$M_i = \frac{A \cap B_i}{B_i} \tag{4.1}$$

```
def vytvor_pary():
    serad sestupne kvetiny dle skore jistoty
    vypocti plochu B vsech ramecku kvetinacu
    for kvetina in serazene_detekovane_kvetiny:
        vypocti metriku M kvetiny se vsemi kvetinaci
        vyber kvetinac s nejvetsi hodnotou metriky
        if metrika > prahova hodnota
            vytvor par kvetina-kvetinac
            odeber kvetinac ze seznamu nesparovanych kvetinacu
```

Výsledek detekcí a párování ilustruje obrázek 4.5 pořízený z kamery robotu. Páry jsou zde označeny indexy v anotacích za zkratkou třídy, následuje skóre jistoty. V případě květináčů je určena metrická vzdálenost, čímž se zabývá podkapitola 4.4.

Algoritmus byl navržen pro předpokládané scénáře zalévání, kdy nejsou květiny příliš blízko sebe. V případě shluknutých květin, čemuž se blíží obrázek, existují situace, kdy květináče algoritmus rozdělí nesprávně. Pokud by se například malý květináč, na obrázku označen indexem 2, nacházel v prostoru výše, byl by přidělen květině s indexem 1. Metrika by se dala upravit jako vážený součet současné metriky M a metriky, kde by ve jmenovateli figurovalo sjednocení ($A \cup B_i$). To by pracovalo více s předpokladem, že velká květina bude umístěna ve velkém květináči.

Obrázek 4.5: Spárované květiny a květináče

4.4 Určení polohy květin

Tato podkapitola se zabývá tvorbou nástrojů pro určení vzdálenosti a polohy květin vůči robotu. Na krátkou vzdálenost byly zvoleny identifikační štítky ArUco, na velkou vzdálenost hloubkové mapy, které bylo dále potřeba sloučit s měřením z LiDARu. Oba způsoby vyžadovaly kalibraci kamery.

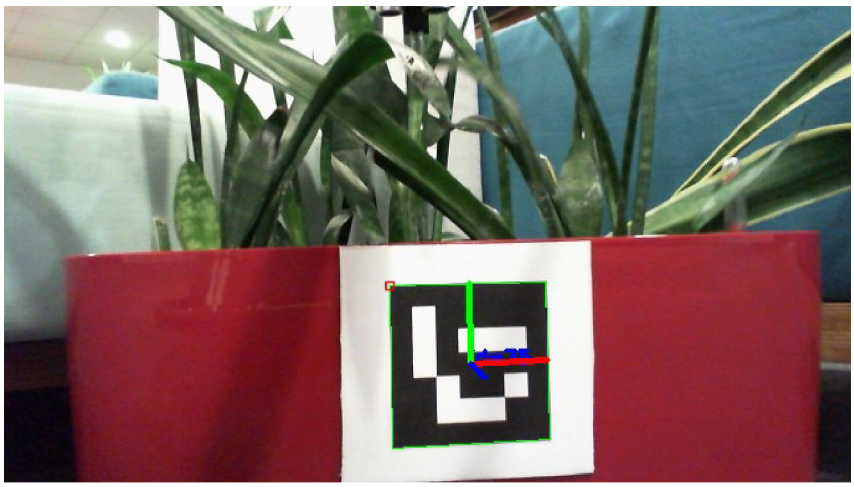
4.4.1 Identifikační štítky

Používaná kamera byla kalibrována podle [49] pomocí vzoru *ChArUco board*, který kombinuje šachovnici a ArUco štítky, vytištěném na papíře A4. Pro každé ze 3 rozlišení bylo pořízeno okolo 50 kalibračních snímků. Získány tak byly matice kamery a koeficienty deformace.

Pro práci s ArUco štítky byla využita knihovna ArUco z rozšířené verze OpenCV [49]. Štítky pocházely ze slovníku *DICT_ARUCO_ORIGINAL*, jehož štítky obsahují 5×5 bitů. Pro testování robotu byly vytištěny štítky o rozměru (76×76) mm, aby byly dobře rozlišitelné i ze vzdálenosti 1,5 m při menším rozlišení obrazu.

Souřadnicový systém štítku je v knihovně definován stejně jako na obrázku 4.6, kde osy x,y,z jsou označeny popořadě červeně, zeleně a modře. Orientace souř. systému kamery byla ukázána v podkapitole 2.4.1. K detekci štítků a čtení identifikátoru nabízí knihovna funkci *detectMarkers*, k určení polohy a orientace štítku funkci *estimatePoseSingleMarkers*. Polohu štítku vrací funkce jako translační vektor středu štítku v souř. systému kamery a orientaci štítku v podobě rotačního (Rodriguezova) vektoru. Je to vektor, kolem kterého je potřeba otočit souř. systém štítku, aby se zarovnal se souř. systémem kamery. Velikost vektoru udává úhel rotace v radiánech. Výsledný rotační vektor byl převeden na rotační matici pomocí funkce *Rodrigues*. Souř. systém štítku byl otočen o π rad kolem své osy x proto, aby byly souř. systémy totožné v případě, že je kamera orientovaná ke štítku kolmo. Výsledná translace a rotace hledaného štítku byly převedeny na kvaternion pomocí

knihovny transformací *tf* kvůli komunikaci v ROSu.



Obrázek 4.6: ArUco štítek na květináči s estimovaným osovým křížem

4.4.2 Syntéza relativní hloubkové mapy s LiDARem

Metoda pro monokulární predikci hloubky MiDaS byla implementována pomocí originálního zdrojového kódu pro Python z repozitáře [90]. Využit byl kód ze skriptu *run.py*. V něm může být velikost vstupního obrazu automaticky upravena tak, aby delší (volba *upper_bound*) nebo kratší (volba *minimal*) strana obrazu měřila 384 px u modelu DPT-hybrid a 256 px u modelu MiDaS2.1small. Druhý rozměr je roven nejbližšímu násobku 32. Jak bylo zmíněno v kapitole 2.3.2, výstupem modelu je relativní inverzní hloubka d, z které lze určit metrickou hloubku D pomocí lineárního vztahu 4.2. Testováním na smcích stejné scény, jen nepatrně odlišného úhlu závěru, bylo ověřeno, že parametry α a β byly často nekonzistentní, α mohla nabývat až 4× větších hodnot. Parametry bylo proto nutno stanovit pro každý obraz zvlášť dodatečným měřením.

$$\frac{1}{D} = \alpha d + \beta \tag{4.2}$$

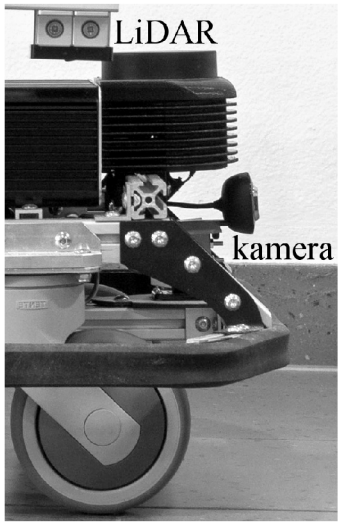
V této podkapitole je rozebrán způsob nalezení α a β pomocí paprsků LiDARu, který se nachází pod kamerou, případně nad ní. Hodnotě vzdálenosti z každého paprsku LiDARu nacházejícího se v zorném úhlu kamery byla přiřazena hodnota z relativní hloubkové mapy. Body tvořené dvojicí těchto hodnot byly dle vztahu 4.2 proloženy přímkou metodou nejmenších čtverců, čímž byly získány α a β a relativní hloubka byla převedena na metrickou. Každá fáze algoritmu je detailně popsána níže.

Tabulka 4.4: Parametry robotů pro syntézu

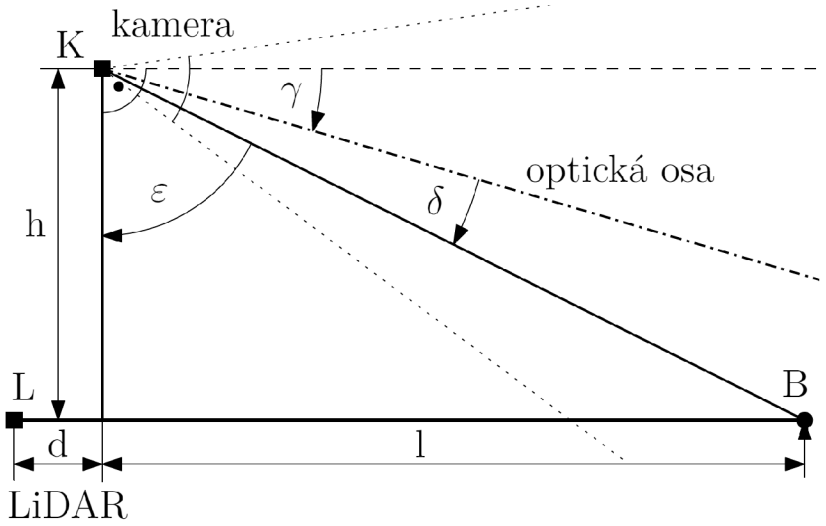
	h [mm]	d [mm]	offset LiDARu θ_{x0} [°]	náklon kamery γ [°]	výška kamery nad zemí [mm]
Breach	−80	55	90	0	160
Leela	83	30	270	0	187

Geometrie a parametry robotů

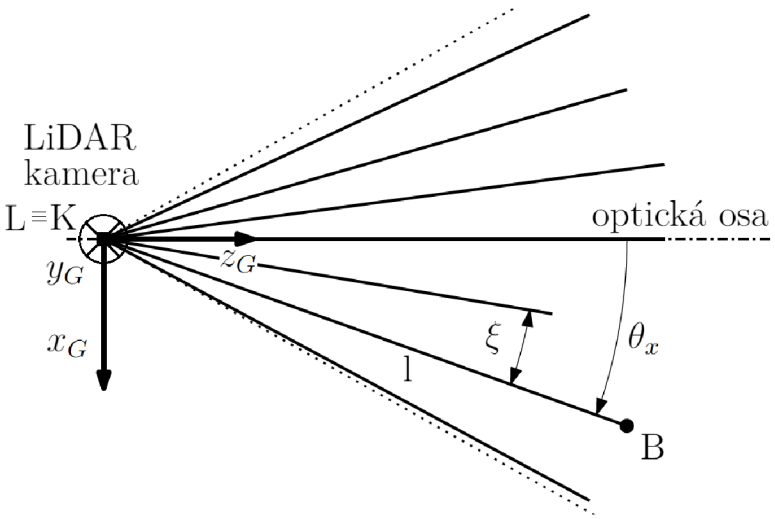
Vzájemné uspořádání kamery (K) a LiDARu (L) na robotu Breach je na obrázku 4.7, zarovnány jsou v podélné ose robotu. Obrázek 4.9 ilustruje paprsky LiDARu v pohledu shora, obrázek 4.8 zachycuje jeden paprsek v pohledu z boku. Na Leele se nacházela kamera nad LiDARem, rovnice byly odvozovány pro její případ. Jsou ale platné i pro konfiguraci na Breachi, vzdálenost mezi kamerou a LiDARem h je u něj záporná. Střed LiDARu je umístěn o vzdálenost d za ohniskem kamery, avšak předpokládá se, že je tato vzdálenost pro odvozené vztahy zanedbatelná. Optická osa kamery by mohla být vychýlená od horizontálního směru o úhel γ. LiDAR měří s úhlovým rozlišením ξ rovným 1°. LiDAR rotuje při pohledu shora v kladném směru, jeho výstupem je kruhový sken vzdáleností. Offset LiDARu θ_{x0} udává, jaký úhel skenu odpovídá přímému směru. Hodnoty zmiňovaných parametrů se liší od použitého robotu a jsou uvedeny v tabulce 4.4. Potřebná je i kalibrace kamery: ohniskové vzdálenosti f_x, f_y a souřadnice hlavního bodu c_x, c_y.



Obrázek 4.7: Breach



Obrázek 4.8: Schéma jednoho paprsku LiDARu v pohledu z boku

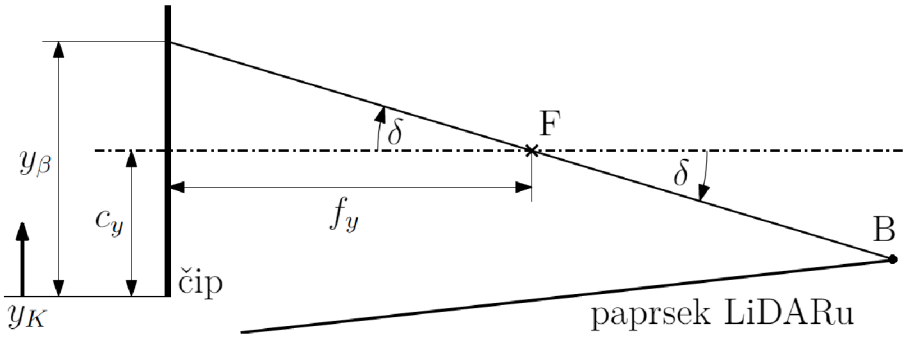


Obrázek 4.9: Schéma paprsků LiDARu v pohledu shora

Párování paprsků LiDARu s relativní hloubkovou mapou

Cílem této části algoritmu bylo přiřadit koncové body paprsků LiDARu B bodům $[x_k, y_k]$ na čipu kamery. X-ová složka byla určena pomocí schématu 4.9. Sken LiDARu byl nejprve zarovnán na základě offsetu θ_{x0} a natočení mechanismu kamery. Ze skenu byly vybrány paprsky nacházející se v zorném poli kamery (tečkovaně) a bylo invertováno jejich pořadí. Paprskům byl přiřazen orientovaný úhel θ_x s počátkem v optické ose. Souřadnice v obrazu x_k pro daný paprsek byla určena podle rovnice 4.3, což opodstatňuje analogický případ na schématu 4.10.

$$x_k = f_x \operatorname{tg} \theta_x + c_x \tag{4.3}$$



Obrázek 4.10: Schéma

Výpočet souřadnice y_k vychází ze schématu 4.8. Úhel ε se určil dle rovnice 4.4, kde $l + d$ je vzdálenost ze skenu. Z něj se vypočetl úhel δ dle rovnice 4.5. Na základě schématu 4.10 se dopočetla y-ová souřadnice y_k dle rovnice 4.6. V případě, že byl bod B mimo horizontální zorné pole kamery, nebyl dále uvažován.

$$\varepsilon = \operatorname{arctg} \frac{l}{h} \tag{4.4}$$

$$\delta = \frac{\pi}{2} - \gamma - \varepsilon \tag{4.5}$$

$$y_k = f_y \operatorname{tg} \delta + c_y \tag{4.6}$$

Pro každý validní bod $[x_k, y_k]$ na čipu kamery byla z hloubkové mapy odečtena relativní hodnota vzdálenosti d. Metrická vzdálenost $D = |KB|$ byla určena z rovnice 4.7.

$$D = \sqrt{l^2 + h^2} \tag{4.7}$$

Robustní lineární regrese a převod na metrickou hloubku

Uspořádané dvojice $[1/D, d]$, kterých vzniklo na robotech Breach a Leela až 60, byly vstupem lineární regrese rovnice 4.2. Jak bude ukázáno v kapitole 4.6.2, na hranách objektů nebo listnatých částech květin docházelo někdy k tvorbě chybných dvojic. Tyto odlehlé hodnoty zapříčiňovaly chybné proložení metodou obyčejných nejmenších čtverců. Byla proto zvolena metoda robustní lineární regrese, konkrétně Huber regresor z knihovny Scikit Learn. Ten odlehlé hodnoty nevynechá úplně, ale zohledňuje je výrazně nižší vahou, což bylo v některých případech proložení užitečné.

