Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Visual tracking systém pro UAV

Diplomová práce

Bc. Michal Kolář

Vedoucí práce: PhDr. Milan Novák, Ph.D.

České Budějovice 2018

Bibliografické údaje

Bc. Kolář Michal, 2018: Visual tracking systém pro UAV. [Visual tracking system for UAV. Mgr. Thesis, in Czech.] – 62 p. , Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Diplomová práce se zabývá analýzou současných možností pro sledování objektů v obraze, na základě které je navržen postup pro tvorbu systému schopného sledovat objekt zájmu. Součástí práce je navržení virtuální reality pro potřeby implementace sledovacího systému, jež je ve finále nasazen a testován na reálném prototypu bezpilotního prostředku.

Annotation

This master thesis deals with the analysis of the current possibilities for object tracking in the image, based on which is designed a procedure for creating a system capable of tracking an object of interest. Part of this work is designing virtual reality for the needs of implementation of the tracking system, which is finally deployed and tested on a real prototype of unmanned vehicle.

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ..... Podpis autora .....

Poděkování

Děkuji Adamu Blažkovi a Lukáši Vencálkovi za společné konzultace a čas. Dále děkuji své rodině a přítelkyni Lucii Jandákové za podporu při psaní této práce.

Slovník pojmů

- **VTS** - Visual Tracking System
- **UAV** - Unmanned Aerial Vehicle
- **VR** - Virtual Reality
- **SVM** - Support Vector Machines
- **mAP** - Mean Average Precission
- **CNN** - Convolutional Neural Networks
- **R-CNN** - Region-based Convolutional Neural Networks
- **RP** - Region Proposal
- **Yolo** - You only look once
- **SSD** - Single Shot MultiBox Detector
- **SORT** - Simple Online Realtime Tracking
- **Deep SORT** - Simple Online Realtime Tracking with a Deep Association Metric
- **UDP** - User Datagram Protocol
- **TwD** - Tracking with Darkflow
- **FPS** - Frames per second

Obsah

1. Úvod

Vizuální zpracování objektů z obrazu v reálném čase, uvědomění si jejich pozice a konkrétní vlastnosti, bylo v lidem známém světě doménou pouze organismů založených na uhlíkové bázi. Díky každoročnímu nárůstu výkonu výpočetních technologií a novým nápadům z oboru vizuální detekce, se do této množiny pomalu hlásí systémy založené na konvolučních neuronových sítích, které svou rychlostí zpracování umožňují reagovat na vizuální podněty z reálného světa.

1.1 Cíle práce

Cílem této práce je navrhnout autonomní systém pro bezpilotní prostředek, který dokáže sledovat konkrétní objekt zájmu. Součástí práce bude analýza současných trackovacích metod, ze které se bude vycházet při výběru technologií pro zpracování obrazu. Na základě technologických možností bude ve finále sestrojen prototyp bezpilotního prostředku pro otestování vytvořeného Visual Tracking Systému (dále jen VTS).

1.2 Struktura práce

Práce je dělena do samostatných tematických celků popisujících jednotlivé kroky k dosažení stanovených cílů. V následující kapitole se nachází analýza několika řešení problematiky visual trackingu, jejich princip fungování a vyhodnocení. Další, třetí kapitola se zabývá vysvětlením jednotlivých logických kroků k navržení VTS. Čtvrtá kapitola popisuje důležitost a modelování virtuální reality pro testování. Pátá kapitola analyzuje technologie pro detekování objektů v obraze a využití těchto technologií pro vytvoření VTS. Šestá kapitola popisuje využité trackovací algoritmy. Sedmá kapitola se zabývá implementací VTS a její nasazení na virtuální a skutečný svět. Osmá kapitola obsahuje diskusi, která rozvádí a hodnotí dosažené výsledky. Poslední kapitola krátce shrnuje celou práci a vyhodnocuje stanovené cíle.

2. Tracking systémy

Díky velkému pokroku v oblasti výkonu, miniaturizace a cenově dobře dostupných výpočetních systémů, se staly v posledních letech bezpilotní prostředky (UAV) velmi aktivní oblastí výzkumu a dosáhly obrovského pokroku na poli automatizované navigace, dozoru, vojenské aplikace, záchranných úkolů a zemědělství. Jedním z odvětví výzkumu se staly UAV systémy založené na vizuálním rozpoznávání okolí. Tento způsob se hodí například v místech, kde je nutné udržet zaměření na určitý cíl nebo není možné bezpilotní prostředek navádět podle GPS signálu.

Pro vytvoření kvalitního autonomního bezpilotního prostředku založeného na vizuálním rozpoznávání okolí je nutné zvolit vhodnou metodu sledování objektu zájmu z obrazu. Níže se nachází seznam vybraných, nejpoúživanějších metod pro trackování objektu.

2.1 Color filtering

Prvotní pokusy o sestrojení VTS byly založeny na sledování objektu zájmu na základě barevné filtrace. Jedná se o jednoduchý a velmi rychlý způsob trackování objektu z obrazu. Objekt či objekty musí mít vůči zbytku obrazu významný barevný rozdíl, aby je bylo možné úspěšně segmentovat. V případě rozpoznávání více objektů, musí mít každý jinou barvu, aby je bylo možné odlišit. Jedná se tedy pouze o selekci určitých rozsahů barevného spektra, což tento způsob činí výpočetně velmi nenáročným. [1][2]

2.1.1 Výhody

- hardwarově nenáročné
- jednoduché na implementaci
- snadné pro tracking

2.1.2 Nevýhody

- v obraze nesmí být stejná barva
- náchylné na změnu světelných podmínek
- obtížné sledování více objektů

2.2 Augmented Reality Marker

Jedním z dalších možných přístupů pro tracking objektu je použití binárních čtverečních markerů s černým ohraničením, ve kterém se nachází vzor podobně jako QR kód. Markery jsou využívány především pro rozšířenou realitu, kde se podle zkosení a natočení vnitřního vzoru určí pozice a rotace v prostoru vůči kameře. Vzhledem k možnosti dosazení unikátního vzoru do markeru je možné trackovat více objektů najednou. Výhodou oproti barevné filtraci je nezávislost na barvách v obrazu a tím i lepší možnost trackování více objektů. [3]

2.2.1 Výhody

- hardwarově nenáročné
- snadné pro tracking
- vynikající pro tracking více objektů

2.2.2 Nevýhody

- nutnost přítomnosti markeru

2.3 OpenTLD

Jeden z nejznámějších algoritmů pro sledování se jmenuje Tracking, learning and detection. Jak jeho anglický název napovídá, skládá se ze tří komponent:

- Tracking - sledování
- Learning - učení
- Detection - detekce

Tento algoritmus nepotřebuje v obraze žádné podpůrné prvky, aby sledoval objekt zájmu. Tracker sleduje objekt nebo objekty snímek za snímkem. Detektor lokalizuje veškeré dosud pozorované zobrazení objektu a opravuje tracker, pokud je potřeba. Učením trackeru podle předchozích přítomností objektu v obraze se koriguje chyba detektoru, který je aktualizován, aby se chybě v budoucnu předešlo. Vzhledem k možnosti učení dokáže tracker navázat zpět na objekt, který byl dočasně zakryt nebo zmizel z obrazu. [4] Díky této schopnosti se algoritmus dočkal aplikace na bezpilotní prostředky. [5][6]

2.3.1 Výhody

- hardwarově nenáročné
- tracking bez pomocných prvků v obraze
- možnost trackovat více objektů
- obnova trackingu zmizelého objektu

2.3.2 Nevýhody

- nepřesnost až ztráta objektu při změně úhlu pohledu
- ztráta identity při podobných objektech

2.4 Active Track

Společnost DJI vyvinula pro svoje bezpilotní prostředky vlastní proprietární trackovací systém jménem Active Track. Bohužel k tomuto systému nejsou veřejně přístupné žádné specifikace. Jediné dohledatelné informace pocházejí z tiskových zpráv, kde se uvádí využívání vizuálního procesoru Movidius Myriad 2 VPU, který se běžně používá například pro hluboké strojové učení. I přes nedostatek informací je Active Track v této práci uveden kvůli jeho skvělým trackovacím schopnostem, kde svou přesností viditelně předčí trackovací algoritmy jako Compressive Tracking a OpenTLD. Podle dostupné video dokumentace, využitého hardware a schopnosti, s jakou tento algoritmus dokáže sledovat objekt zájmu, se lze domnívat, že algoritmus využívá nějakou formu hlubokých neuronových sítí pro sledování objektů. [7][8][9]

2.4.1 Výhody

- tracking bez pomocných prvků v obraze
- perfektní zaměření sledovaného objektu

2.4.2 Nevýhody

- veřejnosti nepřístupný proprietální algoritmus

2.5 Tracking za pomoci detekce

V posledních letech se díky zvyšující se oblibě ve zpracování obrazu začal velmi často používat tracking za pomoci detekce, kde jsou předem známy pozice objektů v obraze. Tyto pozice jsou vygenerovány jinou komponentou, například konvolučními neuronovými sítěmi. Práce pro tracking algoritmus je tím v základu zjednodušena na pouhé rozlišování jednotlivých objektů podle předchozích snímků obrazu. Další základní nadstavbou této myšlenky je přidání algoritmů pro předpovídání pohybu, které se hodí v případě dočasného zákrytu objektu. Za ideálního stavu dokáže

systém obnovit ztracenou identitu sledovaného objektu. Pokročilejší verze těchto trackovacích systémů obsahují i neuronové sítě vylepšující indentifikaci po ztrátě, takže nedochází tak často k nechtěné změně identity.

2.5.1 Výhody

- tracking bez pomocných prvků v obraze
- perfektní zaměření sledovaného objektu
- open source řešení

2.5.2 Nevýhody

- velmi náročné na hardware

3. Strategie řešení

Pro dosažení hlavního cíle práce, tedy vytvoření kvalitního VTS, nelze v dnešní době výkonných počítačů počítat s jednoduchými trackovacími metodami jako Color Filtering používající selekci barev nebo Augmented Reality Marker, kde jsou k identifikaci používány statické vzory. Tyto formy jsou limitovány nutností výskytu určitých prvků v obraze. Částečným řešením tohoto problému je použití tradičních trackovacích algoritmů jako například BOOSTING, MIL, KCF, TLD, MEDIANFLOW nebo GOTURN. Problém těchto algoritmů ovšem spočívá v absenci kontextu obrazu. Algoritmus neví, co sleduje a pouze hledá změny vůči předchozím snímkům, jež porovnává. Ve trojrozměrném světě potom často každá rychlejší změna pozice kamery znamená ztrátu sledovaného objektu, neboť změnou snímaného úhlu se mění jak sledovaný objekt, tak jeho pozadí. Tento problém je zásadní pro plánovaný systém bezpilotního prostředku, protože se očekává aktivní sledování objektu zájmu, čímž se znemožnuje nasazení těchto algoritmů pro tvorbu VTS.

Na základě výše popsaných problémů tato práce navrhuje řešení v dodání kontextu pro trackovací algoritmy, kde je trackovací algoritmus podpořen detekčními systémy udávajícími pozice nalezených objektů v obraze, nezávisle na jejich předchozí poloze. Tímto způsobem se zvýší přesnost trackingu, protože pozice objektu se neodvíjí od nálezu trackovacího algoritmu, ale od nalezené detekce, čímž dochází ke korekci trackovacího algoritmu pro další snímky. Předpokládá se, že využitím metody trackování za pomoci detekce bude nový VTS dosahovat minimálně podobných kvalit jako u proprietárního algoritmu Active Track od společnosti DJI.

Detekce bude postavena na konvolučních neuronových sítích, které zažívají v posledních letech velký rozmach a existuje k nim velké množství materiálů. Dá se předpokládat, že detekce za pomoci konvolučních neuronových sítích, bude extrémně náročná na hardwarové prostředky a bude jí nutno minimálně v rané fázi práce vykonávat na externí výpočetní stanici, tedy mimo bezpilotní prostředek.

V rámci vývoje VTS bude do práce zakomponovaná virtuální realita (dále jen VR), která bude simulovat reálnou scénu ze skutečného světa. Cílem tohoto kroku je

naprogramovat do VR bezpilotní prostředek pohybem simulující svůj reálný vzor. Předpokládá se, že vývoj a ladění chování bezpilotního prostředku na základě výstupních dat z trackování za pomoci detekce, bude zdlouhavá a opakovaná činnost, která by se díky výměně zdrojů či nepřízni okolního prostředí mohla v reálném světě rapidně protáhnout. Výsledkem tohoto kroku bude:

- řešení umožňující zaměnění obrazu z VR za ten skutečný
- vytvoření rozhraní pro simulovaný a fyzický prostředek využitím výstupu trackovacího systému

4. Testovací virtuální realita

Hlavní cíl práce je navržení autonomního systému pro bezpilotní prostředek. Vzhledem k tomuto cíli je nutné mít k dispozici kontrolované prostředí, ve kterém je možné bezpečně otestovat rozhodovací schopnosti prostředku. V raných fází projektu se velmi snadno může stát, že prostředek není stoprocentně pod kontrolou a může se při špatně ošetřeném stavu programu například ztratit z dosahu, někoho zranit nebo se poškodit.

V rámci eliminace těchto černých scénářů byla do projektu zařazena vrstva VR, která je ideálním prostředím pro ladění VTS. Kromě prostředí, ve kterém se nemůže nic stát, je velkou výhodou této formy možnost testování jednotlivých částí na předem definovaném místě a prostoru prakticky okamžitě a opakovaně. Tím odpadá režie na připravení skutečného prostředí, výměnu napájecích zdrojů, zabezpečení prostoru atp. Níže následuje popis jednotlivých komponent testovacího virtuálního prostředí.

