FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Sledovací system pro modely aut
Student:	Jan Maček
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Navrhněte a realizujte programové vybavení pro sledování pohybu aut nebo jiných modelů na podlaze laboratoře inteligentních vestavných systémů. Na stropě laboratoře bude umístěna kamera a výkonný vestavný systém kombinující riskový a grafický procesor. Proveďte rešerši vhodných řešení, navrhněte a implementujte programové vybavení, které bude v reálném čase sledovat pohyb modelů aut na podlaze, poskytovat informaci o jejich poloze, orientaci, rychlosti a trajektorii pohybu. Kromě lokalizace modelů na základě jednoduchých značek umožněte sofistikované sledování s využitím hlubokých neuronových sítí, které dovolí obecnou detekci a klasifikaci objektů na podlaze. Uvažujte možnost rychlé adaptace sledovacího systému na nové objekty malým množstvím trénovacích dat s využitím augmentace dat nebo přístupů one-shot learningu. Vyhodnoťte přesnost lokalizace a klasifikace objektů. Výsledky práce řádně zdokumentujte včetně zdrojových kódů. Rozsah práce upřesněte po dohodě s vedoucím práce.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 17. ledna 2022 v Praze.

Bakalářská práce

SLEDOVACÍ SYSTÉM PRO MODELY AUT

Jan Maček

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Miroslav Skrbek, Ph.D.
12. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Miroslavu Skrbkovi, Ph.D za odborné vedení, za pomoc a rady při zpracování této práce a Bc. Peteru Guĺovi za výpomoc s prací v laboratoři inteligentních vestavýnch systémů. Také bych rád poděkoval své rodině a přátelům za dlouhodobou podporu mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

Abstrakt

Tato bakalářská práce se věnuje problému sledování pohybu aut nebo jiných modelů na podlaze laboratoře inteligentních vestavných systémů v reálném čase pomocí statické kamery. Cílem bylo kromě lokalizace modelů na základě jednoduchých značek umožnit sofistikované sledování s využitím hlubokých neuronových sítí s důrazem na rychlou adaptaci sledovacího systému na nové modely. K lokalizaci pomocí značek jsem využil ArUco Markers. K detekci objektů s využitím hlubokých neuronových sítí jsem využil model YOLO. Navrhl a implemenotval jsem způsob automatického označování dat pomocí metody odečítání pozadí, který značně urychlil adaptaci sledovacího systému na nové modely.

Klíčová slova strojové vidění, detekce objektů, sledování objektů, odečítání pozadí, označování dat, YOLO, ArUco

Abstract

This bachelor thesis addresses the problem of tracking the movement of cars or other models on the floor of an intelligent embedded systems laboratory in real time using a static camera. The goal was to enable sophisticated tracking using deep neural networks in addition to model localization based on simple markers, with an emphasis on fast adaptation of the tracking system to new models. I used ArUco Markers for localization using markers. For object detection using deep neural networks, I used the YOLO model. I designed and implemented a method of automatic data labeling using background subtraction, which greatly accelerated the adaptation of the tracking system to the new models.

Keywords computer vision, object detection, object tracking, background subtraction, data labeling, YOLO, ArUco

Seznam zkratek

ROI	Region Of Interest
CAD	Computer Aided Design
stl	Standard Triangle Language
FOV	Field Of View
GPU	Graphics Processing Unit
fps	Frames Per Second
CBOT	Centroid Based Object Tracking
OSOD	One Shot Object Detector

Kapitola 1
Úvod

Detekci objektů s využitím hlubokého učení, jako jednomu z nejzásadnějších a nejnáročnějších problémů počítačového vidění, se v posledních letech dostává velké pozornosti. Každým rokem se přichází s novými metodami jak detekci vylepšit. Problém jak obecně naučit model rychle a přesně s malým množstvím dat, aby podával dobré výsledky, je velice složitý. Počítači nestačí se podívat na jeden obrázek s předmětem, aby následně mohl daný předmět správně detekovat ve všech ostatních obrázcích. K použitelnému detekování objektů stále potřebují dostatečné množství dat k trénování. Což získat může být v některých případech i obtížné, ale největší přítěží je, že se musí v každém jednotlivém obrázku ručně objekty vyznačit, aby se je počítač mohl naučit detekovat.

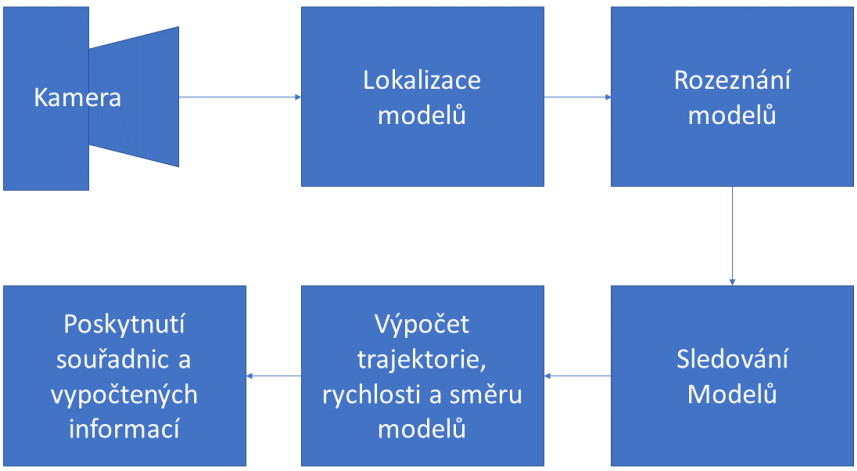
Předmětem práce je navrhnutí sledovacího systému pro pohyb aut nebo jiných modelů na podlaze laboratoře vestavných inteligentních systémů v reálném čase, přičemž se zabývám rychlou adaptací sledovacího systému na nové objekty, takže časově náročnému manuálnímu označování dat se budu snažit předejít.

Laboratoř se nachází v místnosti THA:1054 a plocha, po které se modely aut budou pohybovat, je velká přibližně 6×3 metry. Kamera, pomocí které se objekty budou sledovat, se bude nacházet uprostřed této plochy, připevněna ke stropu místnosti pohledem kolmo dolů. V laboratoři jsou aktuálně 4 identická pohyblivá auta, která jsou veliká 25×14 centimetrů a dokáží jet rychlostí až 60 cm/s.

Kapitola 2
Cíle práce

Na podlaze laboratoře inteligentních vestavných systémů se budou pohybovat modely aut. Cílem je pomocí kamery, která bude umístěna ke stropu laboratoře, jednotlivé modely v obraze lokalizovat, rozeznat je od sebe a následně je po obraze sledovat a pro každý vypočítat a poskytnout trajektorii, rychlost a směr jízdy.

Výsledný proces by měl vypadat následovně:



Obrázek 2.1 Cíl práce

Jelikož v laboratoři zatím žádná kamera připevněna ke stropu není, jedním z cílů práce je také zvolení vhodné kamery a její instalace.

S využitím zvolené kamery budu následně muset nějakým způsobem jednotlivé modely aut v obraze lokalizovat. Po podlaze se ale může pohybovat více různých typů modelů aut a proto, abych dokázal při následném poskytování informací určit, ke kterému typu modelu se informace vážou, musím jednotlivé typy modelů od sebe dokázat rozeznat. Po podlaze se také ale může pohybovat více modelů aut stejného typu a vzhledu. K tomu abych vždy dokázal určit, o který konkrétní model se jedná, budu je muset nějakým způsobem unikátně identifikovat a po celou dobu, co se budou po podlaze pohybovat, je sledovat a udržovat si jejich unikátní přiřazení. Po celou dobu sledování se také bude muset vypočítávat jejich rychlost a směr jízdy. Současně se

všechny informace musí nějakým způsobem poskytovat.

Celý tento proces musí ještě k tomu běžet v reálném čase. To znamená, že s použitím kamery, které mají pro živé video snímací rychlost 30 snímků za vteřinu, se bude muset pro jeden snímek celý tento proces stihnout provést rychleji, než kamera předá další snímek ke zpracování.

Velice důležitým cílem této práce je také navržení způsobu, jak umožnit přidání nového typu modelu do tohoto systému za nejkratší dobu, aby se mohl co nejdříve začít sledovat.

Kapitola 3

Metody lokalizace objektů

V této kapitole rozeberu různé možnosti, jak v obraze lokalizovat a rozeznat hledané objekty.

3.1 Lokalizace pomocí značek

Jeden způsob, jak dosáhnout lokalizace objektů, je s využitím jednoduchých značek. ArUco (Augmented Reality library from the University of Cordoba) Markers[1, 2, 3] jsou binární čtvercové značky určené přímo pro lokalizaci ve strojovém vidění. Na první pohled mohou připomínat známější QR kódy, ale je v nich značný rozdíl. Oproti QR kódům, které jsou určené k uchování většímu množství dat, ArUco dokáže zakódovat pouze jedno celé nezáporné číslo v rozsahu 0 až 999, ale navíc lze jednoduše získat i vzdálenost nebo orientaci značky.

Obrázek 3.1 Porovnání QR kódu a ArUco značky

ArUco se často používá například pro navigaci autonomních robotů vybavených kamerou. Vytištěné značky se připevní po stěnách či podlaze a robot je pomocí svojí kamery detekuje. Rychle a přesně tím zjistí tím vlastní pozici vůči okolí a může podle toho upravit svůj směr jízdy.

V mém případě ale můžu ArUco využít tak, že značky připevním na modely aut, které chci sledovat. Mohl bych potom pomocí kamery detekovat jejich polohu, a protože by každá značka byla unikátní, dokázal bych auta od sebe rozeznat a nepotřeboval bych ani jakýkoliv algoritmus na sledování.

Generování

Při vytváření značek se prvně musí určit z jakého slovníku se budou vybírat. Slovníkem se zde myslí seznam binárních kodifikací jednotlivých značek. Nejdůležitějšími parametry slovníku jsou velikost slovníku, který udává počet značek ve slovníku a velikost značky, který určuje rozměr matice značky v počtu bitů.[4] Například značka 3.1 je z největšího možného slovníku o velikosti 1000 (lze tedy vygenerovat ID 0 až 999) s velikostí značky 7×7.

Na velikosti záleží. *„The more bits, the more words in the dictionary, and the smaller the chance of confusion. However, more bits means that more resolution is required for correct detection.“*[5] Nutno dodat, že binární matice značek o konkrétní velikosti se s různými velikostmi slovníku nemění. Při detekci má algoritmus pouze na výběr z menší množiny a tím také zmenšuje pravděpodobnost záměny.

Aby proces detekce byl schopen nalézt značku i otočenou – a určit její původní natočení – je každý roh jednoznačně identifikován. Tohle je rovněž součástí binární kodifikace.[4]

K vytváření značek je možné využít online generátory nebo si napsat vlastní skript pomocí knihovny OpenCV[6], což je nejpopulárnější open source knihovna pro strojové vidění.

Detekce

Proces detekce má pro daný obrázek za úkol vrátit list všech detekovaných značek. Každý detekovaná značka obsahuje:

- souřadnice všech čtyř rohů (v původním pořadí)
- ID značky

Algoritmus pro detekci se skládá z následujících kroků:

1. Detekce kandidátů. V tomto kroku je obraz analyzován s cílem najít čtvercové tvary. Z obrázku se extrahují obrysy a ty, které nejsou konvexní nebo se nepřibližují čtvercovému tvaru, se vyřadí.

2. Po detekci kandidátů je třeba analýzou jejich vnitřní kódování určit, zda se skutečně jedná o značky. Tento krok začíná extrakcí bitů každé značky. K tomu se nejprve použije perspektivní transformace, aby se získala značka v její kanonické podobě. Obraz je následně rozdělen do různých buněk podle dopředu určené velikosti značky. Poté se pro každou buňku určí, zda se jedná o bílý nebo černý bit. Nakonec se bity analyzují, aby se určilo, zda značka patří do konkrétního slovníku.[4]

3.2 Detekce objektů

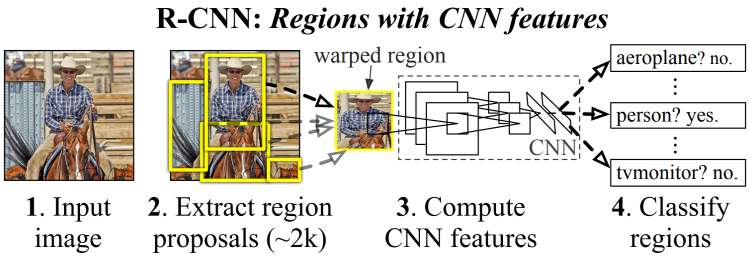
Detekce objektů (object detection) je technika strojového vidění, která umožňuje v obrázku lokalizovat a identifikovat objekty obvykle s využitím hlubokého učení. Tedy oproti například klasifikačnímu problému, kde je za úkol pouze určit jaký objekt se na snímku nachází, jsou zde úkoly dva. A to prvně v obrázku najít všechny hledané objekty a pomocí ohraničujících rámečků (bounding boxů) je označit, a následně každý klasifikovat.