Jak bylo zjištěno v práci [1], natáčecí mechanismus kamery mohl dosahovat odchylek až ±3°, proto byla zavedena automatická korekce této odchylky. K úhlům paprsků θ_x byly

přičteny hodnoty z intervalu očekávaných odchylek s krokem např. 0,5°. Stanovena byla taková odchylka, pro kterou dosahovala regrese nejvyšší vhodnosti.

Po nalezení α a β mohla být relativní inverzní hloubková mapa přepočtena na metrickou. Nešlo rovnou aplikovat vztah 4.8, jelikož zřídka nastávalo dělení číslem blízkým nule a vznik nesmyslných hodnot hloubky některých pixelů obrazu. Proto byla první relativní hloubková mapa saturována podle vztahů 4.9 a 4.10 tak, aby byla výsledná mapa saturována na minimální a maximální metrickou hloubku D_{min} a D_{max}. Tyto meze byly zvoleny s ohledem na měřicí rozsah LiDARu jako 0,3 m a 8,0 m.

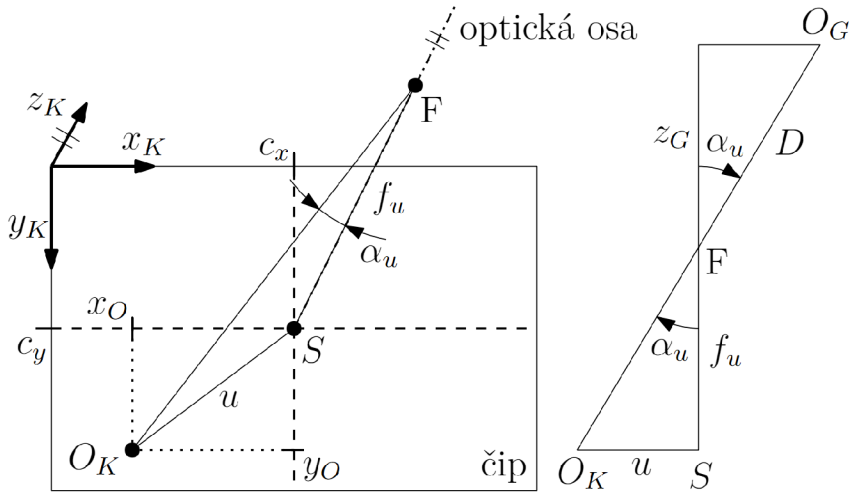
$$D = \frac{1}{\alpha d + \beta} \tag{4.8}$$

$$d_{max} = \frac{1 - \beta D_{max}}{\alpha D_{max}} \tag{4.9}$$

$$d_{min} = \frac{1 - \beta D_{min}}{\alpha D_{min}} \tag{4.10}$$

4.4.3 Určení polohy z metrické hloubkové mapy

Na základě rámečků spárovaných květináčů z detekce a metrické hloubkové mapy mohla být určena poloha těchto květináčů vůči kameře. Situaci znázorňuje schéma 4.11, kde osa z_K směřuje do papíru. Střed rámečku květináče $O_K = [x_O, y_O]$ v obraze byl přes ohnisko F promítnut na bod v prostoru $O_G = [x_G, y_G, z_G]$. Úhel α_u podobných trojúhelníků se určil podle vztahů 4.11 z úhlopříčky u a ohniskové vzdálenosti f. Vzdálenost D byla zjištěna z metrické hloubkové mapy jako medián čtvercového okolí o straně 3 px. Souřadnice bodu O_G byly vypočteny pomocí vztahů 4.12.



Obrázek 4.11: Schéma zobrazení bodu O na čipu kamery a v prostoru

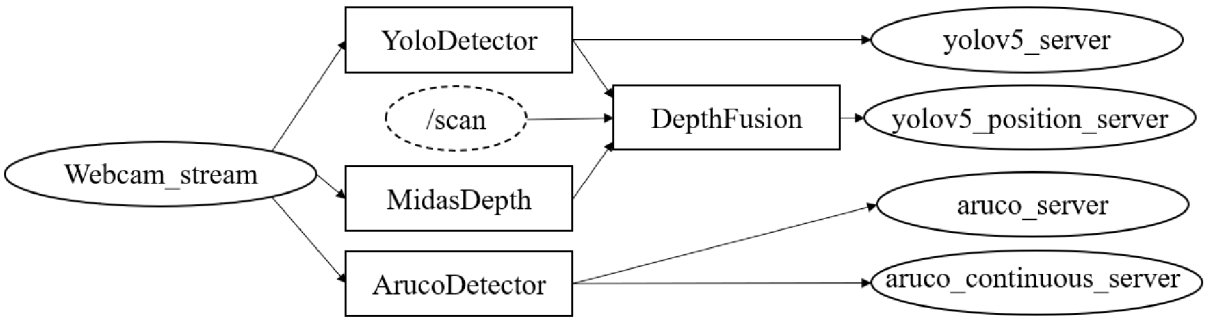
$$\begin{aligned} f_u &= \sqrt{f_x^2 + f_y^2} \\ u &= \sqrt{(y_O - c_y)^2 + (x_O - c_x)^2} \\ \alpha_u &= \operatorname{arctg} \frac{u}{f_u} \end{aligned} \tag{4.11}$$

$$\begin{aligned} z_G &= D \cos \alpha_u \\ x_G &= z \frac{x_O - c_x}{f_x} \\ y_G &= z \frac{y_O - c_y}{f_y} \end{aligned} \tag{4.12}$$

4.5 Implementace v ROSu

Vytvořené funkční celky popsané v předchozích podkapitolách byly implementovány do dvou balíčků (*packages*) v ROSu - *aruco* a *yolov5*. Balíček *aruco* obsahoval skript pro pořízení obrazu z webkamery a skripty týkající se identifikančích štítků. Balíček *yolov5* obsahoval skripty pro detekci a určení vzdálenosti květin. Do projektu měly být funkcionality zasazeny pomocí komunikačního mechanismu akce (*action*), jelikož jsou obecně vhodné pro asynchronní procesy jako např. zpracování obrazu. Příslušné akční servery vytvořené v této práci byly spouštěny akčními klienty, kteří byli součástí řídicí struktury v práci [2]. Pro potřeby testování byli vytvořeni jednodušší klienti i v této práci.

Propojení funkčních celků a akčních serverů ilustruje diagram 4.12. Akční servery byly součástí uzlů (*nodes*), v diagramu ovály vpravo. Uzel *webcam_stream* obsluhoval kameru. Funkční celky byly implementovány formou objektově orientovaného programování, jejich třídy jsou v diagramu obdélníky.



Obrázek 4.12: Aplikace tříd v uzlech

4.5.1 Vytvořené třídy

YoloDetector

Třída *YoloDetector* byla definována ve skriptu *yolov5/scripts/classes/yolo_class.py*. Z ROS parameter serveru očekávala cestu k modelu natrénovanému v PyTorch *yolo_model_path* vzhledem k adresáři balíčku, rozlišení delší strany modelu *img_size* a práh skóre jistoty pro úspěšnou detekci květiny *min_conf*. Kód pro inicializaci modelu, dopředný běh a základní zpracování detekcí vychází z originálního skriptu *detect.py* z [34], který byl v této práci zjednodušen pro konkrétní aplikaci. Dopředný běh zajišťuje metoda *perform_detection*, jejíž vstupem je obraz z kamery. Metody *process_detections* a *form_pairs* sloužily k roztřídění detekcí a vytvoření párů květina-květináč, jak bylo popsáno v kapitole 4.3.3. Metody *anotate_pairs* a *anotate_pairs_distances* vycházely z originálního kódu pro vykreslení detekcí z metody *anotate_raw*, navíc pouze vypsaly roztříděné páry, případně i

předpovězenou vzdálenost.

MidasDepth

Třída *MidasDepth*, definovaná ve skriptu *yolov5/scripts/classes/midas_class.py*, prakticky všechen kód byl převzat z originálního skriptu *run.py* z [90]. Cesta k modelu (DPT-hybrid nebo MiDaS2.1small) *midas_model_path* byla načtena z ROS parameter serveru. Metoda *estimate_depth_map* pro dopředný běh přijímala na vstupu obraz z kamery, výstupní inverzní relativní hloubková mapy byla zapsána do atributu *depth_map_rel*.

DepthFusion

Ve skriptu *yolov5/scripts/classes/fusion_class.py* byla definována třída pro syntézu relativní hloubkové mapy s měřeními z LiDARu a pro určení vzdálenosti a polohy květináčů na základě detekovaných rámečků a vzniklé metrické hloubkové mapy. Z ROS parameter serveru byla přečtena cesta *calib_path* k datům z kalibrace kamery. Dále byly definovány parametry robotů z tabulky 4.4 společně s mezemi pro saturaci hloubkové mapy. Metoda *fuse_depth* provedla syntézu tak, jak bylo vysvětleno v podkapitole 4.4.2. Metody *get_distance* a *get_position* určily vzdálenosti a polohy všech detekovaných květináčů podle principů popsaných v podkapitole 4.4.3.

ArucoDetector

Skript *aruco/scripts/aruco/aruco_class.py* obsahoval třídu pro práci s ArUco štítky. Z ROS parameter serveru byly načteny cesta ke kalibračnímu souboru *calib_path*, slovník *aruco_dict* a velikost markeru v metrech *marker_size*. Detekci štítků a určení jejich polohy a orientace zajišťovala metoda *perform_detection* a zpracování výsledků v případě nalezení hledaného štítku prováděla metoda *process_detections*.

4.5.2 Vytvořené uzly

Vedle uzlů, které zajišťují běh akčních serverů a jsou popsány níže, byly vytvořeny jednoduché pomocné uzly pro testování a vykreslování.

webcam_stream

Tento uzel byl definován ve skriptu *aruco/scripts/webcam_pub.py*. Z ROS parameter serveru načetl rozlišení kamery. Vytvořil instanci objektu *cv2.VideoCapture*, který kontinuálně načítal snímky z kamery. Snímky publikoval na topic *video_frames* zprávu (*message*) typu *Image*, z kterého odebíraly snímky ostatní uzly, a to frekvencí 10 Hz.

yolov5_server

Tento uzel ze skriptu *yolov5/scripts/yolov5_server.py* obsahoval akční server pro detekci květin. V inicializační fázi serveru byl načetl objekt třídy YoloDetector do GPU. Ve vlastní části serveru běžela smyčka s omezením frekvence na 5 Hz, ve které byly prováděny detekce. Pokud byla detekována květina se skórem jistoty přesahujícím definovaný práh, server vrátil klientovi exitcode značící úspěšnou detekci. Tento akční server byl klientem spuštěn ve chvíli, když se robot blížil místu s předpokládanou polohou květiny.

yolov5_position_server

Skript *yolov5/scripts/yolov5_position_server.py* definoval uzel s akčním serverem, který určoval vzdálenost detekovaných květináčů spárovaných květin. Uzel vychází z uzlu yo-

lov5_server. V hlavní smyčce odebíral uzel kromě obrazů také skeny z LiDARu ve formě zprávy typu LaserScan, konkrétně z topicu /scan, který již byl součástí robotů Breach i Leela. Prováděna byla detekce květin objektem třídy YoloDetector, v případě detekce květiny i květináče byl proveden pokus o jejich spárování. Po úspěšném spárování byla vykonána procedura pro určení vzdálenosti květináčů. Skládala se z predikce relativní hloubkové mapy objektem třídy MidasDepth, syntézou a určením polohy květináčů objektem třídy DepthFusion. Jelikož byl při testování robotu uvažován případ osamocených květin, do výsledku akce byla zapsána poloha pouze květiny s nejvyšším skóre jistoty. Poloha byla vracena v rámci zprávy typu *Pose*, ve které byly vyplněny pole pouze pro polohu.

Modely YOLO a MiDaS2.1small bylo možné na Jetson Nano spustit současně tak, aby jeden z těchto modelů běžel na GPU (podrobněji v kapitole 4.6). V případě použití modelu DPT-hybrid nestačila paměť RAM, sdílená mezi procesorem a GPU, pro současné načtení modelu spolu s modelem YOLO, kdy alespoň jeden z modelů měl běžet na GPU. Nepomohly ani pokusy o navýšení paměti zram. Nicméně nebylo nutné, aby oba modely běžely současně. Bylo počítáno s tím, že po predikci detekcí bude uvolněna paměť vymezená modelem YOLO a alokována pro model DPT-hybrid. PyTorch to ale v rámci jednoho skriptu neumožňoval, skript by musel být pro uvolnění paměti nejprve ukončen. Možné řešení se nabízelo v rozdělení obou modelů do samostatných skriptů, *service* serverů, které by se z tohoto akčního serveru postupně programově spouštěly a vypínaly s využitím Roslaunch API. Signály pro vypnutí ale nebylo možné použít ze skriptu akčního serveru, pouze z hlavního vlákna, což situaci komplikovalo. Nakonec bylo u modelu DPT-hybrid přistoupeno k inferenci při určování vzdálenosti pouze na CPU a predikce hloubkové mapy tak trvala okolo 10 nebo 30 sekund v závislosti na vstupním rozlišení modelu.

aruco_server

Ve skriptu *aruco/scripts/aruco_server.py* byl obsažen uzel pro identifikaci ArUco štítků. Uzel obsahoval hlavní smyčku, omezenou na 10 Hz, která v odebíraných obrazech detekovala štítky pomocí objektu třídy ArucoDetector. V případě detekce hledaného identifikátoru byl vrácen klientovi úspěšný exitcode a poloha a orientace detekovaného štítku ve zprávě typu *Pose*.

aruco_continuous_server

Skript *aruco/scripts/aruco_continuous_server.py* definoval stejný uzel jako *aruco_server* s tou výjimkou, že detekce štítků a estimace jejich polohy probíhala kontinuálně do té doby, než ji nevypl klient.

4.6 Výsledky

V této kapitole jsou ověřovány implementované metody převážně na datech pořízených z robotu Breach při testování. Posuzovány jsou i výpočetní nároky jednotlivých řešení.