4.1 Unity 3D

Pro vytvoření simulace ve VR byl použit engine Unity3D, který nabízí široké spektrum možností pro tvorbu 3D aplikací. Kromě grafického prostředí podporuje vytváření skriptů v jazyce C# k jednotlivým objektům scény, což je ideální pro plánované kontrolované prostředí. [11]

4.1.1 Assets Store

Unity obohacuje svou developerskou scénu o online obchod Assets Store, na kterém se nechají koupit nebo prodat hotové kompatibilní komponenty pro Unity jako textury, 3D modely, animace, zvuky, efekty, kamery a mnohem více. Pokud není třeba žádných speciálních požadavků, je možné jednoduchou scénu poskládat z nabízených komponentů Assets Store. [12]

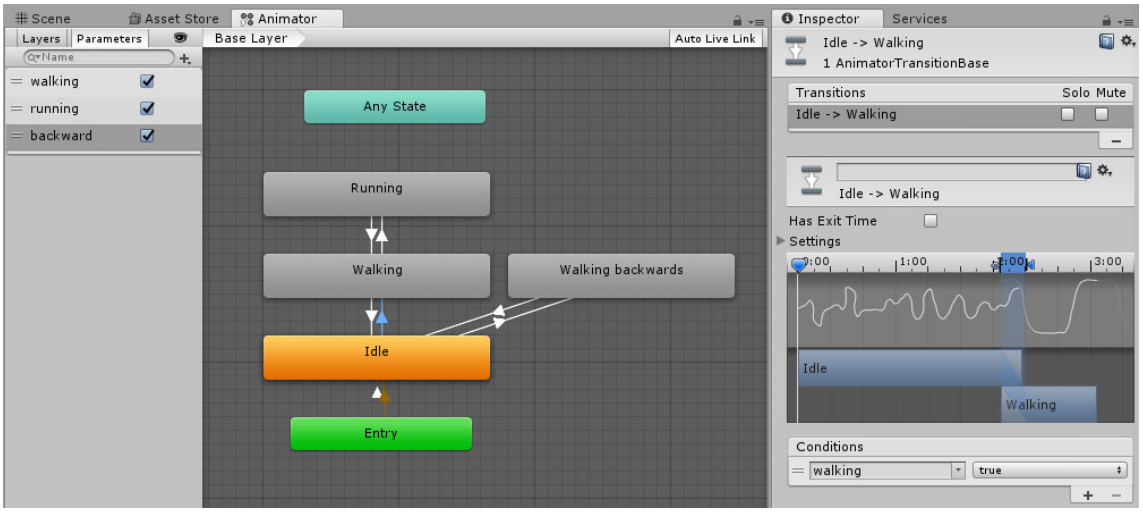
4.1.2 Konfigurace postavy

Obrázek 4.1: Model postavy

Pro navržení VTS byla zvolena jako sledovaný objekt zájmu postava člověka, která je nejlépe dostupná pro testování v reálném světě a může jí vykonávat samotný operátor bezpilotního prostředku.

Skrze Assets Store byly staženy generické animace a modely postav lidí, které budou sloužit jako testovací objekt zájmu pro navrhovaný VTS. Výhodou humanoidních modelů stažených z Assets Store je kompatibilita se zabudovaným Unity Animator rozhraním. Ke každé postavě z kategorie humanoid lze přiřadit libovolná animace ze stejné kategorie. Model tedy neobsahuje animaci přímo v sobě, ale je animován skrze klouby. [10]

Ovládání animace postavy zaštiťuje Unity Animator, který je vyobrazen graficky jako přehled možných stavů. Jednotlivé stavy představují konkrétní pohyb postavy. Přechody z jednotlivých stavů jsou vyhodnoceny dle definovatelných podmínek. Parametry pro vyhodnocení podmínek jsou nastavitelné skrze scripty vázané na objekty scény.



Obrázek 4.2: Přehled stavů - přechod pohybu stání do chůze

V rámci postavy byly definovány stavy:

- Idle (animace – stát na místě)
- Walking (animace – chůze vpřed)
- Walking Backwards (animace – chůze vzad)
- Running (animace – běh vpřed)

V rámci Unity Animator byly definovány Boolean proměnné:

- Walking (default: false)
- Running (default: false)
- Backward (default: false)

Postava se může dostat vždy do konkrétního stavu pouze pokud je s ním její současný stav propojený. Například, aby se postava dostala do stavu Running (běh), musí nejprve být ve stavu Walking (chůze). Tato logika má své opodstatnění v ukončení pohybu běh a přechodu do pohybu chůze. Pokud by byl stav běhu propojen přímo se stáním, či chůzí vzad, došlo by tak k nerealistické extrémní změně pohybu, což by mohl být nechtěný rušící element pro VTS.

Předávání parametrů pro vyhodnocení pohybu je řešeno nastavováním akce podle stisknutých kláves W, S a levý Shift.

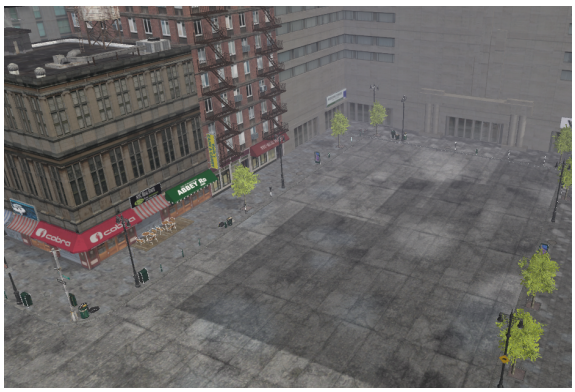
```
if (Input.GetKey(KeyCode.W)) {
    setMovement("walking", true);
    if (Input.GetKey(KeyCode.LeftShift)) {
        setMovement ("running", true);
    }
}
if (Input.GetKey(KeyCode.S)) {
    setMovement ("backward", true);
}
```

Aby mohla postava přejít do stavu Running, musí se tedy držet klávesa levý shift pro běh podmíněně s klávesou W, která slouží pro chůzi.

4.1.3 Konfigurace prostředí

Pro navrhování samotného VTS bylo třeba zvolit scénu, která by se svým vzhledem podobala skutečnému prostředí. V konečném důsledku je totiž třeba testovat bezpilotní prostředek ve světě bohatém na detaily, aby se co nejvíce usnadnil přechod z virtuálního světa do toho skutečného. Ideálně by měl být virtuální svět natolik autentický, aby stačilo systému zaměnit na vstupu virtuální obrazový materiál za skutečný a jeho chod zůstal stejný jako v simulaci.

Z výše popsaných důvodů byla z dostupných modelů v Assets store poskládána část fiktivního města, které obsahuje budovy s obchody, chodníky, stromy a další modely pro navýšení úrovně autenticity prostředí. V rámci plánovaného pohybu vícero potencionálních objektů ke sledování, byla scéna stavěna jako veřejná plocha simulující náměstí.



Obrázek 4.3: Virtuální náměstí

4.1.4 Simulace rušného náměstí

Aby bylo možné testovat VTS, bylo potřeba do scény vložit další autonomní humanoidní postavy, které měly za úkol záměrně pracovat jako rušivé elementy při sledování postavy zájmu. Tento způsob testování měl prověřit, zdali nedochází k nechtěné záměně postavy zájmu za jinou okolní humanoidní entitu.

Vloženým autonomním postavám byla naprogramována jednoduchá strategie pohybu, kde je postava náhodně natočena a vložena na náhodné souřadnice omezené pozicí a rozměry náměstí. Postavy jsou vloženy s nastaveným automatickým pohybem chůze vpřed. Když postava odejde z vymezené plochy, proces náhodného vložení se opakuje.

Obrázek 4.4: Virtuální náměstí s autonomními postavami

5. Detekce

Lidé se podívají na obraz a okamžitě je jim jasné, jaké objekty se zde nachází a co znamenají. Vizuální systém člověka je tak rychlý a přesný, že umožňuje vykonávat relativně složité úkony jako třeba řízení auta s minimálním úsilím. Podobně rychlé a přesné vyhodnocování donedávna nebylo na běžně dostupných zařízeních možné. Detekční systémy, řešící tento problém, sice existovaly, ale buď nedosahovaly použitelné přesnosti v detekci nebo byly pro plynulé zpracování snímků i na nejvýkonnějších spotřebních strojích moc náročné. Revoluci v tomto odvětví způsobily až detekční systémy, lišící se tím, že dokáží rozpoznávat objekty v jednom průchodu vyhodnocení obrazu, což je několikanásobně rychlejší než u dosavadních systémů, které provádějí tuto činnost i několiksetkrát.

Počítač vidí obraz jako pole hodnot jednotlivých pixelů. Jejich počet se odvíjí od šířky a výšky obrazu a barevného modelu. Vezme-li se v úvahu RGB bitmapový obrázek o rozlišení 512x512 a 24bitové hloubce barev, tak jeho reprezentací jako pole bude 786432 (512x512x3) čísel nabývajících hodnot od 0 do 255, které popisují intenzitu pixelu v tomto bodě. Tato čísla jsou pro člověka při klasifikaci obrazu nic neříkající, ale jsou to jediné vstupy, které má počítač k dispozici. Aby mohl rozpoznávat a rozlišovat objekty v obraze, bylo ho nutné naučit, jak zjistit jedinečné konkrétní vlastnosti objektu. Když se například podívá člověk na obraz kočky, okamžitě vidí snadno identifikovatelné rysy jako čtyři nohy, tlapky atp. Podobného principu využívají umělé neuronové šítě typu Convolutional Neural Networks (dále jen CNN), které aplikováním různých filterů na části obrazů vyhledávají a extrahují prvky jako okraje, křivky, a ty pak kombinacemi spojují do větších celků reprezentujících další složitější prvek, např. tlapku kočky. Výsledek CNN je pak předán Support vector machines (dále jen SVM) klasifikátoru, který určí pravděpodobnost s jakou se v obraze nachází konkrétní objekt. [13][15][16]

Kapitola popisuje několik nejčastěji používaných detekčních systémů založených na konvolučních neuronových sítích. Cílem tohoto přístupu je zhodnotit použitelnost jednotlivých systémů v praxi. Poslední podkapitola shrnuje zjištěné informace a stanovuje systém pro výrobu VTS. V dovětku každého systému je krátké shrnutí hodnotící systém dle zjištěných informací jako počet snímků za vteřinu, přesnost dle metriky mAP na datasetu Pascal VOC 2007[14] aj.

5.1 Region-based Convolutional Neural Networks

Aby CNN mohla lokalizovat objekt v obraze pro klasifikaci, je bez pomocných algoritmů potřeba vzít na vědomí veškeré možnosti výskytu objektu. To znamená zkusit všechny možné lokace, výšku a šířku. Tento proces je velmi neefektivní a výpočetně náročný, protože každý vyzkoušený segment musí opět projít náročnými výpočty konvoluční sítě.

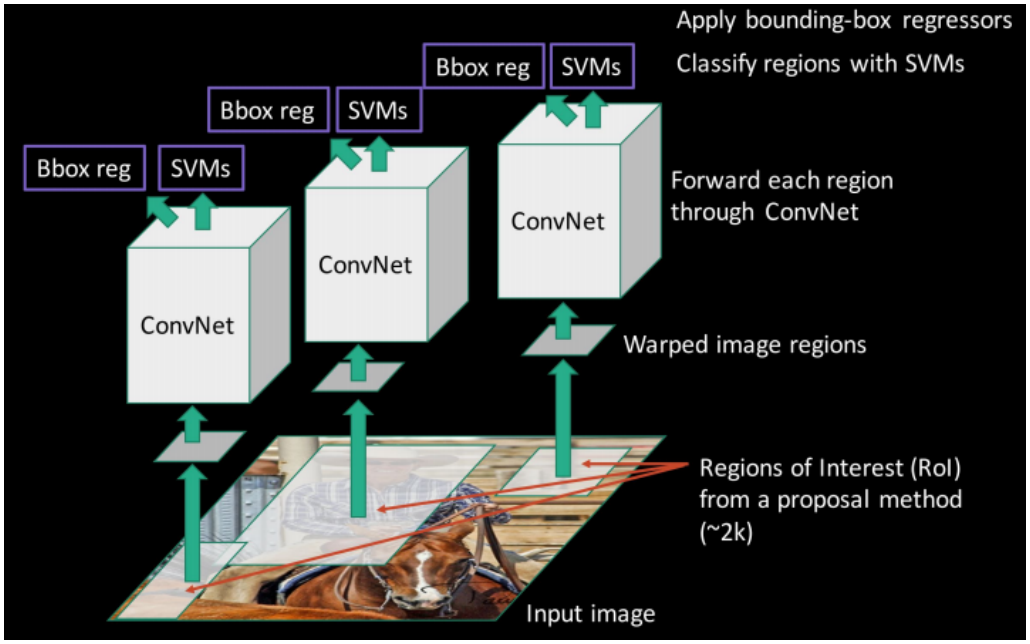
Obrázek 5.1: Testování všech možných lokací objektu

5.1.1 R-CNN

Aby se snížil počet pokusů pro nalezení objektu, byly do detekčních systémů rozpoznávajících obraz použity algoritmy typu Region Proposal (dále jen RP), kde se na obraz aplikuje tzv. agnostický klasifikátor, používající několik grafických filtrů segmentujících obraz. Agnostický klasifikátor tedy pouze určí, kde se v obraze nachází nějaký objekt. Tímto způsobem se nechá počet možností zredukovat pouze na několik stovek až tisíců potencionálních objektů, které jsou dále jednotlivě

zpracovány CNN. Po průchodu CNN se výsledek předá SVM klasifikátoru, který už určuje pravděpodobnost konkrétních tříd. [17][18]