Object detection modelu, který využívá hlubokou neuronovou síť, může proces detekce jednoho snímku trvat delší dobu, než například lokalizaci s využitím značek. V této práci se zabývám konkrétně detekcí objektů v reálném čase. Což znamená, že je potřeba, aby algoritmus dokázal zpracovat alespoň 30 snímků za jednu vteřinu.

3.2.1 YOLO

YOLO (You Only Look Once)[7, 8, 9, 10] je nejmodernější algoritmus detekce objektů v reálném čase. K porozumění, proč je YOLO tak inovativní, je důležité se prvně podívat jakým způsobem se detekce objektů provádí u jiných algoritmů.

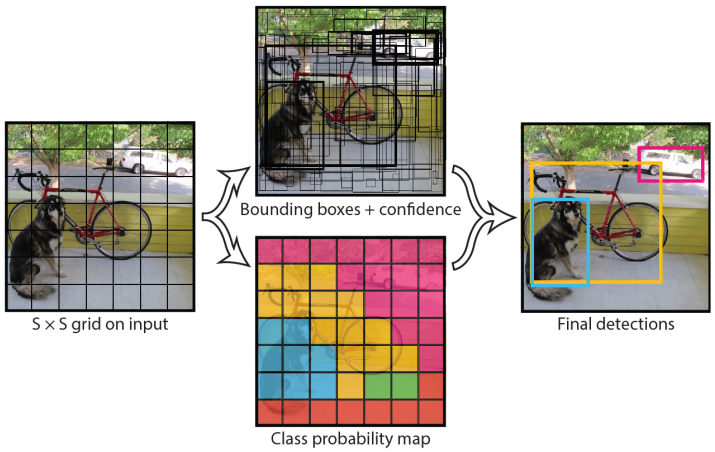
Proces detekce objektů se skládá ze dvou úkolů – lokalizace a klasifikování objektu. A většina object detection modelů se k tomu také tak postavila – jako první model predikuje regiony, kde by se objekty mohly nacházet a následně provede klasifikaci. Nejpopulárnější model, který detekuje objekty tímto způsobem (tzv. two stage detector) je R-CNN (Region-based Convolutional Neural Network)[11] a jeho vylepšení Fast R-CNN[12] a následné Faster R-CNN[13].



Obrázek 3.2 Model R-CNN, zdroj: [11]

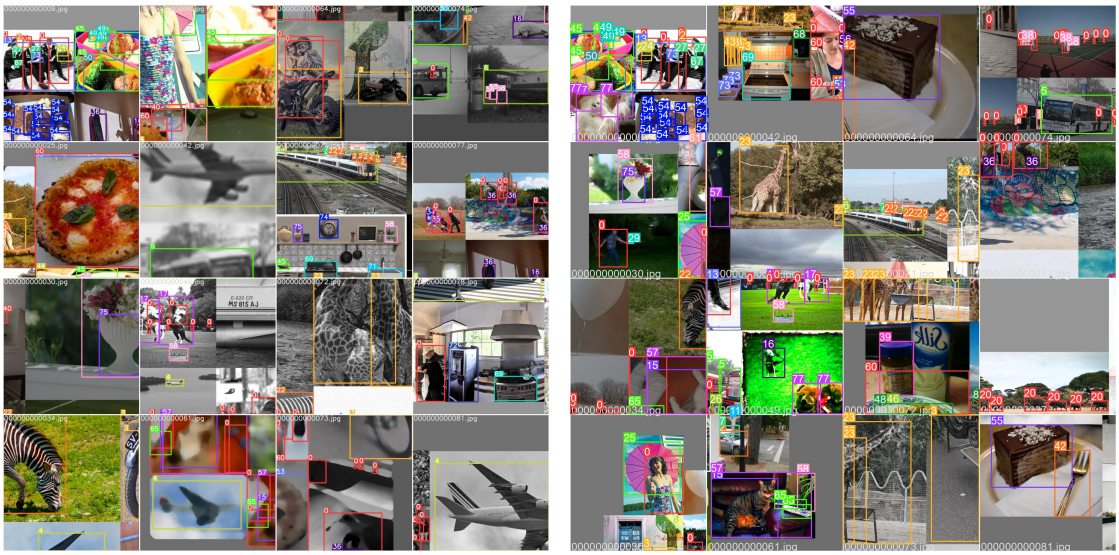
Faster R-CNN, i přes velké zrychlení, nedosahuje rychlosti dostačující k detekci v reálném čase. V [14] se uvádí maximální dosažené fps 17. V dalším porovnání object detection modelů [15] dosáhlo 22 fps s použitím GPU, které je z pohledu výkonnosti řádově lepší, než které bude využito pro tuto práci.

Oproti tomu YOLO, jak může název napovědět, zpracuje celý snímek pouze na jeden průchod. Pomocí jediné neuronové sítě provede predikci bounding boxů i klasifikaci. Díky tomu dokáže detekovat objekty rychleji jak v reálném čase. Základní myšlenka modelu YOLO je rozdělit obrázek na čtvercovou mřížku. Každá buňka mřížky předpovídá několik bounding boxů a důvěryhodnost (confidence) pro tyto boxy. Každá buňka také predikuje pravděpodobnosti tříd. Všechny tyto predikce použije k výsledné detekci.[7]



Obrázek 3.3 Model YOLO, zdroj: [7]

Další vylepšení, se kterým později YOLO přišlo, je mozaiková augmentace dat. Při augmentaci se vezmou 4 obrázky z datasetu a náhodně se promixují s tím, že jim zůstanou původní ohraničené rámečky. To umožňuje detekci objektů mimo jejich normální kontext a značně to vylepšuje výsledky trénování.[10]



Obrázek 3.4 Ukázka augmentace dat pomocí mozaik, zdroj: [16]

3.2.2 One/Few-Shot learning

Jedním z cílů této práce je rychlé nasazení nových objektů do modelu. Object detection modely, jako je například YOLO, potřebují na trénování spoustu obrázků. A ty se musí nejdřív nějakým způsobem získat a následně označit, což může být někdy obtížné a časově náročné. Cílem One-shot (nebo Few-shot) object detection modelů je právě detekce objektů s využitím pouze jednoho (nebo pár) obrázků. Existuje více různých přístupů, které se tento problému snaží vyřešit.

Frustratingly Simple Few-Shot Object Detection

Frustratingly Simple Few-Shot Object Detection je nejnovější metoda Few-Shot learnigu pro modely detekce objektů. Obecnou myšlenkou Few-Shot learningu je rozdělení tříd na základní (base classes) a nové (novel classes), kde základní třídy mají mnoho označených dat a nové jenom pár (few). Cílem je přenést znalosti naučené na základních třídách s velkým množstvím vzorků na nedostatečně zastoupené nové třídy, aby byl model schopen efektivně detekovat objekty patřící do nových tříd, i když během trénování viděl pouze pár případů.[17]

Metoda využívá tzv. Two-stage Fine-tuning Approach (TFA). Model se prvně natrénuje na základním datasetu s velkým množstvím dat a s využitím Feature extraction[17]. Ve druhé fázi se dotrénuje už jenom s využitím pár obrázků z nové a základní třídy. Oproti klasickým metodám detekcí objektů samozřejmě nedosahují tak velké přesnosti, ale největším problémem je využívání Two-Stage Detectorů, které nedokáží detekovat objekty v reálném čase.

One Shot Object Detection s využitím augmentace dat

Dalším možným přístupem k trénování modelů s malým množstvím je pomocí augmentace dat, kde se s využitím jednoho obrázku objektu různými úpravami vytvoří celý dataset. Jeden takový One-shot object detektor (OSOD) je z knihovny Turi Create[18] od firmy Apple. Knihovna

Turi Create nepřichází v ohledu detekce či trénování s žádnou inovací. K tomu používá vlastní implementaci algoritmu YOLO. Jediné co přináší je generování syntetických trénovacích dat. Syntetické obrázky se generují přidáním původního obrázku na obrázky ze sady pozadí (background images), s použitím rotací (yaw, pitch a roll), posunu a změny velikosti.

Tento OSOD je vhodný pro dvourozměrné objekty, které mají určitou pravidelnost.[18] Například pro detekci osobních aut není možné si s jedním obrázkem vystačit. Každé auto má jinou barvu, tvar a je snímáno z jiné strany. Oproti tomu například dopravní značka se vyskytuje vždy ve stejné podobě, a proto může k natrénování stačit pouze jeden obrázek.

Obrázek 3.5 Příklad augmentace dat, zdroj: [18]

Touto metodou je tedy možné, s použitím jediného obrázku, získat dataset, který obsahuje stovky správně označených obrázků a tím značně urychlit proces přidávání nového objektu do modelu.

V této práci sice sleduji modely aut, které jsou trojrozměrné, ale pomocí statické kamery, která má pohled kolmo dolů. Auta jsou tedy v záběru vidět pouze ze shora, takže teoreticky by se tato metoda dala využít.

3.2.3 Důležité pojmy

Pro vyhodnocení object detection modelů se používá spousta metrik. Oproti klasifikačním problémům, zde přibývají i funkce, které mají za úkol ohodnotit správnost či přesnost ohraničujících rámečků detekovaných objektů. V této kapitole vysvětlím všechny pojmy, které jsou důležité k porozumění výsledků těchto funkcí.

Přesnost (Precision) měří z kolika procent byly predikce správné.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

TP (True Positives) je počet správně predikovaných (tam kde bounding box má být, tam byl označen).

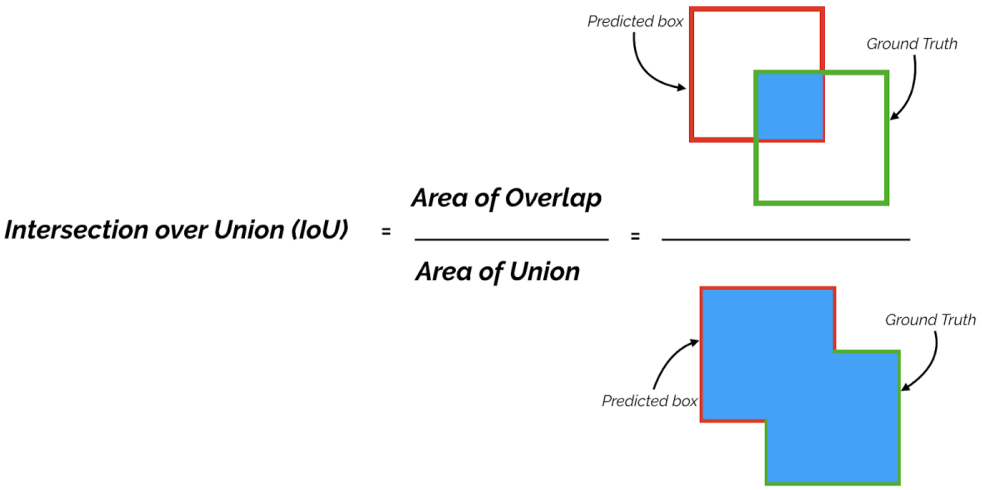
FP (False Positives) je počet špatně predikovaných, kdy model označil pozitivně místo kde žádný box být nemá.

Výtěžnost (Recall) měří z kolika procent model našel všechny ohraničující rámečky, které by správně měl najít.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

FN (False Negatives) je počet výskytů, kdy model špatně neoznačil místo, které by správně označeno být mělo.[19]

IoU (Intersection over Union) slouží k vyhodnocení predikce. Počítá se jako podíl průniku a sjednocení obsahů bounding boxů správného a predikovaného objektu.



Obrázek 3.6 IoU, zdroj: [19]

Následně se určí prahová hodnota (threshold), která rozhodne správnost predikce.

Pokud IoU $\geq$ threshold, detekce se vyhodnotí jako True Positive (TP).

Pokud IoU $<$ threshold, detekce se vyhodnotí jako False Positive (FP).

V případě, kdy v obrázku bude bounding box, který model nedokázal predikovat, vyhodnotí se jako False Negative (FN).

mAP (mean average precision) je metrika používaná k vyhodnocení modelu detekce objektů. *„When a model has high recall but low precision, then the model classifies most of the positive samples correctly but it has many false positives (i.e. classifies many Negative samples as Positive). When a model has high precision but low recall, then the model is accurate when it classifies a sample as Positive but it may classify only some of the positive samples.“*[20] A k tomu, aby se predikování modelu dalo lépe vyhodnotit, slouží právě mAP.

Počítá se přes jednotlivé třídy s několika různými hodnotami prahové hranice.

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k, \tag{3.3}$$

kde AP_k je AP (average precision) pro jednotlivou třídu k a n je počet tříd.

$$P = \sum_{k=0}^{n-1}(Recall(k) - Recall(k+1)) \cdot Precision(k)$$

$$Recall(n) = 0, Precision(n) = 1, \tag{3.4}$$

kde n je počet prahových hranic.[20]

Účelové funkce (loss function) slouží object detecion modelům k učení. Jednoduše řečeno dostávají zpětnou vazbu svých predikcí v podobě hodnoty „chybovosti“, kterou se snaží co nejvíce snížit. Různé architektury mohou používat jiné účelové funkce i rozdílné způsoby výpočtu těchto funkcí. Obecně jsou ale dvě, které využívají všechny.

box loss měří jak blízko k sobě jsou ohraničující rámečky predikovaného a správného objektu.

cls loss (classification loss) určuje správnost klasifikace detekovaných objektů.