4.6.1 Trénování modelu pro detekci květin

V této části jsou prezentovány výsledky experimentů s nastavením hyperparametrů při trénování detektoru. Za výchozí stav byly považovány doporučené hyperparametry ze zdrojového kódu YOLOv5 [34], zvolena byla verze s nejmenší mírou tréninku (soubor *hyp.scratch-low.yaml*). Kvalita detektoru byla pro každý případ vyhodnocena na validačních datech, dodatečně i na testovacích. Posuzována byla podle dosažené přesnosti detekce

mAP0.50 a *mAP*. Pod označením *mAP* je chápána metrika mAP0.5:0.05:0.95 vysvětlená v podkapitole 2.2.1. V potaz byl brán také *recall*, aby bylo zřejmé, zda model příliš nezpřesňuje lokalizaci rámečků s detekcemi na úkor rozpoznávání různých typů květin a květináčů. Při vyhodnocení těchto metrik byl uvažován práh IoU algoritmu NMS rovný 0,45; práh skóre jistoty byl zvolen 0,25.

Velikost dávky a počtu epoch

Na modelu YOLOv5s s rozlišením 416 px byla hledána vhodná velikost dávky (*batch size*) a počet epoch trénování. Nejčastěji přidělená GPU v Google Colab disponovala 11 GB paměti, maximální dávka vycházela na 128. Výsledky zobrazuje tabulka 4.5, první 3 řádky přísluší validačním datům, zbylé 3 testovacím datům. Z validačních dat je patrné, že nejlepších hodnot recall při vysoké mAP0.50 dosahuje batch 128 při trénování na 100 epochách. Proto byl zvolen pro další experimenty.

Tabulka 4.5: Hodnoty metrik při různé dávce a počtu epoch

	128 batch 200 epoch	128 batch 100 epoch	128 batch 50 epoch	64 batch 100 epoch	32 batch 100 epoch	32 batch 50 epoch
*mAP*0.50	0.78	**0.80**	0.79	**0.80**	0.79	**0.80**
mAP	0.57	0.57	0.53	**0.59**	0.57	**0.59**
recall	0.65	**0.70**	0.67	0.68	0.66	0.69
*mAP*0.50	0.88	**0.92**	**0.92**	0.88	0.90	0.90
mAP	0.67	0.66	0.62	0.67	**0.68**	**0.68**
recall	0.81	**0.85**	0.84	0.84	0.82	0.83

Váhy chybové funkce

Jak bylo uvedeno v kapitole 2.2.3, chybová funkce se skládala ze tří částí. V souboru s hyperparametry jsou definovány váhy ovlivňující chybu lokalizace (box), klasifikace (cls) a objektovosti (obj). Tabulka 4.6 zobrazuje v druhém sloupci výchozí hodnoty, v dalších byly zvyšovány nebo snižovány. Z výsledků na validačních datech v tabulce je patrné, že výchozí váhy, naladěné na dataset COCO, jsou vhodné i pro dataset s květinami, což se potvrdilo i na testovacích datech.

Tabulka 4.6: Hodnoty metrik při různých vahách chybové funkce

	0.05 box; 0.5 cls; 1.0 obj	0.025 box; 0.5 cls; 2.0 obj	0.0125 box; 0.5 cls; 4.0 obj	0.1 box; 0.5 cls; 0.5 obj
*mAP*0.50	**0.80**	**0.80**	**0.80**	**0.80**
mAP	**0.57**	0.56	0.55	**0.57**
recall	**0.70**	0.69	0.66	0.69
*mAP*0.50	**0.92**	**0.92**	0.88	0.91
mAP	**0.66**	0.65	0.63	0.65
recall	**0.85**	**0.85**	0.78	0.82

Chyba *focal loss*

Rozšíření chybové funkce *focal loss* disponovalo v nastavení parametrem γ, který zapříčiňoval vyšší hodnoty chybové funkce na obtížnějších trénovacích datech. Jelikož obsahuje vytvořený dataset obrazy o výrazně odlišné obtížnosti, očekával se přínos této chyby, ve výchozí hodnotě nastavené na 0. Avšak z tabulky 4.7 je vidět, že se výrazný přínos nepotvrdil, dále tedy uvažován nebyl. Až testovací data ukázala, že je vhodné menší míru této chyby zvážit. Navíc focal loss způsoboval hladší průběh chyby na validačních datech. Nenulové hodnoty β musely být větší než 1, jinak způsobovaly nestabilitu trénování.

Tabulka 4.7: Hodnoty metrik při použití *focal loss*

	$\gamma = 0$	$\gamma = 1.1$	$\gamma = 1.5$	$\gamma = 2.0$	$\gamma = 4.5$
$mAP0.50$	**0.80**	0.79	**0.80**	0.78	0.74
mAP	**0.57**	**0.57**	**0.57**	0.56	0.51
$recall$	**0.70**	0.66	0.69	0.66	0.64
$mAP0.50$	0.92	0.92	**0.93**	0.92	0.85
mAP	0.66	**0.69**	0.68	0.68	0.59
$recall$	0.85	0.85	**0.88**	0.83	0.78

Augmentace dat

V další fázi byla upravována míra augmentace dat, tedy mírné pozměnění charakteru obrazu - nejčastěji barev, jasu, kontrastu a geometrie. Jeho smyslem je minimalizovat přeučení modelu na konkrétních datech. Tabulka 4.8 ukazuje, že vyšší míra augmentace může mírně pomoci, nepřináší však oproti výchozímu nastavení znatelné zlepšení.

Tabulka 4.8: Hodnoty metrik při různé míře augmentace dat

	původní	vyšší	nižší
$mAP0.50$	0.80	**0.81**	0.80
mAP	**0.57**	0.56	**0.57**
$recall$	**0.70**	**0.70**	0.67
$mAP0.50$	**0.92**	0.91	0.91
mAP	**0.66**	0.65	0.65
$recall$	**0.85**	0.83	**0.85**

Zmrazení vrstev a menší krok učení

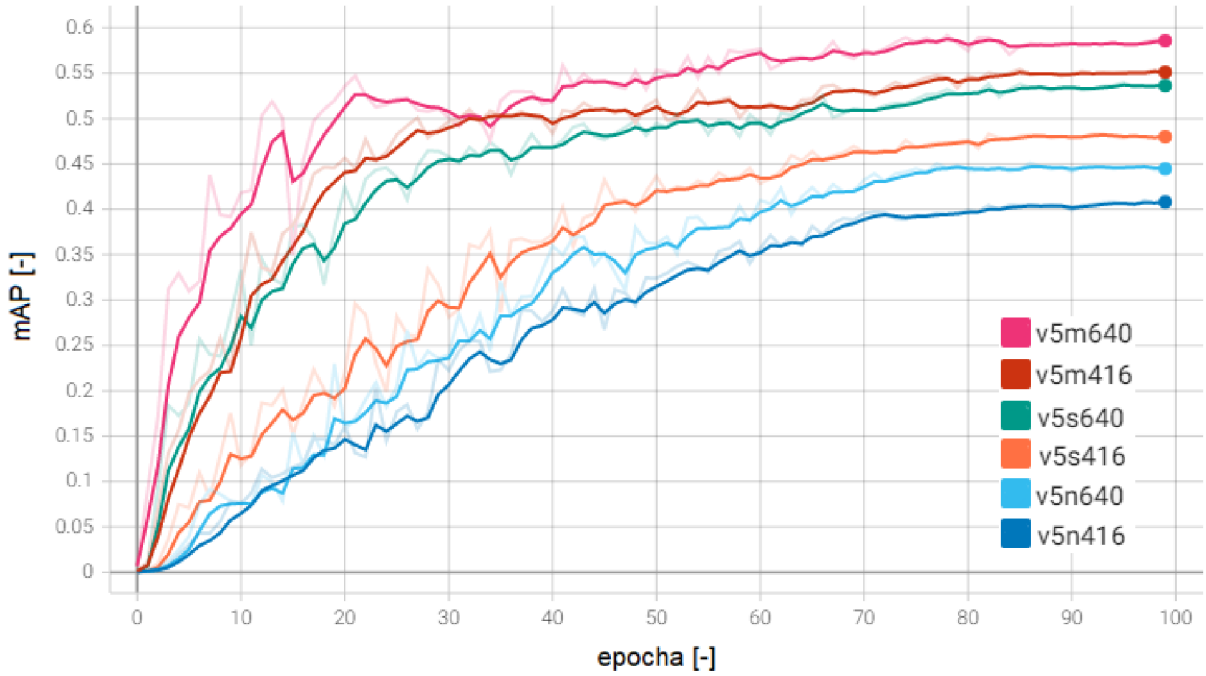
Tato část rozebírá techniky fine-tuningu, tedy snižování kroku učení a zmrazení vybraných vrstev. Výsledky shrnuje tabulka 4.9. Výchozí krok učení je označen *lr1*, *lr0.1* značí 10× menší krok učení. Zmrazení všech vrstev kromě poslední, detekční, udává případ *f24*, zmrazení pouze části *backbone* případ *f10*. Z tabulky lze vidět, že pouhé zmrazení výrazně zhoršuje přesnost detekce. Stejně tak snížení kroku učení má na přesnost negativní vliv. Lepších výsledků dosahuje zmrazení s následným trénování celku, a to především s krokem učení *lr1*. Zde jsou však vyšší hodnoty *mAP* způsobeny hlavně větším celkovým počtem trénovacích epoch a na základě nižšího *recall* lze pozorovat postupně přeučení sítě. Původní

volba kroku učení bez zmrazování dosahovala nejlepších výsledků.

Na základě proběhlých experimentů se ukázalo, že původní nastavení hyperparametrů je vhodný i pro použitou kombinaci modelu a datasetu, bylo proto při tréninku zachováno.

Tabulka 4.9: Hodnoty metrik při různé dávce a počtu epoch

	lr1	f10	f24	lr0.1	f10 + lr0.1	f24 + lr0.1	f10 + lr1	f24 + lr1
*mAP*0.50	**0.80**	0.69	0.63	0.79	0.79	0.78	**0.80**	**0.80**
mAP	0.57	0.44	0.39	0.51	0.55	0.56	**0.58**	**0.59**
recall	**0.70**	0.58	0.51	0.66	0.67	0.67	0.68	0.66
*mAP*0.50	0.92	0.88	0.78	0.92	0.90	**0.93**	0.90	0.90
mAP	0.66	0.62	0.52	0.57	0.64	0.65	0.67	**0.69**
recall	0.85	0.77	0.53	0.83	0.82	**0.86**	0.82	0.83



Obrázek 4.13: Metrika mAP napříč testovanými modely

Modely o různé komplexnosti

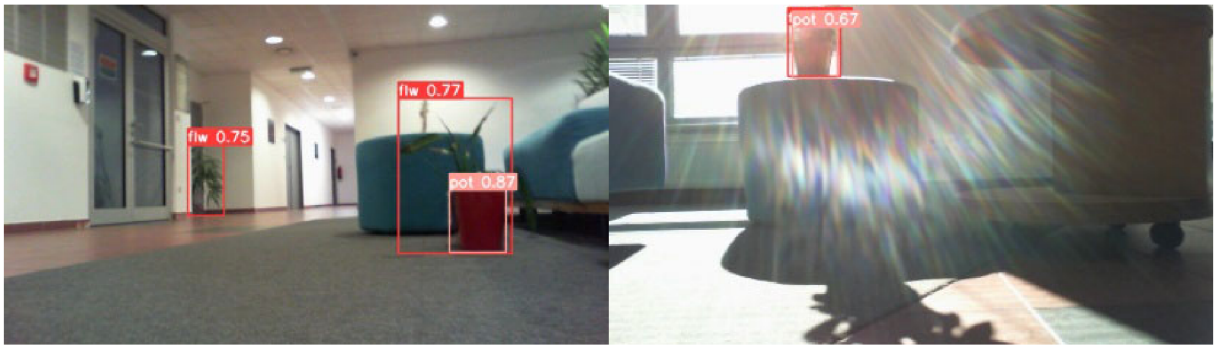
Poslední experiment s detekcí zkoumá další relevantní modely metody YOLOv5 a to i s vyšším rozlišením. Průběh trénování 6 testovaných modelů zobrazuje obrázek 4.13. V průběhu tréninku je mAP vyhodnocována s mírně odlišnými prahy, proto se hodnoty liší od hodnot v tabulce 4.10. Kromě metrik na validačních a následných testovacích datech obsahuje tabulka 4.10 i měření výpočetní náročnosti modelů na Jetson Nano při běhu na GPU. Na něm běh probíhal na testovacích datech s poměrem stran 16:9 a tedy pro rozlišení (416×256) px nebo (640 × 352) px. Z validačních výsledků lze vidět, že výborného poměru přesnosti *mAP0.50* a rychlosti dosahuje právě zvolený model v5s s rozlišením 416

px. Na testovacích datech dosáhl překvapivých **0,92 *mAP0.50***. V metrice *mAP* přísnější na lokalizaci za komplexnějšími modely zaostává, nicméně podstatné je, že udržuje *recall* na vysoké hodnotě.

Ačkoliv samotný běh zvoleného modelu trval 50 ms, dalších cca 10 ms zabralo předzpracování vstupního obrazu a filtrace detekcí algoritmem Non Maximum Supression. Při implementaci v ROSu probíhala detekce květin paralelně s pohybem robotu a mohly tedy vznikat špičková zatížení. Detekce modelu byla proto omezena na zcela dostatečných 5 FPS, s kterými mohla běžet spolehlivě i v případě mírného kolísání trvání běhu hlavně kvůli vytížení paměti.

Tabulka 4.10: Hodnoty metrik při různé dávce a počtu epoch

	v5n 416	v5n 640	v5s 416	v5s 640	v5m 416	v5m 640
*mAP*0.50	0.77	0.79	0.80	0.82	0.80	**0.83**
mAP	0.50	0.53	0.57	0.62	0.63	**0.66**
recall	0.62	0.68	0.70	0.69	0.68	**0.71**
*mAP*0.50	0.88	0.82	**0.92**	0.89	0.90	0.90
mAP	0.58	0.66	0.66	0.70	0.73	**0.75**
recall	0.79	**0.86**	0.85	0.82	0.84	0.84
dopředný průchod Jetson Nano [ms]	45	/	50	100	120	/
sdílená paměť pro CPU a GPU [GB]	2,9	/	2,9	2,9	2,9	/



Obrázek 4.14: Správné detekce za horších světelných podmínek

Ukázky detekcí

Obrázek 4.14 zobrazuje správné detekce při horším osvětlení. Obraz vlevo byl pořízen za jízdy robotu při slabším umělém osvětlení, ale i přes menší rozmazání fungovala detekce správně. Obraz vpravo ukazuje extrémní případ, kdy svítilo slunce přímo do kamery. Jelikož odlesk objektivu květinu nepřekrýval, detekce zafungovala.