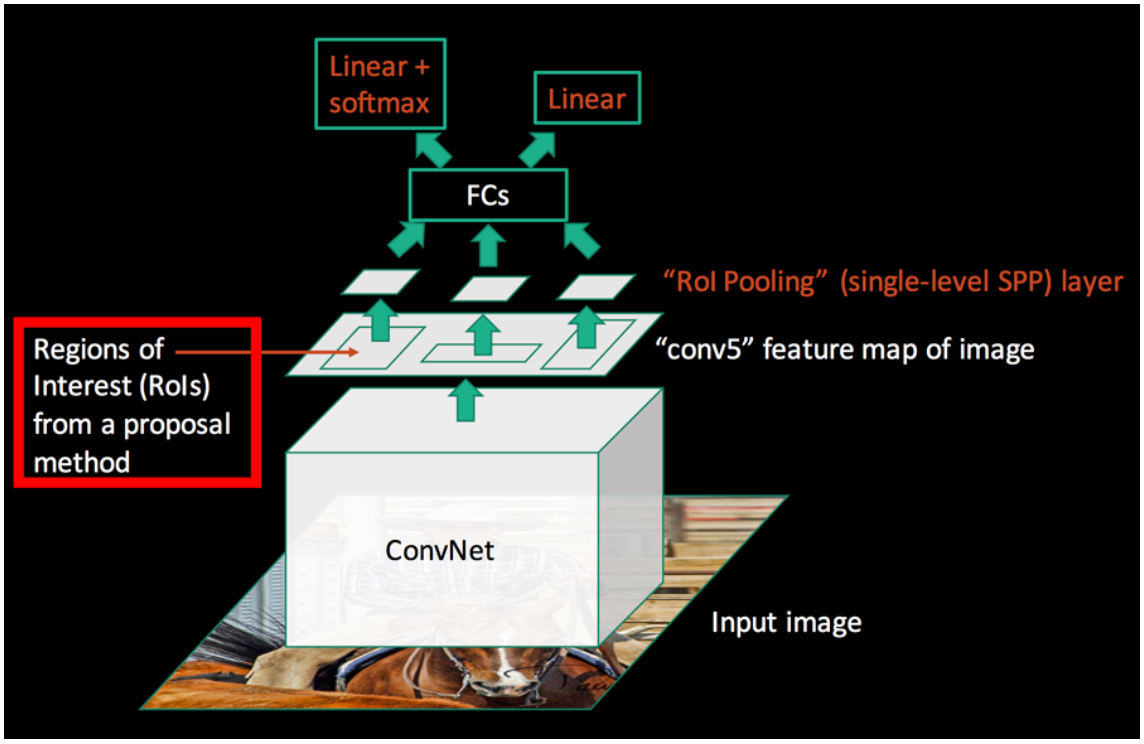
Obrázek 5.2: Region proposal (Selective Search)



Obrázek 5.3: R-CNN model

I přes značnou redukci možností výskytu objektu je tento systém velmi pomalý a na současně dostupném hardware generuje výstup kolem 50 vteřin. [18][19]

5.1.2 Fast R-CNN

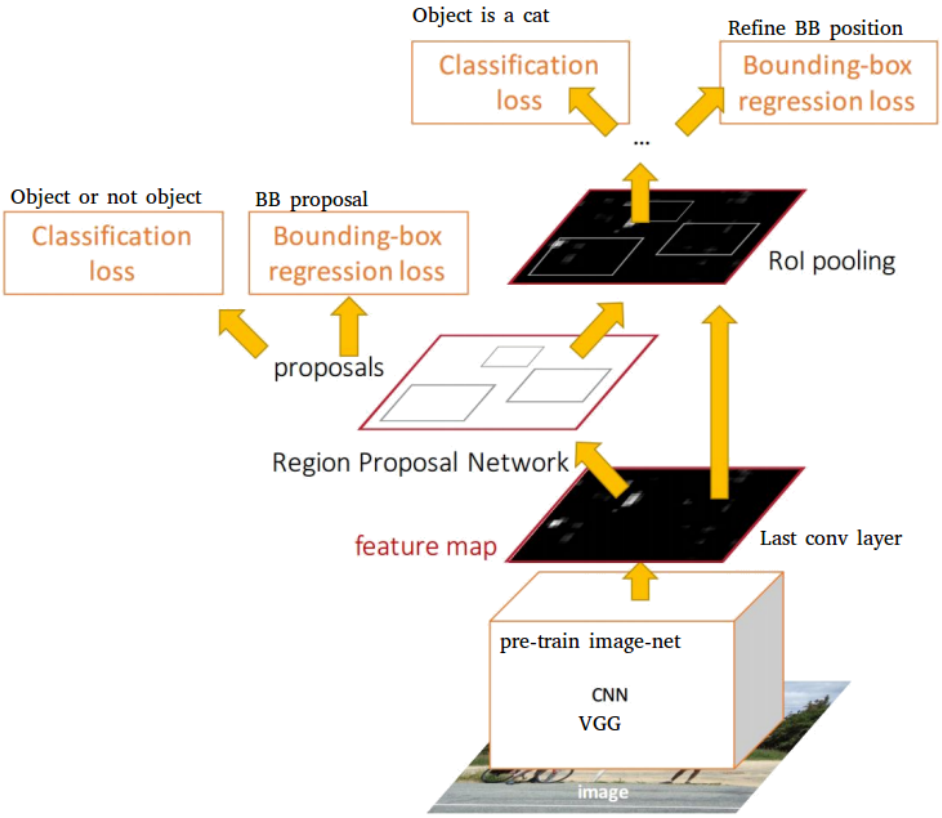


Obrázek 5.4: Fast R-CNN model

Oproti svému předchůdci (R-CNN) neposílá každý potenciální objekt k náročné extrakci prvků (hrany, křivky, ...) k vyhodnocení skrze CNN, ale nejprve vytvoří tzv. *feature map* z celého obrazu, která je reprezentací původního obrazu po průchodu CNN. Ta je pak dále opětovně použitelná pro navrhování regionů a klasifikaci objektů. Tímto způsobem odpadlo opakované zpracování, neboť R-CNN tuto proceduru opakovalo u každého potencionálního objektu. Výsledek se tedy de facto zahodil a pro potencionální objekty z blízkého okolí se musel vypočítat částečně znovu. [18][19][20]

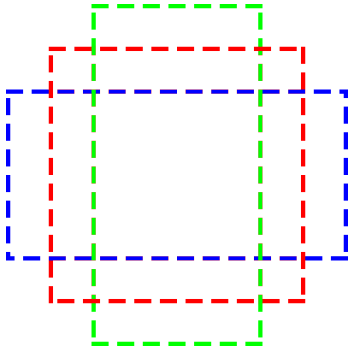
Touto změnou v návrhu se systém rapidně zrychlil oproti předchozí verzi. Výsledek dokáže vygenerovat za 2 vteřiny. [18][19]

5.1.3 Faster R-CNN



Obrázek 5.5: Faster R-CNN model

Testováním se ukázalo, že algoritmy typu RP jsou v celkovém výpočtu brzdícím faktorem celého systému. Faster R-CNN nahrazuje RP algoritmy za velmi malou konvoluční síť zvanou Region Proposal Network predikující menší množství regionů ke zpracování. V každém regionu se vygenerují tři tzv. Anchor Boxes, které zabírají sledovanou část obrazu v poměrech 1:1, 2:1 a 1:2. Na všechny tři boxy se pak použije Bounding Box Regression metoda upřesňující pozici případného objektu a agnostický klasifikátor určující pravděpodobnost výskytu objektu. Výhodou těchto Anchor Boxes je, že se ve vybraném obrazu může vyskytovat a překrývat více objektů, které by regresní metoda při aplikaci na sledovanou oblast mohla zanedbat. Tím, že se zkouší více poměrů, má postup větší šanci určit správnou pozici objektů, i když se například překrývají. [18][19][21][22]



Obrázek 5.6: Anchor Boxes

Nahrazením tradičních RP metod se celý systém několikanásobně zrychlil. Vygenerování výstupu nyní trvá 0,2 vteřiny. [18][19]

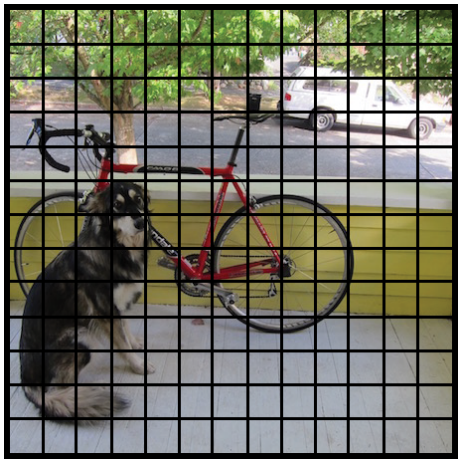
Bohužel ani po všech úpravách původního systému není možné aplikovat systém pro zpracování snímků v reálném čase. Díky latenci 0,2 vteřiny tedy dokáže systém poskytnout pouhých pět snímků za vteřinu.

5.2 Regression-based object detectors

Všechny výše popsané systémy využívaly k detekci objektu klasifikaci, pro kterou bylo nutné vyhledat v celém obraze pravděpodobné objekty, které byly dále postoupeny regresním metodám upřesňujícím polohu. Tyto procedury se ale při současně dostupném hardware ukázaly jako nepoužitelné pro vývoj VTS, neboť nedosahovaly dostatečného počtu snímků za vteřinu. Nicméně se v poslední době objevilo několik metod, které problematiku detekce řeší jako regresní problém. Zásadním rozdílem je, že tyto systémy nemají zabudované RP algoritmy, nýbrž používají předdefinovanou mřížku a CNN, která dokáže zároveň určit pravděpodobnost objektu v konkrétní buňce a míru její příslušnosti jednotlivým třídám klasifikátoru.

5.2.1 Yolo

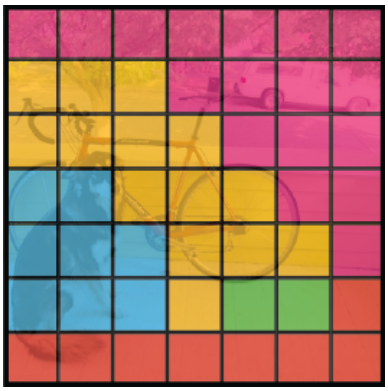
*Y*ou only look once (dále jen Yolo) v prvním kroku rozdělí obraz do SxS mřížky. Každá buňka mřížky predikuje N pravděpodobných pozic s výškou a šířkou, které definují potencionální umístění objektů v obraze. Yolo zároveň každý z těchto ob-



Obrázek 5.7: Yolo - mřížka

jektů ohodnotí s jakou jistotou se na pozici skutečně nachází objekt. Objekt buňky s nejvyšší jistotou se předá ke zpracování klasifikátorem, který určí s jakou pravděpodností spadá objekt do jednotlivých tříd.

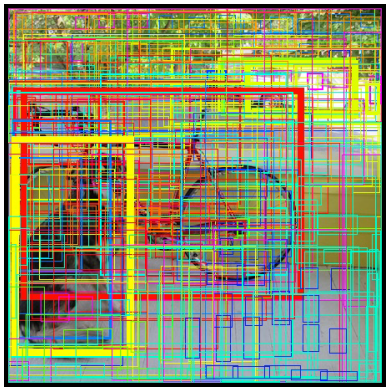
Obrázek 5.8: Yolo - ohraničení potencionálních objektů



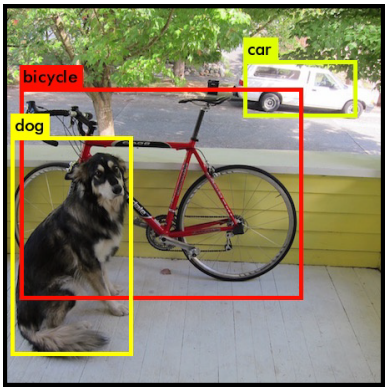
Obrázek 5.9: Yolo - pravděpodobnostní mapa tříd objektů

Součinem pravděpodobnosti výskytu objektu s pravděpodobností příslušnosti do nějaké třídy se získá finální pravděpodobnost výskytu objektu v dané buňce. Výsledkem celé operace je zredukování počtu objektů eliminací finálních hodnot nižších než nastavená hranice.

Absence hledání regionů a fixní počet míst pro hledání objektů umožnily tomuto způsobu další několikanásobné zrychlení detekce, které s sebou ovšem nese určitá negativa. Buňka dokáže predikovat pouze jednu třídu, což limituje detekci malých objektů objevujících se ve skupině jako například pták v hejnu.