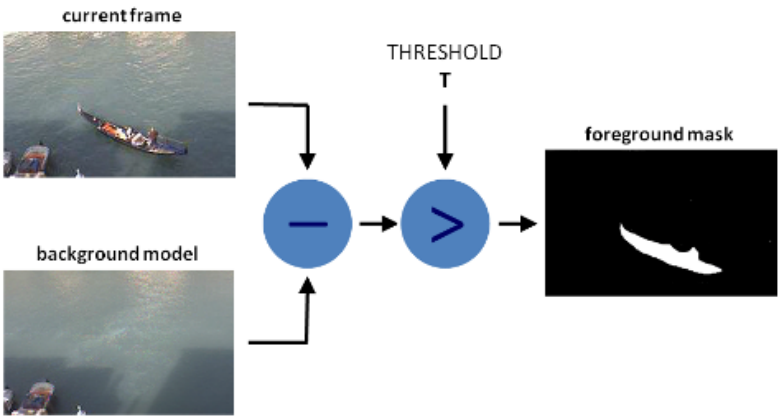
Například architektura YOLO používá navíc ještě třetí účelovou funkci.

obj loss (objectness loss) říká s jakou jistotou se v predikovaném bounding boxu objekt nachází.[21]

3.3 Odečítání pozadí

Další způsob, jak lokalizovat objekty, je pomocí odečítání pozadí (background subtraction). Tato metoda slouží k detekci pohybujících se objektů ve videu a funguje pouze s použitím statické kamery. Touto metodou je tedy možné objekty pouze lokalizovat, k následnému rozeznání jednotlivých objektů by se muselo využít nějakého klasifikátoru.

Metoda odečítání pozadí je založena na principu sledování změn hodnot pixelů v čase. Výsledkem pro každý snímek ve videu je tzv. maska popředí (foreground mask), což je binární maska obsahující bílé pixely patřící pohybujícím se objektům a černé pixely jako pozadí. Algoritmus vypočítá foreground masku odečtením aktuálního snímku od modelu pozadí, který obsahuje statickou část scény. Pixely, kde je hodnota rozdílu větší než prahová hranice (threshold), která se předem určila, se považují za pohybující se objekt. Modelování pozadí se skládá ze 2 kroků – inicializace a aktualizace. V prvním kroku je vypočítán počáteční model pozadí a ve druhém je tento model aktualizován, aby se přizpůsobil možným změnám ve scéně.[22]



Obrázek 3.7 Proces background subtraction, zdroj: [22]

Přestože se v této práci zabývám sledováním pohybujících se modelů aut pomocí statické kamery, v tomto případě tato metoda pro lokalizaci objektů není vhodná. Důvody jsou následující:

- Pokud auto stojí na místě, není možné jej detekovat;
- V případě, kdy se auto bude dotýkat jiného pohybujícího se objektu, bílé pixely ve výsledné binární masce budou tvořit jednu konturu a nebude možné je od sebe oddělit. Jejich rozeznání je opět problém lokalizace objektů ve snímku, pouze na menším ROI (region of interest).

Kapitola 4
Metody sledování objektů

Sledování objektů (object tracking) je proces strojového vidění, při kterém se vezmou detekované objekty a pro každý se vytvoří unikátní identifikátor. Následně pokračuje sledování objektů při jejich pohybu po snímcích ve videu, přičemž se zachovává přiřazení jedinečných ID.[23] V následujících sekcích popíšu různé metody jak sledování objektů docílit.

4.1 Centroid based object tracking

Centroid based object tracking (CBOT) je jednoduchý, ale velice účinný algoritmus. Funguje na základě měření Euklidovské vzdálenosti detekovaných objektů mezi jednotlivými snímky. Algoritmus se dělí do následujících kroků:

1. Algoritmus pro každý snímek dostane na vstup ohraničující rámečky objektů a prvním krokem je vypočítat jejich geometrický střed.
2. Následně spočítá Euklidovskou vzdálenost mezi geometrickým středem nových objektů a již existujících objektů (získaných z předchozího snímku viz 4. krok).
3. Pokud je nový objekt dostatečně blízko některému z existujících, považuje se za ten samý a pouze se tomu existujícímu aktualizuje souřadnice.
4. V případě, kdy nový objekt není poblíž žádnému z existujících, musí se zaregistrovat. Myšleno, že se mu přiřadí nový unikátní identifikátor a s jeho souřadnicemi se uloží do seznamu existujících.
5. Zkontroluje se, zda by se neměl některý z existujících objektů smazat. To nastane v případě, kdy nebyl poblíž žádnému novému objektu. Znamená to, že například objekt opustil snímek a takový už nadále sledovat nechceme.[24]

Kvůli své jednoduchosti má ale i pár nevýhod. Aby sledování fungovalo správně, objekty mezi jednotlivými snímky se musí vždy nacházet blízko sebe. Tento předpoklad obvykle platí, ale problém může nastat v případě, kdy se objekt překryje s jiným objektem. V té chvíli by se mohly zaměnit jejich identifikátory. Tahle chyba ale v mém případě nemůže nastat. Objekty sleduji pomocí statické kamery pohledem kolmo dolů, takže není možné aby jedno auto bylo zakryté druhým. Také je důležité dodat, že tenhle problém není specifický pro sledování pomocí Euklidovské vzdálenosti. Stává se to i u mnoha jiných metod, včetně pokročilých.[24]

Přestože bych si teoreticky s metodou Centroid based object tracking měl vystačit, jsou i pokročilejší způsoby, které bych mohl využít. Existují algoritmy sledování objektů založené na

hlubokém učení, které se snaží vypořádat s problémem prohození identifikátorů. Jeden z nejpoužívanějších modelů je DeepSORT[25], který si popularitu získal hlavně díky svojí rychlosti. Ačkoli je oproti ostatním algoritmům založených na hlubokém učení ten nejrychlejší, tak požadovaných 30 fps nedosáhne[26]. DeepSORT, jak sám název napovídá, je rozšířením jednoduššího algoritmu SORT[27], který žádné hluboké učení nepoužívá, ale lze použít v reálném čase.[28]

4.2 SORT

SORT (Simple Online and Realtime Tracking) je architektura sledování objektů založená na základních technikách sdružování dat a odhadu stavu. Využívají se zde dvě klasické metody – Kalmanův filtr[29] a Maďarská metoda[30].[27] Přestože tento algoritmus problém s prohozením ID neřeší, je velice populární díky své rychlosti a slouží jako základ pro vývoj dalších sledovacích algoritmů.[31]

Kalmanův filtr je algoritmus, který poskytuje odhady některých neznámých proměnných na základě měření pozorovaných v čase. Slouží tedy k nalezení optimálního váženého průměru pro kombinaci dat z více zdrojů a je založen na bayesovské statistice.[29] Maďarská metoda je efektivní algoritmus jak řešit přiřazovací problémy. Přiřazovací problém (assignment problem) je problém kombinatorické optimalizace, kde cílem je najít optimální přiřazení pro danou matici „nákladů“ (cost matrix).[30]

Algoritmus SORT, poté co se detekují objekty, funguje následovně:

1. Predikce polohy objektů (detekovaných v aktuálním snímku) na následujícím snímku. Tento posun objektů v po sobě jdoucích snímcích je odhadován pomocí modelu lineární konstantní rychlosti. *„When a detection is associated to a target, the detected bounding box is used to update the target state where the velocity components are solved optimally via a Kalman filter framework. If no detection is associated to the target, its state is simply predicted without correction using the linear velocity model.“*[27]

2. Přiřazování predikcí k existujícím cílům. Jako první se vypočítá matice, která se skládá z IoU (Intersetion over Union) mezi každou detekcí a všemi předpovězenými bounding boxy. Přiřazení se následně řeší optimálně pomocí Maďarského algoritmu. Navíc je stanovena prahová hranice IoU_{min} pro odmítnutí přiřazení, kde je překrytí detekce k cíli menší než IoU_{min}.

3. Vytváření a mazání identifikátorů. V momentě kdy se v obrázku objeví nový objekt, je třeba mu přiřadit ID. Objektu se přiřadí nové ID v okamžiku, kdy má překrytí s ostatními detekcemi menší než IOU_{min}. Sledování je ukončeno, pokud objekt již není detekován.[27]

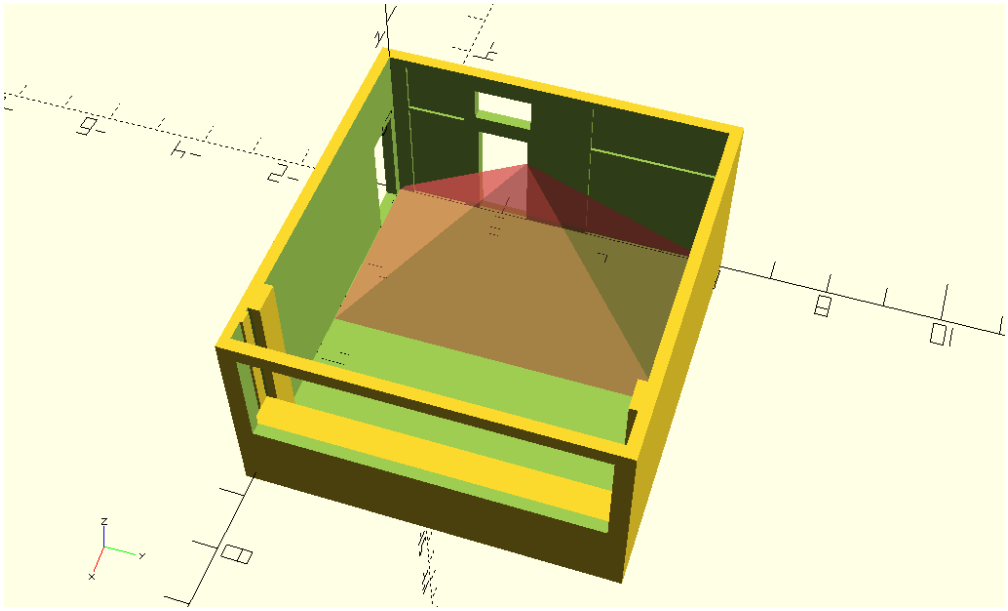
Kapitola 5

Návrh řešení

V této kapitole je popsán výběr kamery a následně dva návrhy řešení této práce.

5.1 Výběr kamery

Pro tuto práci je důležitý i výběr kamery. Kamera bude umístěna na stropě laboratoře ve výšce přibližně 314 cm. Je tedy potřeba, aby měla dostatečné rozlišení, protože modely aut jsou velké jenom 25 × 14 centimetrů. Také záleží na velikosti zorného pole (FOV). Ideálně by kamera měla pokrýt co největší plochu, ale zároveň je zbytečné aby zabírala okolní stěny místnosti. Ideální FOV si můžu vypočítat. Kamera bude situována uprostřed první poloviny laboratoře u dveří, kde je největší prázdná plocha podlahy. Šířka místnosti je 588 cm a předpokládá se, že kamera bude mít standardní poměr stran (aspect ratio) 16:9.



Obrázek 5.1 Ideální FOV kamery

Zjistil jsem, že u různých kamer se liší, jaké FOV uvádí. Může být:

- diagonální (dFOV)
- horizontální (hFOV)
- vertikální (vFOV)

Nejčastěji se udává diagonální nebo horizontální. Vypočítal jsem tedy obě dvě.

$$dFOV = 2 \cdot \arctan \frac{\sqrt{l^2 + w^2}}{2 \cdot h} \cdot \frac{180}{\pi} \tag{5.1}$$

$$hFOV = 2 \cdot \arctan \frac{l}{2 \cdot h} \cdot \frac{180}{\pi} \tag{5.2}$$

kde l je délka, w je šířka a h je výška jehlanu zorného pole.

$$FOV = 2 \cdot \arctan \frac{588}{2 \cdot 314} \cdot \frac{180}{\pi} \approx 86° \tag{5.3}$$

Pro výpočet dFOV je potřeba zjistit w. Využil jsem pro to předpokládaný pomšr stran kamery.

$$w = \frac{l \cdot 9}{16} = \frac{588 \cdot 9}{16} = 330,75\ cm \tag{5.4}$$

$$FOV = 2 \cdot \arctan \frac{\sqrt{588^2 + 330,75^2}}{2 \cdot 314} \cdot \frac{180}{\pi} \approx 94° \tag{5.5}$$

Ideální kamera by tedy měla mít následující parametry:

- alespoň Full HD rozlišení (1920×1080)
- horizontální FOV nejméně 86°
- připojení přes USB-A (nutné pro zapojení k PC)

V laboratoři je přístupná kamera s dFOV 78°, ale to je nedostatečné. Velká spousta kamer se vyrábí s 90° FOV, ale někdy je složitější dohledat, zda je to horizontálně, nebo diagonálně. Ideální by byla s hFOV 90° a takové i existují, ale bohužel nebyla žádná dostupná na českém trhu a musela by se posílat až z Číny, ale to by mohlo trvat týdny než dorazí. Pořídila se tedy webkamera Logitech c930e, která splňuje všechny požadavky s jedinou výjimkou. Stejně jako většina webkamer nemá 90° horizontálně, ale diagonálně, což odpovídá přibližně hFOV 82°. Ale není to problém. Je to jenom o 4° méně a na jedné straně místnosti je u stěny stůl, který v záběru není potřeba celý, takže když jsem kameru posunul přibližně o 26 cm opačným směrem, bez problému pokryje na šířku celou požadovanou plochu.