Obrázek 4.15 ukazuje případy, kde se kromě korektních detekcí objevují i chybové detekce. Na levém horním obraze došlo ke sloučení dvou dotýkajících se květin. Pravý horní obraz ukazuje taburet, který byl z vybraných úhlů detekován s vysokým skóre jistoty jako květináč. Modul určování vzdálenosti by tento případ mohl narušit, pokud by se

s taburetem překrývala květina. Levý spodní obraz ukazuje nezachycenou květinu, pravý spodní obraz shluknuté květiny, kde květináče jsou rozlišeny (až na jeden nedetekovaný) správně, detekce květin jsou sloučené.

Další výsledky detekcí spolu s párováním květin a květináčů jsou ukázány v následující podkapitole.

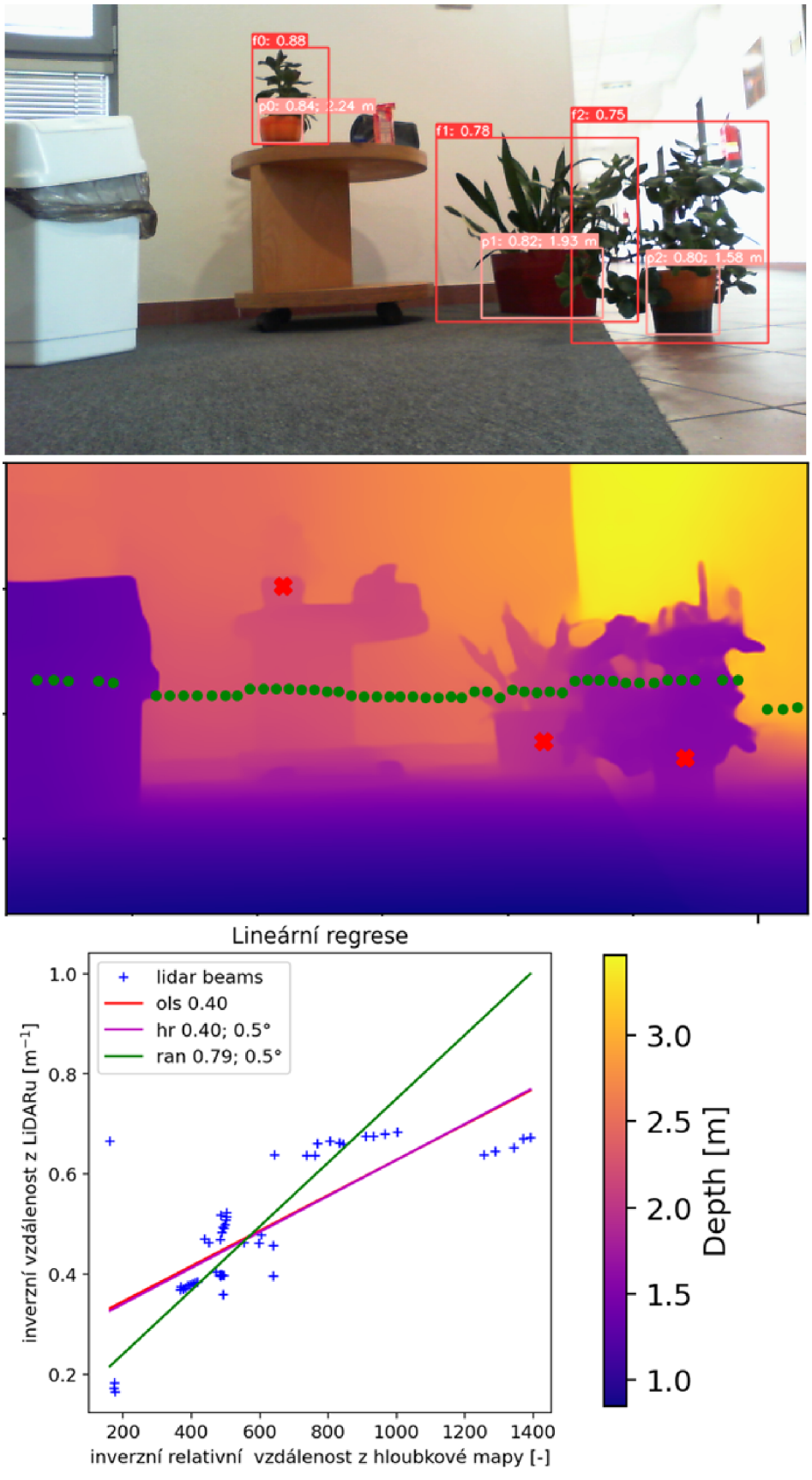
Obrázek 4.15: Příklady chyb v detekci

4.6.2 Vyhodnocení určování vzdálenosti květináčů

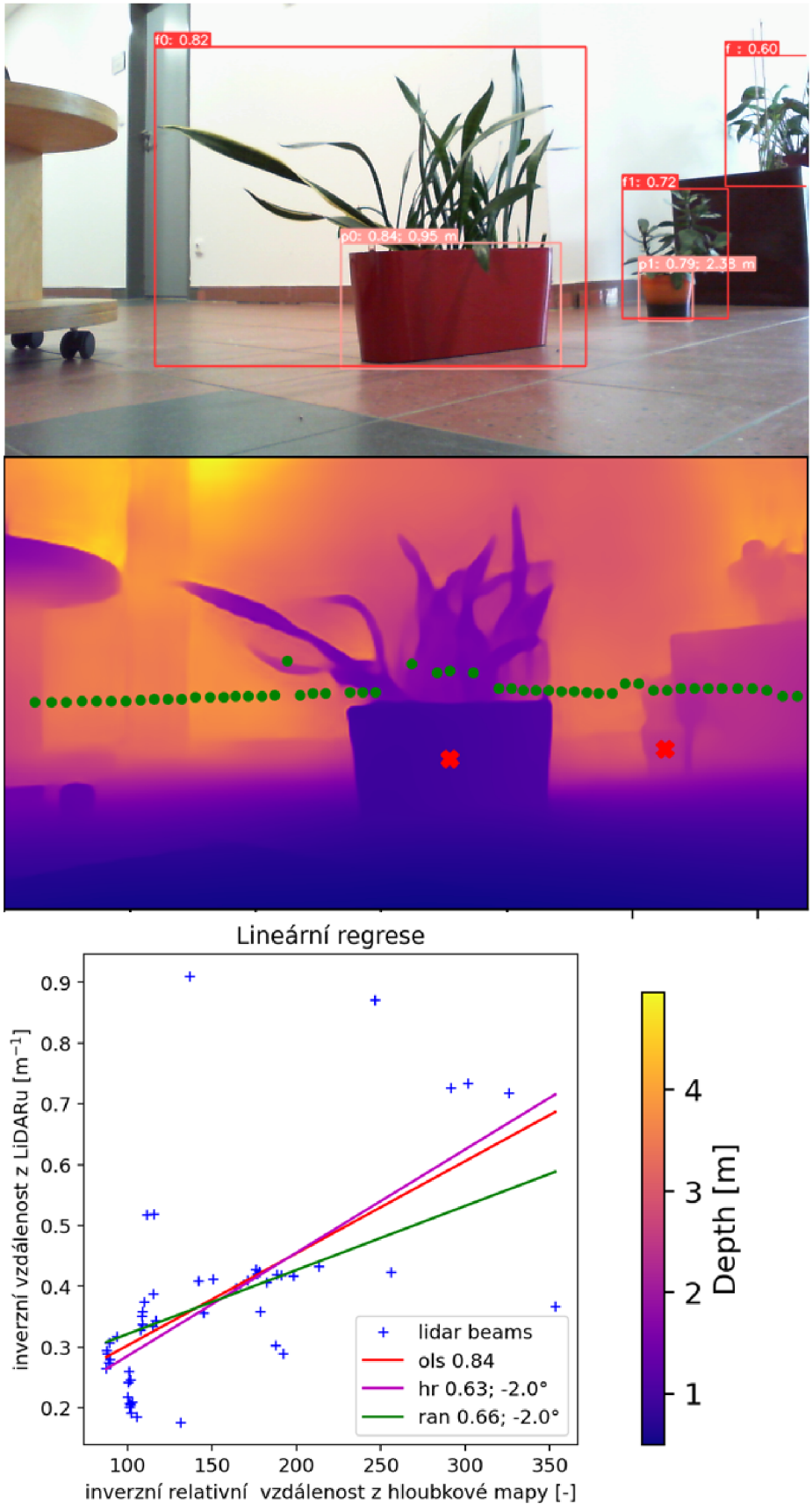
Určení vzdálenosti květináčů, příslušejících k detekovaným květinám, byla část práce, která kombinovala nejvíce vytvořených celků. Zahrnovala detekci modelem YOLOv5s 416 px, párování květin s květináči, predikci relativní hloubkové mapy, její syntézu s daty z LiDARu a tím převod na metrickou hloubkovou mapu. V této části je nejprve na příkladech ukázáno, jakých výsledků systém dosahoval. Kvantitativně je vyhodnocena chybovost jednotlivých řešení a v závěru výpočetní náročnost a vhodné kombinace použití na Jetson Nano.

Funkčnost celého systému

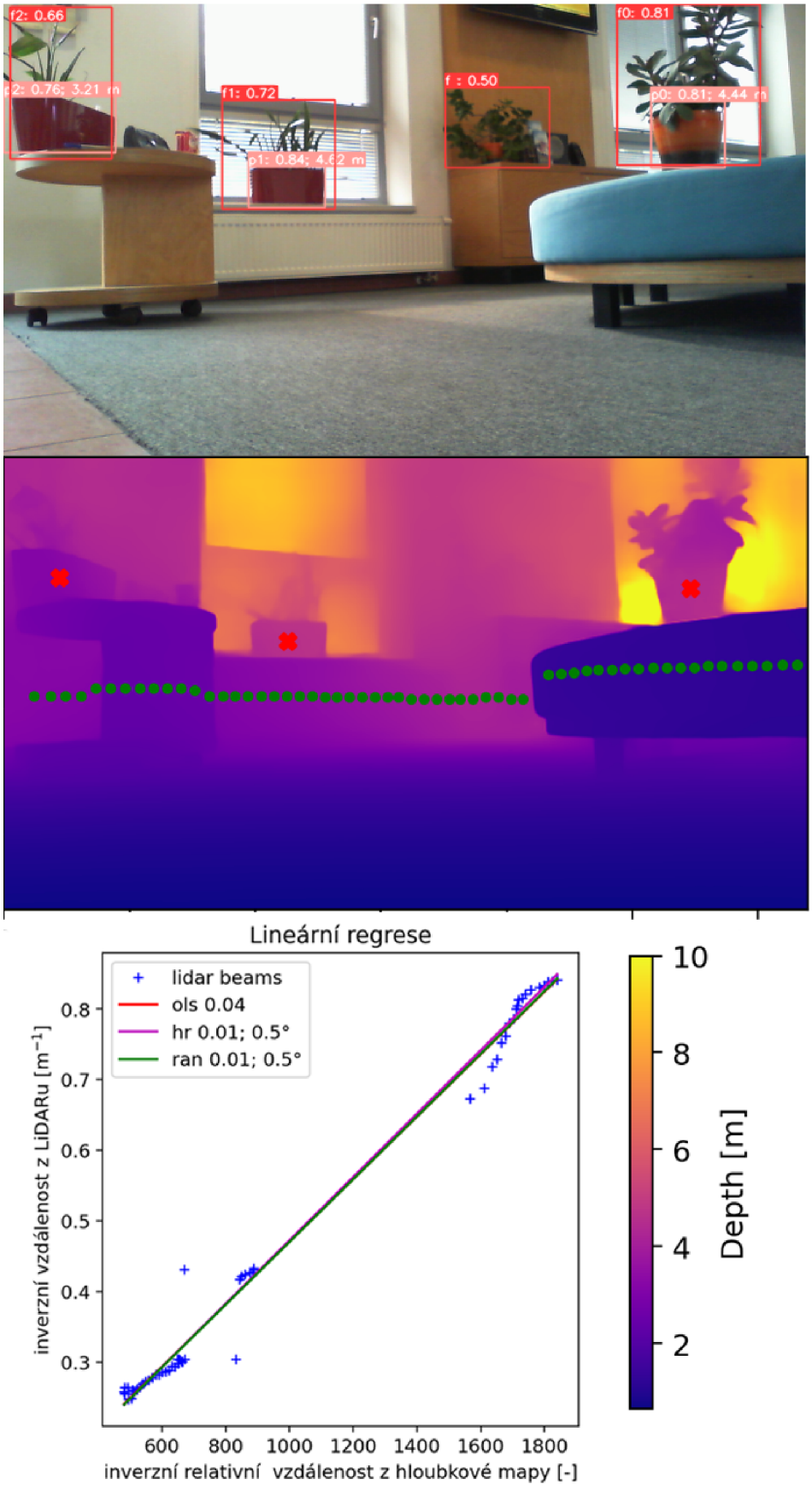
Následující tři trojice obrázků 4.16, 4.17 a 4.18 ilustrují vybrané testovací situace a demonstrují funkčnost i silné a slabé stránky systému. Horní obraz z trojice vizualizuje detekované rámečky, klasifikaci třídy objektu (f je květina, p je květináč), v případě spárování index páru, skóre jistoty od 0 do 1 a predikovanou vzdálenost květináče v metrech. Na prostředním obraze je vzniklá metrická hloubková mapa s barevnou škálou uvedenou pod ní. Zelené puntíky odpovídají koncovým bodům paprsků LiDARu, červené křížky značí středy rámečků predikovaných květináčů. Jelikož byla na robotu Breach kamera umístěna pod LiDARem, bližší body se na čip kamery promítly výše. Spodní obraz ukazuje proložení těchto bodů robustní lineární regresí, čímž byly nalezeny parametry α a β pro převod relativní hloubkové mapy na metrickou. Modré křížky příslušejí jednotlivým bodům.



Obrázek 4.16: Určení vzdálenosti modelem DPT-hybrid 384×224



Obrázek 4.17: Určení vzdálenosti modelem MiDaS2.1small 448×256



Obrázek 4.18: Určení vzdálenosti modelem DPT-hybrid 672×384

Veličiny na osách jsou v takové podobě, aby odpovídaly rozměru v rovnici 4.2. Grafy ukazují různé metody lineární regrese – obyčejné nejmenší čtverce (ols), Huberův regresor (hr) a RANSAC (ran). V legendě je dále uvedena chyba regrese v podobě 1− koeficient korelace, za ním úhel vybraný korekcí natočení mechanismu kamery.