Obrázek 5.10: Yolo - Pravděpodobnosti objekt x klasifikace



Obrázek 5.11: Yolo - finální výstup po eliminaci malých hodnot

Yolo

První verze Yolo vznikla v roce 2015. Yolo představilo nový přístup k detekci a otevřelo novou kapitolu zpracování obrazu v reálném čase. Vůči ostatním detekčním systémům té doby bylo extrémně rychlé, ale také méně přesné. Z principu, ze kterého vycházelo, mělo relativně problémy s malými objekty a objekty s neočekávaným poměrem stran objektu. [24]

Díky novému přístupu dokázal systém zpracovat 45 snímků za vteřinu při přesnosti 63.4mAP, což otevřelo dveře pro zpracování obrazu v reálném čase. [25]

YoloV2

V rámci zdokonalování přišly na konci roku 2016 autoři Yolo s novou verzí tohoto detekčního systému, která vylepšila nejen přesnost, ale i rychlost. Nové YoloV2 přišlo se změnou rozlišení vstupních dat z dříve používaných 256x256 na vyšší, což umožnilo modelu lépe pracovat s obrazy ve větším rozlišení. Další markantní změnou byla implementace formy Anchor Boxes, známé z Faster R-CNN, která umožnila lépe a těsněji ohraničovat objekty. [25]

Tyto a další změny pomohly dosáhnout přesnosti 78.6mAP při 40 snímcích za vteřinu při vstupním obraze o rozlišení 544x544. Se vstupním obrazem o rozlišení 416x416 dosahuje 76.8mAP při 67 snímcích za vteřinu. [25]

Tiny Yolo

Autoři Yolo využili koncept systému a vytvořili model na bázi YoloV2 s názvém Tiny Yolo. Model obsahuje veškeré nové aspekty YoloV2, ale jeho model obsahuje méně konvolučních vrstev. Tato redukce umožnila zpracovávat 207 snímků za vteřinu při přesnosti 57.1mAP, což je téměř jako původní verze Yolo. Toto extrémní zrychlení je i přes ztrátu přesnosti perfektní variantou pro zařízení, kde není dostupný masivní výkon. [23]

5.2.2 SSD

Single Shot MultiBox Detector (dále jen SSD), podobně jako Yolo, dělí obraz do mřížky. V jednotlivých buňkách se předpovídá výskyt objektu a jeho posun vůči Anchor boxes, podobně jako u Faster R-CNN detekčního systému, aby se našlo co nejpřesnější ohraničení a povedlo se najít i objekty uvnitř větších objektů. Ke každému upřesněnému objektu jsou systémem zároveň předpovězeny příslušnosti jednotlivých tříd. Na rozdíl od Yolo systémů, SSD je složena z několika progresivně zmenšujících se konvolučních vrstev, na kterých probíhá tato detekce. Tento způsob umožňuje detekci objektů menších velikostí, což je například problém Yolo systémů, které generují detekce pouze na jedné konvoluční vrstvě. Nevýhodou tohoto přístupu je, že tento způsob generuje více detekcí, což je výpočetně náročnější. [26][27]

Nejlepší výsledek tohoto systému má SSD se vstupními obrázky o rozlišení 512x512, se kterými dosáhl 76.8mAP při 19 snímcích za vteřinu. Jeho slabší varianta se vstupními obrázky 300x300 dosáhla 74.3mAP při 45fps, což z něj dělá vhodného kandidáta pro vytvoření VTS. [26]

5.3 Porovnání detekčních systémů

Hlavním faktorem pro výběr detekčního systému byla rychlost, neboli počet snímků za vteřinu, kde je potřeba, aby vybraný systém dosahoval kolem třiceti snímků za vteřinu, což je potřebné pro plynulé zpracování videa. Sekundárním faktorem byla přesnost detekcí měřená v metrice mean average precision na datasetu Pascal

VOC 2007.[14]

5.3.1 Měření

Následující měření probíhala na grafické kartě Nvidia Geforce GTX Titan X. Dataset pro měření: Pascal VOC 2007 [14].

Detekční systém	Mean average precision	Počet snímků za vteřinu
R-CNN	66.0	0,02
Fast R-CNN	70.0	0,5
Faster R-CNN	76.4	5
Yolo	63.4	45
YoloV2 416	76.8	67
YoloV2 544	78.6	40
Tiny Yolo	57.1	207
SSD 300	74.3	45
SSD 512	76.8	19

[24][26]

5.3.2 Vyhodnocení

Pro vytvoření VTS byl vybrán detekční systém Yolo s Tiny Yolo neuronovým modelem. Model je sice výrazně méně přesný, ale vůči ostatním výše popsaným možnostem zpracování je zdaleka nejrychlejší. Hlavní argument pro toto rozhodnutí byla idea ve finální přesunutí výpočetních algoritmů přímo na bezpilotní prostředek, kde je v současnosti k dispozici podstatně méně výkonu.

6. Tracking

V rámci práce se na základě průzkumu dostupných řešení povedlo nalézt trackovací systémy pro sledování nalezených detekcí. Bohužel většinu těchto systémů nešlo použít z důvodu uzavřeného kódu nebo použitého programovacího jazyka nekompatibilního s detekčním systémem Yolo. Z těchto důvodů nebyla provedena rozsáhlá analýza a níže jsou zmíněny pouze dva potencionální trackovací systémy pro VTS.

6.1 SORT

Simple Online and Realtime Tracking (dále jen SORT) je trackovací systém sloužící k identifikaci nalezených detekcí. V práci je zmíněn z důvodu, že je předchůdcem vybraného systému pro tvorbu VTS a přímo z něj vychází.

Trackovací systém využívá v zásadě dvě známé rudimentární techniky: Kalman filter a Hungarian algorithm. Kalman filter se v systému používá pro predikci pohybu, což je vhodné nejen pro lepší sledování, ale i pro stavy, kde je sledovaný objekt dočasně zakryt. Hungarian algorithm řeší asociaci jednotlivých detekcí ke svým identitám. [30]

Největší silou algoritmu je, že dokáže zpracovat 60 snímků za vteřinu v testovací scéně s mnoha objekty ke sledování. Jeho jednoduché zpracování ovšem způsobuje časté ztráty nebo záměny identit detekovaných objektů. [28]

6.2 Deep SORT

Tracking with a Deep Association Metric (dále jen Deep SORT) je nástupcem výše uvedeného systému, který zásadně eliminuje ztráty a záměny identit nahrazením Hungarian algorithm za konvoluční síť, která se za běhu učí informace o vzhledu sledovaného objektu. Díky tomuto vylepšení dokáže systém mnohem lépe reindetifikovat objekty jevící se jako dočasně zakryté nebo zcela zmizelé. Dalším kladem vylepšení je snížení záměny identit až o 45%. [28]

Nevýhodou použití tohoto algoritmu je nutnost naučení na konkrétní typy objektů. V rámci projektu autoři dali k dispozici síť naučenou na lidské postavy a zveřejnili zdrojové kódy pro učení. [29]

Výměna asociačního algoritmu za konvoluční si vybrala daň v podobě potřebného výkonu, který klesl zhruba o třetinu - 40 snímků za vteřinu v testovací scéně. [28]

7. Implementace VTS

7.1 Využité technologie

Na základě výše uvedených analýz byl VTS sestaven z následujích komponent.

7.1.1 Tiny Yolo

Jako detekční systém byl pro práci vybráno Yolo s redukovaným modelem Tiny Yolo, který dokáže s přijatelnou přesností klasifikovat objekty v obraze v reálném čase. Tento detekční systém byl postaven na frameworku pro neuronové sítě Darknet. [23]

7.1.2 Darkflow

Darkflow je založený na open source frameworku Darknet pro stavění neuronových sítí, který byl napsán v programovacích jazycích C a CUDA C. Darkflow je jeho implementace pro populární framework TensorFlow. V práci je použit kvůli možnosti psát v jazyce Python, ve kterém jsou naprogramovány další součásti této práce. [32][33]

7.1.3 Deep SORT

Deep SORT je trackovací systém založený na trackování za pomoci detekce. Systém je napsaný v programovacím jazyce Python a kombinuje základní výpočetně rychlé techniky s neuronovou sítí, která pomáhá sledovat objekty po delší časový úsek a zároveň snižuje počet nechtěných změn identit při překryvu podobných objektů. [31]

7.1.4 OpenCV

Open Source Computer Vision Library je svobodná a otevřená multiplatformní knihovna pro manipulaci s obrazem. Je zaměřena především na počítačové vidění a zpracování obrazu v reálném čase. OpenCV je implementovaná pro programovací

jazyky C++, Java a Python. V práci je použita například pro získání obrazu z kamery, transformaci kamery a vykreslování ohraničení detekovaných a trackovaných objektů. [34]

7.1.5 Tracking with Darkflow

Tracking with Darkflow (dále jen TwD) zapouzdřuje výše popsané technologie do jednoho celku a poskytuje konfigurační parametry pro jednotlivé komponenty. [35]

7.2 Inicializace

Do projektu byl naklonován GIT repozitář TwD s jeho upravenou verzí repozitářů Darkflow a Deep SORT. Zdrojový kód je, stejně jako další dále popsané komponenty, psán v jazyce Python, aby bylo celkové řešení jednotné. Zdrojový kód komponenty Darkflow je v repozitáři upraven pro použití trackovacích algoritmů Deep SORT a SORT, které rozšiřují detekční systém o identifikaci těchto detekcí.

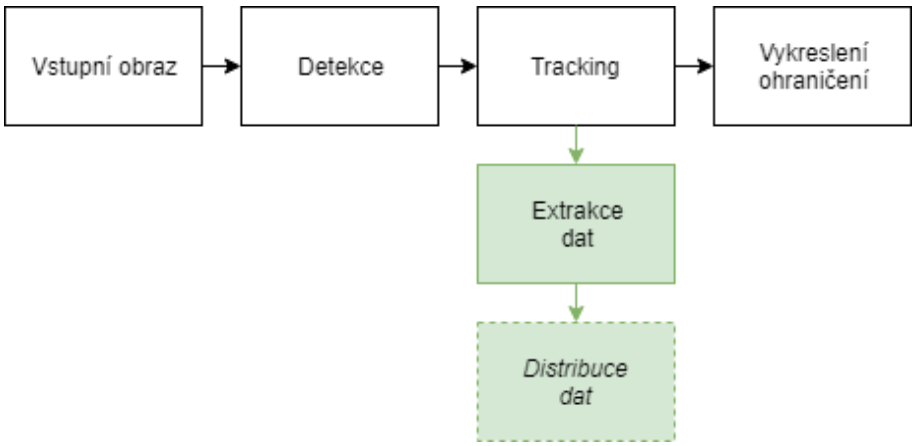
TwD obsahuje několik inicializačních parametrů, které je třeba nastavit pro správný chod:

- **demo** - (string|int) cesta k souboru nebo id zachytávacího zařízení
- **model** - (string) cesta ke konfiguračnímu souboru Yolo
- **load** - (string) cesta k kompilovaným vahám Yolo
- **threshold** - (float) prahová hodnota (0 - 1) eliminující hodnoty s nízkou pravděpodobností výskytu detekce
- **gpu** - (float) hodnota (0 - 1) určující kolik výkonu se má využít z GPU, 0 znamená výpočet na CPU
- **track** - (bool) trackování detekcí
- **trackObj** - (string[ ]) pole názvů tříd k detekci objektů z natrénované množiny definované v konfiguračním souboru Yolo

- **saveVideo** - (bool) uložení obrazového výstupu po zpracování
- **BK_MOG** - (bool) substrakce pozadí
- **tracker** - (string) název trackovacího systému
- **skip** - (int) počet snímků k přeskočení pro zvýšení výkonu
- **csv** - (bool) zápis pozic trackovaných detekcí do souboru
- **display** - (bool) vykreslení zpracovaného obrazu na obrazovku

7.3 Modifikace

Koncept programu TwD spočívá v demonstraci kombinovaného řešení, kde se přijme na vstupu obraz a do něj vykreslí očíslované ohraničení sledovaných objektů. Pro vytvoření VTS bylo nutné pozměnit další části kódu pro získání těchto číslovaných ohraničení objektů, se kterými se dále pracuje při předávání dat do bezpilotního prostředku. Aby bylo možné s těmito objekty pracovat, bylo nutné upravit kód zodpovědný za vykreslování ohraničení do obrazu a zpřístupnit tak pole s daty o jednotlivých objektech pro další části aplikace (viz obr. 7.1). Z tohoto důvodu byla v metodě, kde se generuje ohraničení, vytvořena globální proměnná schraňující data o sledovaných objektech, aby je šlo dále distribuovat napříč aplikací.

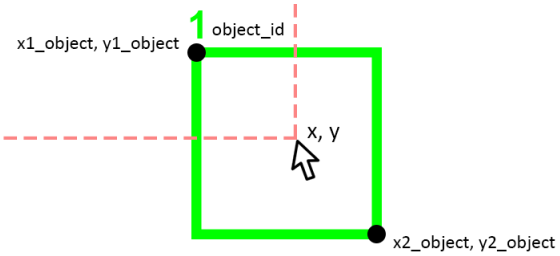


Obrázek 7.1: Schéma - modifikace kódu

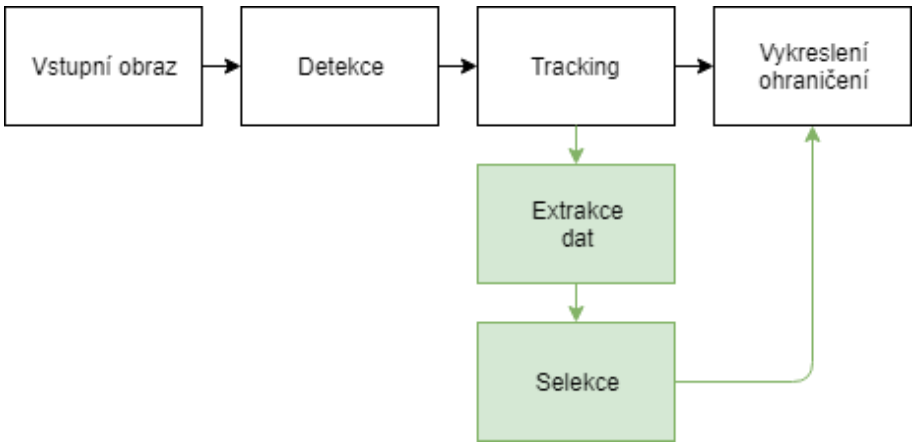
7.4 Selekce

Jak je patrné ze schématu 7.3, pro selekci konkrétního objektu bylo potřeba využít nově zpřístupněná data, díky kterým bylo možné získat pozici, šířku, výšku a přidělené identifikační číslo objektu. Vybrání objektu probíhalo za pomoci OpenCV knihovny, která zaštiťuje v projektu zobrazování obrazu v okně. Tomuto oknu bylo nastaveno zachytávání událostí myši a každé kliknutí levým tlačítkem do obrazu porovnávalo, zdali se pozice kliknutí vyskytuje uvnitř nějakého ohraničeného objektu. Pokud k takovému případu došlo, ohraničení bylo pro přehlednost obarveno zelenou barvou a id objektu bylo nastaveno do globální proměnné, čímž se stal objekt sledovaným viz 7.2.

```
...
if (x > x1_object) and (x < x2_object) and (y > y1_object) and (y < y2_object):
    selected_object_id = object_id
...
```