Obrázek 5.2 Kamera Logitech c930e připevněna ke stropu laboratoře THA:1054

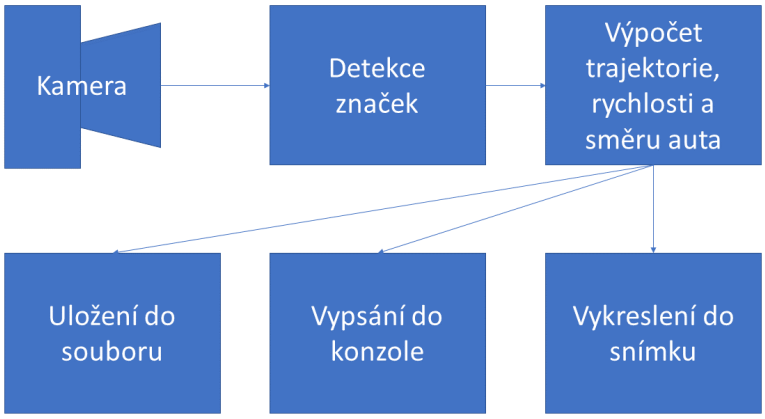
5.2 Řešení s využitím značek

Prvním řešením je využití ArUco značek. Lokalizace, rozeznání a sledování modelů se pomocí ArUco značek zajistí v jediném kroku. Je to díky tomu, že každý model auta by na sobě měl připevněnou unikátní značku, takže se žádné rozeznávání a následovné sledování nemusí řešit. K tomu se ale značky musí prvně připravit.

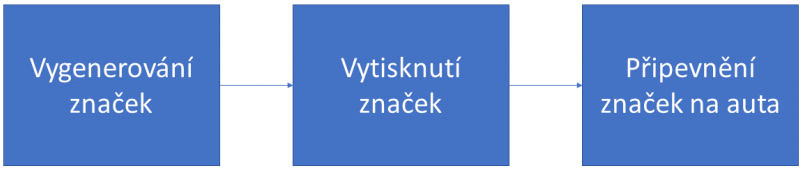
V knihovně OpenCV[6] je připraven modul pro generování i detekci s názvem aruco[1]. A prvním krokem, který je třeba zvážit, je výběr slovníku. V OpenCV jsou již předdefinované s velikostmi značek od 4×4 po 7×7 a velikostmi slovníku 50 až 1000. Jelikož je kamera umístěna ve výšce až 3,14 m a rozlišení má „pouze“ 1080p, vybral jsem značky o velikosti 4×4. A protože objektů k detekci nebude mnoho a chceme předejít možným záměnám, využil jsem ten nejmenší slovník, který byl možný – 50.

Vygenerované značky pomocí OpenCV je následně potřeba vytisknout a připevnit k objektům, které chceme sledovat. Zde je samozřejmě důležitý i rozměr vytištěné značky. Čím větší značka, tím lépe bude detekovatelný. Ale nemůže být zase příliš velká, například nemůže být větší jak předmět, ke kterému bude připevněna. A důležité také je, aby se dala připevnit k co největšímu množství různých objektů. Tudíž v tomhle ohledu zase platí čím menší, tím lepší. Snažil jsem se tedy najít co nejmenší rozměr značky, při které bude detekce fungovat stále spolehlivě. Jako ideální velikost jsem vyhodnotil 10 cm (viz 8.4).

[1] `https://docs.opencv.org/4.x/d9/d6a/group__aruco.html`



Obrázek 5.3 Návrh řešení s využitím značek



Obrázek 5.4 Postup připravení značek k detekci

Při detekci získám souřadnice každého rohu ve směru hodinových ručiček počínaje levým horním rohem. Tuto vlastnost jsem využil k přesnému výpočtu směru objektů. Výsledek bude ve stupních od 0° do 360° (stejně jako na jednotkové kružnici), kde objekt bude vždy uprostřed pomyslné kružnice. Jelikož obrázky mají souřadnici [0,0] v levém horním rohu, můžeme si je představit jako 4. kvadrant kartézské soustavy souřadnic, kde akorát místo záporného y máme kladné. Tudíž pro správný výpočet je potřeba vynásobit y souřadnice -1.

$$\begin{aligned} A &= [x_1, y_1] \\ B &= [x_2, y_2] \\ \alpha &= \arctan \frac{-(y_1 - y_2)}{x_1 - x_2} \cdot \frac{180}{\pi} \mod 360, \end{aligned} \tag{5.6}$$

kde α je výsledný úhel, A je souřadnice levého horního a B levého spodního rohu značky.

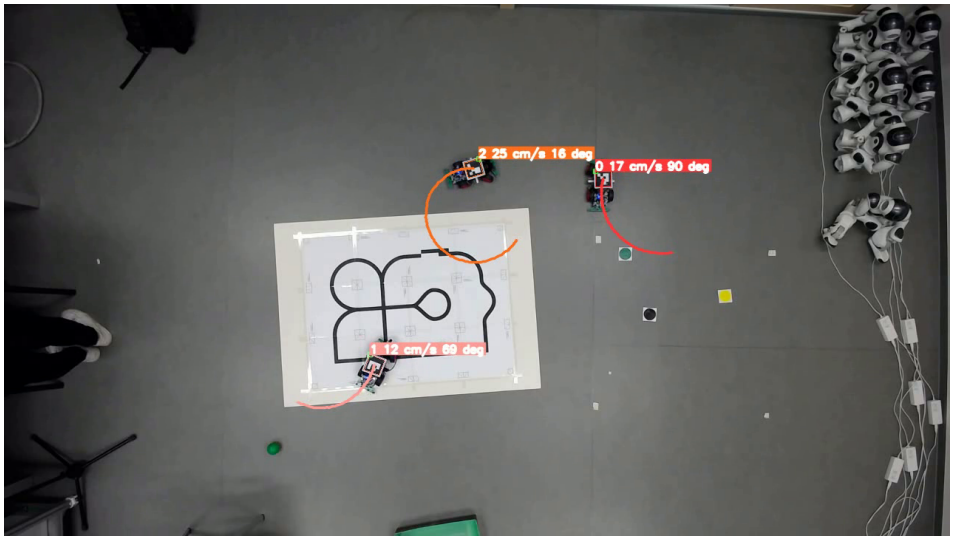
Pro výpočet rychlosti jsem využil uraženou vzdálenost mezi jednotlivými snímky. Vzdálenost jsem počítal mezi středy značek.

$$v = \frac{d \cdot fps}{ppm}, \tag{5.7}$$

kde v je výsledná rychlost, fps jsou snímky za vteřinu a ppm pixely na metr.

Poskytování informací bude podobou grafického znázornění ve videu, výpis do konzole a uložení do souboru. Pro vykreslení trajektorie si ukládám posledních 100 detekcí a jednotlivě pro každé ID zvlášť nakreslím čáru mezi středy značek každých 2 po sobě jdoucích detekcí.

U každého objektu se vykresluje více hodnot. První číslo značí ID značky, druhá hodnota je vypočítaná rychlost auta v centimetrech za vteřinu a poslední je směr auta ve stupních. Barevná čára za každým autem je jejich trajektorie.



Obrázek 5.5 Ukázka detekce pomocí ArUco

Výhody řešení s využitím značek ArUco:

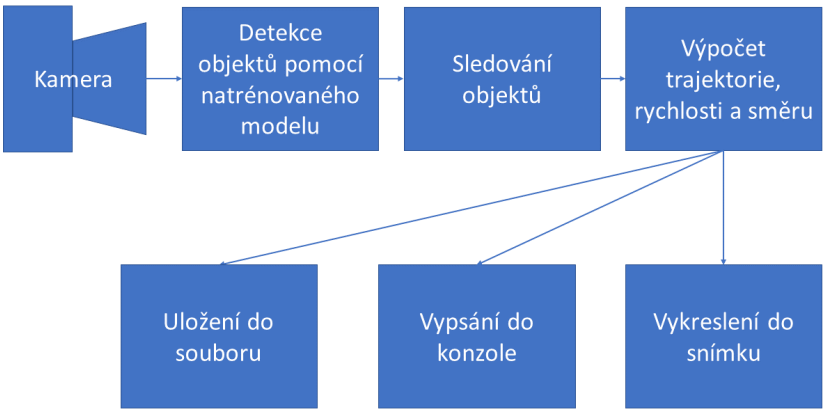
- přesnost
- rychlost
- není potřeba učit neuronovou síť
- nejsou potřeba žádná data
- známe přesnou orientaci značky

Nevýhody:

- musí se vygenerovat, vytisknout a nějakým způsobem značka připevnit ke každému objektu, který chceme sledovat
- značka musí být vždy celá viditelná, jakékoli minimální zakrytí způsobí nefunkčnost detekce

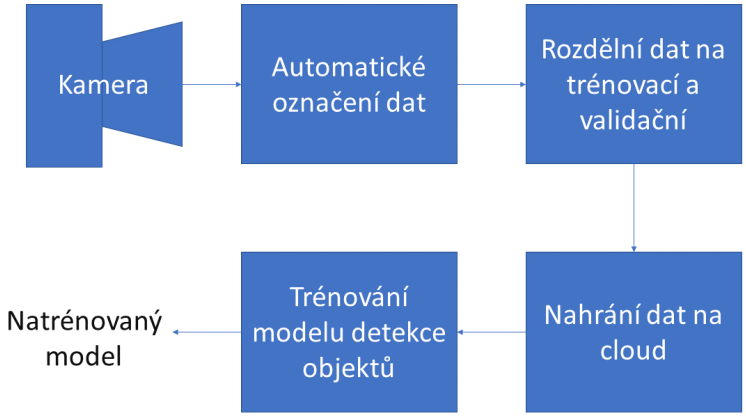
5.3 Řešení s využitím detekce a sledování objektů

Tento návrh řešení využívá detekci objektů k lokalizaci a rozeznání jednotlivých modelů aut a následně sledování objektů pro sledování aut.



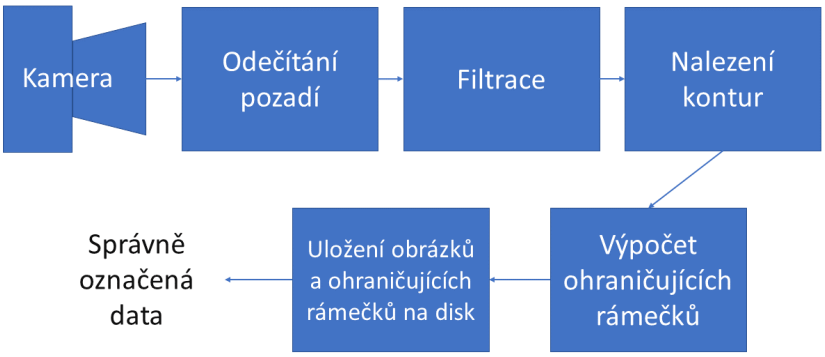
Obrázek 5.6 Návrh řešení s využitím detekce a sledování objektů

Pro detekci objektů se využije model YOLO, který se musí prvně natrénovat na označených datech. To proběhne následovně:



Obrázek 5.7 Postup trénování modelu detekce objektů

Aby trénování bylo co nejrychlejší, využije se cloudového prostředí Colab[2], kde je nabízena zdarma hardwarová akcelerace s vysoce výkonným grafickým procesorem NVIDIA Tesla T4. Data, na kterých se bude model trénovat, se musí nějakým způsobem na Colab odeslat. K tomu jsem zvolil platformu Weights & Biases[32], která mimo jiné slouží k automatickému verzování datasetů, vyhodnocování výkonnosti modelů či vizualizaci výsledků. Data se tedy odešlou na W&B a následně použitím připravené Python knihovny se v prostředí Colab budou moct načíst pro trénování modelu. YOLO potřebuje k natrénování dostatečné množství dat. Získat a označit data zabere značnou dobu procesu. Manuálně vyznačit ohraničující rámeček pro každý objekt ve stovkách záznamů může trvat desítky minut nebo i pár hodin. Navrhl jsem tedy způsob jak automaticky označit data. Myšleno, že pro každý snímek algoritmus najde konkrétní objekt a určí jeho bounding box. Proces je následovný:



Obrázek 5.8 Postup automatického označování dat

Pro získání a automatické označení dat se využije tedy videa z kamery. Funguje to tak, že pohybující se objekty ve videu s využitím metody odečítání pozadí lokalizuji, provedu filtraci a pomocí nalezených kontur vypočítám jejich ohraničující rámečky. Následně každý obrázek s jeho příslušným textovým souborem, ve které budou zapsány ohraničující rámečky se uloží na disk.