Detekce na všech prezentovaných obrazech (i v elektronické příloze) zajišťoval natrénovaný model YOLOv5s z předchozí kapitoly při rozlišení (416 × 256) px, lineární regrese byla provedena Huberovým regresorem. Na prvním obraze (4.16) proběhla predikce relativní hloubky modelem DPT-hybrid při rozlišení (384 × 224) px. Skutečné vzdálenosti květináčů v pořadí indexů párů činily $2,03, 2,06a1,52$ m. Patrné jsou správné predikce rámečků i při výrazném překryvu květin. Hloubková mapa vykazovala chybu v predikci vzdálenosti zelených částí pravé květiny, což se na grafu projevovalo několika nejbližšími body v prostřední části. Predikce vzdálenosti květináčů dosahovala i tak malé chyby (do 10 %) a to i navzdory tomu, že je LiDAR nezachytil.

Druhý obrázek (4.17) ukazuje případ, kdy bylo zachyceno velké množství proměnlivých zelených částí květin. K predikci hloubky byl využit model MiDaS small při rozlišení (448 × 256) px. Skutečné vzdálenosti detekovaných květináčů byly 1,25 a 1,90 m, predikovány byly s větší chybou 25 %. Navzdory chybě je vidět, že i tento malý model může predikovat hloubkové mapy o přijatelné kvalitě a ve snazších situacích dosahoval uspokojivých výsledků.

Predikce hloubky na třetím obrázku (4.17) byla provedena modelem DPT-hybrid při rozlišení (672 × 384) px. Z grafu lineární regrese je patrná výborná lineárnost predikce. Jak je vidět v elektronické příloze, model MiDaS small takové lineárnosti nedosahovala poměr vzdáleností mezi sedačkou, stolkem a stěnou určil rozdílně. Obrovskou komplikaci však ve všech případech představovala predikce vzdálenosti květin proti oknům. Skutečné vzdálenosti květin na tomto obrázku činily 1,36, 3,62 a 2,42 m, model predikoval vzdálenost pravého květináče s chybou 267 %. Jelikož se vzdálenost neblížila maximální uvažované vzdálenosti, nemohla být snadno odhalena jako chybná. V případě modelu MiDaS small byly vzdálenosti obou květináčů u okna předpovězeny dále než 8 m, proto bylo možné je z predikce vyřadit.

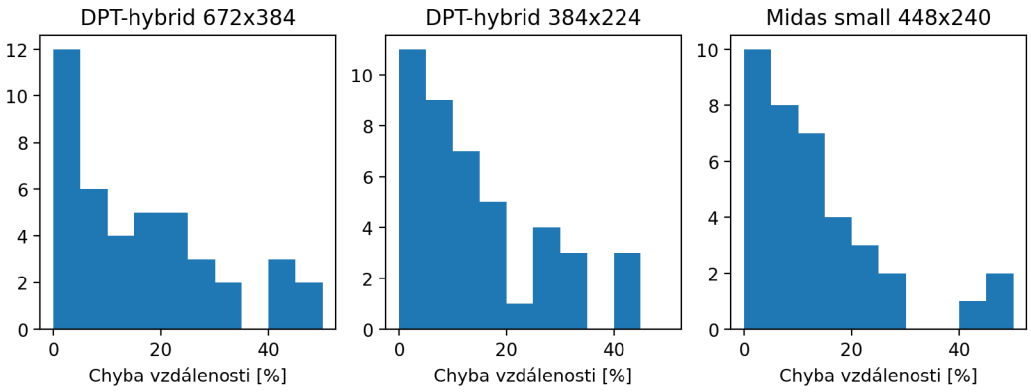
Obecně lze konstatovat, že modely DPT-hybrid fungovaly v obou rozlišeních velmi podobně. Model s nižším rozlišením byl negativně ovlivněn např. rozhraními koberce a podlahy, což je částečně vidět i na obrázku 4.16. Pro syntézu to ale nemělo vliv. Menší úroveň detailů v olistění syntézu také nezhoršovala výrazně, proto byl model s nižším rozlišením a tedy i nižšími výpočetními nároky zcela použitelný.

Z metod robustní lineární regrese se jevil lepší Huber, který na rozdíl od RANSACu částečně zohledňoval okrajové body. Těch se z důvodu nedokonalostí predikce vyskytovalo v obrazech vyskytovalo větší množství a mnohdy by nebyly uvažovány ani v případě, že byly určeny správně. RANSAC byl ovlivněn například špatnou predikcí sklonu velkých jednolitých ploch a nezohledňoval menší množství správně určených bližších nebo vzdálenějších bodů. Je proto vhodnější spíše pro velmi kvalitní modely predikce hloubky se spolehlivě lineární predikcí. Výpočetní čas jedné regrese metodou Huber činil pro 30 měření (47 ± 16) ms a metodou RANSAC (41 ± 11) ms.

Chybovost systému

Chybovost určení vzdálenosti byla vyhodnocena pro všechny 3 modely kvantitativně, a to na 46 květináčích, detekovaných na obrazech z elektronické přílohy. Obrázek 4.19 zobrazuje histogramy procentuální chyby (předpovězené vzdálenosti ku skutečné) v rozsahu 0 – 50

%. Obsahují spíše mírně těžší případy, než na kterých byl robot testován v rámci celého projektu.



Obrázek 4.19: Histogram procentuálních chyb určených vzdáleností

Vzhledem k tomu, že se jedná o experimentální řešení a navíc robot nevyžadoval určení polohy květináče pomocí tohoto systému s vysokou přesností, je chyba do 10 % výborná, do 20 % dostačující. Komplikace mohou nastat při chybě přesahující 40 %, tyto případy také nastávaly. V histogramech nejsou zahrnuty hrubé chyby nad 50 %. Těch se model DPT-hybrid při obou testovaných rozlišeních dopouštěl pouze v případě květináčů v situacích jako na obrázku 4.18, ve 4 případech při rozlišení 672 px, ve 3 případech při 384 px. Model MiDaS small v těchto situacích vykazoval hrubou chybu v 7 případech, v lehčích situacích ve 2 případech. V 5 případech byla predikovaná vzdálenost tak vysoká, že ji bylo možné na základě prahu označit za chybnou. Z histogramů je patrné, že všechny modely dosahují podobných přesností, překvapivě nejmenších odchylek model MiDaS small je ale méně spolehlivý.

Výpočetní nároky modelů pro detekci a predikci hloubky

V této části byla testována výpočetní náročnost na Jetson Nano jak modelů pro predikci hloubky, tak jejich kombinace s natrénovaným modelem YOLOv5s. Jetson Nano kromě 4 GB paměti sdílené pro GPU a CPU disponoval pamětí zram o doporučené velikosti 2 GB, která v případě potřeby ukládala komprimované bloky paměti na microSD kartu. Pokusy o její rozšíření vedly k nestabilitě systému. Samotný systém bez dalších spouštěných programů zabíral 1,5 GB.

Tabulka 4.11: Běh modelů MiDaS na Jetson Nano s GPU

	DPT-hybrid 672×384	DPT-hybrid 384×224	MiDaS small 448×256	MiDaS small 256×128
běh (*half*) [s]	3,2	1,3	0,13	0,13
paměť (*half*) [GB]	3,4	3,1	2,5	2,5
běh [s]	málo paměti	0,9	0,20	0,20
paměť [GB]	málo paměti	4,1	2,9	2,9

Tabulka 4.11 zobrazuje výsledky samostatného běhu modelů DPT-hybrid a MiDaS small na GPU. Jednou byl běh spuštěn s optimalizací *half precision* (16bitový float),

jednou bez ní (32bitový float). Použití optimalizace bylo u modelu DPT-hybrid klíčové, i tak jí zabíral mnoho.

Dále byly zkoušeny různé kombinace běhu modelů tak, aby na Jetson Nano šly spustit současně. Tři relevantní kombinace zobrazuje tabulka 4.12, průměrné hodnoty byly získány z 20 měření. Model DPT-hybrid mohl spolu s YOLO běžet pouze na CPU. Jeho predikce tak trvala téměř 12 sekund, při rozlišení 672×384 dokonce 30 s. To by nebylo na závadu, robot během predikce hloubky stál na místě. Běh YOLO se ale zpomalil k frekvenci 1 FPS, což už nebylo pro detekci během jízdy robotu ideální.

Nejpraktičtější kombinaci pro použití na robotu představoval běh YOLO na GPU a MiDaS small na CPU. V tomto případě byl běh modelu YOLO víc než 2x pomalejší než v případě jeho samostatného běhu. To bylo ale způsobeno střídáním běhu s modelem MiDaS a aktivním zapojením paměti zram. Při pohybu robotu k tomuto přesunu paměti docházet nemuselo, proto tento případ nebyl problémový.

Nejrychlejší kombinací byl běh YOLO na CPU a MiDaS small na GPU. V takovém případě dosahoval běh i při započtení syntézy s LiDARem téměř 0,5 FPS. Díky tomu by bylo možné určovat vzdálenost i během pomalé jízdy robotu a tu tím pádem iterativně zpřesňovat. Tato možnost však v praxi testována nebyla.

Tabulka 4.12: Kombinace YOLO a MiDaS na Jetson Nano

kombinace modelů	v5s 416×256 CPU hybrid 384×224 CPU	v5s 416×256 GPU small 448×256 CPU	v5s 416×256 CPU small 448×256 GPU
běh YOLO [s]	$0,63 \pm 0,05$	$0,12 \pm 0,03$	$0,63 \pm 0,05$
běh MiDaS [s]	$11,7 \pm 0,3$	$5,2 \pm 0,3$	$0,282 \pm 0,014$
paměť [GB]	1,8	3,3	2,8

Výpočetní nároky webkamery a detekci ArUco

Původně byly pracováno s maximálním rozlišením použité webkamery (1280×800) px. Díky tomu bylo možné určovat vzdálenost z ArUco štítků o velikosti 2,2 cm až na vzdálenost 1 m. Jak se ukázalo při testování projektu na robotu Leela, zátěž CPU byla při pohybu robotu a paralelní detekci ArUco štítků příliš vysoká. Proto bylo rozlišení sníženo na 640×360. Zátěž procesoru pro jednotlivá rozlišení je v tabulce 4.13.

Z požadavku pro určení vzdálenosti a i přesné orientace štítku na vzdálenost 1,5 m byla pro testování projektu zvolena velikost štítku 7,6 cm. Robot by ale mohl v tomto případně přijet blíže k detekované květině a určit polohu s orientací i s použitím menších štítků.

Tabulka 4.13: Zátěž procesoru webkamerou na Jetson Nano

Rozlišení kamery	424×240	640×360	1280×800
zátěž CPU kamerou [%]	2	3	11
zátěž CPU ArUco [%]	5	10	23
běh ArUco [ms]	9	15	25

5 Závěr

Tato práce byla součástí projektu řešeného 5 studenty. Jeho smyslem bylo rozšíření platformy mobilního robotu Breach tak, aby se robot dokázal autonomně pohybovat ve vnitřním prostřední a zalévat v něm pokojové rostliny. Náplní této práce bylo veškeré obrazové zpracování související s projektem. Jednalo se o detekci květin, určování jejich vzdálenosti na vzdálenosti několika metrů s menší přesností i na vzdálenost v řádu desítek centimetrů s vyšší přesností pomocí štítků, které zároveň sloužily k identifikaci dané rostliny. Uvažována byla aplikace metod na embedded počítači NVIDIA Jetson Nano.

První cíl práce spočíval v rešerši relevantních metod pro detekci hledaných květin. Prozkoumány byly metody s využitím konvolučních neuronových sítí. Pro použití na robotu s nižšími výpočetními kapacitami se jevila nejvhodnější metoda YOLOv5, která byla podrobněji popsána. Dalším cílem bylo provedení průzkumu metod určení vzdálenosti květin. Nalezeny a zváženy byly metody od specializovaných kamer až po metody využívající konvoluční neuronové sítě. Nejmenší nároky na cenu, místo a nutnost robotu vykonávat dodatečné pohyby kladly metody pro monokulární predikci hloubky (MPH). Představeno bylo rozdělení metod MPH a nastíněny principy vybraných metod. Zmíněny byly i možné typy štítků pro přesnější určení polohy květin a jejich identifikaci z menší vzdálenosti.

V praktické části bylo nejdříve zprovozněno a na vlastních datech otestováno 8 metod MPH. Přesnost výstupu metod a jejich výpočetní náročnost byla posuzována kvalitativně převážně na obrazech z vnitřního prostředí s květinami. Na základě vzájemného srovnání byla pro další postup zvolena metoda MiDaS a její rozšíření DPT, jelikož převyšovaly ostatní metody v kvalitě predikce, nabízely také odlehčené modely s dobrou kvalitou výstupu, které byly již natrénované a dostatečně robustní.

Proběhl sběr a anotace fotografií květin a květináčů ve vnitřním prostředí. Tvořily menší dataset čítající přes 1000 fotografií, který byl použit pro přetrénování méně náročné varianty detektoru metody YOLOv5 původně trénovaném na datasetu COCO. Vytvořen byl algoritmus, který detekce významově roztřídí na páry květina-květináč.

Identifikační štítky byly zvoleny ArUco, které byly k dispozici v knihovně OpenCV. K jejich aplikaci byla kalibrována použitá webkamera.

Metoda MiDaS predikovala lineární relativní hloubkovou mapu s nekonzistentní škálou α a posunutím β, pro získání metrickou hloubkové mapy musela být po každé predikci kalibrována. Proto byl vytvořen algoritmus syntézy hloubkové mapy s daty z LiDARu, který byl na robotu již přítomen. Algoritmus přiřadil vzdálenosti koncových bodů LiDARu patřičným pixelům hloubkové mapy. Robustní lineární regresí Huber byly nalezeny α a β a proběhl přepočet na metrickou hloubku. Na základě polohy detekovaných květináčů byla odečtena z mapy jejich vzdálenost a určena jejich poloha.

Veškerý kód, psaný v jazyce Python, byl implementován do robotického frameworku ROS ve formě dvou balíčků. První z nich, *aruco*, zajišťoval obsluhu kamery a detekci ArUco štítků s určením jejich polohy a orientace na krátkou vzdálenost. Druhý, *yolov5*, zajišťoval detekci květin a určování jejich polohy na základě hloubkové mapy.

Vytvořený program obsahující aplikované metody byl zprovozněn na zařízení Jetson Nano. Byla otestována výpočetní náročnost metod, především možnost současného použití modelů YOLO a MiDaS na tomto zařízení s nízkou výpočetní kapacitou. Nejvhodnější varianta představovala běh YOLO na GPU s naprosto dostatečnou frekvencí 5-10 FPS a odlehčeného modelu MiDaS small na CPU, jehož běh tak trval 5 sekund. Jelikož v době určování vzdálenosti robot stál na místě, doba výpočtu nehrála roli a požadavky na výpočetní náročnost byly zcela splněny.