Obrázek 7.2: Náčtrt vybírání objektu



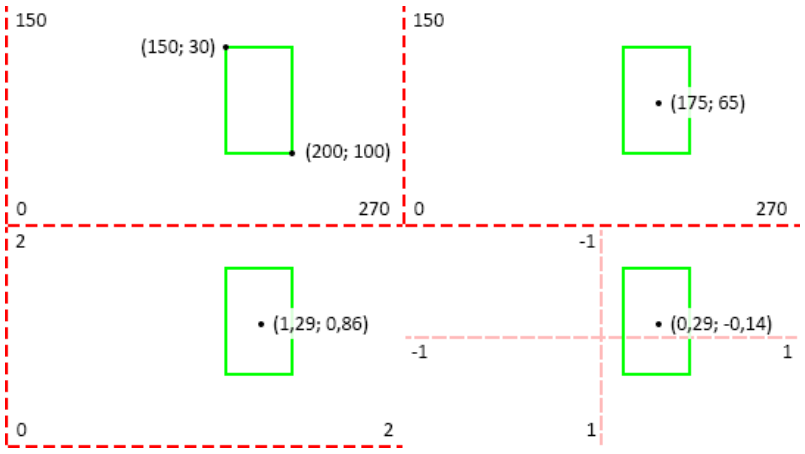
Obrázek 7.3: Schéma - selekce

7.5 Normalizace

Obdélníkové ohraničení sledovaného objektu je reprezentováno dvěma protilehlými body, jejichž souřadnice jsou uvedeny v obrazových bodech. Tyto jednotky ovšem nedávají pro bezpilotní prostředek, který potřebuje směrový vektor, smysl. Jak je patrné z náčrtu 7.4, pro sledování objektu stačí pouze střed obdélníkového ohraničení a proto byla data normalizována formou:

- výpočet x a y souřadnic středu obdelníku
- přepočítání souřadnic do rozsahu hodnot (0; 2)
- posunutí výsledku o hodnotu -1

```
def calculate_dimension_normalized_units(first_value, second_value, max_value):
    middle_object_value = first_value + ((second_value - first_value) / 2)
    return ((2 / max_value) * middle_object_value) - 1
```

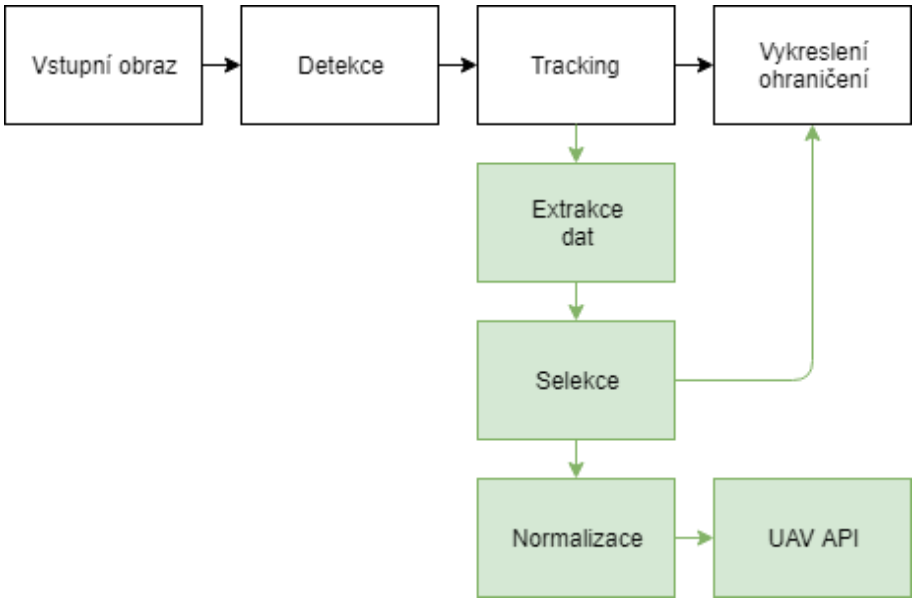


Obrázek 7.4: Normalizace souřadnic

Tímto způsobem byla data normalizována do rozsahu (-1; 1), kde hodnota 0 reprezentovala střed obrazovky v dané ose. Výhodou tohoto přepočítání je, že výsledná hodnota se již dá použít přímo pro směr a rychlost pohybu. Pokud je sledovaný objekt na kraji obrazovky, rychlost se bude blížit v závislosti na směru k -1 nebo 1. Naopak, pokud bude objekt téměř ve středu obrazovky, systém vypočítá téměř nulovou rychlost ve směru ku středu, čímž se lze vyhnout okamžitému zrychlení bezpilotního prostředku, který by v důsledku mohl znamenat ztrátu plynulého pohybu

a rychlou změnou v obraze i ztrátu sledovaného objektu.

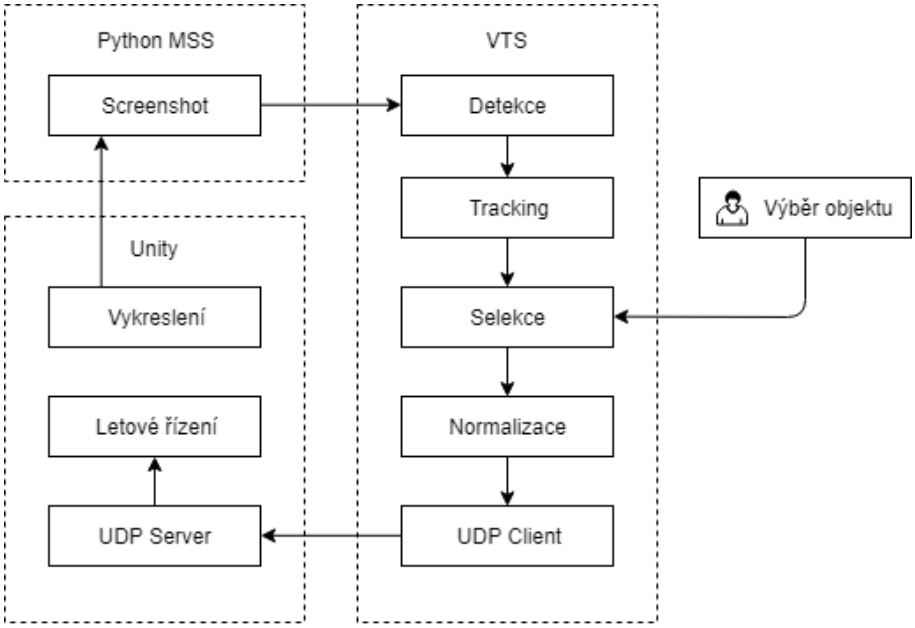
Protože bezpilotní prostředek získává data pouze z obrazu, bylo udržování vzdálenosti od sledovaného objektu možné řešit pouze podle výšky obdélníkového ohraničení objektu. Tato výška byla normalizována jako poměr výšky ohraničení k výšce obrazu.



Obrázek 7.5: Schéma - normalizace

Takto normalizovaná data jsou dále poskytnuta řídícím prvkům VTS, které zajišťují pohyb bezpilotního prostředku viz 7.5.

7.6 Virtuální svět



Obrázek 7.6: Schéma zapojení ve virtuálním světě

7.6.1 Komunikace

Propojení vytvořené VR v Unity engine s VTS s sebou přineslo dva významné problémy. První problém spočíval v předání obrazu do TwD a druhým problémem byl zpětné předávání dat do Unity.

Získání obrazu

Předávání obrazu z Unity do TwD bylo vyřešeno vytvořením jednoduché třídy ScreenCapture, která slouží ke snímání obrazovky za pomoci knihovny Python MSS. [36] Třída v konstruktoru přijímá nově vytvořené inicializační parametry:

- **capture_screen_width** - (int) šířka obrazovky
- **capture_screen_height** - (int) výška obrazovky
- **capture_screen_offset_top** - (int) odsazení odshora
- **capture_screen_offset_left** - (int) odsazení zleva

- **capture_screen_scale** - (float) měřítko

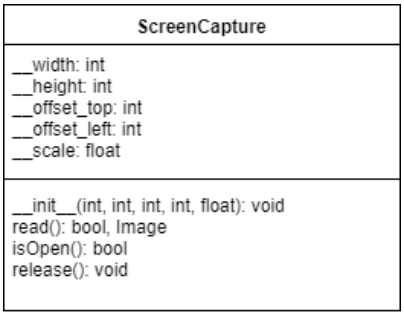
Inicializační parametr demo byl v kódu rozšířen o nabytí dalšího stavu: záporné číslo/id zachytávacího zařízení. Pokud je parametr demo nastaven na tuto hodnotu (< 0), vytvoří se objekt ScreenCapture namísto VideoCapture z knihovny OpenCV. Pokud je do konstruktoru třídy VideoCapture předána kladná celočíselná hodnota, tak se OpenCV pokusí otevřít zachytávací zařízení (např. webkameru) s tímto identifikačním číslem. Pokud je předán řetězec, TwD zjistí, zdali uvedený název a cesta souboru existuje.

```
...
file = self.FLAGS.demo
if isinstance(file, int):
    if file < 0:
        from visual_tracking_system.screen_capture import ScreenCapture
        camera = ScreenCapture(
            self.FLAGS.capture_screen_width,
            self.FLAGS.capture_screen_height,
            self.FLAGS.capture_screen_offset_top,
            self.FLAGS.capture_screen_offset_left,
            self.FLAGS.capture_screen_scale,
        )
    else:
        camera = cv2.VideoCapture(file)
else:
    assert os.path.isfile(file), \
    'file {} does not exist'.format(file)
    camera = cv2.VideoCapture(file)
...
```

Vzhledem ke struktuře zbytku kódu TwD bylo nutné implementovat použité metody VideoCapture, aby bylo možné objekty jednoduše zaměnit a program mohl bez problémů pracovat s instancí třídy ScreenCapture jako s VideoCapture. Třída ScreenCapture tedy ve skutečnosti pouze napodobuje pojmenováním metod funkčnost VideoCapture.

- __init__ - konstruktor přijmající šířku, výšku, odsazení odshora, odsazení zleva a škálu obrazu
- isOpen - návratová hodnota trvale nastavena na bool hodnotu True

- release - nevrací nic (void)
- read - vrací tuple ((bool) obraz dostupný a (Image) nasnímaný obraz obrazovky)



Obrázek 7.7: Class diagram - ScreenCapture

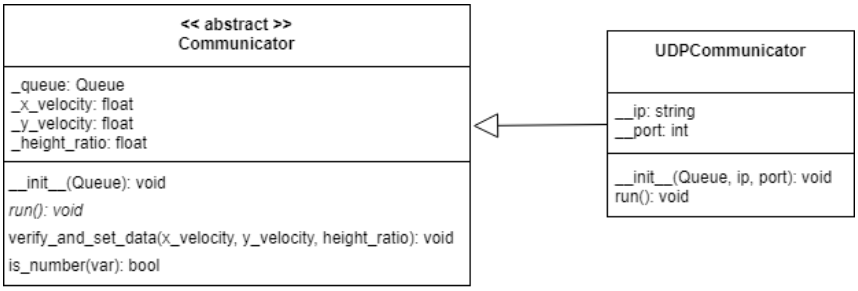
Posílání dat

Druhým zmiňovaným problémem byla komunikace s Unity prostředím, kde bylo třeba předávat vypočítaná data z VTS. Vzhledem k použití programovacího jazyka C# ve VR bylo rozhodnuto, že se bude komunikovat skrze transportní síťovou vrstvu TCP/IP. Protože se aplikace nesmí zdržovat zasíláním zpráv ztracených po cestě a i v reálném prostředí se mohou ztratit nějaká data, byl vybrán jako transportní protokol User Datagram Protocol (dále jen UDP).

V rámci předpokladu, že implementace pro komunikaci s bezpilotním prostředkem bude v podobném duchu, byla vytvořena abstraktní třída jménem Communicator, zaštiťující základní funkcionalitu. Kvůli plynulosti generování obrazového výstupu, který je klíčový pro VTS, bylo potřeba odesílat datagramy tak, aby nezdržovaly hlavní vlákno. Z tohoto důvodu dědí abstraktní třída třídu Thread z balíčku threading. Po inicializaci této třídy, nebo jejich potomků, dojde k založení nového nezávislého vlákna. Abstraktní třída přijímá jako parametr objekt Queue patřící také do balíčku threading, který dovoluje bezpečně předávat data napříč vlákny.

Pro komunikaci mezi TwD a Unity byla vytvořena třída UDPCommunicator dědící z abstraktní třídy Communicator, která rozšiřuje konstruktor o dva nové parametry

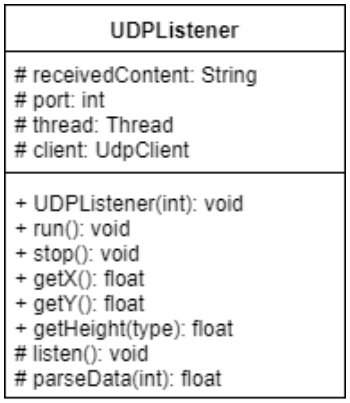
ip a port sloužící pro nastavení síťové komunikace. V rámci abstraktní třídy bylo pro zdědění potřeba implementovat metodu run(), která spouští hlavní logiku třídy. V této metodě se nachází nekonečný cyklus čekající na příchozí pole dat do objektu Queue. Po přijetí pole jsou jednotlivé položky kontrolovány, jestli se jedná o validní hodnoty. Pokud jsou hodnoty validní, přijatá čísla jsou nastavena pro celou instanci. V opačném případě jsou hodnoty všech položek nastaveny na 0. Nastavené hodnoty jsou poté s oddělujícím znakem „středník“ zapsány do řetězce a odeslány jako datagram přes UDP.