K tomu, aby označení dat proběhlo správně, se musí zajistit určité podmínky:

1. Ve snímku se nesmí nacházet modely aut z různých tříd. Aut tam může být více, ale vždy pouze z jedné třídy. Jelikož nedokážu pomocí metody background subtraction objekty rozeznat (to je totiž přesně úkol object detection), k tomu abych správně každému objektu přiřadil třídu, se musí ve snímku vždy nacházet objekty pouze z jedné třídy.

2. Modely aut nesmí narazit do žádného objektu. Pokud se objekty srazí, jejich bílé pixely ve výsledné binární masce budou tvořit jednu konturu a nedokáži je od sebe oddělit. Tohle by opět způsobilo špatné označení obrázků a zhoršilo výsledky učení object detection modelu.

3. Po celou dobu detekce musí být v záběru kamery stejný počet pohybujících se aut. Aby se předešlo možným chybám při detekci, vždy se kontroluje, zda počet detekovaných objektů odpovídá předem definovanému počtu. Modely aut se tedy musí po dobu detekce pohybovat, nesmí se ani na chvíli zastavit. Pokud by se některé auto zastavilo, počet detekovaných objektů nebude správný a žádné označování nenastane.

[2] `https://colab.research.google.com/`

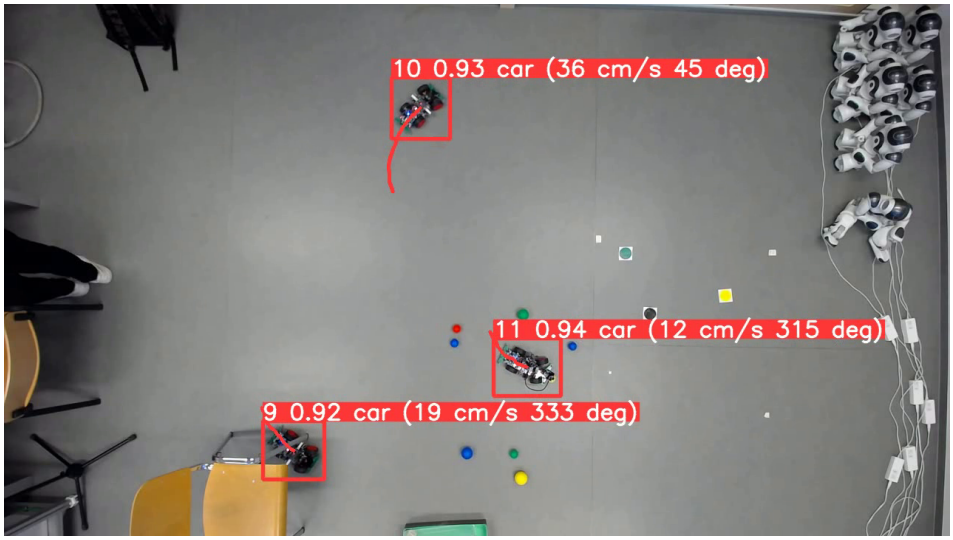
Kdyby se počet detekovaných objektů nekontroloval, tak v případě, že by se některé auto zastavilo, nebylo by detekováno a způsobilo by to problém v učení object detection modelu. Říkal bych mu tím, že na tom konkrétním místě v obrázku žádný objekt neexistuje, ale ve skutečnosti tam je. Snažil by se tedy naučit, aby toto místo nedetekoval, přestože by správně měl.

4. Není to nutné, ale pro lepší výsledky je vhodné, aby auta jezdila po více různých pozadí. A to z důvodu, aby si model při trénování nemyslel, že to konkrétní jedno pozadí je součástí sledovaného objektu.

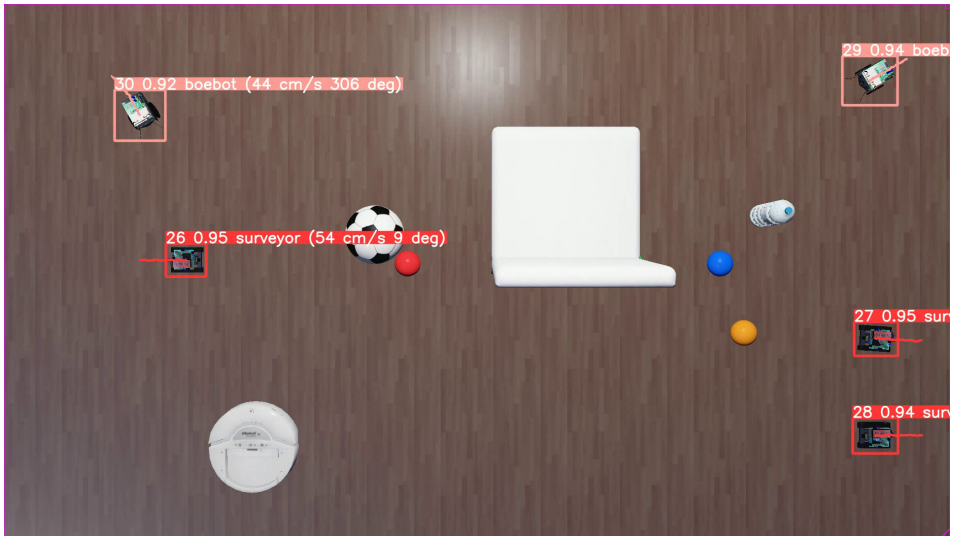
5. Pro lepší výsledky a prevenci False Positives je také vhodné, aby v záběru bylo více objektů, než pouze sledovaná auta. Je totiž důležité také modelu ukázat, jak auto nevypadá. Možnost je také přidat do datasetu obrázky s různými jinými předměty, kde se žádné auto nenachází.

Za těchto podmínek se během pár vteřin dokáže získat stovky správně označených dat a ušetřit tím spoustu času a manuální práce.

Po úspěšné detekci objektů využiji pro sledování algoritmu SORT. Poskytnutí a výpočet informací zde probíhá obdobným způsobem jako v případě detekce s využitím značek. Jedinou výjimkou je, že při detekci značek ArUco se získají souřadnice všech 4 rohů markeru, takže úhel směru je vypočítán přesně pro každý snímek. Ale v případě detekce objektů se znají pouze souřadnice ohraničujících rámečků, takže pro zjištění směru počítám úhel vektoru posunu mezi jednotlivými snímky.



Obrázek 5.9 Ukázka detekce a sledování objektů na reálných datech



Obrázek 5.10 Ukázka detekce a sledování objektů v simulaci

Obdobně jako u ArUco markerů, se zde vykresluje ke každému objektu více hodnot. První zde značí unikátní ID přiřazené pomocí algoritmu sledování objektů, druhé značí s jakou důvěrou object detecion model objekt klasifikoval, třetí je název klasifikované třídy a poslední dvě hodnoty v závorkách jsou opět rychlost a směr auta.

Výhody řešení s využitím detekce a sledování objektů:

- objekty lze detekovat i v případě, že jsou z nějaké části zakryté
- není potřeba vyrábět a přidělávat na auta žádné značky

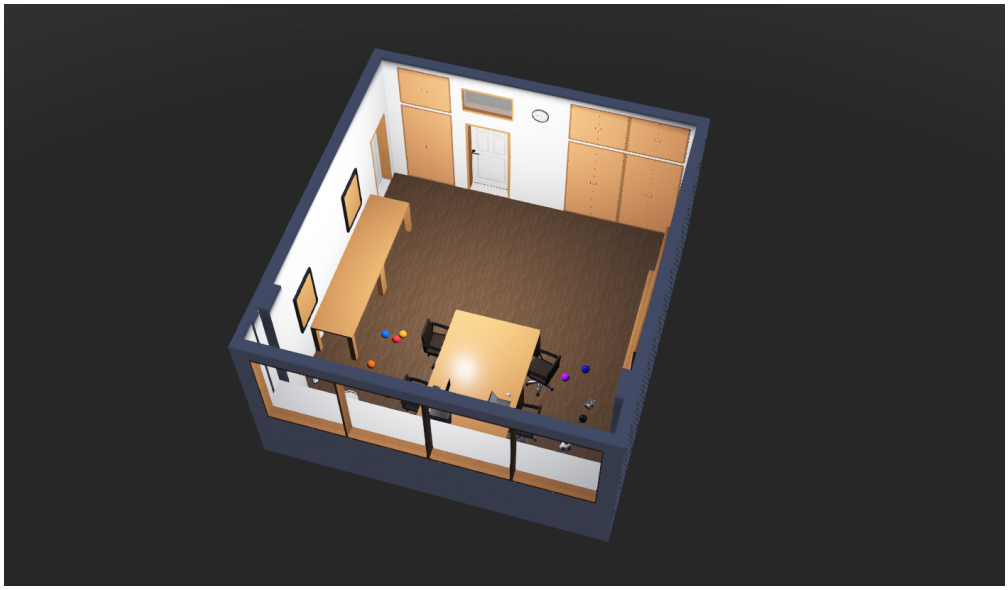
Nevýhody:

- je potřeba natrénovat model detekce objektů
- je potřeba získat a označit data
- horší přesnost telemetrie

Kapitola 6

Simulační prostředí

V době, kdy jsem začal pracovat na této práci, nebylo ještě možné přichytit kameru ke stropu laboratoře, a tak jsem si vytvořil simulaci. Používat simulaci je také časově efektivnější, protože přístup k laboratoři jsem neměl každý den a ve virtuálním prostředí můžu rychle a jednoduše přidávat nové objekty nebo měnit podmínky. Simulace samozřejmě slouží pouze k testování různých technik strojového vidění. Pokud bych například chtěl použít object detection model – který fungoval skvěle ve virtuálních podmínkách – neuronovou síť budu muset na reálných datech znovu naučit.



Obrázek 6.1 Virtuální místnost ve Webots

V programu Webots[33] jsem sestavil virtuální kopii laboratoře THA:1054, ve které lze simulovat pohyblivé roboty a statickou kameru přesně jak potřebuji. Prvně jsem ale musel vytvořit 3D model laboratoře. Programů určených k vytváření CAD objektů je spousta. Jelikož s počítačovým projektováním nemám žádné zkušenosti, 3D model jsem vytvořil ve svobodném software OpenSCAD[34], který je pro programátory ideální, protože na rozdíl od jiných, které jsou zpravidla interaktivní, se zde píše skript, který model vygeneruje. Přeměřil jsem tedy celou místnost a sepsal jsem kód.

Model jsem poté exportoval ve formátu stl a importoval do Webots. Přidal jsem několik základních objektů, aby místnost připomínala laboratoř THA:1054.

Následně jsem vybral dva pohyblivé roboty již připravené k použití přímo ve Webots, které mi nejvíce připomínaly modely aut z laboratoře THA:1054. Konkrétně to jsou *Surveyor SRV-1*[1] a *BoeBot*[2].

Nakonec jsem do virtuální místnosti přidal kameru, která má totožné parametry s kamerou Logitech c930e, která se vybrala pro použití v této bakalářské práci. Má tedy dFOV 90 %, rozlišení 1920×1080 a snímá rychlostí 30 fps.

[1] `https://cyberbotics.com/doc/guide/surveyor`
[2] `https://cyberbotics.com/doc/guide/boebot`

Kapitola 7
Implementace

V této kapitole je popsána implementace jednotlivých metod. Všechny soubory s okomentovaným zdrojovým kódem jsou dostupné na `https://gitlab.fit.cvut.cz/macekja9/bap`. Požadavkem pro spuštění skriptů je počítač s operačním systémem Windows nebo Linux, na kterém je nainstalován Python verze 3.8 nebo vyšší. Všechny následující metody, pokud není řečeno jinak, používají pouze knihovny NumPy a OpenCV.

7.1 Značky ArUco

Modul aruco pro generování a detekci ArUco značek není v základní knihovně OpenCV a musí se nainstalovat i rozšíření *opencv-contrib-python*.

Generování

Skript pro generování ArUco značek je v souboru *aruco_generate.py*. Tento skript má na vstupu jediný parametr, který požaduje seznam číselných ID značek, které chce uživatel vygenerovat. Velikost slovníku a značek je už dopředu určenýá, použije se *cv2.aruco.DICT_4X4_50*. Rozsah ID je tedy 0 až 49. Slovník se získá funkcí *cv2.aruco.Dictionary_get*. Pro každé ID se následně pomocí metody *cv2.aruco.drawMarker* značka nakreslí a výsledek se uloží na disk v aktuální složce ve formátu *aruco_ID.png*.