Všechny nástroje byly testovány na květinách ve vnitřním prostředí. Přesnost detekce byla testována na mírně obtížnějších případech, než byly pro běh robotu vyžadovány. Dosahovala metriky mAP0.50 rovné 0,92. Z kvalitativního posouzení bylo patrné, že pokud se květiny výrazně nepřekrývají, je detekce a párování na květiny a květináče spolehlivé. Určování vzdálenosti bylo testováno ve stejném prostředí. Odchylka od skutečné vzdálenosti se pohybovala ve většině případů do 20 %, výjimečně nad 40 % ve speciálních případech. Vzhledem k tomu, že tato část práce využívala metody staré několik málo let a byla spíše experimentální, jsou tyto výsledky slušné a chod robotu jako celku nenarušily. Všechny cíle práce tak byly splněny.

Literatura

[1] PODOLINSKÝ, Ondřej. *Komplexní návrh zalévacího mobilního robotu* [online]. Brno, 2022 [cit. 2022-05-04]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140183. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[2] BAJER, Jan. *Návrh a realizace komunikačního rozhraní autonomního mobilního robotu* [online]. Brno, 2022 [cit. 2022-05-05]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140227. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Stanislav Věchet.

[3] DOSEDĚL, Miroslav. *Návrh a realizace strategie plánovaní pohybu mobilního robotu* [online]. Brno, 2022 [cit. 2022-05-04]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140228. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Stanislav Věchet.

[4] VIZVÁRY, Peter. *Návrh zalévacího modulu pro mobilní robot* [online]. Brno, 2022 [cit. 2022-05-05]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140184. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[5] MICHELUCCI, Umberto. *Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection*. Berkeley: Apress, 2019. ISBN 978-1-4842-4975-8

[6] *CS231n Convolutional Neural Networks for Visual Recognition* [online]. [cit. 2022-04-30]. Dostupné z: https://cs231n.github.io/

[7] HARSH. Understanding convolutional neural network (CNN). *CodeSpeedy* [online]. [cit. 2022-04-30]. Dostupné z: https://www.codespeedy.com/convolutional-neural-network-cnn/

[8] NERD, Random. Swish activation function by Google. *Medium* [online]. [cit. 2022-04-30]. Dostupné z: Swish activation function by Google

[9] LIU, Li, Wanli OUYANG, Xiaogang WANG, Paul FIEGUTH, Jie CHEN, Xinwang LIU a Matti PIETIKÄINEN. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision* [online]. 2020, **128**(2), 261-318 [cit. 2022-04-30]. ISSN 0920-5691. Dostupné z: doi:10.1007/s11263-019-01247-4

[10] HUANG, Jonathan, Vivek RATHOD, Chen SUN, et al. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition* [online]. 2017, s. 3296-3305 [cit. 2022-04-30]. ISBN 978-153860457-1. Dostupné z: doi:10.1109/CVPR.2017.351

[11] HUI, Jonathan. MAP (mean Average Precision) for Object Detection. In: *Medium*. [online]. [cit. 2022-05-17]. Dostupné z: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

[12] COCO - Common Objects in Context [online]. [cit. 2022-05-17]. Dostupné z: https://cocodataset.org/#detection-eval

[13] GIRSHICK, Ross, Jeff DONAHUE, Trevor DARRELL a Jitendra MALIK. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* [online]. 2014, s. 580-587 [cit. 2022-04-30]. ISBN 978-147995117-8. ISSN 1063-6919. Dostupné z: doi:10.1109/CVPR.2014.81

[14] GIRSHICK, Ross. Fast R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision* [online]. 2015, s. 1440-1448 [cit. 2022-04-30]. ISBN 978-1-4673-8391-2. ISSN 15505499. Dostupné z: doi:10.1109/ICCV.2015.169

[15] REN, Shaoqing, Kaiming HE, Ross GIRSHICK a Jian SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2017, **39**(6), 1137-1149 [cit. 2022-04-30]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2016.2577031

[16] BOCHKOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan MARK LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv* [online]. 2020 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2004.10934

[17] FU, Cheng-Yang, Wei LIU, Ananth RANGA a Akexander C. BERG. Dssd: Deconvolutional single shot detector. *ArXiv* [online]. 2017 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/pdf/1701.06659.pdf

[18] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement. *CoRR* [online]. 2018 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/1804.02767

[19] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, et al. SSD: Single Shot MultiBox Detector. In: *Computer Vision – ECCV 2016 Lecture Notes in Computer Science* [online]. 2016, s. 21-37 [cit. 2022-04-30]. ISBN 978-331946447-3. ISSN 0302-9743. Dostupné z: doi:10.1007/978-3-319-46448-0_2

[20] LIN, Tsung-Yi, Priya GOYAL, Ross GIRSHICK a Piotr DOLLÁR. Focal Loss for Dense Object Detection. In: *Proceedings of the IEEE International Conference on Computer Vision* [online]. 2017, s. 2999-3007 [cit. 2022-04-30]. ISBN 978-153861032-9. ISSN 1550-5499. Dostupné z: doi:10.1109/ICCV.2017.324

[21] TAN, Mingxing, Ruoming PANG a Quoc V. LE. EfficientDet: Scalable and Efficient Object Detection. In: 2020 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2020, 2020, s. 10778-10787 [cit. 2022-04-30]. ISBN 978-1-7281-7168-5. ISSN 10636919. Dostupné z: doi:10.1109/CVPR42600.2020.01079

[22] Automl/efficientdet at master · google/automl. *GitHub* [online]. [cit. 2022-04-30]. Dostupné z: https://github.com/google/automl/tree/master/efficientdet

[23] CARION, Nicolas, Francisco MASSA, Gabriel SYNNAEVE, Nicolas USUNIER, Alexander KIRILLOV a Sergey ZAGORUYKO. End-to-End Object Detection with Transformers. VEDALDI, Andrea, Horst BISCHOF, Thomas BROX a Jan-Michael FRAHM, ed. *Computer Vision – ECCV 2020 [online]. Cham: Springer International Publishing*, 2020, 2020-11-03, s. 213-229 [cit. 2022-04-30]. Lecture Notes in Computer Science. ISBN 978-3-030-58451-1. Dostupné z: doi:10.1007/978-3-030-58452-8_13

[24] LIU, Ze, Yutong LIN, Yue CAO, Han HU, Yixuan WEI, Zheng ZHANG, Stephen LIN a Baining GUO. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. IEEE, 2021, 2021, s. 9992-10002 [cit. 2022-04-30]. ISBN 978-1-6654-2812-5. ISSN 15505499. Dostupné z: doi:10.1109/ICCV48922.2021.00986

[25] LAW, Hei a Jia DENG. CornerNet: Detecting Objects as Paired Keypoints. FERRARI, Vittorio, Martial HEBERT, Cristian SMINCHISESCU a Yair WEISS, ed. *Computer Vision – ECCV 2018* [online]. Cham: Springer International Publishing, 2018, 2018-10-09, s. 765-781 [cit. 2022-04-30]. Lecture Notes in Computer Science. ISBN 978-3-030-01263-2. Dostupné z: doi:10.1007/978-3-030-01264-9_45

[26] TIAN, Zhi, Chunhua SHEN, Hao CHEN a Tong HE. FCOS: Fully Convolutional One-Stage Object Detection. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. IEEE, 2019, 2019, s. 9626-9635 [cit. 2022-04-30]. ISBN 978-1-7281-4803-8. Dostupné z: doi:10.1109/ICCV.2019.00972

[27] DUAN, Kaiwen, Song BAI, Lingxi XIE, Honggang QI, Qingming HUANG a Qi TIAN. CenterNet: Keypoint Triplets for Object Detection. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. IEEE, 2019, 2019, s. 6568-6577 [cit. 2022-04-30]. ISBN 978-1-7281-4803-8. Dostupné z: doi:10.1109/ICCV.2019.00667

[28] DUAN, Kaiwen, Song BAI, Lingxi XIE, Honggang QI, Qingming HUANG a Qi TIAN. CenterNet++ for Object Detection. *ArXiv* [online]. 2022 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2204.08394

[29] YU, Guanghua, Qinyao CHANG, Wenyu LV, et al. PP-PicoDet: A Better Real-Time Object Detector on Mobile Devices. *ArXiv* [online]. 2021 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2111.00902

[30] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. You Only Look Once: Unified, Real-Time Object Detection. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2016, s. 779-788. ISBN 978-146738850-4. ISSN 1063-6919. Dostupné z: doi:10.1109/CVPR.2016.91

[31] REDMON, Joseph a Ali FARHADI. YOLO9000: Better, Faster, Stronger. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* [online]. 2017, s. 6517-6525 [cit. 2022-04-30]. ISBN 978-153860457-1. Dostupné z: doi:10.1109/CVPR.2017.690

[32] WANG, Chien-Yao, Alexey BOCHKOVSKIY a Hong-Yuan Mark LIAO. Scaled-YOLOv4: Scaling Cross Stage Partial Network. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) [online]. IEEE, 2021, 2021, s. 13024-13033 [cit. 2022-04-30]. ISBN 978-1-6654-4509-2. Dostupné z: doi:10.1109/CVPR46437.2021.01283

[33] WANG, Chien-Yao, I-Hau YEH a Hong-Yuan MARK LIAO. You Only Learn One Representation: Unified Network for Multiple Tasks. *ArXiv* [online]. 2021 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2105.04206

[34] JOCHER, Glenn. Ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite. *GitHub* [online]. [cit. 2022-04-30]. Dostupné z: https://github.com/ultralytics/yolov5

[35] ZHANG, Junguo, Wenbin LI, Zhongxing YIN, Shengbo LIU a Xiaolin GUO. Forest fire detection system based on wireless sensor network. In: *2009 4th IEEE Conference on Industrial Electronics and Applications* [online]. IEEE, 2009, 2009, s. 520-523 [cit. 2022-04-30]. ISBN 978-1-4244-2799-4. Dostupné z: doi:10.1109/ICIEA.2009.5138260

[36] YUN, Wonsub, J. Praveen KUMAR, Sangjoon LEE, Dong-Soo KIM a Byoung-Kwan CHO. Deep learning-based system development for black pine bast scale detection. *Scientific Reports* [online]. 2022, **12**(1) [cit. 2022-04-30]. ISSN 2045-2322. Dostupné z: doi:10.1038/s41598-021-04432-z

[37] NEPAL, Upesh a Hossein ESLAMIAT. Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *Sensors* [online]. 2022, 22(2) [cit. 2022-04-30]. ISSN 1424-8220. Dostupné z: doi:10.3390/s22020464

[38] ZHANG, Haoting, Mei TIAN, Gaoping SHAO, Juan CHENG a Jingjing LIU. Target Detection of Forward-Looking Sonar Image Based on Improved YOLOv5. *IEEE Access* [online]. 2022, 10, 18023-18034 [cit. 2022-04-30]. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2022.3150339

[39] LIU, Shu, Lu QI, Haifang QIN, Jianping SHI a Jiaya JIA. Path Aggregation Network for Instance Segmentation. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* [online]. Salt Lake City, 2018, s. 8759-8768 [cit. 2022-04-30]. ISBN 978-1-5386-6420-9. ISSN 2575-7075. Dostupné z: doi:10.1109/CVPR.2018.00913

[40] SAMBASIVARAO, K. Non-maximum Suppression (NMS). In: *Medium.* [online]. [cit. 2022-05-17]. Dostupné z: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

[41] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Deep residual learning for image recognition. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* [online]. 2016, s. 770-778 [cit. 2022-04-30]. ISBN 978-146738850-4. ISSN 1063-6919. Dostupné z: doi:10.1109/CVPR.2016.90

[42] WANG, Chien-Yao, Hong-Yuan MARK LIAO, Yueh-Hua WU, Ping-Yang CHEN, Jun-Wei HSIEH a I-Hau YEH. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* [online]. IEEE, 2020, 2020, s. 1571-1580 [cit. 2022-04-30]. ISBN 978-1-7281-9360-1. Dostupné z: doi:10.1109/CVPRW50498.2020.00203

[43] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2015, **37**(9), 1904-1916 [cit. 2022-04-30]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2015.2389824

[44] YOLOv5 (6.0/6.1) brief summary Issue #6998 ultralytics/yolov5. *GitHub* [online]. [cit. 2022-04-30]. Dostupné z: https://github.com/ultralytics/yolov5/issues/6998

[45] ZHENG, Zhaohui, Ping WANG, Wei LIU, Jinze LI, Rongguang YE a Dongwei REN. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. In: *34th AAAI Conference on Artificial Intelligence, AAAI 2020* [online]. New York, 2020, s. 12993-13000 [cit. 2022-04-30]. ISBN 978-157735835-0. Dostupné z: https://arxiv.org/abs/1911.08287

[46] LIN, Tsung-Yi, Piotr DOLLÁR, Ross GIRSHICK, et al. Feature pyramid networks for object detection. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* [online]. s. 936-944 [cit. 2022-04-30]. ISBN 978-153860457-1. Dostupné z: doi:10.1109/CVPR.2017.106

[47] LIN, Tsung-Yi, Michael MAIRE, Belongie SERGE, et al. Microsoft COCO: Common objects in context. In: *Computer Vision – ECCV 2014. Lecture Notes in Computer Science* [online]. 2014, s. 740-755 [cit. 2022-04-30]. ISSN 0302-9743. Dostupné z: https://arxiv.org/abs/1405.0312

[48] Benewake. *Sklep Elektroniczny Botland - Sklep z Elektroniką* [online]. [cit. 2022-04-30]. Dostupné z: https://botland.cz/laserove-skenery/16638-laserovy-snimac-vzdalenosti-lidar-tf-luna-8-m-uart-i2c-5903351249041.html

[49] OpenCV modules. *OpenCV* [online]. [cit. 2022-04-30]. Dostupné z: https://docs.opencv.org/4.5.3/

[50] BHATTI, Asim, ed. *Current Advancements in Stereo Vision* [online]. InTech, 2012 [cit. 2022-04-30]. ISBN 978-953-51-0660-9. Dostupné z: doi:10.5772/2611

[51] DONG, Xingshuai, Matthew A. GARRATT, Sreenatha G. ANAVATTI a Hussein A. ABBASS. Towards Real-Time Monocular Depth Estimation for Robotics: A Survey. *IEEE Transactions on Intelligent Transportation Systems* [online]. 1-22 [cit. 2022-04-30]. ISSN 1524-9050. Dostupné z: doi:10.1109/TITS.2022.3160741

[52] *Intel® RealSense™ Depth and Tracking Cameras* [online]. [cit. 2022-04-30]. Dostupné z: https://www.intelrealsense.com/

[53] GRIFFIN, Brent A. a Jason J. CORSO. Depth from Camera Motion and Object Detection. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2021, 2021, s. 1397-1406 [cit. 2022-04-30]. ISBN 978-1-6654-4509-2. ISSN 10636919. Dostupné z: doi:10.1109/CVPR46437.2021.00145

[54] HASEEB, Muhammad A., Jianyu GUAN, Danijela RISTIC-DURRANT a Axel GRÄSER. DisNet: *A novel method for distance estimation from monocular camera* [online]. In: . 2018 [cit. 2022-04-30]. Dostupné z: https://project.inria.fr/ppniv18/files/2018/10/paper22.pdf

[55] ZHU, Jing a Yi FANG. Learning Object-Specific Distance From a Monocular Image. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. 2019, s. 3839-3848 [cit. 2022-04-30]. Dostupné z: https://openaccess.thecvf.com/content_ICCV_2019/papers/Zhu_Learning_Object-Specific_Distance_From_a_Monocular_Image_ICCV_2019_paper.pdf

[56] VAJGL, Marek, Petr HURTIK a Tomáš NEJEZCHLEBA. Dist-YOLO: Fast Object Detection with Distance Estimation. *Applied Sciences* [online]. 2022, 12(3) [cit. 2022-04-30]. ISSN 2076-3417. Dostupné z: https://www.mdpi.com/2076-3417/12/3/1354

[57] ZHAO, ChaoQiang, QiYu SUN, ChongZhen ZHANG, Yang TANG a Feng QIAN. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences* [online]. 2020, **63**(9), 1612-1627 [cit. 2022-04-30]. ISSN 1674-7321. Dostupné z: doi:10.1007/s11431-020-1582-8

[58] GODARD, Clement, Oisin Mac AODHA, Michael FIRMAN a Gabriel BROSTOW. Digging Into Self-Supervised Monocular Depth Estimation. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. IEEE, 2019, 2019, s. 3827-3837 [cit. 2022-04-30]. ISBN 978-1-7281-4803-8. Dostupné z: https://ieeexplore.ieee.org/document/9009796/

[59] GARG, Ravi, Vijay Kumar B.G., Gustavo CARNEIRO a Ian REID. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. LEIBE, Bastian, Jiri MATAS, Nicu SEBE a Max WELLING, ed. *Computer Vision – ECCV 2016* [online]. Cham: Springer International Publishing, 2016, 2016-09-17, s. 740-756 [cit. 2022-04-30]. Lecture Notes in Computer Science. ISBN 978-3-319-46483-1. Dostupné z: http://link.springer.com/10.1007/978-3-319-46484-8_45

[60] EIGEN, David, Christian PUHRSCH a Rob FERGUS. Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In: *Advances in Neural Information Processing Systems* [online]. Curran Associates, 2014, 2366 - 2374 [cit. 2022-04-30]. ISSN 10495258. Dostupné z: https://proceedings.neurips.cc/paper/2014/file/7bccfde7714a1ebadf06c5f4cea752c1-Paper.pdf

[61] ALHASHIM, Ibraheem a Peter WONKA. High Quality Monocular Depth Estimation via Transfer Learning. *ArXiv* [online]. 2018 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/1812.11941

[62] FAROOQ BHAT, Shariq, Ibraheem ALHASHIM a Peter WONKA. AdaBins: Depth Estimation Using Adaptive Bins. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2021, 2021, s. 4008-4017 [cit. 2022-04-30]. ISBN 978-1-6654-4509-2. Dostupné z: https://ieeexplore.ieee.org/document/9578024/

[63] ZHOU, Tinghui, Matthew BROWN, Noah SNAVELY a David G. LOWE. Unsupervised Learning of Depth and Ego-Motion from Video. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2017, 2017, s. 6612-6619 [cit. 2022-04-30]. ISBN 978-1-5386-0457-1. Dostupné z: http://ieeexplore.ieee.org/document/8100183/

[64] JIAWANG, Bian, Li ZHICHAO, Wang NAIYAN, et al. Unsupervised Scale-consistent Depth and Ego-motion Learning from Monocular Video. In: *Advances in Neural Information Processing Systems* [online]. 2019 [cit. 2022-04-30]. Dostupné z: https://proceedings.neurips.cc/paper/2019/file/6364d3f0f495b6ab9dcf8d3b5c6e0b01-Paper.pdf

[65] MCCRAITH, Robert, Lukas NEUMANN a Andrea VEDALDI. Calibrating Self-supervised Monocular Depth Estimation. In: *ArXiv* [online]. [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2009.07714

[66] HUANGYING, Zhan, Huangying ZHAN, Jia-Wang WANG, et al. Auto-Rectify Network for Unsupervised Indoor Depth Estimation. In: *ArXiv* [online]. 2020 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2006.02708

[67] GEIGER, Andreas, Philip LENZ, Christoph STILLER a Raquel URTASUN. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research* [online]. 2013, **32**(11), 1231-1237 [cit. 2022-04-30]. ISSN 0278-3649. Dostupné z: http://journals.sagepub.com/doi/10.1177/0278364913491297

[68] SILBERMAN, Nathan, Derek HOIEM, Pushmeet KOHLI a Rob FERGUS. Indoor Segmentation and Support Inference from RGBD Images. FITZGIBBON, Andrew, Svetlana LAZEBNIK, Pietro PERONA, Yoichi SATO a Cordelia SCHMID, ed. Computer Vision – ECCV 2012 [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 2012, s. 746-760 [cit. 2022-04-30]. Lecture Notes in Computer Science. ISBN 978-3-642-33714-7. Dostupné z: http://link.springer.com/10.1007/978-3-642-33715-4_54

[69] DOSOVITSKIY, Alexey, Lucas BEYER, Alexander KOLESNIKOV, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR* [online]. 2021 [cit. 2022-04-30]. Dostupné z: https://openreview.net/forum?id=YicbFdNTTy

[70] WOFK, Diana, Fangchang MA, Tien-Ju YANG, Sertac KARAMAN a Vivienne SZE. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In: *2019 International Conference on Robotics and Automation (ICRA)* [online]. IEEE, 2019, 2019, s. 6101-6108 [cit. 2022-04-30]. ISBN 978-1-5386-6027-0. Dostupné z: https://ieeexplore.ieee.org/document/8794182

[71] WANG, Linda, Mahmoud FAMOURI a Alexander WONG. DepthNet Nano: A highly compact self-normalizing neural network for monocular depth estimation. *ArXiv* [online]. 2020 [cit. 2022-04-30]. Dostupné z: https://arxiv.org/abs/2004.08008

[72] GODARD, Clement, Oisin Mac AODHA a Gabriel J. BROSTOW. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2017, 2017, s. 6602-6611 [cit. 2022-04-30]. ISBN 978-1-5386-0457-1. Dostupné z: http://ieeexplore.ieee.org/document/8100182/

[73] CASSER, Vincent, Soeren PIRK, Reza MAHJOURIAN a Anelia ANGELOVA. Depth Prediction without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos. In: *Proceedings of the AAAI Conference on Artificial Intelligence* [online]. 2019, 8001 - 8008 [cit. 2022-04-30]. ISBN 978-157735809-1. Dostupné z: https://ojs.aaai.org/index.php/AAAI/article/view/4801

[74] CASSER, Vincent, Soeren PIRK, Reza MAHJOURIAN a Anelia ANGELOVA. Unsupervised Monocular Depth and Ego-Motion Learning With Structure and Semantics. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* [online]. 2019, s. 381-388 [cit. 2022-04-30]. ISBN 978-1-7281-2506-0. Dostupné z: https://ieeexplore.ieee.org/document/9025608

[75] LI, Hanhan, Ariel GORDON, Hang ZHAO, Vincent CASSER a Anelia ANGELOVA. Unsupervised Monocular Depth Learning in Dynamic Scenes. In: *Proceedings of the 2020 Conference on Robot Learning* [online]. 2021, s. 1908–1917 [cit. 2022-04-30]. Dostupné z: https://proceedings.mlr.press/v155/li21a.html

[76] RANFTL, René, Katrin LASINGER, David HAFNER, Konrad SCHINDLER a Vladlen KOLTUN. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2020, **44**(3), 1623 - 1637 [cit. 2022-04-30]. Dostupné z: https://ieeexplore.ieee.org/document/9178977

[77] RANFTL, René, Alexey BOCHKOVSKIY a Vladlen KOLTUN. Vision Transformers for Dense Prediction. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* [online]. 2021, 12159 - 12168 [cit. 2022-04-30]. Dostupné z: https://ieeexplore.ieee.org/document/9711226

[78] MIANGOLEH, S. Mahdi H., Sebastian DILLE, Long MAI, Sylvain PARIS a Yağız AKSOY. Boosting Monocular Depth Estimation Models to High-Resolution via Content-Adaptive Multi-Resolution Merging. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2021* [online]. 2021, s. 9680-9689 [cit. 2022-04-30]. ISBN 978-166544509-2. ISSN 10636919. Dostupné z: doi:10.1109/CVPR46437.2021.00956

[79] PRINCE, Simon J.D. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012. ISBN 9781107011793.

[80] QRcode.com DENSO WAVE. *Qrcode.com* [online]. 2020 [cit. 2022-04-30]. Dostupné z: https://www.qrcode.com/en/

[81] AprilTag: A robust and flexible visual fiducial system. In: *2011 IEEE International Conference on Robotics and Automation* [online]. 2011 [cit. 2022-04-30]. ISBN 978-1-61284-385-8. Dostupné z: doi:10.1109/ICRA.2011.5979561

[82] GARRIDO-JURADO, Sergio, Rafael MUÑOZ-SALINAS, Francisco José MADRID-CUEVAS a Manuel J. MARÍN-JIMÉNEZ. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* [online]. 2014, **47**(6), 2280-2292 [cit. 2022-04-30]. Dostupné z: doi:10.1016/j.patcog.2014.01.005

[83] Bender Robotics | Mobilní robotická platforma Breach. *Bender Robotics* [online]. [cit. 2022-05-13]. Dostupné z: https://www.benderrobotics.cz/breach.html

[84] Qengineering/Jetson-Nano-Ubuntu-20-image: Jetson Nano with Ubuntu 20.04 image. *GitHub* [online]. [cit. 2022-05-20]. Dostupné z: https://github.com/Qengineering/Jetson-Nano-Ubuntu-20-image

[85] STIPPEL, Christian. Depth estimation and segmentation for 3D object tracking. *Cognitive XR* [online]. [cit. 2022-05-20]. Dostupné z: https://cognitivexr.at/blog/2021/11/05/depth-estimation-and-segmentation-for-3d-object-tracking.html

[86] ASHWATH, Balraj. Indoor Training Set (ITS) [RESIDE-Standard]. *Kaggle*. [online]. [cit. 2022-05-20]. Dostupné z: https://www.kaggle.com/datasets/balraj98/indoor-training-set-its-residestandard

[87] AHMAD, Muhammad. MIT Indoor Scenes. *Kaggle*. [online]. [cit. 2022-05-20]. Dostupné z: https://www.kaggle.com/datasets/balraj98/indoor-training-set-its-residestandard

[88] *Make Sense* [online]. [cit. 2022-05-20]. Dostupné z: https://www.makesense.ai/

[89] JOCHER, Glenn. Ultralytics/yolov3: YOLOv3 in PyTorch > ONNX > CoreML > TFLite. *GitHub* [online]. [cit. 2022-05-12]. Dostupné z: https://github.com/ultralytics/yolov3

[90] RANFTL, Rene. isl-org/MiDaS: Code for robust monocular depth estimation described in "Ranftl et. al., Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer, TPAMI 2020". *GitHub* [online]. [cit. 2022-05-13]. Dostupné z: https://github.com/isl-org/MiDaS

Seznam zkratek

ROS	Robot Operating System
MP	Max Pooling
FC	Fully Connected
R-CNN	Region Based Convolutional Neural Network
R-FCN	Region Based Fully Convolutional Network
YOLO	You Only Look Once
SSD	Single Shot Detector
mAP	Mean Average Precision
COCO	Common Objects in Context
DSSD	Deconvolutional Single Shot Detecto
FPN	Feature Pyramid Network
BiFPN	Weighted Bi-directional Feature Pyramid Network
GPU	Graphics Processing Unit
DETR	Detection Transforme
FCOS	Fully Convolutional One-Stage
YOLOR	You Only Learn One Representation
PANet	Path Agregation Network
SPPF	Spatial Pyramid Pooling Fast
SPP	Spatial Pyramid Pooling
CIoU	Complete-IoU Loss
IoU	Intersection over Union
BCE	Binary Cross Entropy
CSP	Cross Stage Partial
LiDAR	Light Detection And Ranging
MPH	monokulární predikce hloubky
RGB-D	Red Green Blue Depth
PE	Pose Estimation
FPS	Frames Per Second
ViT	Vision Transformer
MiDaS	Mixing Datasets

DPT Dense Prediction Transformer

BMD Boosting Monocular Depth

CPU Central Processing Unit

ITS Indoor Training Set

MIT MIT Indoor Scenes

GFLOPs Giga FLoating-point Operations

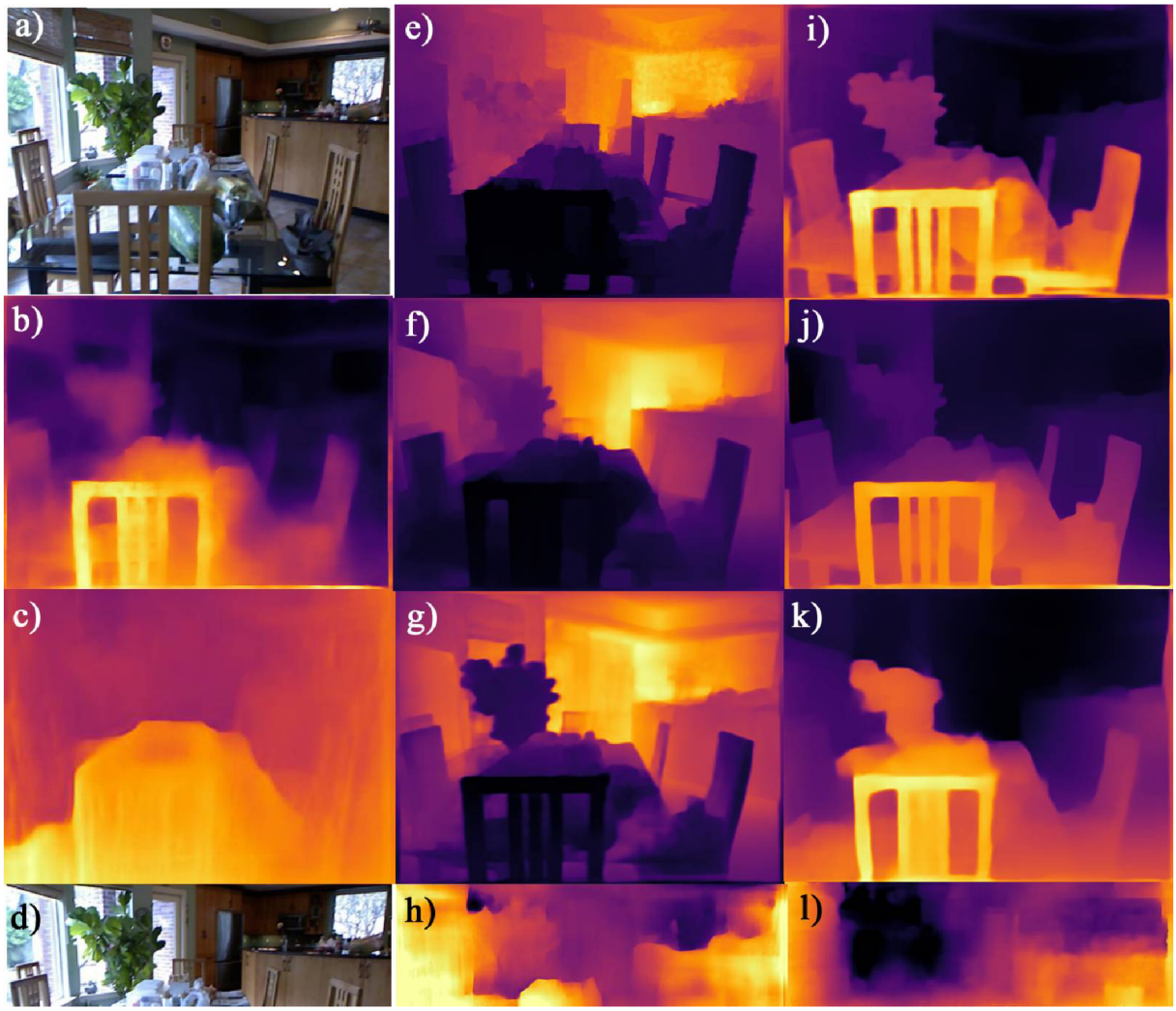
Seznam obrázků

Seznam tabulek

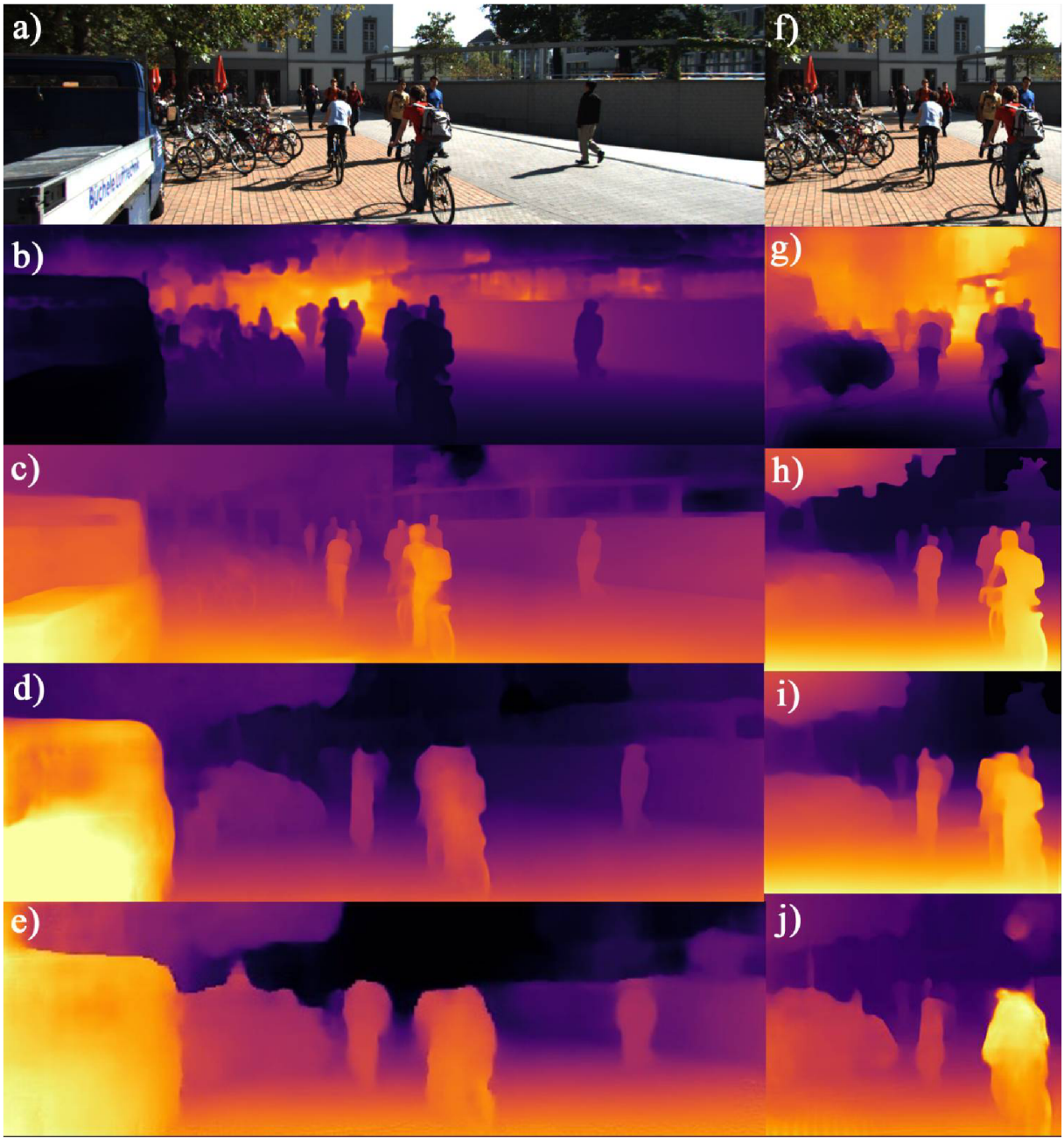
A Elektronické přílohy

2022_DP_Sladky_Jiri_200885_priloha.zip
- Monokulární predikce hloubky (MPH) – *další obrazy srovnávající metody MPH*
  - Obrazy_z_psane_prace
  - odkaz_na_dalsi_obrazy.txt
- Syntéza relativních hloubkových map s LiDARem – *další testovací obrazy*
- Ukázka datasetu – *několik trénovacích, validačních a testovacích obrazů s anotacemi*
- Zdrojový kód ROS – *vytvořené balíčky*
  - aruco
  - yolov5

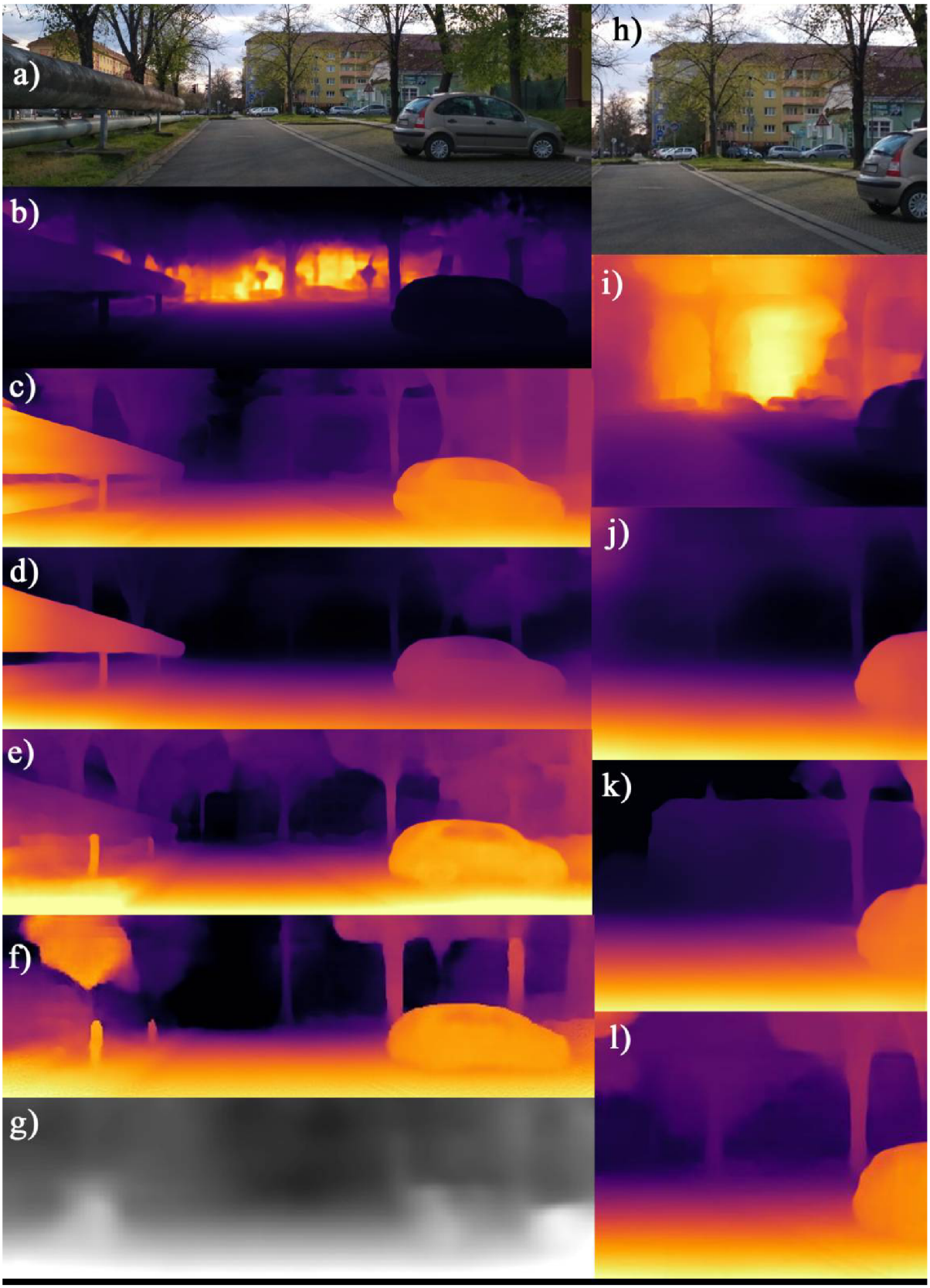
B Srovnání metod MPH



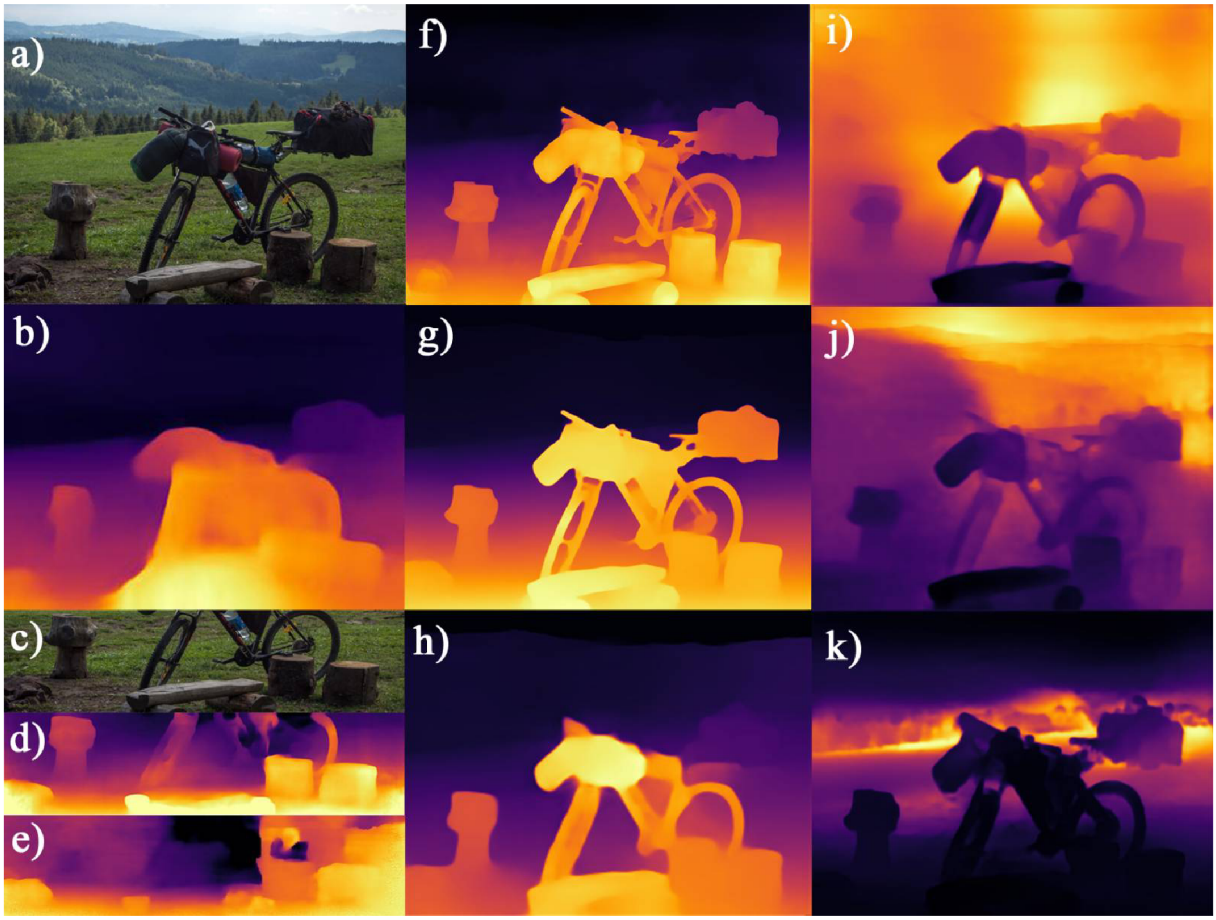
Obrázek B.1: Metody MPH na NYUv2: a) původní obraz, b) BMD, c) SC-SfMLearner, d) výřez (1024 × 320) px, e) *ground-truth*, f) DenseDepth, g) AdaBins NYUv2, h) Monodepth2, i) DPT-hybrid, j) DPT-hybrid NYUv2, k) MiDaSv2.1 small, l) Struct2Depth.



Obrázek B.2: Metody MPH na KITTI: a) původní obraz, b) AdaBins KITTI, c) BMD, d) Monodepth2, e) Struct2Depth, f) výřez (640 × 480) px, g) DenseDepth, h) DPT-hybrid, i) MiDaSv2.1 small, j) SC-SfMLearner.



Obrázek B.3: Metody MPH v dopravním prostředí: a) původní obraz, b) AdaBins KITTI, c) BMD, d) DPT-hybrid, e) Monodepth2, f) Struct2Depth, g) SfMLearner, h) výřez (640 × 480) px, i) DenseDepth, j) DPT-hybrid, k) MiDaSv2.1 small, l) SC-SfMLearner.



Obrázek B.4: Metody MPH v různém prostředí: a) původní obraz, b) SC-SfMLearner, c) výřez (1024×320) px, d) Monodepth2, e) Struct2Depth, f) BMD, g) DPT-hybrid, h) MiDaSv2.1 small, i) DenseDepth, j) AdaBins NYUv2, k) AdaBins KITTI.