Obrázek 7.8: Class diagram - UDPCommunicator

Přijímání dat

V Unity byla vytvořena třída s názvem UDPListener sloužící k příjmu UDP datagramů. Třída obsahuje jediný konstruktor s povinným celočíselným parametrem, reprezentujícím číslo portu, na kterém předávání dat probíhá. Stejně jako na straně TwD i v Unity bylo třeba založit paralelní vlákno, aby čekání na datagramy a jejich zpracovávání nebrzdilo chod zbytku aplikace. Inicializace nového vlákna se nachází v metodě run(), kde probíhá nekonečný cyklus zachycující data z UDP. Tato data jsou dále zpracována metodou parseData(String incommingString), která rozdělí řetězec do pole, jednotlivé položky převede z textu do čísel s desetinnou čárkou a uloží je do instanční proměnné. Třída pak nabízí zpracované hodnoty pomocí veřejných metod pro další práci.



Obrázek 7.9: Class diagram - UDPListener

7.6.2 Bezpilotní prostředek

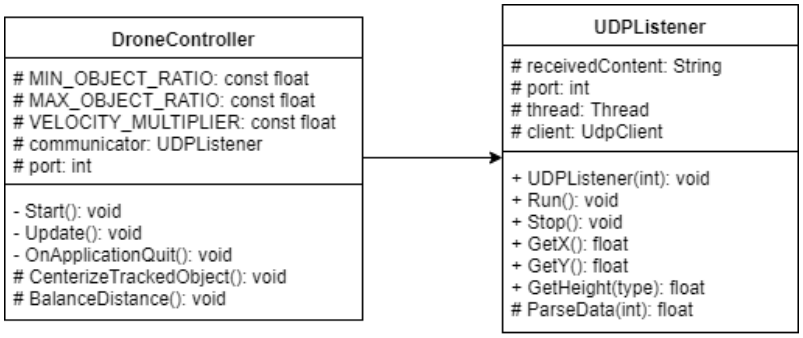
Bezpilotní prostředek je ve virtuálním světě reprezentován herním objektem Drone, na který je vázán další herní objekt Camera, snímající scénu podobně jako kamera v reálném světě. K objektu Drone je připojen herní script DroneController obsahující logiku pohybu bezpilotního prostředku.

V rámci VR bylo o bezpilotním prostředku uvažováno jako o létajícím dronu s možností udržet konstantní výšku.
Scripty, které se v Unity přiřazují přímo na herní objekty, musí dědit z třídy MonoBehaviour, čímž se zajistí spuštění scriptu při načtení scény. Unity kontroluje, jestli existují některé metody, které souvisí s životním cyklem programu. [37] V práci byly implementovány metody start() a update(). V metodě start() dochází k nastavení a spuštění UDP komunikátoru. Metoda update() se volá při každém překreslení scény, proto bylo do těla metody vloženo volání metod pro aktivní sledování objektu: centerizeTrackedObject() a balanceDistance().

Udržování středu

```
protected void CenterizeTrackedObject() {
    float x = communicator.GetX();
    transform.Translate(new Vector3(x * VELOCITY_MULTIPLIER, 0, 0) * Time.deltaTime);
}
```

Metoda centerizeTrackedObject() má za úkol udržet objekt zájmu horizontálně uprostřed, proto vytváří směrový vektor, udávající bezpilotnímu prostředku směr



Obrázek 7.10: Class diagram - DroneController

a rychlost letu. K vytvoření směrového vektoru se využívá obdržená normalizovaná hodnota z příchozího UDP datagramu. Aby se bezpilotní prostředek pohyboval dostatečně rychle, byla tato hodnota násobena konstantou VELOCITY_MULTIPLIER. Celý vektor byl poté vynásoben delta časem, který posun rozloží do jedné vteřiny místo posunu v každém vypočteném snímku. [38] Tímto se stal posun nezávislý na počtu snímků a zároveň zajistil plynulost pohybu.

Udržování vzdálenosti

```
protected void BalanceDistance() {
    float currentRatio = communicator.GetRatio();
    if (currentRatio > 0) {
        if (currentRatio < MIN_OBJECT_RATIO) {
            transform.Translate(new Vector3(0, 0, 1f) * Time.deltaTime);
        }
        if (currentRatio > MAX_OBJECT_RATIO) {
            transform.Translate(new Vector3(0, 0, -1f) * Time.deltaTime);
        }
    }
}
```

Protože bezpilotní prostředek nemá žádné prostředky k určení vzdálenosti sledovaného objektu, využívá se k tomuto účelu poměr sledovaného objektu vůči obrazu. V rámci práce se nejvíce osvědčilo neomezovat poměr na konkrétní hodnotu, nýbrž na rozsah dvou blízkých hodnot. Poměrová hodnota objektu v ideální vzdálenosti většinou kolísá, což nutilo bezpilotní prostředek neustále kompenzovat vzdálenost.

7.6.3 Testování

VTS byl testován na počítačové sestavě Intel i5 2500K 3,3@4,6GHz, 2x 4GB DRR3 2133MHz CL9. Virtuální svět byl původně testován na rozlišení obrazu 1280x1080. Testovaný VTS ve VR dokázal zpracovat na přetaktovaném CPU pouhé 3 FPS. Samotný návrh CPU je uzpůsoben spíše pro zpracování obecných úloh, proto bylo nutné přesunout výpočty na GPU, které je navržené pro paralelní zpracovávání instrukcí a je tedy vhodnější pro počítání hodnot tisíce vrstvených neuronů CNN. Díky podpoře Nvidia CUDA na Tensorflow byla do počítačové sestavy zakomponována grafická karta Nvidia GeForce 1060GTX 3GB. Použitím grafické karty došlo k navýšení výkonu a VTS dokázal zpracovat až 25 snímků za vteřinu, což i tak bylo překvapivě malé číslo vzhledem k avizovaným rychlostem využitých komponent uvedených v této práci. Zkoumáním tohoto výkonnostního problému se došlo k závěru, že se jedná o kombinaci faktorů renderování virtuální scény a získávání obrazového vstupu skrze snímání obrazu.

Testování probíhalo na vytvořené scéně simulující rušné náměstí, do které bylo umístěno 10 chodících autonomních postav a postava ovládaná operátorem skrze klávesnici. Do scény byla vložena kamera. Její pozice byla pro testování zvolena tak, aby byla ovladatelná postava zhruba ve středu záběru. Vzhledem k tomu, že kamera simuluje vznášející se bezpilotní prostředek, byla jí definována pevná výška odpovídající 2,5 metru.

Jedním z cílů tohoto testování bylo dokázat, že VTS dokáže sledovat vybraný objekt zájmu v obraze. Pro reálné nasazení bylo důležité vyzkoušet, jestli VTS dokáže rozpoznat a obnovit identitu dočasně zakrytých nebo zmizelých objektů. Z tohoto důvodu byly do scény zasazeny postavy lidí, které jsou si vzhledem podobné a měly tak mást VTS. Dalším cílem bylo otestování bezpilotního prostředku, jenž měl pomocí předaných směrových vektorů z VTS udržovat sledovaný objekt uprostřed obrazovky v nastavené vzdálenosti.

Testováním bylo zjištěno, že vytvořený systém je nedostatečný pro sledování postav. Problém spočíval v detekci s trackingem, kde docházelo k častým ztrátám identity nebo vůbec nedošlo k detekci. Vyšetřováním tohoto problému bylo odhaleno, že Tiny

Yolo sice dokáže víceméně ohraničovat objekty, ale toto ohraničení mělo s každým snímkem lehce odlišnou velikost, což způsobovalo potíž asociačním algoritmům Deep SORT. V článku, ve kterém autoři prezentovali tracking systém SORT, bylo zmíněno, že kvalitní detektor dokázal zvednout kvalitu trackování až o 18.9%. [30] Z tohoto důvodu byl v práci vyměněn Yolo model Tiny Yolo za náročnější model YoloV2. Ohraničení objektů YoloV2 oproti své redukované verzi skvěle obtékala detekované objekty, což mělo za následek prudký nárůst trackovacích schopností Deep SORT. Výměna detekčního modelu si bohužel vyžádala další výpočetní prostředky a tím se počet snímků za vteřinu propadl na 8. Pro navýšení rychlosti zpracování byla scéna v Unity nastavena na nejnižší detaily a rozlišení obrazu bylo změněno na 640x480. Počet snímků za vteřinu se tímto kompromisem zvedl zhruba k 16. Bohužel pro plynulé sledování tento počet nestačil. Systém při rychlém pohybu často nedokázal obnovit identitu objektu a prostředek se pohyboval v místech, kde se již objekt nenacházel. V rámci dalšího zrychlení byl pro YoloV2 použit vstupní obraz 320x320 místo 544x544, což umožnilo zvýšit počet snímků za vteřinu na 23. Zmenšení vstupního obrazu se viditelně neprojevilo na kvalitě detekcí a vyšším počtem FPS se eliminovaly zmíněné problémy.

Sledování bylo řešeno udržováním objektu zájmu ve středu obrazu v definovaném rozsahu výšky objektu popisované v kapitole **7.6.2 Bezpilotní prostředek**. Aby tento přístup fungoval, bylo nutné najít koeficienty pro normalizované výstupní hodnoty VTS. Koeficienty pro směrový vektor bylo nutné nastavit na hodnoty umožňující sledovat rychle se pohybující objekt, ale zároveň tyto hodnoty nesměly způsobit příliš velkou akceleraci, jež by zapříčinila náskok dronu před sledovaným objektem. Koeficienty byly testováním nastaveny na hodnoty 5 pro pohyb do stran a 1,5 pro udržování vzdálenosti.

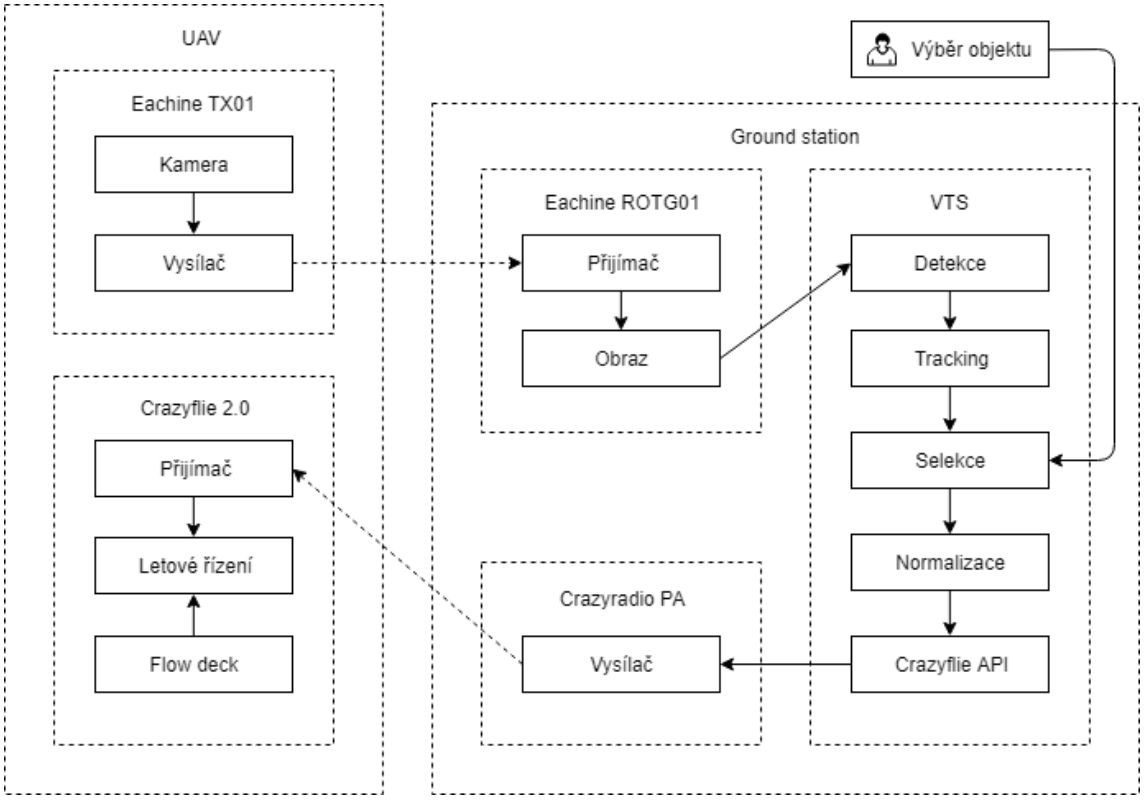
Detekce YoloV2 předpovídá pravděpodobnost, s jakou je v daném ohraničení konkrétní typ objektu. Aby se zamezilo dalšímu zpracovávání objektů s nízkou pravděpodobností, byly detekce s nižší hodnotou než hodnota z inicializačního parametru threshold potlačeny. Vzhledem k virtuálním objektům, podobajícím se skutečným postavám a rozlišení 640x480, bylo nutné tento parametr nastavit na relativně níz-

kou hodnotu 0,35, aby byly nalezeny například i vzdálenější objekty.

Označená postava byla díky odladění výše popsaných skutečností sledována a virtuální dron ovládaný vytvořeným vektorem pohybu z normalizovaných dat dokázal tuto postavu následovat a obnovovat její identitu při dočasném zákrytu jinou postavou. VTS byl tímto úspěšně otestován a byl připraven pro nasazení do reálného světa.

Obrázek 7.11: Ukázka - Visual Tracking Systém ve virtuální realitě

7.7 Reálný svět



Obrázek 7.12: Schéma zapojení v reálném světě

7.7.1 Komunikace

Získání obrazu

Obrázek 7.13: FPV přijímač - Eachine ROTG01

Pro získání obrazu byl použit přijímač Eachine ROTG01 operující v pásmu 5.8GHz. [42] Po připojení do počítače se příjmač identifikuje jako webová kamera. Vytvořený VTS

obsahuje knihovnu OpenCV podporující přijímat obraz z webových kamer. Při vytváření VTS se s tímto počítalo a byl připraven inicializační parametr demo popsaný v kapitole **7.6.1 Komunikace: Získání obrazu**, kterým se přepíná obraz mezi VR a web kamerou.