Výpis kódu 7.1 Ukázka spuštění skriptu pro generování značek ArUco

```
python aruco_generate.py 0 1 2 3
```

Detekce

Skript pro detekci značek se nachází v souboru *aruco_detect.py* a má více vstupních parametrů:

source	cesta k videu nebo označení webkamery
show-video	určuje, zda se bude ukazovat video s detekovanými značkami
print-output	určuje, zda se budou detekce a telemetrie vypisovat do konzole
save-video	cesta k souboru, do kterého se uloží video s detekovanými značkami; pokud je argument vynechán, video se neuloží
save-txt	cesta k souboru, do kterého se detekce a vypočítané telemetrie vypíšou; pokud je argument vynechán, text se neuloží

Formát výpisu do konzole a souboru je následující:
snímek id [x y] [x y] [x y] [x y] **rychlost směr**, kde jednotlivé části znamenají:

snímek	číslo snímku
id	id detekované značky
[x y]	souřadnice rohů detekované značky v pořadí: levý horní, pravý horní, levý dolní a pravý dolní
rychlost	rychlost auta v cm/s
směr	směr auta ve stupních

Slovník značek se získá stejným způsobem jako ve skriptu pro generování. Detekci zajišťuje funkce *cv2.aruco.detectMarkers*. Funkci se předává nijak neupravený snímek a všechno předzpracování zajišťuje sama. Lze jí pouze předat různé parametry. Některé z nich jsem ručně vyladil pro konkrétní kameru a velikost značek použité v této práci. Pro vykreslování detekovaných značek je připravena funkce *cv2.aruco.drawDetectedMarkers*. Vypočítané telemetrie a trajektorie se vykreslují zvlášť.

Stisknutím ctrl+C se detekce ukončí. Pokud byl použit parametr *show_video*, je také možné detekci ukončit stisknutím klávesy Q. V případě použití videozáznamu, se detekce ukončí sama po zpracování všech snímků.

Výpis kódu 7.2 Ukázka spuštění skriptu pro detekci značek ArUco

```
python aruco_detect.py --source video.mp4 --show-video \
--save-video detected.mp4
```

7.2 Automatické označení dat

Skript pro automatické označení dat se nachází ve složce *labeling* v souboru *auto_label.py* a má následující parametry:

source	cesta k videu nebo označení webkamery
cnt	počet aut, které se budou ve snímcích pohybovat
cls	číselné označení třídy objektů
show	určuje, zda se budou ukazovat videa s maskou a vypočítanými bounding boxy
save	určuje, zda se snímky a textové soubory s označeními uloží na disk
n	bude se ukládat pouze každý n-tý snímek
after	určuje, po kterém snímku detekce začne
threshold	prahová hranice pro metodu odečítání pozadí
history	parametr historie pro metodu odečítání pozadí

Pro odečítání pozadí jsem využil *cv2.createBackgroundSubtractorMOG2*. Po získání masky odečtením pozadí se provede filtrace funkcemi *cv2.GaussianBlur* a *cv2.dilate*. Následně se pomocí *cv2.findContours* naleznou kontury a pro odstranění šumu se ty, které jsou příliš malé, zahodí. Pro každou výslednou konturu se vypočítá ohraničující rámeček pomocí *cv2.boundingRect* a spolu s obrázkem se uloží na disk. Obrázky do souborů *původníNázevVidea_N.jpg* do složky *images* a textové soubory s ohraničujícími rámečky do souborů *původníNázevVidea_N.txt* do složky *bbox_txt*, kde N je rostoucí index zpracovaného obrázku počínaje od 0. Formát výstupu v textovém souboru odpovídá formátu YOLO (viz 7.5).

Výpis kódu 7.3 Ukázka souboru spuštění automatického označení dat

```
python auto_label.py --source video.mp4 --cnt 3 --cls 0 --show --save
```

7.3 Manuální označení dat

Pro manuální označení dat jsem si upravil open source nástroj k označování dat [35]. Všechny soubory jsou ve složce *labeling*. Interaktivní aplikace se spustí skriptem *label.py*. Předtím, než se ale aplikace spustí, se musí do textového souboru *class_list.txt* jednotlivě po řádcích napsat názvy tříd. Obrázky k označení se musí nacházet ve složce *images*. Pro každý obrázek se ve složce *bbox_txt*, vytvoří příslušný textový soubor, do kterého se budou ukládat informace o ohraničujících rámečcích, které uživatel vyznačí. Pokud textový soubor pro obrázek již existuje, nepřepíše se.

Aplikace má jednoduché ovládání. Pomocí kláves A a D se přepíná mezi jednotlivými obrázky. Klávesy W a S přepínají mezi jednotlivými třídami. Pro označení ohraničujícího rámečku se jednou klikne levým tlačítkem myši pro označení prvního rohu a podruhé pro označení protějšího rohu rámečku. Pro smazání rámečku stačí jednou kliknout pravým tlačítkem myši doprostřed rámečku. Aplikace se ukončí stiskem klávesy Q.

7.4 Rozdělení dat

Pro rozdělení dat na trénovací a validační jsem si upravil skript, který byl součástí [35]. Tento skript využívá pouze základní Python knihovny. Skript se nachází ve složce *labeling* v souboru *train_test_split.py* a má dva parametry:

dest	název složky do které se uloží rozdělená data
split	poměr rozdělení dat do validační množiny

Skript rozdělí obrázky ve složce *images* a jejich příslušné textové soubory ve složce *bbox_txt*. Výsledkem budou kopie obrázků a jejich příslušných textových souborů náhodně rozdělených v požadovaném poměru ve složkách *train* a *val* v adresáři zadaném parametrem.

7.5 Detekce a sledování objektů

Pro YOLO jsem využil nejnovější a nejpopulárnější open source implementaci YOLOv5[16]. Tato verze je napsána v jazyce Python s využitím knihovny pro strojové učení Pytorch[36]. Pro SORT jsem využil open source implementaci [31]. Všechny soubory k detekci a sledování objektů jsou ve složce *yolov5*. Pro nainstalování knihovny Pytorch s podporou GPU na zařízení s operačním systémem Windows je nutné zavolat speciální příkaz. Všechny ostatní knihovny jsou vypsané v souboru *requirements.txt* a nainstalují se druhým příkazem. Pro instalaci na Linuxovém zařízení stačí zavolat pouze druhý příkaz v 7.4.

Výpis kódu 7.4 Příkazy pro nainstalování knihoven

```
pip install torch torchvision torchaudio --extra-index-url \
https://download.pytorch.org/whl/cu113

pip install -r requirements.txt
```

Trénování modelu YOLO

YOLO k označení ohraničujících rámečků trénovacích dat využívá textový soubor pro každý jednotlivý obrázek. Textový soubor musí mít vždy stejný název jako obrázek, ke kterému patří. Každý řádek textového souboru popisuje jeden ohraničující rámeček v příslušném obrázku. Formát každého řádku je **cls x y w h**, kde jednotlivé části znamenají:

cls	číselné označení třídy
x y	souřadnice středu rámečku normalizované podle šířky a výšky obrázku
w h	šířka a výška rámečku normalizované podle šířky a výšky obrázku

Aby se model mohl v cloudovém prostředí začít trénovat, musí se data předtím odeslat na Weights & Biases. K použití Weights & Biases je nutné, aby se uživatel na `https://wandb.ai` zdarma zaregistroval a vytvořil nový projekt. Potom se musí vytvořit soubor formátu yaml, do kterého se napíše cesta k datasetu a počet a názvy tříd.

Výpis kódu 7.5 Ukázka souboru yaml

```
train: path_to_dataset/train/
val: path_to_dataset/val/
nc: 2
names: ['surveyor', 'boebot']
```

Následně se data už mohou odeslat. Po uživateli to bude vyžadovat klíč k autorizaci, který se získá na `https://wandb.ai/authorize`.

Výpis kódu 7.6 Odeslání dat na W&B

```
python utils/loggers/wandb/log_dataset.py --project my_project \
--data my_dataset.yaml
```

Po úspěšném odeslání se automaticky vytvoří soubor *my_dataset_wandb.yaml*, který se při trénování použije k načtení dat z Weights & Biases.

Nyní se může přejít na `https://colab.research.google.com/`, kde je opět potřeba, aby byl uživatel přihlášen. Pro hardwarovou akceleraci se pod záložkou *Runtime* zvolí volba *Change runtime type* a vybere GPU. K trénování je připraven Jupyter Notebook *colab_train.ipynb*, který se musí se souborem *my_dataset_wandb.yaml* na Colab nahrát. Notebook obsahuje pouze dvě buňky – první se nainstalují všechny potřebné knihovny a naklonuje repozitář yolov5 a druhou se spustí trénování. Trénování zajišťuje skript *train.py* a má spoustu parametrů, které jsou ve zdrojovém kódu vysvětlené. Těmi nejdůležitějšími ale jsou:

data	cesta k yaml souboru vytvořeného po nahrání datasetu do Weight & Biases
project	název vytvořeného projektu na Weight & Biases
weights	počáteční váhy neuronové sítě
epochs	počet epoch

Výpis kódu 7.7 Příklad spuštění trénování modelu

```
!python train.py \
  --data ../my_dataset_wandb.yaml \
  --epochs 20 \
  --project my_project \
  --batch-size -1 \
  --weights yolov5m.pt \
  --cache
```

Po úspěšném trénování se přejde opět na `https://wandb.ai`, odkud se uvnitř projektu ze záložky *Artifcats* natrénovaný model stáhne.

Detekce

Skript pro detekci objektů pomocí YOLO je v souboru *detect.py*, do kterého jsem zakomponoval SORT i výpočet telemetrie. Třída *Sort* pro sledování objektů se nachází v souboru *sort/sort.py*.

Skript pro detekci má opět spoustu argumentů a všechny jsou ve zdrojovém kódu vysvětleny. Těmi nejdůležitějšími jsou:

weights	cesta k souboru natrénovaného modelu
source	cesta k videu nebo označení webkamery
view-img	určuje, zda se bude ukazovat video s detekovanými objekty
print-output	určuje, zda se budou detekce a telemetrie vypisovat do konzole
save-txt	určuje, zda se budou detekce a telemetrie ukládat do souboru
nosave	určuje, zda se nebude výsledné video s detekcemi ukládat do souboru

Výsledné video a textový soubor budou uloženy ve složce *runs/detect/exp*. Formát výpisu do konzole a souboru je následující: **snímek třída id x y w h rychlost směr**, kde jednotlivé části znamenají:

snímek	číslo snímku
třída	číselné označení detekované třídy objektu
x y	souřadnice levého horního rohu ohraničujícího rámečku detekovaného objektu
w h	šířka a výška rámečku
rychlost	rychlost auta v cm/s
směr	směr auta ve stupních

Stisknutím ctrl+C se detekce ukončí. Pokud byl použit parametr *view-img*, je také možné detekci ukončit stisknutím klávesy Q. V případě použití videozáznamu, se detekce ukončí sama po zpracování všech snímků.

Výpis kódu 7.8 Ukázka spuštění detekce

```
python detect.py --weights my_model.pt --source 0 --viev-img
```

Kapitola 8

Experimenty a testování

V této kapitole jsou popsáno jakým způsobem jsem zkoušel, porovnával a testoval jednotlivé metody. Trénování modelů proběhlo v cloudovém prostředí s vysoce výkonným grafickým procesorem NVIDIA Tesla T4. Pro testování byl používán grafický procesor NVIDIA GeForce GTX 1060.

8.1 Velikosti modelů

YOLOv5 nabízí 5 modelů různých velikostí. V této sekci se zabývám výběrem vhodného modelu pro tuto práci.

Obrázek 8.1 Modely YOLOv5

S rostoucí velikostí se zvyšuje přesnost detekce, ale také doba učení a snižuje se rychlost zpracování jednoho snímku. Cílem práce je rychlé nasazení, takže čas strávený trénováním je velice důležitým parametrem při výběru velikosti. Nutnou podmínkou ale je, aby model dokázal detekovat objekty v reálném čase. Všechny modely jsem tedy prvně otestoval, abych vyřadil ty, které nedokáží objekty detekovat rychlostí alespoň 30 snímků za vteřinu.

Tabulka 8.1 Porovnání modelů YOLOv5

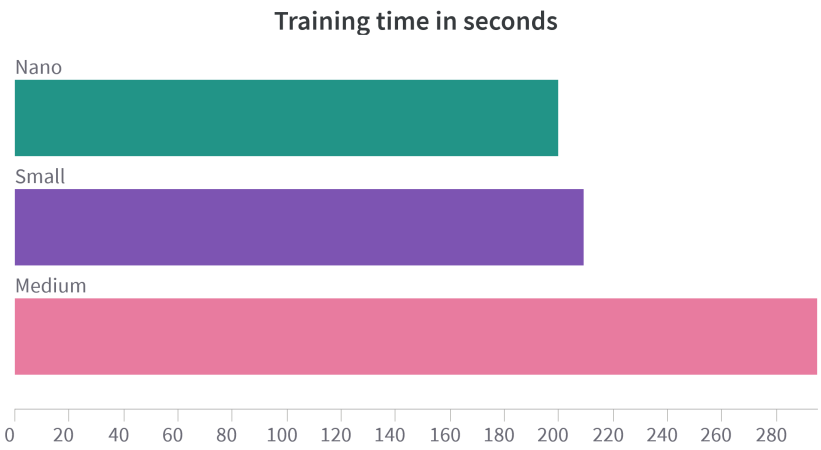
Model	fps
Nano	73
Small	67
Medium	45
Large	22
XLarge	14

Z porovnání vychází, že modely Large a XLarge pro tuto práci vhodné nejsou. Modely Nano, Small a Medium jsem následně začal trénovat na vlastních datech. V oficiální dokumentaci[37] se pro nejlepší výsledek doporučuje více než 1500 obrázků na jednu třídu a trénovat alespoň 300 epoch, což může trvat i několik hodin než se model natrénuje. Tam se však očekávají složitější detekce než v mém případě, myšleno detekce trojrozměrných objektů v reálném světě jako jsou například auta, lidé nebo zvířata. Kde v jedné třídě se každý objekt může lišit v barvě, tvaru, velikosti a úhlu záběru. Já sice také sleduji trojrozměrné objekty, ale pomocí statické kamery, takže objekty v záběru jsou vidět vždy pouze ze shora a každý model auta v jedné třídě až na maličkosti vypadá totožně. Měl bych si tedy vystačit se zlomkem toho.

Dataset obsahuje 444 obrázků, rozdělených v poměru 80:20 na trénovací a validační. Má pouze jednu třídu a skládal se z obrázků získaných pomocí kamery, která bude využita i k detekci. V jednom obrázku jsou až 3 modely aut. Po mozaikové augmentaci mají obrázky velikost 640×640 a vypadají následovně:

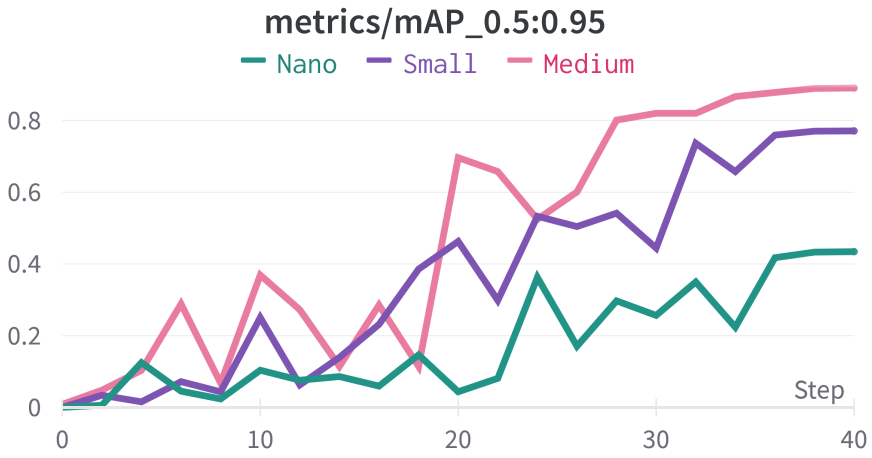
Obrázek 8.2 Vzorek dat po augmentaci

Pro porovnání jsem vyzkoušel všechny modely trénovat 20 epoch.

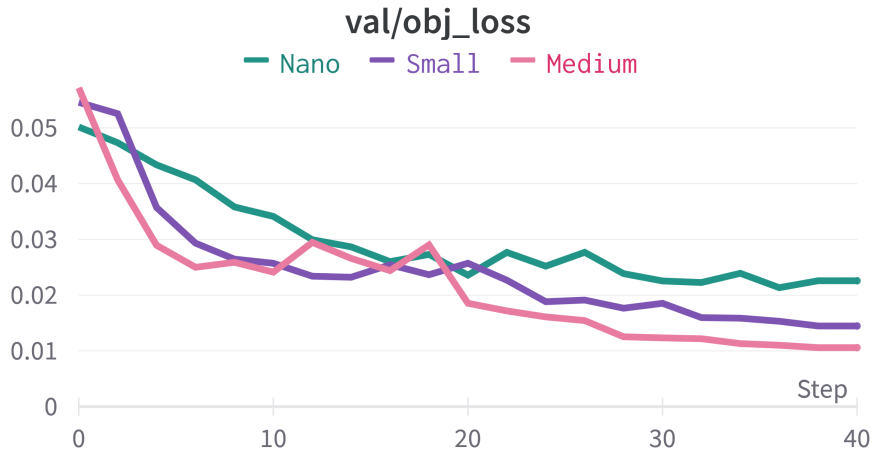


Obrázek 8.3 Doba trénování modelů

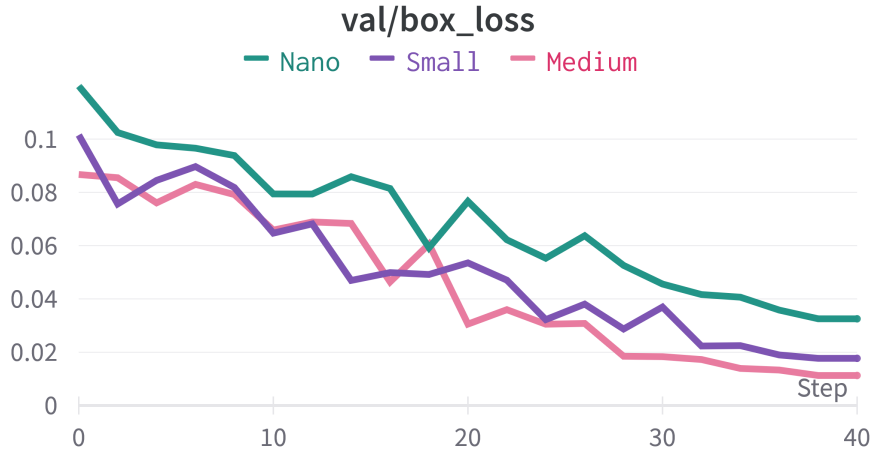
Modelům Nano a Small trvalo trénování o přibližně 30 % času méně než Medium. Odpovídaly tomu také ale i výsledky.



Obrázek 8.4 mAP po 20 epochách

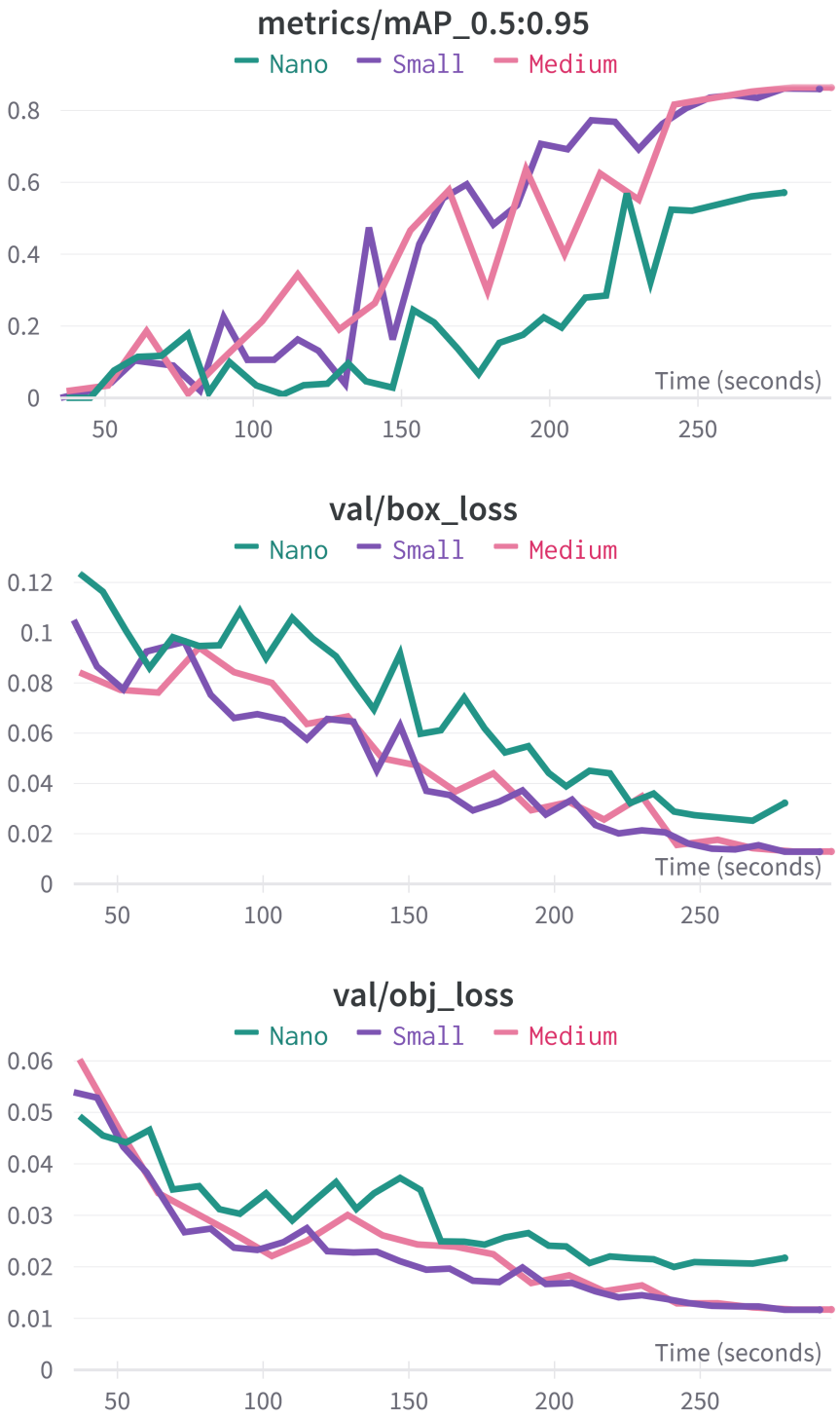


Obrázek 8.5 Obj loss po 20 epochách



Obrázek 8.6 Box loss po 20 epochách

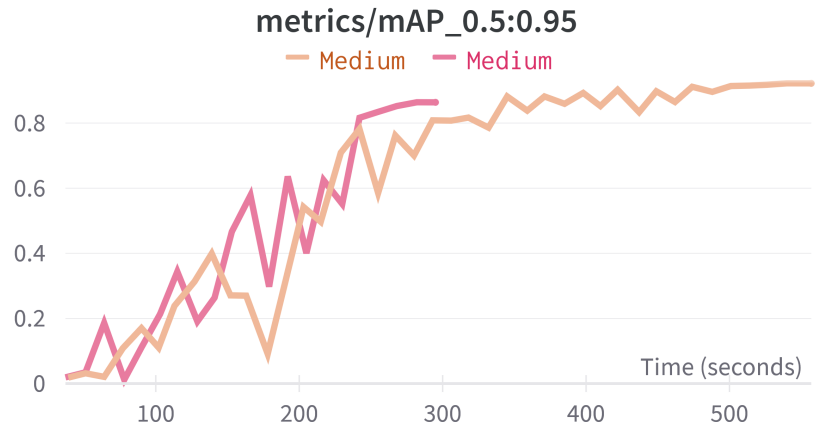
Trénovaly se zatím ale jenom po dobu 20 epoch. Pokračoval jsem tedy v trénování pro modely Nano a Small tak, aby každý celkově běžel po dobu přibližně stejnou jako Medium.



Obrázek 8.7 Finální výsledky

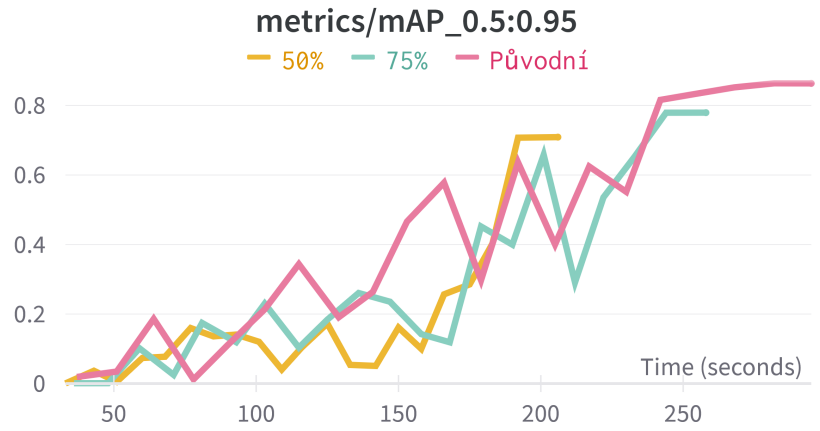
I po delším trénování Nano a Small, kdy měly všechny modely stejné podmínky, je přesto vítězem opět Medium. Small má sice výsledky trénování velice dobré, ale na testovacích datech se stále občas vyskytují False Positives. Snížit výskyt False Positives nemusí být obtížné. Ale protože Medium oproti Small žádnou nevýhodu nemá[1], tak není důvod proč se snažit o vylepšení horšího modelu, když ten lepší již existuje. Vybral jsem tedy model Medium.