Posílání dat

Obrázek 7.14: Crazyradio PA

Přenos dat do bezpilotního prostředku je řešen pomocí rádiového vysílače Crazyradio PA. Vysílač je vybaven zesilovačem s výkonem 20dBm, který umožňuje ovládat kvadrakoptéru až do 1km při přímé viditelnosti. [43]

Při vývoji VR byla vytvořena abstraktní třída Communicator definující společnou logiku zasílání příkazů do bezpilotního prostředku (popsáno v kapitole **7.6.1 Komunikace: Posílání dat**). Do projektu byla vytvořena nová třída CrazyFlieCommunicator dědící z této abstraktní třídy. V rámci podědění bylo nutné implementovat metodu run(), jenž zavádí komunikaci a v nekonečném cyklu předává zpracovaná data zasílaná z VTS. Do projektu byl přidán balíček cflib obsahující The Crazyflie Micro Quadcopter library API pro komunikaci s Crazyflie. API umožňuje skenovat, otvírat/zavírat komunikaci a výměnu dat. V prvním kroku dochází k navázání komunikace, kde se hledají dostupné Crazyflie. V kroku druhém jsou již skrze API posílány samotné letové příkazy. V projektu je používána metoda send_hover_setpoint(vx, vy, yawrate, zdistance) využívající připojené perifierie Flow Deck, která umožňuje udržet kvadrakoptéru na konkrétním místě v prostoru.

Parametry metody send_hover_setpoint:

- **vx** - směr a rychlost v m/s v ose x
- **vy** - směr a rychlost v m/s v ose y
- **yawrate** - otočení v úhlových stupních
- **zdistance** - vzdálenost od země v m

[44]
Udržování středu a vzdálenosti je řešeno stejně jako ve VR za pomoci vypočítaných směrových vektorů z normalizovaných hodnot VTS. Vektory jsou pouze násobeny jinými koeficienty v závislosti na poskytnutém obraze a možnostech pohybu Crazyflie. Tyto konstanty byly definovány na základě testování.

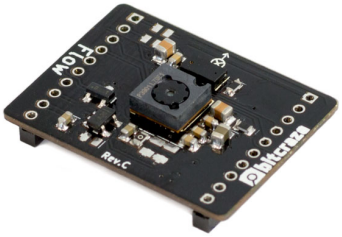
7.7.2 Bezpilotní prostředek

Crazyflie 2.0

Obrázek 7.15: Crazyflie 2.0

Jako bezpilotní prostředek byla zvolena malá kvadrakoptéra o rozměrech 9x9cm z důvodu bezpečného a jednoduchého testování uvnitř budovy. Crazyflie komunikuje buď přes rádio, nebo bezdrátový standard Bluetooth a poskytuje svůj kód jako Open Source. Horní a spodní část kvadrakoptéry obsahuje rozšiřující rozhraní, na které lze připojit různé periferie. Skrz toto rozhraní má uživatel přístup ke sběrnicím jako UART, I2C a SPI, PWM, analogové vstupy a výstupy a GPIO. [39]

Flow deck



Obrázek 7.16: Flow deck

Jedním z rozšíření kvadrakoptéry je deska Flow deck vybavená snímačem optického toku a snímačem vzdálenosti za pomoci laseru. Tento způsob umožňuje řídícímu systému znát směr a rychlost a také dává kvadrakoptéře schopnost vznášet se na konkrétní pozici v prostoru až 2 metry nad zemí. [40]

FPV Kamera

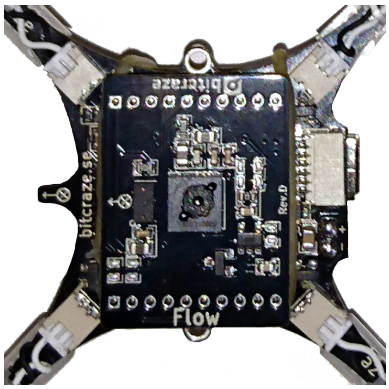
Obrázek 7.17: FPV kamera - Eachine TX01

Kvadrakoptéra byla vybavena malou kamerou s vysílačem Eachine TX01 o velikosti 20x13x6mm a hmotnosti 4,48g. Kamera analogově vysílá v pásmu 5,8GHz s vysílacím výkonem 25mW. Její vstupní napětí je v rozmezí 3,7 - 5V, což umožnilo vyrobením rozbočovače napojení přímo na baterii Crazyflie. [41]

Sestavení

Obrázek 7.18: Zkompletovaný bezpilotní prostředek

Na spodní stranu stranu Crazyflie byla bez potíží připojena periferie Flow deck. Malý problém ovšem nastal při zapojení kamery, kterou bylo potřeba uchytit na horní stranu kvadrakoptéry. Jelikož Crazyflie není na připojení tohoto druhu rozšíření připravená, byla kamera přilepena oboustrannou páskou. Aby bylo možné kameru provozovat, musel být vyroben rozbočovač napájení.



Obrázek 7.19: Připevněný Flow deck na spodní straně Crazyflie

Obrázek 7.20: Rozbočovač nápájení pro kameru

7.7.3 Testování

VTS byl testován s nastavením, které vyplynulo z testování simulace (popsáno v kapitole **7.6.3 Testování**).

Narozdíl od VR testování posloužily jako testovací objekty skutečné postavy lidí, jež byly snímány malou bezdrátovou kamerou komunikující v pásmu 5,8GHz. Přijímaný obraz měl rozlišení 640x480. Bohužel vzhledem k použití analogového přenosu a využití pásma pro Wi-Fi sítě docházelo občas k výpadkům, obraz trpěl šumem a malá kamera s nízkým rozlišením neměla velmi kvalitní obraz, což se značně podepsalo na výkonech VTS. Na krátké vzdálenosti, za dobré světelnosti a nerušeném vysílacím pásmu ovšem pracoval VTS relativně dobře a dosahoval podobné přesnosti jako ve VR. Díky absenci snímání obrazovky a renderování virtuální scény byl počítač schopen zpracovat 30 snímků za vteřinu s použitím YoloV2 s navýšeným vstupním obrazem 544x544, což pomohlo generovat detekce v nekvalitním obraze.

Díky připojenému rozšíření Flow deck pro Crazyflie bylo možné nastavit konstantní výšku kvadrakoptéry na 1m. Vzhledem ke špatné kvalitě obrazu byl rozsah poměrů výšky sledovaného objektu vůči výšce obrazu nastaven na vyšší hodnoty než u VR, a to na 0,30 až 0,35, aby kvadrakoptéra létala blíže a systém lépe rozpoznával objekt zájmu.

Prvotní testování bylo o nalezení potřebných koeficientů pro normalizované hodnoty z VTS. Stejně jako pro dron ve VR bylo nutné najít takové koeficienty, jaké by umožňovaly sledovat rychle se pohybující objekt, ale zároveň tyto hodnoty nesměly způsobit příliš velkou akceleraci, jež by zapříčinila náskok dronu před sledovaným objektem. Koeficienty byly stanoveny na hodnoty umožňující dronu bez problému sledovat lehce klusající postavu.

Jako zásadní problém se ukázala baterie Crazyflie, kde k vybití docházelo v průměru za 3,5 min, během kterých bylo nutné vše testovat. Kvadrakoptéra by měla podle specifikací létat déle, ale napojení bezdrátové kamery na stejný zdroj napájení zkrátilo dobu letu skoro na polovinu. [45] Nabití baterie trvalo kolem 30 minut, což značně zdržovalo kalibraci.

Testování probíhalo ve velkém uzavřeném vnitřním prostoru, kde se volně procházeli dobrovolníci. Cílem bylo vyzkoušet, jestli bezpilotní prostředek dokáže udržet sledovaného člověka a doletět ho v případě dočasného zákrytu. Testováním se ukázalo, že pokud je postava dočasně zakryta, například průchodem jiné postavy, VTS nemá většinou problém obnovit indentitu. Bohužel se ale občas stalo, že sledovaná posatava, jenž náhle změnila svůj směr během zákrytu, nebyla správně identifikována a dostala nové id.

Obrázek 7.21: Ukázka - Visual Tracking Systém v reálném světě

8. Diskuse

Hlavním cílem diplomové práce bylo vytvořit trackovací systém schopný sledovat objekt zájmu z obrazu. Z analýzy současně dostupných trackovacích systémů z kapitoly **2 Tracking Systémy** bylo rozhodnuto, že se práce vydá směrem hledání objektů v obraze za pomoci konvolučních neuronových sítí, neboť ostatní analyzované metody potřebovaly pomocné prvky v obraze nebo byly svou kvalitou nedostačující pro VTS. Výhodou tohoto zpracování je, že systém nejprve najde a klasifikuje objekty v obraze, které až následně identifikuje podle předchozích snímků. Tím se zásadně liší od běžných tracking algoritmů, jež sledují vybranou část obrazu bez jakéhokoli vědomí o tom, co konkrétně sledují, čímž snadno dojde ke ztrátě sledování nebo sledování jiné části obrazu. Nevýhodou použití konvolučních neuronových sítí pro zpracování obrazu je jejich vysoký nárok na výpočetní výkon.

VTS byl původně stavěn pro přímou aplikaci na bezpilotním prostředku, proto byl za základě výsledků analýzy z kapitoly **5.3 Porovnání systémů** vybrán pro detekování objektů velmi rychlý systém Tiny Yolo. Později byl ale v práci pro generování nepřesných detekcí nahrazen jeho robustnější verzí YoloV2, jenž dokázal skvěle ohraničovat objekty v obraze. Tímto krokem ovšem bylo jasné, že se současným mobilním hardwarem nepůjde systém provozovat, neboť už tak vysoké nároky na výkon byly podle zjištěných hodnot několikanásobně vyšší (**5.3.1 Měření**).

Aby bylo možné vygenerované detekce identifikovat, byl do projektu integrován trackovací systém Deep SORT, který byl zvolen, protože společně se svým méně přesným předchůdcem to byly jediné dva algoritmy vhodné pro tuto práci. Další nalezené systémy buď nebyly kompatibilní s plánovaným systémem nebo byl jejich zdrojový kód uzavřen (**6 Tracking**).

Vzhledem k nutnosti testování VTS na dynamickém kontrolovaném prostředí byla do práce přidána VR, jenž byla vytvořena v engine Unity (**4 Testovací virtuální realita**). Vytvořená scéna měla za úkol nasimulovat co nejreálnější protředí, aby mohl být systém použit i ve skutečném světě. Z těchto důvodů byla scéna poskládána jako část fiktivního města s chodníky, budovami s obchody a stromy pro navýšení

autenticity prostředí. Bezesporu se jako největší výhoda použití VR ukázalo opakované zkoušení systému na předem definovaném místě virtuálního prostoru a také absence výměny a nabíjení napájecích zdrojů, které značně brzdily odladění a testování VTS v reálném světě.

Pro odladění jednotlivých komponent VTS bylo do VR přidáno několik autonomních postav, které měly za úkol záměrně pracovat jako rušivé elementy při detekci a indetifikaci (**4.1.4 Simulace rušného náměstí**). Vyrenderovaný obraz Unity byl zachytáván přímo z obrazovky, což si vyžádalo daň v podobě dalšího potřebného výkonu (**7.6.1 Komunikace: Získání obrazu**). Získaný obraz byl dále zpracován a nalezné objekty byly zpřístupněny uživateli pro vybrání objektu zájmu (**7.4 Selekce**). Souřadnice sledovaného objektu byly poté normalizovány do hodnot umožňujících vytvoření směrového vektoru pro bezpilotní prostředek (**7.5 Normalizace**). Vzhledem k tomu, že kód VTS byl psaný v programovacím jazyce Python a kód VR byl v programovacím jazyce C#, byla komunikace vyřešena předáváním dat po síti za pomoci UDP (**7.6.1 Komunikace: Posílání dat**, **7.6.1 Komunikace: Přijímání dat**). O bezpilotním prostředku bylo ve VR uvažováno jako o létajícím dronu a byla mu nastavena pevná výška, ze které sledoval chodící postavy lidí. Použitím normalizovaných hodnot zaslaných VTS dron udržoval objekt zájmu ve středu obrazovky. Vzdálenost od sledovaného objektu byla řešena jako minimální a maximální poměr výšky jeho ohraničení vůči výšce obrazu, kterou dron musel kompenzovat (**7.6.2 Bezpilotní prostředek**).

Při zkoušení možností VTS bylo zjištěno, že provádění výpočtů na CPU je nevyhovující, neboť VTS nedokázal zpracovat dostatečný počet snímků pro plynulý chod. Z tohoto důvodu byly výpočty díky kompatibilnímu softwaru přesunuty na GPU. Výměna detekčního systému Tiny Yolo za YoloV2 společně se zachytáváním obrazu zapříčinily, že VTS nedokázal v předpokládaném nastavení zpracovat dostatek snímků a pro dosažení plynulosti bylo nutné snížit kvalitu detekce, rozlišení zachytávaného obrazu a grafické detaily VR. (**7.6.3 Testování**)

Označená postava byla díky aplikaci výše popsaných postupů skvěle sledována a virtuální dron ovládaný vytvořeným vektorem pohybu z normalizovaných dat plynule

tuto postavu následoval. VTS byl tímto úspěšně otestován a byl připraven pro nasazení do reálného světa.

Díky vyzkoušení konceptů sledování ve VR bylo velmi snadné aplikovat stejné postupy na skutečný bezpilotní prostředek přesně, jak bylo plánováno v kapitole **3. Strategie řešení**. Jako prototyp reálného bezpilotního prostředku byla vybrána malá kvadrakoptéra Crazyflie 2.0 z důvodu snadného a bezpečného testování uvnitř budovy (**7.7.2 Bezpilotní prostředek: Crazyflie 2.0**). Zachytávání obrazu bylo vyřešeno namontováním malé bezdrátové kamery Eachine TX01 s vysílačem (**7.7.2 Bezpilotní prostředek: FPV Kamera**). Vysílaný obraz byl zpracován přijímačem Eachine ROTG01, který se v počítači identifikoval jako webová kamera, čímž bylo možné poslat obraz bez dalších úprav na vstup VTS (**7.7.1 Komunikace: Získání obrazu**). Obraz byl dále zpracován stejně jako ve VR pro získání normalizovaných hodnot objektu zájmu, ze kterého byl sestaven směrový vektor pro bezpilotní prostředek. Předávání letových instrukcí bylo provedeno skrze vysílač Crazyradio a použitého softwarového API pro Crazyflie (**7.7.1 Komunikace: Posílání dat**).

Jedním z nedostatků postaveného prototypu byla velmi krátká doba letu zapříčiněná zapojením kamery na malou baterii Crazyflie. Během 3,5 minuty bylo nutné vyzkoušet veškeré potřebné věci, což značně časově limitovalo kalibraci zařízení a vývoj.

Zásadním nedostatkem postaveného prototypu byl ale získaný obraz z malé bezdrátové kamery, který trpěl na občasné rušení a výpadky obrazu způsobující ztrátu sledovaného objektu. Dalšímy problémy obrazu byla kvalita a nízké rozlišení, jež zapříčinily nutnost létat blíže k objektu zájmu, aby byl objekt vůbec detekován. Pokud ovšem byla kvadrakoptéra testována za příznivých podmínek, označená postava byla sledována přibližně stejně dobře, jako bylo testováno ve VR a VTS plnil svůj účel.

Použití vytvořeného prototypu pro tento druh práce nebyla vhodná volba. Prototyp bude maximálně sloužit jako ověření konceptu a pro reálné využití nemá smysl. Pro využití potenciálu VTS je potřeba kvalitní obraz, na němž se přímo odráží schopnosti vytvořeného systému.

9. Závěr

Práce se v prvopočátcích zabývala analýzou současných trackovacích systémů, na základě které byla navržena strategie pro tvorbu VTS. Jako vybraný postup bylo zvoleno trackování objektu za pomoci detekce, což práci rozšířilo o další nutné analýzy pro vybrání vhodných systémů. V rámci usnadnění vývoje byla do práce zakomponována virtuální realita sloužící pro testování jednotlivých komponent a systému jako celku. Vyrobený VTS byl nejprve odladěn a vyzkoušen v simulaci a poté byl portován do skutečného světa na malou kvadrakoptéru Crazyfie 2.0.

Hlavním cílem práce bylo vytvořit funkční prototyp, který dokáže zamknout objekt zájmu a ten sledovat. Tento cíl se povedlo splnit, avšak za cenu přesunutí výpočetního výkonu na ground station, neboť detekce s trackingem tvořící základ sledování objektů využívá hluboké neuronové sítě, a ty vyžadují enormní výkon. Dalším cílem práce bylo vytvořit systém schopný vyrovnání se proprietálnímu algoritmu Active Track od společnosti DJI, což bylo umožněno využitím metod vzešlých z jednotlivých analýz v této práci. Jako další stanovený cíl bylo vytvoření autentické VR, která skvěle posloužila při výrobě VTS a jeho nasazení do skutečného světa tím znamenalo pouze obstarání spojení, výměnu vstupního obrazu a kalibraci kvadrakoptéry.

V rámci práce se podařilo splnit stanovené cíle a byl vytvořen systém schopný sledovat objekt zájmu. Nicméně pro běžné použití není systém vzhledem k potřebě externího výkonu vhodný. Lze ovšem předpokládat, že budoucí hardware bude výkonnostně stoupat a externí výpočet nebude potřeba.

Seznam použité literatury

[1] *International Journal of Scientific & Technology Research: Vision-Based Object Tracking Algorithm With AR. Drone* [online].2016, **2016**(6) [cit. 2018-03-15]. ISSN 2277-8616. Dostupné z: http://www.ijstr.org/final-print/june2016/Vision-based-Object-Tracking-Algorithm-With-Ar-Drone.pdf

[2] *Color Detection & Object Tracking. OpenCV Tutorial C++* [online]. [cit. 2018-03-15]. Dostupné z: https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html

[3] *Marker Tracking with AR.Drone for Visual-based Navigation Using SURF and MSER Algorithms* 2017-04 [online]. [cit. 2018-03-15]. Dostupné z: https://www.researchgate.net/publication/315141712_Marker_Tracking_with_ARDrone_for_Visual-based_Navigation_Using_SURF_and_MSER_Algorithms

[4] *Learn OpenCV: Object Tracking using OpenCV* [online]. [cit. 2018-03-24]. Dostupné z: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python

[5] Bc. Adam Vyškovský: *Object tracking by a flying drone* [online]. 2014-09-04 [cit. 2018-03-24]. Dostupné z: https://is.cuni.cz/webapps/zzp/detail/146263/29577936/

[6] *Vision based GPS-denied Object Tracking and Following for Unmanned Aerial Vehicles* [online]. [cit. 2018-03-24]. Dostupné z: http://robotics.asu.edu/astril/robotics.asu.edu/ardrone2_ibvs/index.html

[7] *Movidius and DJI Bring Vision-Based Autonomy to DJI Phantom 4* 2016-05 [online]. [cit. 2018-03-15]. Dostupné z: https://www.movidius.com/news/movidius-and-dji-bring-vision-based-autonomy-to-dji-phantom-4

[8] *DJI Machine Learning lab: Phantom 4 Active Track algorithm test and comparison: car mountain driving* [online]. 2016-06-01 [cit. 2018-03-15]. Dostupné z: https://www.youtube.com/watch?v=7u_T_5U6cdI

[9] *DJI Machine Learning lab: Phantom 4 Active Track algorithm test and comparison: girl skating* [online]. 2016-06-01 [cit. 2018-03-15]. Dostupné z: https://www.youtube.com/watch?v=T-khvN_pGpM

[10] *Unity Manual: Working with humanoid animations* [online]. 2018-01-29 [cit. 2018-02-18]. Dostupné z: https://docs.unity3d.com/Manual/AvatarCreationandSetup.html

[11] *Unity Manual: Creating and Using Scripts* [online]. 2018-01-29 [cit. 2018-02-18]. Dostupné z: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

[12] *Unity Manual: Importing from the Asset Store* [online]. 2018-01-29 [cit. 2018-02-18]. Dostupné z: https://docs.unity3d.com/Manual/AssetStore.html

[13] *A Beginner's Guide To Understanding Convolutional Neural Networks* [online]. 2016-07-20 [cit. 2018-02-25]. Dostupné z: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[14] *The PASCAL Visual Object Classes Challenge 2007* [online]. [cit. 2018-02-25]. Dostupné z: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

[15] *Computerphile: Neural Network that Changes Everything* [online]. 2016-05-20 [cit. 2018-02-25]. Dostupné z: https://www.youtube.com/watch?v=py5byOOHZM8

[16] Gene Kogan. *What convolutional neural networks see* [online]. 2016-11-16 [cit. 2018-02-25]. Dostupné z: https://www.youtube.com/watch?v=Gu0MkmynWkw

[17] GIRSHICK Ross, Jeff DONAHUE, Trevor DARRELL a Jitendra MALIK. *Rich feature hierarchies for accurate object detection and semantic segmentation* [online]. 2014-10-22 [cit. 2018-02-25]. arXiv:1311.2524v5 Dostupné z: http://arxiv.org/abs/1311.2524v5

[18] *CV-Tricks.com: Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD* [online]. [cit. 2018-02-25]. Dostupné z: http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[19] *MachineLearner: CS231n Lecture 8 - Localization and Detection* [online]. 2016-06-14 [cit. 2018-02-25]. Dostupné z: https://www.youtube.com/watch?v=_GfPYLNQank

[20] GIRSHICK, Ross. *Fast R-CNN* [online]. 2015-09-27 [cit. 2018-02-25]. arXiv:1504.08083v2 Dostupné z: https://arxiv.org/abs/1504.08083v2

[21] Leonardo Araujo dos Santos. *Artificial Inteligence: Object Localization adn detection* [online]. 2016-05-18 [cit. 2018-02-25]. Dostupné z: https://www.gitbook.com/read/book/leonardoaraujosantos/artificial-inteligence

[22] REN Shaoqing, HE Kaiming, Ross GIRSHICK a SUN Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal* [online]. 2016-01-06 [cit. 2018-02-25]. arXiv:1506.01497v3 Dostupné z: https://arxiv.org/abs/arXiv:1506.01497v3

[23] Joseph Redmon a Ali Farhadi. *YOLO: Real-Time Object Detection* [online]. 2018-03-06 [cit. 2018-03-06]. Dostupné z: https://pjreddie.com/darknet/yolo/

[24] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick a Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection* [online]. 2015-06-08 [cit. 2018-02-25]. arXiv:1506.02640v5 Dostupné z: https://arxiv.org/abs/1506.02640v5

[25] Redmon Joseph a Ali Farhadi. *YOLO9000: Better, Faster, Stronger* [online]. 2016-12-25 [cit. 2018-02-25]. arXiv:1612.08242v1 Dostupné z: https://arxiv.org/abs/1612.08242v1

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu a Alexander C. Berg. *SSD: Single Shot MultiBox De-*

tector [online]. 2016-12-29 [cit. 2018-03-24]. arXiv:1512.02325v5 Dostupné z: https://arxiv.org/abs/1512.02325v5

[27] *CV-Tricks.com: Object Detection using Single Shot Multibox Detector* [online]. [cit. 2018-03-24]. Dostupné z: http://cv-tricks.com/object-detection/single-shot-multibox-detector-ssd/

[28] WOJKE, Nicolai, Alex BEWLEY a Dietrich PAULUS. *Simple online and realtime tracking with a deep association metric. 2017 IEEE International Conference on Image Processing (ICIP)* [online]. IEEE, 2017, 2017, 3645-3649 [cit. 2018-03-25]. DOI: 10.1109/ICIP.2017.8296962. ISBN 978-1-5090-2175-8. [online]. IEEE, 2017, 2017, 3645-3649 [cit. 2018-03-25]. DOI: 10.1109/ICIP.2017.8296962. ISBN 978-1-5090-2175-8. Dostupné z: http://ieeexplore.ieee.org/document/8296962/

[29] Wojke, Nicolai and Bewley, Alex (2018) Deep Cosine Metric Learning for Person Re-Identification. In: IEEE Winter Conference on Applications of Computer Vision (WACV). [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/nwojke/cosine_metric_learning

[30] BEWLEY, Alex, Zongyuan GE, Lionel OTT, Fabio RAMOS a Ben UPCROFT. *Simple online and realtime tracking. 2016 IEEE International Conference on Image Processing (ICIP)* [online]. IEEE, 2016, 2016, , 3464-3468 [cit. 2018-03-25]. DOI: 10.1109/ICIP.2016.7533003. ISBN 978-1-4673-9961-6 Dostupné z: http://ieeexplore.ieee.org/document/7533003/

[31] WOJKE, Nicolai, Alex BEWLEY a Dietrich PAULUS. *Deep SORT* [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/nwojke/deep_sort

[32] Joseph Redmon. *Darknet: Open Source Neural Networks in C* [online]. [cit. 2018-03-24]. Dostupné z: http://pjreddie.com/darknet/

[33] *Darkflow: Translate darknet to tensorflow* [online]. [cit. 2018-03-24]. Dostupné z: http://github.com/thtrieu/darkflow

[34] *Itseez: Open Source Computer Vision Library* [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/itseez/opencv

[35] Ouail Bendidi. *Tracking with Darkflow* [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/bendidi/Tracking-with-darkflow

[36] Mickaël Schoentgen. *Python MSS* [online]. 2016-12-25 [cit. 2018-02-25]. Dostupné z: https://github.com/BoboTiG/python-mss

[37] *Unity Manual: Execution Order of Event Functions* [online]. 2018-03-22 [cit. 2018-03-25]. Dostupné z: https://docs.unity3d.com/Manual/ExecutionOrder.html

[38] *Unity Scripting API: Time.deltaTime* [online]. 2018-03-22 [cit. 2018-03-25]. Dostupné z: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html

[39] *bitcraze: Crazyflie 2.0* [online]. [cit. 2018-03-29]. Dostupné z: https://www.bitcraze.io/crazyflie-2/

[40] *bitcraze: Flow deck* [online]. [cit. 2018-03-29]. Dostupné z: https://www.bitcraze.io/flow-deck/

[41] *Eachine: Eachine TX01 Super Mini AIO 5.8G 40CH 25MW VTX 600TVL 1/4 Cmos FPV Transmitter* [online]. [cit. 2018-03-29]. Dostupné z: http://www.eachine.com/Eachine-TX01-Super-Mini-AIO-5.8G-40CH-25MW-VTX-600TVL-1-4-Cmos-FPV-Transmitter-p-418.html

[42] *Eachine: Eachine ROTG01 UVC OTG 5.8G 150CH Full Channel FPV Receiver For Android Mobile Phone Smartphone* [online]. [cit. 2018-03-29]. Dostupné z: http://www.eachine.com/Eachine-ROTG01-UVC-OTG-5-8G-150CH-Full-Channel-FPV-Receiver-For-Android-Mobile-Phone-Smartphone-p-843.html

[43] *bitcraze: Crazyradio PA* [online]. [cit. 2018-03-30]. Dostupné z: https://www.bitcraze.io/crazyflie-2/

[44] *Crazyflie Nano Quadcopter Client: Commander* [online]. [cit. 2018-03-30]. Dostupné z: `https://github.com/bitcraze/crazyflie-lib-python/blob/master/cflib/crazyflie/commander.py`

[45] *Bitcraze Wiki: Frequently Asked Questions* [online]. 2015-07-15 [cit. 2018-04-10]. Dostupné z: `https://wiki.bitcraze.io/projects:crazyflie:faq`

Seznam obrázků