Větší počet epoch, množství dat či ladění hyperparametrů modelu Medium už velké zlepšení nepřináší. Za cenu dvojnásobného času učení se mi podařilo vylepšit mAP o 6 % a na testovacích datech jsem ani žádného zlepšení nedosáhl. Dosažený výsledek je pro účely této práce více než dostačující, a jelikož chci stále udržet dobu trénování co nejnižší, nemyslím si, že téměř nepatrná vylepšení na úkor času trénování za to stojí.



Obrázek 8.8 Nepatrné zlepšení modelu

Přestože ke skvělému výsledku stačí trénovat model necelých 5 minut na jednu třídu, zkusil jsem pro ještě větší zrychlení ubrat množství dat. Poprvé jsem ubral čtvrtinu množství dat a následně další čtvrtinu.



Obrázek 8.9 Porovnání s menším množstvím dat

[1]Kromě rychlosti predikce. Small dokáže zpracovat jeden snímek za kratší dobu než Medium, ale protože požaduji pouze 30 fps (a to oba modely splňují), je to nepodstatné.

Výsledné mAP se s každým ubráním dat snížilo a velké zrychlení to nepřineslo. Když běh s 75 % dat skončil, tak původní model měl ve stejný čas lepší výsledky. Běh s 50 % dat měl sice v době kdy skončil výsledky lepší než ostatní modely, ale pro nasazení je model nepoužitelný a to z důvodu výskytu velkého množství False Positives na testovacích datech.

8.2 Sledování objektů

V této sekci porovnávám metody sledování objektů SORT a CBOT. Testování proběhlo na více různých videozáznamech, ve kterých jeden snímek obsahoval až 5 objektů.

Algoritmu SORT zpracování jednoho snímku trvalo v průměru 2,4 ms a CBOT 0,4 ms, což v obou případech je více než dostačující pro sledování v reálném čase.

Pro vyhodnocení správnosti sledování jsem kontroloval dvě metriky. První bylo nesprávné vytváření nových ID objektům, které již byly sledovány. A druhou počet záměn identifikátorů mezi objekty. Výsledky měli obě metody totožné, nenastalo jediné prohození ani špatné přiřazení nového identifikátoru. Což není vůbec překvapivé.

8.3 One-Shot learning s augmentací dat

Knihovna Turi Create nabízí sadu 951 background obrázků. Trénovací obrázky jsem zde využil dva – roboty Surveyor a Boebot z Webots. Po augmentaci, která trvala přibližně 3 minuty, trénování object detection modelu s tímto množstvím dat po 20 epoch zabralo necelých 9 minut. Výsledek byl ale velice špatný – na testovacích datech nebyl detekován správně jediný objekt. Pokračoval jsem tedy dál v trénování, ale ani po 20 minutách výsledek nebyl použitelný. Jelikož se je mým cílem rychlé nasazení a oproti YOLOv5, kde trénování na jednu třídu trvalo 5 minut a mělo skvělé výsledky, už jsem dále nepokračoval.

Tato metoda se tedy ukázala jako velmi neúčinná a za příčinu považuji následující dva problémy:

1. Tato metoda je doporučená pouze pro detekci dvojrozměrných objektů. A opravdu, i přes to, že jsou modely aut vždy snímány pouze ze shora, pro tuto práci není použitelná. Jelikož se auta volně pohybují, přímý pohled ze shora je pouze na auta, která se nachází přímo pod kamerou. Jakmile jedou dál od středu, začnou být snímány i jejich boční strany, které model při trénování nikdy neviděl.

2. Obrázek s objektem, který chceme detekovat, nemůže kvůli augmentaci sám mít nějaké pozadí. Je to z toho důvodu, že model se při trénování naučí detekovat objekt vždy pouze s tím konkrétním pozadím a celá ta augmentace ve výsledku ztrácí význam. Získat obrázky s průhledným pozadím pro konkrétní modely aut by nebyl problém, ale pro obecné řešení, je to zbytečné přitížení, které ve výsledku žádné výhody nepřináší.

8.4 Velikost ArUco značky

V této sekci se zabývám nalezením ideální velikostí ArUco značky. Snažil jsem se najít co nejmenší rozměr značky, při kterém detekce funguje stále spolehlivě. Způsob testování byl velice jednoduchý. Připravil jsem si několik značek různé velikosti a následně na živém videu se stálým upravováním parametrů detekce zkoumal, zda se značky detekují správně. Jako nejlepší velikost jsem vyhodnotil 10 cm.

Následně jsem pro ověření nahrál několik videozáznamů, ve kterých se pohybovaly modely aut s připevněnými značkami, a počítal jsem, v kolika snímcích se značky správně detekovaly. Značky byly detekovány správně v 99.3 % případů.

Kapitola 9
Závěr

V této práci bylo cílem navrhnout řešení pro sledování pohybujících se modelů aut no podlaze laboratoře inteligentních vestavných systémů v reálném čase s důrazem na rychlou adaptaci nových modelů. Kromě využití značek, se mělo objekty sledovat i pomocí technik s využitím hlubokého učení. Zadání jsem splnil. Pro sledování s využitím značek jsem využil ArUco markers, které jsou určené přímo pro lokalizaci ve strojovém vidění. Pro nasazení nového objektu stačí vytvořit novou značku a připevnit ji k objektu. Pro sledování s využitím hlubokého učení jsem využil state-of-the-art object detection modelu YOLOv5. Trénování na jednu třídu trvalo necelých 5 minut s použitím vysoce výkoného grafického procesoru. K rychlé adaptaci na nové modely aut jsem navrhl způsob automatického označení dat pomocí metody odčítání pozadí a ušetřil jsem tím spoustu času a manuální práce.

Bibliografie

1. ROMERO-RAMIREZ, Francisco; MUÑOZ-SALINAS, Rafael; MEDINA-CARNICER, Rafael. Speeded Up Detection of Squared Fiducial Markers. *Image and Vision Computing*. 2018, roč. 76. Dostupné z DOI: 10.1016/j.imavis.2018.05.004.
2. GARRIDO-JURADO, Sergio; MUÑOZ-SALINAS, Rafael; MADRID-CUEVAS, Francisco; MEDINA-CARNICER, Rafael. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition*. 2015, roč. 51. Dostupné z DOI: 10.1016/j.patcog.2015.09.023.
3. GARRIDO-JURADO, Sergio; MUÑOZ-SALINAS, Rafael; MADRID-CUEVAS, Francisco; MARÍN-JIMÉNEZ, Manuel. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*. 2014, roč. 47, s. 2280–2292. Dostupné z DOI: 10.1016/j.patcog.2014.01.005.
4. BRADSKI, Gary. *Detection of ArUco markers*. 2018. Dostupné také z: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.
5. SALINAS, Rafael Muñoz. *Aruco Library Documentation*. Google, 2018. Dostupné také z: https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc.
6. ITSEEZ. *Open Source Computer Vision Library*. 2015. Dostupné také z: https://github.com/itseez/opencv.
7. REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, roč. abs/1506.02640. Dostupné z arXiv: 1506.02640.
8. REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, roč. abs/1612.08242. Dostupné z arXiv: 1612.08242.
9. REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *CoRR*. 2018, roč. abs/1804.02767. Dostupné z arXiv: 1804.02767.
10. BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*. 2020, roč. abs/2004.10934. Dostupné z arXiv: 2004.10934.
11. GIRSHICK, Ross B.; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, roč. abs/1311.2524. Dostupné z arXiv: 1311.2524.
12. GIRSHICK, Ross B. Fast R-CNN. *CoRR*. 2015, roč. abs/1504.08083. Dostupné z arXiv: 1504.08083.

13. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross B.; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, roč. abs/1506.01497. Dostupné z arXiv: 1506.01497.
14. HUI, Jonathan. *Object detection: Speed and accuracy comparison*. Medium, 2018. Dostupné také z: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359.
15. DWIVEDI, Priya. *Yolov5 compared to faster RCNN*. Towards Data Science, 2020. Dostupné také z: https://towardsdatascience.com/yolov5-compared-to-faster-rcnn-who-wins-a771cd6c9fb4.
16. JOCHER, Glenn; CHAURASIA, Ayush; STOKEN, Alex; NANOCODE012; KWON, Yonghye; BOROVEC, Jirka; TAOXIE; FANG, Jiacong; IMYHXY; MICHAEL, Kalen; LORNA; V, Abhiram; MONTES, Diego; NADAR, Jebastin; LAUGHING; TKIANAI; YXNONG; SKALSKI, Piotr; WANG, Zhiqiang; HOGAN, Adam; FATI, Cristi; MAMMANA, Lorenzo; ALEXWANG1900; PATEL, Deep; YIWEI, Ding; YOU, Felix; HAJEK, Jan; DIACONU, Laurentiu; MINH, Mai Thanh. *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Zenodo, 2022. Ver. v6.1. Dostupné z DOI: 10.5281/zenodo.6222936.
17. WANG, Xin; HUANG, Thomas E.; DARRELL, Trevor; GONZALEZ, Joseph E.; YU, Fisher. Frustratingly Simple Few-Shot Object Detection. *CoRR*. 2020, roč. abs/2003.06957. Dostupné z arXiv: 2003.06957.
18. APPLE INC. *One-shot object detection*. GitHub, 2019. Dostupné také z: https://apple.github.io/turicreate/docs/userguide/one_shot_object_detection/.
19. YOHANANDAN, Shivy. *Map (mean average precision) might confuse you!* Towards Data Science, 2020. Dostupné také z: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.
20. GAD, Ahmed Fawzy. *Mean average precision (MAP) explained*. Paperspace Blog, 2020. Dostupné také z: https://blog.paperspace.com/mean-average-precision/amp/.
21. ARIE, Lihi Gur. *The Practical Guide for Object Detection with Yolov5 algorithm*. Towards Data Science, 2022. Dostupné také z: https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843.
22. BRADSKI, Gary. *How to use background subtraction methods*. 2018. Dostupné také z: https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html.
23. MEEL, Vidushi. *Object tracking in computer vision (complete guide)*. 2022. Dostupné také z: https://viso.ai/deep-learning/object-tracking/.
24. KHACHATRYAN, Levon. *A case study: Centroid based object tracking*. 2019. Dostupné také z: https://github.com/lev1khachatryan/Centroid-Based_Object_Tracking.
25. WOJKE, Nicolai; BEWLEY, Alex; PAULUS, Dietrich. *Simple Online and Realtime Tracking with a Deep Association Metric*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1703.07402.
26. KANJEE, Ritesh. *DeepSORT - deep learning applied to object tracking*. Augmented Startups, 2020. Dostupné také z: https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104.
27. BEWLEY, Alex; GE, ZongYuan; OTT, Lionel; RAMOS, Fabio; UPCROFT, Ben. Simple Online and Realtime Tracking. *CoRR*. 2016, roč. abs/1602.00763. Dostupné z arXiv: 1602.00763.
28. SINGH, Aditya. *Top 5 object tracking methods*. Augmented Startups, 2021. Dostupné také z: https://medium.com/augmented-startups/top-5-object-tracking-methods-92f1643f8435.

29. KALMAN, Rudolph Emil. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*. 1960, roč. 82, č. Series D, s. 35–45.

30. KUHN, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955, roč. 2, č. 1-2, s. 83–97. Dostupné z DOI: https://doi.org/10.1002/nav.3800020109.

31. BEWLEY, Alex. *Sort: Simple, online, and realtime tracking of multiple objects in a video sequence.* 2016. Dostupné také z: https://github.com/abewley/sort.

32. BIEWALD, Lukas. *Experiment Tracking with Weights and Biases*. 2020. Dostupné také z: https://www.wandb.com/.

33. WEBOTS. *http://www.cyberbotics.com*. Ed. LTD., Cyberbotics. [B.r.]. Dostupné také z: http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.

34. KINTEL, Marius. *OpenSCAD*. 2010. Dostupné také z: https://openscad.org/.

35. GONCHAROV, Ivan. *Modified Open Labelling*. 2021. Dostupné také z: https://github.com/ivangrov/ModifiedOpenLabelling.

36. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; D'ALCHÉ-BUC, F.; FOX, E.; GARNETT, R. (ed.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, s. 8024–8035. Dostupné také z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

37. JOCHER, Glenn. 2020. Dostupné také z: https://docs.ultralytics.com/tutorials/training-tips-best-results/.

38. BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000.

39. PYTHON CORE TEAM. *Python: A dynamic, open source programming language*. 2019. Dostupné také z: https://www.python.org/.

40. CANU, Sergio. *Object tracking with Opencv and python*. 2021. Dostupné také z: https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python/.

Obsah přiloženého média

- `README.md` ..... stručný popis obsahu média
- `aruco` ..... zdrojové kódy pro generování a detekci ArUco značek
- `labeling` ..... zdrojové kódy pro označení a rozdělení obrázků
- `webots_project` ..... virtuální místnost v Webots
- `yolov5` ..... zdrojové kódy pro detekci a sledování objektů
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF