FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro navigaci minidronu
Student:	Jan Filip
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Proveďte rešerši knihoven pro řízení minidronů Parrot Mambo. Navrhněte a implementujte aplikaci, která bude řídit dron na základě informace z jeho kamer. Vertikální kameru využijte pro navigaci dronu podle objektů nebo značek ležících na podlaze. Horizontální kameru využijte pro vyhýbání se překážkám v prostoru nebo průletu otvorem. Pro zpracování obrazu použijte knihovnu OpenCV, případně sofistikovanější knihovny například na bázi hlubokých neuronových sítí. Aplikaci implementujte v jazyce Python, případně v kombinaci C/C++. Drony budou řízeny z linuxového počítače přes WiFi nebo BlueTooth. Veškeré programové vybavení řádně zdokumentujte a zdrojové kódy doplňte komentáři tak, aby se z nich dala automaticky generovat dokumentace. Aplikaci otestujte, zhodnoťte přesnost navigace a možnost současné navigace více dronů z jednoho počítače. Rozsah práce upřesněte po dohodě s vedoucím práce.

Elektronicky schválil/a doc. Ing. Hana Kubátová, CSc. dne 12. února 2021 v Praze.

Bakalářská práce

APLIKACE PRO NAVIGACI MINIDRONU

Jan Filip

Fakulta informačních technologií ČVUT v Praze
Katedra číslicového návrhu
Vedoucí: Ing. Miroslav Skrbek, Ph.D.
13. května 2021

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

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 připřípravě vysokoškolských závěrečných prací.Beru na vědomí, že se na moji prácivztahují 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 10. května 2020 .....

Abstrakt

Tato práce je zaměřena na možnosti ovládání minidronů Parrot Mambo. Cílem práce je vytvořit aplikaci schopnou dron samostatně navádět dle informací z jeho senzorů a kamer. Vyhýbání se dronu překážkám je dosaženo za pomocí upraveného Bug algoritmu. Získání informací z obrazu je zajištěno především za použití knihovny OpenCV a neuronové sítě Detectron2.

Klíčová slova dron, autonomní navigace, zpracování obrazu, neuronové sítě, navigace, senzory

Abstract

This thesis is focusing on ways to control minidrone Parrot Mambo. Main goal is to create application able to autonomously navigate dron based on data gained from sensors and camera. Obstacle avoidance of drone is achieved with modified bug algorithm. Images from cameras are processed with neural network Detectron2 and library OpenCV.

Keywords dron, autonomous navigation, image processing, neural netwroks, navigation, sensors

Seznam zkratek

UAV	Unmanned aerial vehicle
FGA	Flexible geometric algorithm
CNN	Convolutional neural network
R-CNN	Region-based CNN
ROI	Region of interest
RRT	Rapidly-exploring random tree
SLAM	Simultaneous localization and mapping

Kapitola 1
Úvod

Autonomní navigace robotů se stále více integruje do každodenního života a v budoucnu se stane jeho nedílnou součástí. Z počátku se autonomní navigace využívala v systémech se kterými běžný člověk nepřišel do styku. Příkladem můžou být autonomní lodě uzpůsobené na přistávání raketových motorů firmy SpaceX. V současnosti se tyto systémy rozšiřují do běžného života například vozidla firmy Tesla s možností autopilota.

Hlavním cílem práce je navrhnout aplikaci pro navigaci dronu, otestovat ji a řádně zdokumentovat. Sekundárním cílem práce je získat informace o možnostech takto ovládat více dronů v rámci aplikace a zhodnotit proveditelnost a efektivitu takového řešení.

Práce je rozdělena do pěti částí: První kapitola seznámí čtenáře s dronem Parrot Mambo a na různé metody jeho ovládání. Mimo jiné se i podíváme na základní informace o možných rozložení kvadrokoptér a základní principy jejich ovládání.

Druhá kapitola se zabývá analýzou zadané problematiky. Zde se seznámíme se základními principy autonomní navigace a různé principy navigace při průzkumu oblasti. Následně se podíváme na základní algoritmy pro navigaci dronu k cíli včetně vyhýbání se překážkám. Nakonec se podíváme na základní principy pro zpracování obrazu a vytěžení potřebných informací z něj.

Třetí kapitola je věnována návrhu aplikace a základním principům chování jejích jednotlivých částí.

Čtvrtá kapitola podrobně popisuje implementaci aplikace a zaobírá se do větší hloubky řešenými problémy při návrhu aplikace.

Pátá kapitola poskytuje informace o aplikaci jako celku a popisuje nutné kroky pro správné spuštění aplikace.

V poslední kapitole se podíváme na další možný rozvoj aplikace, shrneme cíle aplikace a poskytneme vyjádření o jejich splnění.

Kapitola 2

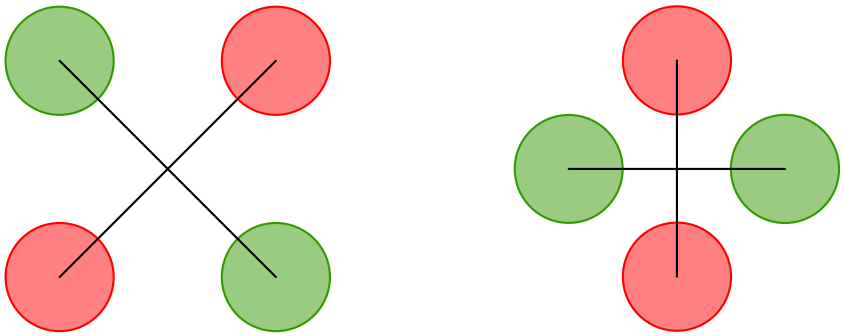
Úvod k ovládání dronů

2.1 Kvadrokoptéry

Kvadrokoptéra je speciální druh helikoptéry se čtyřmi rotory. Jejich použití se rozšířilo s komerčním prodejem dronů pro rekreační účely či pořizování leteckých záběrů pro filmové tvůre.

Takovýto dron má 4 motory, každý ovládající vlastní vrtuli, které umožňují pohyb dronu v šesti základních směrech (do stran, dopředu, dozadu, nahoru a dolu), které je spolu možno kombinovat. Kvadrokoptéry mají dvě možnosti základního rozložení motorů, do tvaru x a $+$. Hlavní rozdíl mezi těmito konfiguracemi je rozdíl mezi příkazy, které se posílají různým motorům pro splnění stejného typu pohybu.

Nejdůležitější část konfigurace motorů dronu je směr rotace. Motory na stejné ose se otáčejí stejným směrem, ale opačným oproti motorům na druhé ose dronu. Toto rozložení umožňuje dronu provádět základní pohyby (pitch, roll, thrust) nezávisle na sobě.[1]



Obrázek 2.1 Rozvržení dronu

2.2 Nástroje pro řízení dronu Parrot

Pro minidron Parrot Mambo existuje několik možností jeho ovládání. Daly by se rozdělit na ovládání pomocí jazyků Python, C/C++ a Matlab.

2.2.1 PyParrot

Pyparrot je knihovna pro jazyk Python navržená a implementovaná Dr. Amy McGovern. Toto rozhraní bylo vytvořeno k výuce STEM konceptů (science, technology, engineering, mathematics) pomocí programování dronů k samostatnému letu.[2]

2.2.2 RollingSpiderEdu

Tato knihovna slouží k navržení a simulaci algoritmů pro ovládání dronu v programu MATLAB či Simulink. Poté sama generuje kód v jazyce C určený k ovládání dronu. Všechna data ze senzorů jsou v průběhu letu zaznamenávána a je možno je později zobrazit a analyzovat.[3]

2.2.3 MATLAB® Support Package for Parrot® Drones

Tato možnost poskytuje vývojáři rozhraní po ovládání dronu z programu MATLAB. Uživateli je poskytnuta možnost ovládání dronu posíláním příkazů k ovládání směru, rychlosti a orientace dronu. Uživatel má zároveň možnost stahovat data ze senzorů, jako například výška, rychlost, orientace a mnoho dalších.

MATLAB zároveň poskytuje balíček pro simulaci dronu určenou k vývoje algoritmů pro kontrolu letu dronu. [4]

2.3 Dron Parrot Mambo

2.3.1 Senzory

Dron disponuje několika senzory. Na spodku dronu nalezneme ultrazvukový senzor pomocí kterého dokážeme měřit vertikální vzdálenost dronu od povrchu. Senzor vyšle zvukovou vlnu a měří jak dlouho trvá než se vlna odražená od země dostane zpět k senzoru. Z naměřeného času lze vypočítat výšku dronu od země.

Dalším senzorem umístěném také na spodku dronu je kamera, která dokáže snímat obraz ve frekvenci 60 FPS. Tato kamera používá optical flow k odhadnutí horizontálního rychlosti a pohybu dronu.

Uvnitř dronu se nachází tlakový senzor určený k měření výšky dronu na základě malých změn tlaku vzduchu.

Dron také disponuje IMU jednotkou vybavenou tříosým akcelerometrem, měřící lineární zrychlení, a tříosým gyroskopem, měřícím natočení dronu. Díky této jednotce dron umí vypočítat další informace jako je například rychlost otáčení.

Žádný z těchto senzorů neumožňuje měřit volnou vzdálenost před kamerou dronu. Absence tohoto senzoru nám velice omezí možné metody pro navigaci dronu a jeho vyhýbání se překážkám.[1]

2.3.2 Ovládání a pohyb dronu

Knihovna pyparrot poskytuje tři základní létací módy: s vertikální a horizontální stabilizací, pouze s vertikální stabilizací a bez stabilizace. Horizontální stabilizace má za cíl zamezení pohybu dronu vytvořeném pohyby vzduchu.

Obrázek 2.2 Dron Parrot Mambo

Samotný pohyb dronu je realizován pomocí funkcí `fly_direct` a `turn_degrees`.

Výpis kódu 2.1 Funkce pro ovládání pohybu dronu

```
def fly_direct(self, roll, pitch, yaw, vertical_movement, duration)
def turn_degrees(self, degrees)
```

Funkce `fly_direct` dostává jako parametry procentuální pohyb dronu po jeho osách a rotaci kolem osy z. Díky správnému rozložení motorů lze tyto pohyby libovolně kombinovat. Příkaz je opakován po celou dobu trvání. Pokud není čas pro provedení manévru specifikován příkaz se odešle pouze jednou.

Parametr	Popis	Očekávaná hodnota
roll	pohyb po ose y dronu	$\langle -1080, 1080 \rangle$
pitch	pohyb po ose x dronu	$\langle -1080, 1080 \rangle$
yaw	rotace dronu	$\langle -1080, 1080 \rangle$
vertical_movement	vertikální pohyb dronu	$\langle -1080, 1080 \rangle$
duration	doba trvání příkazu	> 0

Tabulka 2.1 Parametry funkce

Funkce `turn_degrees` umožňuje rotaci dronu na místě o daný úhel v rozmezí $\langle -180, 180 \rangle$. Na rozdíl od funkce `fly_direct` rotace pomocí této funkce nelze kombinovat s ostatními pohyby dronu.[2]

Kapitola 3
Analýza

Aby se robot mohl správně orientovat ve svém okolí potřebuje o něm získat informace a ty zpracovat do údajů, na základě kterých se bude schopen rozhodovat. Tyto údaje získává ze svých senzorů, které rozdělíme pro naše účely na dva typy:

- senzoru, udávající informace o robotu
- senzory, udávající informace o okolí

První skupina senzorů udává informace jako poloha, rotace robota, či rychlost jeho pohybu. Pro správné zpracování dat z těchto senzorů je nutné vědět v jakém kontextu informace vrací. Například data o poloze můžou být udávána ve třech různých kontextech: data mohou být vracena v GPS souřadnicích nebo můžou být vracena relativně vůči bodu zapnutí robota nebo například u dronu relativně vůči vzletu ze země. To samé může platit u rotace robotu. Tato skupina senzorů bude vracet údaje jako poloha, rotace robota, rychlost pohybu či různé informace o stavu robotu.

Pomocí druhé skupiny senzorů lze získávat informace o okolí. Do této skupiny zařadíme senzory jako kamery, ze kterých lze získat informace o objektech v okolí a v některých případech i vzdálenost k nim a ultrazvukové senzory určující vzdálenost od překážky.[5]

3.1 Autonomní navigace

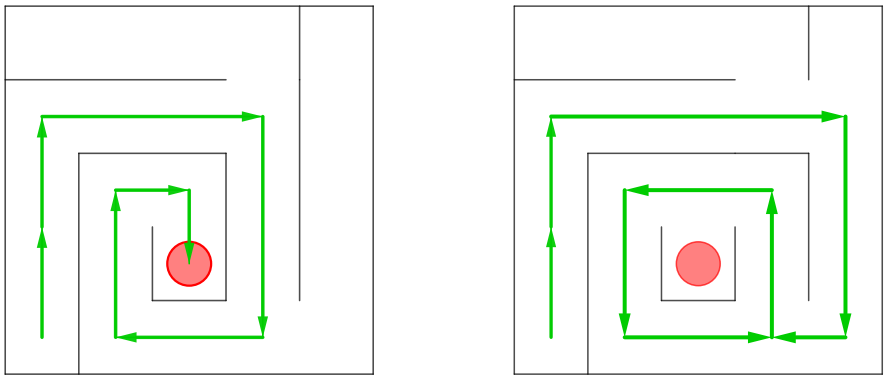
Navigace je postup či činnost, díky které člověk může zjistit svojí polohu a určit cestu k danému cíli. Autonomní navigace je postup či činnost, kterou robot dokáže zjistit svojí pozici a navigovat se ke cíli. Důležité je, že toho docílí sám bez zásahu člověka. Zjednodušeně řečeno je to nějaký postup pravidel, kterými se robot řídí aby nalezl cestu ke svému cíli.

Autonomní navigaci lze rozdělit celé spektrum dle úrovní složitosti. Nejnižší úroveň jsou aktivní systémy pro pomoc ovládání robota. Jedním z takových příkladů by se například dala považovat stabilizace dronu, aby se vznášel na stejném místě a nenechal se unášet proudy větru. Na druhé straně tohoto spektra se nacházejí plně autonomní systémy schopné samostatné navigace v prostředí.[5] Plně autonomní systémy, lze nadále dělit na heuristický a optimální přístup.

3.1.1 Heuristický přístup

Heuristický přístup je soubor pravidel, kterými se robot řídí, aby dosáhl svého cíle. Příkladem tohoto přístupu, může být hledání cíle v bludišti tím, že se robot bude držet vždy levé stěny. Jak můžeme vidět na obrázku 3.1 v případě, že je cíl umístěn na pozici, ke které se dostaneme sledováním stěny problém takovýto algoritmus vyřeší. Bude-li se nacházet cíl na ostrůvku uprostřed bludiště tento postup cíl nikdy nenajde. [5]

Jak je vidět z ukázky, tento postup není použitelný pro obecné řešení, jelikož není možné při vývoji aplikace vymyslet pravidla pro každou situaci ve které by se robot mohl ocitnout.



Obrázek 3.1 Navigace bludištěm

3.1.2 Optimální přístup

Optimální přístup pro hledání cesty vyžaduje mapu prostředí ve kterém se pohybuje. Tuto mapu si robot může sám vytvořit na základě informací ze senzorů, nebo dostane již vytvořenou mapu prostředí, kterou nadále vylepšuje. Na základě této mapy robot může vyhledat optimální cestu prostředím. Díky tomuto přístupu lze nalézt lepší řešení než pomocí heuristického přístupu. Příkladem tohoto přístupu jsou například inteligentní samořídící se vozy. Takovéto vozy si nevystačí s jednoduchým souborem pravidel proto musí hledat lepší metody navigace.[5]

3.1.3 Kombinace přístupů

Optimální přístup sám osobě nemůže vždy fungovat, protože v některých případech by vyžadoval informace aktuálně nedostupné. Například inteligentní vozidlo nemůže zaručit zda předjetí jiného vozidla povede k optimálnímu řešení pokud nezná situaci, která se nachází před tímto vozidlem. Z tohoto důvodu jsou tyto dva přístupy kombinovány a hledání cesty optimálním přístupem, je doplněna heuristickými pravidly. Příkladem u inteligentních vozidel by mohlo být: předjeď vozidlo, pokud je bezpečné tak udělat. Následně bezpečné předjetí by bylo definováno dalším souborem pravidel.[5]

3.2 Průzkum oblasti

K průzkumu zadané oblasti existují dva hlavní přístupy, které se řeší v jejich složitosti na implementaci a efektivitě průzkumu zadané oblasti.

Náhodný průzkum oblasti poskytuje snadnou implementaci, ale nedokáže zaručit plné pokrytí dané oblasti v rozumném čase. Jedním z triviálních přístupů jak tento problém řešit by mohla být metoda jež využívají některé robotické vysavače. Zvolíme si náhodný směr a začneme se pohybovat dopředu. Při nalezení překážky opět zvolíme náhodný směr a pokračujeme dokud nenarazíme na další překážku. Tento algoritmus je jednoduchý na implementaci, ale není efektivní. V čase $t- > \infty$ projde celou oblast, ale omezení dronu takovýto průzkum nedovolí.[5]

3.3 Algoritmy pro navigaci k cíli

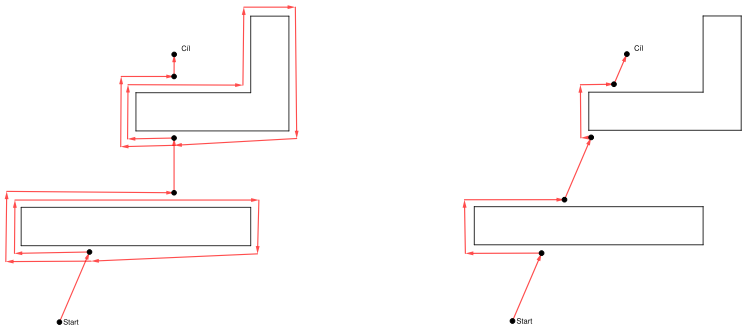
3.3.1 Bug algorithms

Bug algoritmy jsou jednoduché algoritmy, generující cestu k cíli pokud taková existuje. Jejich základní princip vyhýbání se překážkám je založen vytvoření cesty na základě pohybu podél detekované překážky. Tyto algoritmy předpokládají tři věci o robotu, který navigují:

- robot je jednoduchý bod
- robot má přesné informace o poloze
- robot má přesné senzory

Bug1 je nejednoduší a nejméně efektivní algoritmus z této skupiny. Robot se rozjede směrem k cíli, pokud narazí na překážku, začne se pohybovat okolo překážky a zároveň počítat vzdálenost jednotlivých bodů k cíli ze startovního bodu. Robot se zastaví na bodu kde původně narazil na překážku a určí bod s nejkratší vzdáleností k cíli. Následně robot najde tento bod a přesune se na něj. Z tohoto bodu robot může opustit překážku a vydat se k cíli.[6]

Bug2 algoritmus je složitější algoritmus než Bug1 a je více efektivní, jelikož nevyžaduje zkontrolování celého obvodu překážky. Robot se vydá směrem k cíli. Když narazí na překážku opět se začne sledovat její obvod. Jakmile robot narazí na bod podél obvod, který je na původní spojnici mezi startovním a cílovým bodem, opustí překážku a vydá se k cíli.[6]

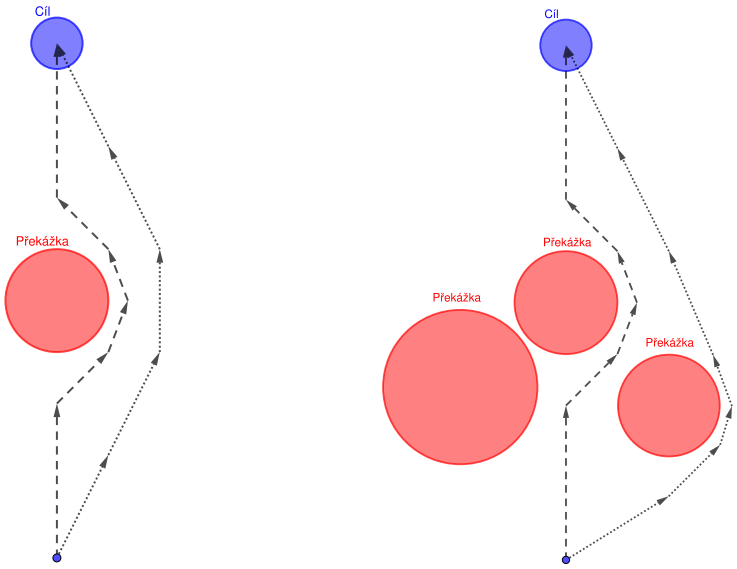


Obrázek 3.2 Bug algoritmusu

Poslední algoritmus je DistBug, který zde nebudeme rozebírat jelikož vyžaduje senzory, jež náš dron neposkytuje.[6]

3.3.2 FGA algoritmus

FGA - *Flexible Geometric Algorithm* - algoritmus předpokládá konstantní rychlost UAV a informace o jeho pozici a rychlosti. FGA nejdříve získá informace o okolí UAV ze senzorů a rozhodne zda se v něm nachází nějaké překážky, které by mohly ohrozit UAV a jsou blíž než daná dělící vzdálenost. Po detekci potencionální překážky jsou vytvořeny dva 2D výseče k překážce, jeden v horizontální a druhý ve vertikální rovině pro výpočet nejsnazšího úhybného manévru. FGA následně najde správný úhybný manévr (statická, dynamická, jedna nebo více překážek) k úspěšnému vyhnutí se překážce. Na základě manévru a informací s ním spojený FGA vypočte optimální časovou funkci pro započetí úhybného manévru. Když UAV dosáhne kritického času k provedení vyhnutí se, FGA spustí manévr a umožní UAV pohyb mimo cestu k cíli dokud není dosažena bezpečná vzdálenost od překážky. Následně se UAV opět vrátí na bod původní cesty k cíli.



Obrázek 3.3 FGA algoritmus

Aktivace úhybného manévru při blízké vzdálenosti k překážce spolu s návratem UAV na původní dráhu drasticky snižuje dráhu, která je třeba přepočítat při použití path-planning algoritmů a tím snižuje výpočetní dobu oproti například vůči metodě optimální cesty, která většinou vyžaduje přepočítání celou cestu k cíli.[7]

3.3.3 Navigace dle mapy

Předchozí algoritmy se nestarají o plánování cesty k vyhnutí se překážkám na které robot narazí. Existují metody plánování, které mají za úkol plánovat cesty pro vyhnutí se statickým překážkám za pomocí mapy prostředí.

3.3.3.1 Tvorba mapy

Pro vytvoření mapy prostředí potřebuje robot možnost měřit vzdálenost od překážek. Robot začne procházet zadanou oblast a získávat údaje ze svých senzorů o svém okolí. Na základě údajů o prostředí a svojí poloze se následně vytvoří mapa prostředí. Tato mapa bude ovšem nepřesná, jelikož nedokážeme zaručit neomylné údaje o pozici robota. Pro vytvoření správné mapy z takovýchto údajů se používá metoda SLAM[**slam**].

Účelem této metody porovnávání jednotlivých měření identifikaci jejich společných prvků. Algoritmus je takto schopen eliminovat chyby pohybové senzoru a vytvořit korektní mapu daného prostředí.

3.3.3.2 Algoritmy pro navigaci dle mapy

Pro navigaci robota na základě vytvořené mapy se používají algoritmy pracující na základě hledání cesty v grafu. Nejznámější takovéto algoritmy jsou A*[8], RRT[8] a RRT*[9].

Hlavní myšlenka algoritmy RRT je náhodné přidávání bodů v prostoru do grafu pomocí kterého následně hledáme cestu k cíli. Nový bod vždy připojíme jako větev již k existujícímu grafu v nejbližším bodě.

Algoritmus RRT* je rozšířením tohoto algoritmu. Rozdíl mezi těmito dvěma je v technice připojování nových bodů k existujícímu grafu. Algoritmus při připojování nového vrcholu prozkoumá jeho oblast a v rámci ní bod připojí tak, aby nová cesta v něm vzniklá byla co nejkratší. Díky tomuto faktoru se se zvětšováním počtu vrcholů v grafu optimalizuje cesta k cíli.

3.4 Zpracování obrazu

Zpracování obrazu se v dnešní době stává nedílnou součástí autonomních robotů. Získání informací z obrazu kamer je nutné pro správné určení prostředí robota a jeho navigaci.

Existuje mnoho metod pro zpracování obrazu k nalezení informací. V komplexních systémech, jako chytrá auta, se používají neuronové sítě k analýze okolí. Tyto metody se používají vzhledem k možnostem vytrénovat si neuronové sítě na specifické potřeby, které robot vyžaduje, s nízkou chybovostí při detekci. Tyto metody jsou ovšem velice náročné na výpočetní sílu pro jejich aplikaci. Z tohoto důvodu je nelze použít ve všech situací.

3.4.1 Detekce hran

Detekce hran (edge detection) je jedna z nezákladnějších metod zpracování obrazu. Tato metoda zvýrazňuje hrany mezi dvěma oblastmi s různými hodnoty jasu, lze jí například použít pro identifikaci popředí a pozadí obrazu. Tato metoda se používá jako základní krok pro další metody zpracování obrazy, jako například: image segmentation, shape features či texture features. [10]

3.4.1.1 R-CNN

Cílem R-CNN k detekci objektů je snížit počet možných kandidátů pro výskyt objektů v obraze nezávisle na ostatních takovýchto oblastech. Na R-CNN staví další neuronové sítě jako třeba Faster R-CNN pro snížení času nutného k detekci objektů.R-CNN samostatně již není příliš používána z důvodu pomalého trénování modelů a zpracování obrazu.[11]

3.4.1.2 Faster R-CNN

Faster R-CNN se skládá ze dvou hlavní části, RPN (region proposal network) a Fast R-CNN. RPN je síť jež navrhuje možné oblasti výskytu objektů. Fast R-CNN vychází z R-CNN a jejím účelem je identifikace objektů v navrhovaných oblastí. [11]

Pro zrychlení výpočtu Faster R-CNN umožňuje sdílení informací mezi jejími dvěma částmi pro zrychlení výpočtu. Výstupem této sítě jsou informace o oblastech, které obsahují detekované objekty a popisky pro jejich identifikaci.[11]

3.4.1.3 Mask R-CNN

Mask R-CNN využívá stejnou dvou fázovou strukturu s první identickou fází, RPN. Ve druhé fázi běží paralelně dva procesy, Fast R-CNN k identifikace objektů a Mask R-CNN pro nalezení masek výskytu jednotlivých objektů. [11]

Kapitola 4
Návrh

Naším úkolem je implementovat aplikaci, která bude vyhledávat pozemní značky a vyhýbat se překážkám. Implementaci lze rozdělit na tři logické celky.

1. zpracování obrazu
2. vyhledání cesty pro průzkum zadané oblasti
3. navigace k jednotlivým bodům průzkumu

4.1 Zpracování obrazu

Aplikace bude zpracovávat obraz ze dvou kamer, horizontální a vertikální. Obraz z těchto kamer musí být zpracován pro potřebné informace k navigaci dronu.

Obraz z horizontální kamery je třeba zpracovat pro nalezení předem známého typu objektu/cíle. Z tohoto důvodu nemusíme pro zpracování obrazu použít hluboké neuronové sítě, ale vystačíme si s funkcemi poskytující například knihovny OpenCV[12].

Záznam vertikální kamery na druhou stranu poskytuje obraz na kterém se budou vyskytovat objekty o jejichž typu nemáme žádné informace. Z tohoto důvodu je třeba pro jejich zpracování využít hluboké neuronové sítě. Tyto sítě by se dali použít i pro zpracování horizontální kamery, ale z důvodu vysoké výpočetní síly pro jejich použití zvolíme jednoduší prostředky pro detekci pozemních cílů.

4.1.1 Detekce pozemních cílů

Díky jednoduchosti vyhledávaných cílů lze pro jejich detekci využít jednodušších principů zpracování obrazu, a není nutno obraz zpracovávat pomocí neuronových sítí.

Pro správné provedení detekce hran, musíme snímek převézt do grayscale, následně rozmazat obrázek pomocí Gaussovského filtru a použít Canny operátor pro samotnou detekci hran. Následně musíme najít způsob, kterým převedeme tyto hrany do datové formy ve které budeme schopni detekovat tvary hran a rozhodnou, zda se jedná o námi hledaný cíl.

Jelikož známe reálnou velikost těchto cílů můžeme poměrně přesně odhadnout jejich vzdálenost od dronu pomocí převedení pixelů na reálnou vzdálenost, díky znalosti velikosti reálných cílů.

4.1.2 Detekce překážek

Pro zpracování snímků z kamery je použita hluboká neuronová síť Detectron2. Kvůli vysoké výpočetní složitosti je tato část aplikace nasazena na serveru speciálně vybaveném ke zpracování obrazu.

Aplikace na obsluhu vertikální kamery předá snímek aplikaci, která komunikuje se serverem přes otevřený socket na specifickém portu. Server po přijetí celého obrázku aplikuje obrázek jako vstup neuronové sítě a výsledné informace o čtvercových obrysech objektů na obrazu odešle zpět dronu pro zpracování.

4.2 Průzkum oblasti

Pro zajištění detekce všech cílů na zadaném území musíme zvolit algoritmus, který bude schopný pokrýt celou oblast. Jelikož dron má relativně krátkou výdrž baterie (8 minut), nelze se spoléhat na naivní náhodné řešení.

Pro nalezení cesty pro optimální pokrytí nejdříve musíme zjistit orientační záběr horizontální kamery. Při počítání optimálního pokrytí cesty musíme počítat s menším záběrem než kamera má. Důvodem je možnost výskytu cílů na hranici záběru jenž náš algoritmus nedetekuje. Některé oblasti tudíž budou pokryty dvakrát pro zaručení spolehlivé detekce všech cílů.[13]

4.3 Navigace

4.3.1 Analýza

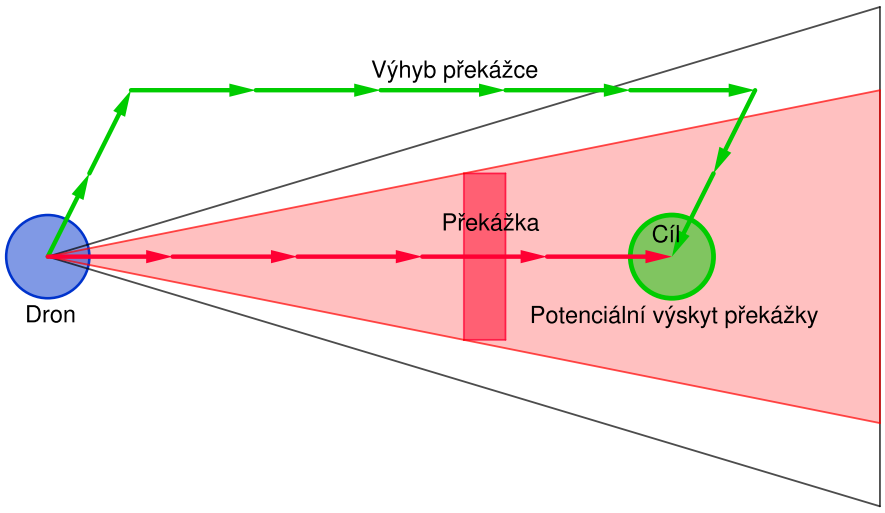
Pro navržení správné metody pro navigaci dronu k zadanému cíli musíme správně porozumět omezením vyplývající z hardwarového vybavení modelu dronu se kterým pracujeme. Z popisu dronu v kapitole 2 známe senzory, kterými je dron vybaven a jaká data z nich můžeme získat. Pro získání informací o prostředí ve kterém se dron vyskytuje lze použít ultrazvukový senzor pro zjištění vzdálenosti od země, horizontální a vertikální kameru. Ani jedna z těchto možností nám neumožňuje získat informace o vzdálenosti překážek od dronu. Z tohoto důvodu naše aplikace nemůže vytvářet mapu prostředí, podle které by byla schopná navigovat dron k cíli.

Dron tedy nemůže předem plánovat bezkolizní let a musí reagovat na překážky po jejich detekci přední kamerou. Ze stejného důvodu, z něhož nemůžeme plánovat cestu, dron nemůžeme využívat ani algoritmus FGA k plánování úhybných manévrů, jelikož nemůže vypočítat optimální dobu pro vyhnutí se.

4.3.2 Řešení

Jelikož dron nemůže detekovat vzdálenost k překážce musíme předpokládat, že se překážka může vyskytovat kdekoliv v daném směru detekovaného objektu jak je vidno z obrázky 4.1. Z tohoto důvodu bereme celý tento prostor jako překážku.

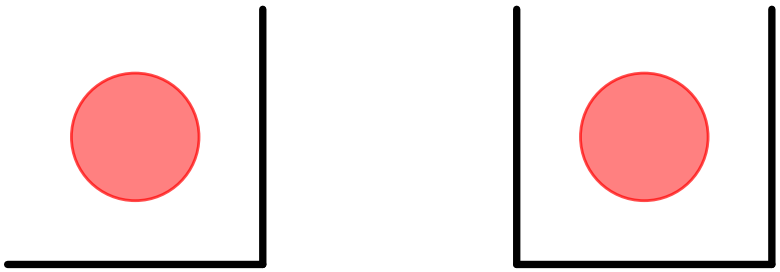
Jakmile dron takovouto překážku detekuje vypočítá si levnější úhybný manévr. To je takový manévr, který vyžaduje nejmenší úhyb z původní kurzu k cíli. Tímto novým směrem dron pokračuje zadanou dobu. Po dokončení tohoto manévru se dron otočí tak, aby jeho nová dráha letu byla rovnoběžná s jeho původní dráhou před zahájením úhybu. Po této dráze pokračuje dokud se jeho vzdálenost k cíli zmenšuje, nebo nenarazí na novou překážku. Jakmile se vzdálenost od cíle začne vzdalovat dron se otočí k cíli a pokračuje dokud se k němu nedostane k dostatečné vzdálenosti.



Obrázek 4.1 Diagram výhybu překážce

4.3.3 Omezení řešení

Postup pro navigaci dronu není dokonalý a existují situace, ve kterých se nedokáže dostat k cíli i přesto, že cesta k němu existuje. Například na obrázku 4.2 lze vidět dvě situace ve kterých dron nenajde cestu k cíli.



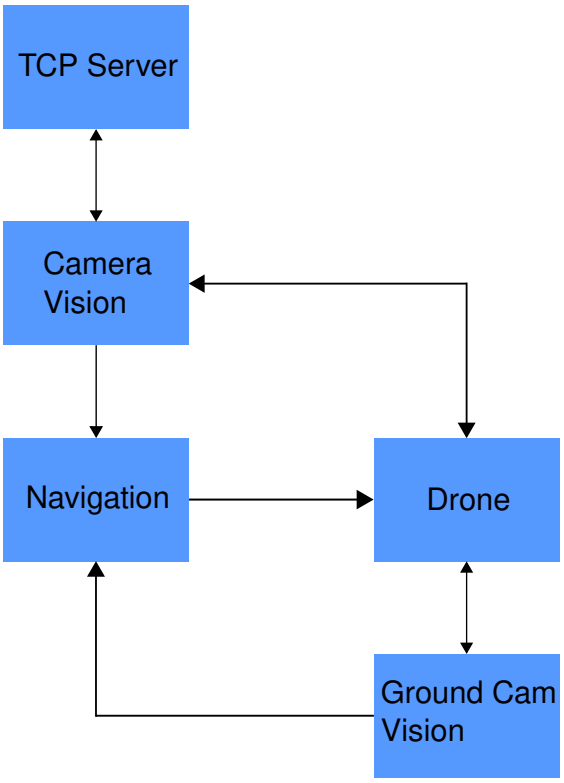
Obrázek 4.2 Omezení navigace dronu

Toto omezení vyplývá z nemožnosti detekovat vzdálenost k překážce. Tato situace by nebyla vyřešena ani při vytváření mapy okolí, jelikož dron by detekoval překážku ve všech možných úhlech snímání a tudíž by detekoval cíl jako nepřístupný.

Naše řešení ovšem tuto situaci neoznačí jako nepřístupný cíl, ale bude se k němu neustále pokoušet přiblížit.

4.4 Aplikace

4.4.1 Struktura aplikace



Obrázek 4.3 Diagram

Z diagramu jde vidět, že aplikace je rozdělena na 5 základních modulů:

- Systém pro navigaci dronu
- Ovládání pohybu dronu a získávání dat ze senzorů
- Obsluha vertikální kamery
- Obsluha horizontální kamery a zpracování jejího obrazu
- Zpracování obrazu z vertikální kamery

Tyto moduly jsou vytvořené jako samostatné části programu a v případě potřeby je při dodržení definovaného rozhraní možno je úplně nahradit. Aplikace je tvořena modulárně především pro potřeby vyměnění modulů pro zpracování obrazu z kamer, jež každý běží na separátním vlákně pro snadné periodické získávání snímků z kamer.

4.4.2 Drone modul

Tento modul slouží především jako wrapper nad knihovnou Pyparrot pro usnadnění ovládání dronu a zpracování potřebných dat ze senzorů. Třída Drone inicializuje Mambo objekt, zkontroluje správné připojení a provede potřebná nastavení pro správné fungování dronu. Hlavním

účelem tohoto obalu je zaručení správné volání funkcích s dostatečným časem pro jejich provedení.

4.4.3 Navigační systém

Tento modul má za účel vypočítat body pro optimální průzkum zadané oblasti na základě zadaného záběru spodní kamery. Zároveň získává informace o pozemních cílech z části aplikace Ground Cam Vision a filtruje je pro vyloučení falešných cílů. Falešný cíl je takový, který je v bezprostřední blízkosti jiného cíle, který je již detekován. Vznik falešných cílů je způsoben nedokonalými senzory snímající polohu dronu.

4.4.4 Obsluha horizontální kamery a zpracování jejího obrazu

Tento modul slouží k automatickému periodickému pořizování snímků z horizontální kamery, jeho zpracování a přeposlání informací získaných z obrazu dalším částí aplikace.

Pyparrot knihovna poskytuje pouze funkci pro pořízení obrázku, získání názvu snímků, stažení obrázku z dronu a jeho smazání. Zásadní problém horizontální kamery je prodleva mezi pořízením snímku a jeho uložení v paměti dronu (1.5-2.5s). Z tohoto důvodu musíme vždy při pořízení fotky uložit aktuální pozici dronu pro zpětné vypočítání přesné polohy případného cíle na obrázku.

Pro zpracování obrazu z horizontální kamery jsou využity základní metody edge detection pomocí canny operátoru[10]. Pro odstranění šumu z obrazu je použit mediánový filtr. Poté na snímku najdeme kruhy a díky fixní velikosti cíle známe od uživatele, pomocí pixel-per-metric metody, získáme vzdálenost cíle od dronu.

4.4.5 Obsluha vertikální kamery

Obsluha vertikální kamery je přímo zabudovaná v Pyparrot knihovně. Po základní incializaci dronu stačí vytvořit objekt spravující kameru a nastavit příslušnou callback funkci, které bude zavolána při pořízení každého nového snímku kamery. Uvnitř callback funkce následně získáme poslední validní snímek a ten následně můžeme uložit/předat ke zpracování.

Tento modul slouží jako obal pro vlákno pro obsluhu kamery. Každý pořízený snímek je odeslán na server přizpůsobený pro zpracování obrazu. Server pošle zpět rámce nalezených objektů a modul je nadále přepošle dalším částí aplikace pro zpracování již získaných dat.

4.4.6 Zpracování obrazu z vertikální kamery

K detekci jednotlivých objektů je použita síť Detectron2[14], která se zakládá na Mask R-CNN. Zpracování pomocí neuronových sítí je vysoce náročné na výpočetní sílu. Z tohoto důvodu tato část aplikace poběží na specializovaném serveru, uzpůsobenému ke zpracování obrazu.

Kapitola 5
Implementace

5.1 Dron

5.1.1 Připojení dronu

Dron Parrot Mambo nabízí připojení přes Wi-fi a přes bluetooth. Jelikož připojení přes bluetooth neposkytuje dostatečně rychlé stahování obrázků z dronu použijeme v naší aplikaci Wi-Fi pro připojení k dronu. Připojení k dronu obstarává třída `Mambo`, která dědí z třídy `Minidrone` správnou komunikaci aplikace s dronem.

Výpis kódu 5.1 Připojení se k dronu

```
mamboAddr="12:34:56:AB:CD:EF"
mambo = Mambo(address=mamboAddr, use_wifi=True)
success = mambo.connect(num_retries=3)
```

Objekt jde inicializovat dvěma parametry, *address* a *use_wifi*. Parametr *address* poskytuje MAC adresu pro připojení dronu pomocí bluetooth. Druhý parametr *use_wifi* specifikuje zda se má aplikace připojit k dronu pomocí Wi-Fi nebo bluetooth. Připojení přes Wi-Fi funguje pouze pokud je ke dronu připojena FPV kamera a počítač je připojen k síti dronu (název sítě odpovídá názvu Mambo_*číslo dronu*). Promněná *sucess* udává zda připojení k dronu proběhlo úspěšně a je možné pokračovat v programu.

V ukázce kódu 5.1 je předána objektu , `Mambo`, při jeho inicializaci, MAC adresa dronu. Tato adresa je nutná pouze při připojování se k dronu přes bluetooth.

5.1.2 Zásady ovládání

5.1.3 Senzory

Objekt `Mambo` obsahuje instanci třídy `MinidroneSensors`, která obsahuje zpracování dat dostupných ze senzorů, provádí dodatečné výpočty a volá callback funkci při obnovení dat ze senzorů, je-li specifikovaná uživatelem.

V souboru `commandsandsensors/minidrone.xml` v knihovně Pyparrot lze najít informace o senzorech, a jaká data odesílají. V třídě `MinidroneSensors` je malá část dat ukládaná v samostatných členských proměnných. Zbytek nespecifikovaných údajů je zachycen v proměnné `sensors_dict`, kde lze získat příslušná data dle adresace jejich jména.

Aplikace pracuje především s údaji o poloze a rotaci dronu, pro určení dalšího pohybu dronu. Dron umožňuje získání informací o souřadnicích dronu vůči bodu vzletu, spolu s rotací dronu a timestamp, údajem o uběhnutém času od posledního přijmutí dat.

5.1.4 Pozice dronu

Dron odesílá řadu informací ze svých senzorů. Jejich celý seznam lze vyčíst ze souboru `minidrone.xml` v Pyparrot knihovně. Nejdůležitější informací, kterou potřebujeme o dronu znát je informace o jeho pohybu či jeho přesnou polohu. Tyto informace poskytuje položka *NavigationDataState* ve výše zmíněném souboru.

1. DronePosition
2. DroneSpeed
3. DroneAltitude
4. DroneQuaternion

Nejdůležitější položkou pro nás bude *DronePosition*. Senzory této kategorie poskytují informace o pozici dronu a jeho rotaci vzhledem k bodu startu (nikoli bodu kde byl dron zapnut). Při běhu programu dron poskytuje informace o poloze dronu. Senzory by měli vracet i hodnotu rotace dronu vůči startovní pozici, senzor bohužel vrací nulovou hodnotu. Informace o rotaci dronu dokážeme nahradit díky senzorům z *DroneQuaternion*. Knihovna Pyparrot sama obsahuje funkci `get_estimated_z_orientation`, která na základě údajů z jiných senzorů vrací odhadovanou rotaci dronu. Hlavním problémem nefunkčnosti tohoto senzoru představuje orientace dat o pozici dronu. Kvůli nedostupné informaci o orientaci dronu při vzletu, jsou data o pozici dronu orientována vzhledem k zapnutí dronu. Z tohoto důvodu musí být dron zapínán již správně orientován pro správnou funkčnost aplikace.

5.1.5 Pohyb dronu

Jelikož detekce překážek dronu je možná jen pomocí FPV kamery, snímající okolí před dronem, pohyb dronu musí být vždy dopředu. Když budeme pohybovat s dronem do jiné strany s vzhledem k jeho aktuální rotaci, riskujeme srážku s překážkou, jež jsme nedetekovali.

Omezení pohybu pouze dopředu nám snižuje počet parametrů ovlivňující rychlost a délku tohoto pohybu.

- parametr `pitch` ve funkci `fly_direct`
- parametr `duration` ve funkci `fly_direct`
- hodnota maximálního náklonu dronu

Tyto tři parametry mají hlavní vliv na charakteristiku pohybu. Parametr `pitch` je procentuální hodnota maximálního dronu. Tato hodnota ovlivňuje rychlost, kterou se dron vydá. Parametr `duration` ovlivňuje dobu po kterou je příkaz k letu opakován, tedy přímo ovlivňuje délku tohoto pohybu. Velikost maximálního náklonu dronu je nastavena pomocí funkce `set_max_tilt`. Tato hodnota spolu s parametry `pitch` a `roll` ovlivňuje rychlost pohybu dronu.

Tyto tři parametry dohromady ovlivňují pohyb dronu dopředu a jejich správná konfigurace je potřebná pro správné fungování aplikace.

5.1.6 Vyhnutí se překážce

Navigace k cíli je implementována ve třídě `Drone` pomocí funkci `moveTo`. Za pomocí tří pomocných funkcí: `calculateDistance`, `turn` a `decideManouver` tato funkce naviguje dron k cíli.

Funkce `calculateDistance` vypočítá Euklidovskou vzdálenost mezi aktuální pozicí dronu a cílem.

5.1.6.1 Funkce turn

Tato funkce implementuje otočení dronu na zadaný cíl či na zadaný úhel ve virtuální mřížce vůči dronu. K implementaci otočení není využita funkce `fly_direct`, jelikož parametr udávající otočení je zadáván jako procentuální zrychlení rotace dronu, nikoli přesný počet stupňů k otočení. Místo toho je užita funkce knihovny `turn_degrees`, která bere jako parametr úhel v intervalu $\langle -180, 180 \rangle$ o který se má otočit.

V případě zadání úhlu na který se má dron otočit, stačí spočítat rozdíl úhlu mezi osou x dronu a úhlem k otočení. Následně stačí jednoduše ověřit zda je úhel v intervalu $\langle -180, 180 \rangle$. Pokud není převedeme úhel do požadovaného intervalu a můžeme zavolat funkci `turn_degrees`. V případě otáčení se na zadaný úhel nedochází k nepřesnostem výpočtu, jelikož pracujeme s úhly jako celými čísly a dochází pouze k jejich sčítání a odčítání.

Při otáčení dronu na cíl musíme vypočítat úhel k cíli buďto pomocí Sinovy, Cosinovy nebo Pythagorovy věty, tudíž může docházet k nepřesnostem ve výpočtu, které se musíme snažit minimalizovat.

Nejdříve vypočítáme úhel, který svírá osa x a pozice dronu s cílem. Poté vypočítáme rozdíl mezi tímto úhlem a úhlem natočení dronu a na závěr ověříme zda se výsledný úhel k otočení nachází v intervalu $\langle -180, 180 \rangle$.

Pro výpočet úhlu α na obrázku si vytvoříme trojúhelník jehož vrcholy jsou C, P a P+10 (bod se souřadnicemi $[P_x + 10, P_y]$). Úhel α poté vypočítáme pomocí Cosinovy věty a následně převedeme úhel otočení tak aby spadal do intervalu $\langle -180, 180 \rangle$.

5.1.6.2 Funkce decideManouver

Účelem této funkce je analyzovat informace o detekovaných překážkách a určit další pohyb dronu. Naše metoda navigace dronu vychází z algoritmů FGA a BUG. Algoritmus FGA provádí úhybné manévry na dvou rovinách, horizontální a vertikální. Z důvodu fixního nastavení záběru spodní kamery pro detekci optimální cesty pro prozkoumání zadané oblasti, nelze měnit výšku letu dronu, tudíž úhybné manévry budou prováděny pouze v horizontální rovině, nikoliv vertikální.

Ke spočítání manévru funkce využívá informace o bounding box detekovaných objektů. Detekce je provedena pomocí Mask R-CNN, která poskytuje přímo masku objektu. Kvůli nasazení zpracování obrazu na serveru, používáme pouze boundig box nikoliv masku objektu, jelikož je tato informace mnohem snazší na odeslání ze serveru a přijetí.

Na základě těchto informací funkce `decideManouver` zjistí zda se nějaký objekt vyskytuje v dráze dronu a případně zjistí, která strana při vyhýbání se objektu, který blokuje cestu, způsobí menší vychýlení z původní dráhy.

Je nutno brát v potaz, že objekty se mohou překrývat a při počítání menší výchylky se musí brát v potaz všechny objekty, nejen objekt blokující cestu.

5.1.6.3 Funkce moveTo

Dron se otočí na cíl *target* specifikovaný parametrem funkce za použití funkce `turn`. Toto otočení je provedeno na místě, není kombinováno s jiným pohybem dronu. Následně se začne pohybovat k cíli.

Pokud má dron informace o překážkách detekovaných FPV kamerou, zjistí zda mu nějaká překáží v cestě dopředu. Pokud má volnou cestu pokračuje v pohybu dopředu dokud se nedostane k dostatečné vzdálenosti od cíle. V případě detekce překážky před dronem, funkce `decideManouver` zjistí, do jaké strany by se měl vyhnout a vrátí příslušnou hodnotu pohybu pro funkci `fly_direct`. Směr vyhnutí se bere jako nejlevnější manévr, tedy manévr při kterém dojde k otočení o nejmenší možný úhel.

Úhybný manévr je prováděn po dobu dvou sekund pro zajištění průletu dronu od potenciální překážky. Po skončení úhybného manévru, dron změní svojí rotaci takovým způsobem, že nová dráha je rovnoběžná s přímou dráhou mezi startovním bodem a cílem. Pokud dron narazí na další překážku provede vyhnutí obdobným způsbem.

Jakmile se dron začne vzdalovat od svého cíle aniž by zrovna prováděl úhybný manévr, otočí se zpět na cíl pomocí funkce `turn`.

Výpis kódu 5.2 Navigace dronu k cíli

```
def moveTo(self, target):
    roll, pitch, yaw, vertical = 0, 20, 0, 0
    targetAngle = self.turn(target=target)
    mTs = 0

    d = con.calculateDistance(target, self.m_position.getPosition())

    while 0.3 < d:
        d = con.calculateDistance(target,
                                  self.m_position.getPosition())

        vertical, yaw = self.makeManouver()

        if vertical + yaw != 0:
            self.avoiding = True
        else:
            self.avoiding = False

        if self.lastDistance < d and self.avoiding == False:
            self.turn(target)
            self.lastDistance = d

        if self.avoiding == False:
            if 2 < time.time() - mTs:
                self.turn(targetAngle)
                mTs = time.time()

        self.m_drone.fly_direct(roll, pitch, yaw, vertical, 0.5)

    self.m_drone.smart_sleep(1)
```

5.2 Navigace

Pro nalezení optimální cesty pro průzkum zadané oblasti musíme vzít v potaz šířku záběru spodní kamery. Pro správné určení této cesty si musíme ovšem uvědomit rozdíl mezi skutečným záběrem kamery a záběrem kamery, ve kterém jsme schopní detekovat pozemní cíle.

Jelikož je detekce pozemních cílů prováděna pomocí knihovny OpenCV[12] a nikoliv neuronových sítí, nedokážeme detekovat části cílů, které se mohou vyskytovat na hranách snímků kamer. Překrývání záběru je také nutné vzhledem k nedokonalému letu dronu. Nemůžeme spoléhat na rovný pohyb dronu, jelikož může docházet k nepřesnostem například při otáčení.

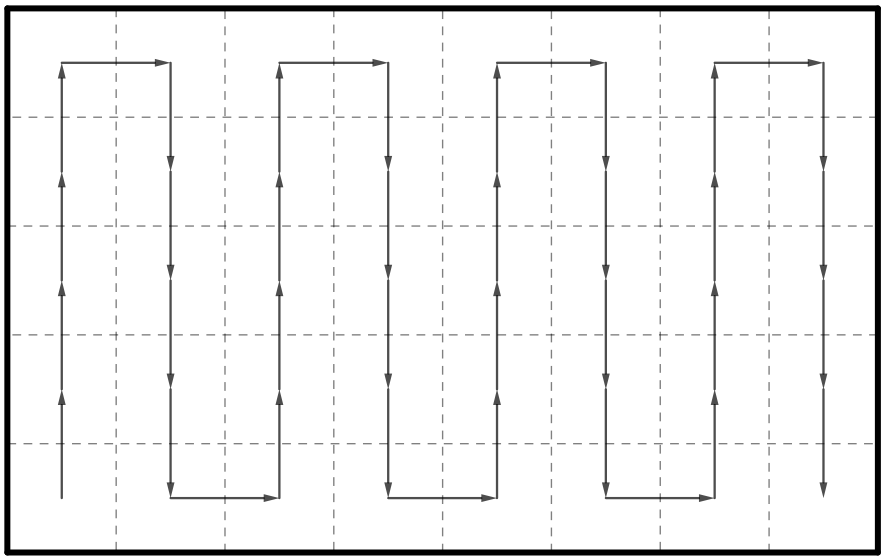
Z překrývání záběru kamer ovšem vzniká problém duplicitní detekce cílů. Z tohoto důvodu je třeba filtrovat cíle a určit nějakou mezní vzdálenost mezí cíli, při které se dva cíle budou považovat za totožné.

5.2.1 Hledání cesty

Hledání cesty v zadané oblasti funguje na principu jejího rozdělení do mřížky. Oblast rozdělím do mřížky, kde je jedna dlaždice této mřížky rovna velikosti efektivního záběru kamery. Získáme si středy jednotlivých dlaždic a rozdělíme si je dle sloupců.

Pro vytvoření cesty z těchto bodů nám stačí seřadit body v každém sloupci. A postupně odebírat jednotlivé body.

Sloupce rozdělíme do skupin dle stejné souřadnice x. Tyto skupiny seřadíme dle souřadnice x. Následně každý sudý sloupec seřadíme vzestupně dle souřadnice y a liché sloupce seřadíme sestupně. Takováto metoda nám vytvoří optimální průchod mřížkou jak je vidět na obrázku 5.1



Obrázek 5.1 Optimální průchod mřížkou

5.3 Horizontální kamera

5.3.1 Stažení obrázku

Pořízení snímku pomocí horizontální kamery je implementováno metodou `take_pictue` třídy `Minidrone`. Pokud je dron připojen přes Wi-Fi, tato funkce zkontroluje zda v dronu je uloženo méně jak 35 snímků z kamery. Při překročení tohoto počtu smaže všechny snímky, jelikož při překročení počtu 40 snímků v paměti dron ignoruje další snímky.

Stažení obrázku je realizováno třídou `MamboGroundcam`. Instance této třídy je vytvořena při inicializaci třídy `Minidrone`, pouze v případě připojení dronu přes Wi-Fi. Tato třída inicializuje FTP připojení na adrese 192.168.99.3. Tato možnost nefunguje pro nejnovější verzi firmwaru **3.0.26**, proto pro správné fungování aplikace je nutné, aby verze firmwaru dronu byla **3.0.17**.

Metoda `get_groundcam_picture`, vyžaduje název souboru v souborovém systému dronu, pro stažení snímku do počítače. Pomocí metody `get_groundcam_pictures_names` lze získat seznam snímků uložených v dronu. Nemůžeme se spoléhat na stahování obrázku dle názvu, tudíž musíme porovnat seznam snímků před a po pořízení snímku nebo na začátku aplikace paměť dronu vyčistit a vždy po stažení snímku ho z dronu vymazat.

5.3.2 Obsluha horizontální kamery

Jelikož snímek se nejdříve musí zapsat do souborového systému dronu nastává prodleva mezi odesláním příkazu k pořízení snímku a možností jeho stažení. Tato prodleva se pohybuje v rozmezí 1.5-2.5 sekundy. Z tohoto důvodu nemůžeme implementovat řešení, které by detekovalo cíle během letu a okamžitě na ně reagovalo. Systém pro detekci pozemních cílů musíme navrhnout takovým způsobem, aby byl schopný pořizovat snímky během letu a následně je stahoval se zpožděním. Pro následné detekování cílů a zjištění jejich pozice je tudíž nutné, aby tento modul uchovával informace o pozici a rotaci dronu v době pořízení snímku.

Tento mechanismus je implementován v třídě `groundCamHandler`. Při zavolání metody `start` se vytvoří nové vlákno programu, které se stará o stahování snímků z dronu.

Výpis kódu 5.3 Obsluha spodní kamery dronu

```
while self.running is True:
        pos, rot = self.getDroneInfo()
        self.drone.getPicture()

        time.sleep(1.5)
        list = self.drone.m_drone.groundcam.get_groundcam_pictures_names()

        if len(list) == 1:
                pic = self.drone.downloadPicture(list[0])
                self.callback(pic, pos, rot)
        else:
                self.drone.deletePictures()
```

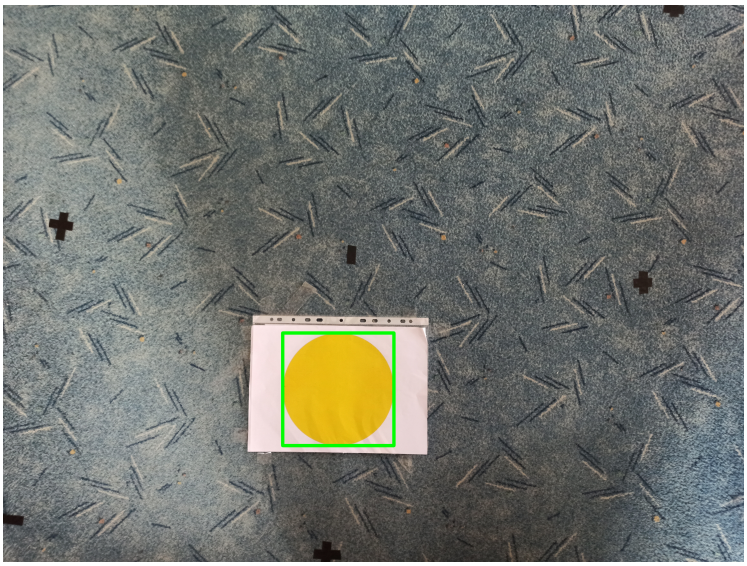
Vlákno si nejdříve uloží informace o pozici a rotaci dronu, poté odešle příkaz pro pořízení snímku a po časové prodlevě si stáhne seznam souborů v dronu. Pokud tento seznam obsahuje více než jednu položku, nemůžeme stahovat snímek a hledat cíle, jelikož nedokážeme určit, ke kterému snímku přiřadit naše informace o pozici a rotaci dronu. Z tohoto důvodu jsou všechny snímky z dronu před spuštěním vlákna smazány. Pokud je v seznamu jen jedna položka, můžeme stáhnout snímek a zavolat funkci pro zpracování obrazu, zadanou při inicializaci `groundCamHandler`. Pokud jsou ve snímku detekovány nějaké cíle, vlákno zavolá všechny callback funkce, které se zadávají pomocí metody `setCallback`.

5.3.3 Zpracování obrazu

Jelikož hledané cíle jsou všechny stejné a všechny snímky jsou foceny téměř ze stejného úhlu, použijeme k jejich detekci knihovnu OpenCV[12] místo zpracování pomocí neuronových sítí. Pro nalezení cílů použijeme jednoduchou metodu detekce hran pomocí Canny operátoru.

Pro správnou detekci hran musíme nejdříve převést snímek do greyscale podoby pomocí funkce `cv2.cvtColor`. Následně je obraz třeba rozmazat funkcí `cv2.GaussianBlur` pro správnou detekci hran pomocí `cv2.Canny`. Nyní máme zvýrazněné hrany. Pro zjištění a popis obrysů pro následné zpracování provedeme pomocí funkce `cv2.findContours`. Není projdeme všechny obrysy a zjistíme, které jsou kruhovitého tvaru. Na základě informací u velikosti kruhu v pixelech a velikosti reálného cíle v cm, můžeme vypočítat reálnou vzdálenost cíle od středu obrazu.

Tímto nezískáme přesný vzdálenost, jelikož snímek není přesná prezentace plochy, ale působí na něj zkreslení kamery. Kamera má ovšem malý záběr, tudíž takto vzniklá nepřesnost bude zanedbatelná (v řádech milimetrů). Nepřesnost v řádech milimetrů či jednotkách centimetrů je zanedbatelná vůči nepřesnosti informací o poloze dronu.



Obrázek 5.2 Detekce pozemních cílů

5.4 Vertikální kamera

5.4.1 Obsluha vertikální kamery

Obsluha FPV kamery je implementována v Pyparrot knihovně pomocí třídy `DroneVision`. Tato třída inicializuje připojení k FPV kameře, stahování a ukládání snímků z kamery. Metoda této třídy, `open_video`, spustí nový proces obstarávající stažení a ukládání snímků z kamery do počítače. Pro okamžité zpracování obrázků při jejich uložení do fronty lze použít metodu `set_user_callback_function`. Tato metoda uloží callback funkci jako nové vlákno.

Výpis kódu 5.4 Inicializace obsluhy FPV kamery

```
def __init__(self, drone):
    self.vision = DroneVision(drone, False, buffer_size=30)
    self.vision.set_user_callback_function(self.processImage,
                                           user_callback_args=None)
```

5.4.2 Zpracování obrazu

Zpracování obrazu z FPV kamery je provedeno pomocí neuronové sítě Detectron2. Použití neuronové sítě je nutností, jelikož nedokážeme předvídat žádné informace o objektech, které by se mohli vyskytovat v cestě dronu. I s předem daným typem překážek by jejich detekce bez použití neuronových sítí byla obtížná.

Neuronová síť Detectron2 musí běžet na specializovaném serveru. Kvůli prodlevě nutné pro inicializaci sítě Detectron2 není možné kopírovat snímek pomocí `scp` na server a vzdáleně spouštět skript, který by ho zpracoval. Na serveru tudíž musí běžet část aplikace, která bude komunikovat s klientem přes otevřený socket. Tato část nakonfiguruje síť Detectron2 a počká na připojení klienta. Klient následně odesílá snímky z kamery, které server zpracuje a zpět odešle informace o bounding box detekovaných objektů.

Komunikace ze strany klienta je implementována v třídě `Client`. Tato třída inicializuje připojení na server `10.10.48.91`. Pomocí metody `send` můžeme odesílat snímky dronu na server, následně získáme informace o objektech v poli, které je vráceno touto metodou.

Toto pole má dimenzi $[2, 2, n]$, kde n je přirozené číslo. Každá položka v tomto poli signalizuje dvojici bodů, které označují levý horní a pravý dolní roh boundig box objektu.

5.4.2.1 Detectron2

Detectron2 je knihovna nové generace výzkumu AI od firmy Facebook. Knihovna poskytuje moderní algoritmy pro segmentaci a detekci objektů. Tato síť vychází z jejího předchůdce Detectron a projektu Mask R-CNN benchamrk.

Výpis kódu 5.5 Inicializace neuronové sítě Detectron2

```
cfg = get_cfg()

cfg.merge_from_file(model_zoo.get_config_file(
                "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(
                "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")

cfg.freeze()
predictor = DefaultPredictor(cfg)
```

Pro detekci objektů Detectron2[14] využívá model zoo. Ten obsahuje trénované modely pro různé úkoly jako detekce objektů či sémantickou segmentaci.

Obrázek 5.3 Detekce objektů

Kapitola 6
Specifikace aplikace

6.1 Instalace

6.1.1 Knihovna Pyparrot

Aplikace vyžaduje několik úprav knihovny Pyparrot pro správné fungování. Nejnovější firmware **3.0.26** dronu Mambo nepodporuje stahování obrázků z horizontální kamery dronu a připojení k dronu přes FTP je vypnuto. Pro povolení možnosti připojení je nutno upravit část kódu v souboru `Minidrone.py` na řádcích 237-243 na následující:

Výpis kódu 6.1 Připojení k dronu přes FTP

```
try:
    self.ftp = FTP('192.168.99.3')  # IP-Address of the drone itself
    login = self.ftp.login()
    print("FTP login success is %s" % login)
except:
    print("ERROR: ftp login is disabled by parrot firmware
          3.0.25 and 26.  Groundcam will not work.")
    self.ftp = None
```

Následně je nutné změnit hodnotu promněné `fullpath` v souborech `Minidrone.py` na řádku 246 a v souboru `DroneVision`na řádku 95.

Výpis kódu 6.2 Původní cesta k uložení snímků

```
fullPath = inspect.getfile(Mambo)
```

Hodnota této promněné je třeba změnit na následující hodnotu.

Výpis kódu 6.3 Nová cesta k uložení snímků

```
fullPath = os.getcwd()
```

Tato změna umožní ukládání stažených obrázků do podsložky `images` v aktuální pracovní cestě na místo vytvoření této podsložky v cestě knihovny Pyparrot, tudíž spuštění aplikace nebude vyžadovat práva superuživatele/administrátora pro zápis obrázků.

Další nutná změna pro správné fungování aplikace je třeba změnit frekvenci volání callback funkce specifikované uživatelem ke zpracování snímků z FPV kamery. Na řádku 208 ve funkci `_user_callback` je nutno změnit dobu uspání na čas odpovídající rychlosti zpracování obrazu. V případě této aplikace je doba k uspání nastavená na 1 sekundu.

Výpis kódu 6.4 Změna frekvence volání callback funkce

```
time.sleep(1.0 / (3.0 * self.fps))
```

Následně je třeba v metodě `update` třídy `MinidroneSensors` dodat ukládání dat ze senzorů do vlastních proměnných. Naše aplikace k těmto datům přistupuje rovnou jako k samostatným členským proměnným nikoliv přes seznam dat ze senzorů, `sensor_dict`, jež sbírá ostatní data jež nebyli specifikována.

Výpis kódu 6.5 Změna frekvence volání callback funkce

```
if (name == "DronePosition_posx"):
    self.position_x = value
elif (name == "DronePosition_posy"):
    self.position_y = value
elif (name == "DronePosition_posz"):
    self.position_z = value
elif (name == "DronePosition_ts"):
    self.position_ts = value
elif (name == "DronePosition_psi"):
    self.position_psi = value
```

6.2 Aplikace

Aplikace je rozdělena na dvě hlavní části: část aplikace běžící na serveru a část běžící na počítači.

Část aplikace nasazená na serveru vyžaduje soubory uložené ve složce `server` a spouští se pomocí soubory `tcp_client.py`.

Druhá část aplikace běžící na serveru je uložena ve složce `local` a je spuštěna pomocí souboru `main.py`.

6.3 Testování a funkčnost aplikace

Aplikace jako celek bohužel nebyla otestována kvůli problémům týkajících se modelu dronu.

Dron zasílá mnoho dat o sobě a o svém okolí. Dva typy informací nám mohou pomoci s určením jeho polohy v prostoru a to jsou: data o poloze a údaje o rychlosti dronu. Údaje o rychlosti jsou dronem aktualizovány s frekvencí 1 Hz a jsou vraceny v kontextu aktuální rotace dronu. Kvůli pomalému obnovování dat není možné tyto informace využít k výpočtu pozice při dlouhodobém letu. Problémem na stává při zrychlování a zpomalování dronu. Rychlost z tohoto okamžiku je mnohem vyšší než rychlost, kterou dron letí zbytek letu a zanášejí velikou nepřesnost do výpočtu pozice. Data o pozici dronu jsou také vracena s vysokou nepřesností.

Bez relativně spolehlivých informacích o pozici dronu není možné implementovat systém pro navigaci dronu, jelikož nelze implementovat spolehlivý průchod zadanou oblastí a ani není možné spolehlivě dopočítat jednotlivě detekované pozemní cíle.

Kapitola 7
Závěr

Hlavním cílem této práce byl návrh a implementace aplikace schopné autonomní navigace dronu. Dron by měl být schopný detekovat pozemní cíle a zároveň se umět vyhýbat překážkám v prostoru.

V rámci této práce bylo vytvořeno celkem čtyři částí aplikace: zpracování obrazu ze spodní kamery, detekce objektů pomocí FPV kamery, navigační systém a modul pro ovládání dronu.

Obsluha spodní kamery byla úspěšně implementována jako spuštění nového vlákna pro pravidelné stahování obrázků a jejich zpracování. Detekce pozemních cílů bylo dosaženo pomocí knihovny OpenCV a dosahuje relativně vysoké spolehlivosti v omezeném testovacím prostředí.

Detekce objektů pomocí FPV kamery se musí v aktuálních podmínkách provádět na specializovaném serveru a z toho důvodu bylo nutné vytvořit část aplikace, které bude běžet na serveru zpracovávat obraz z FPV kamery. Pro část aplikace běžící na lokálním počítači bylo nutné implementovat třídu schopnou komunikace se serverem. Tato část aplikace funguje a je řádně otestována. Aplikace bez problému komunikuje se servem, odesílá snímky ke zpracování a přijímá data o nalezených objektech.

Navigačním systém byl implementován pr nalezení bodů po kterých by se dron měl vydat k průzkumu oblasti. Tento modul byl samostatně otestován (mimo propojení se zbytkem aplikace) a spolehlivě funguje pro hledání bodů dle zadané šířky kamery.

Modul pro ovládání dronu byl otestován a dosahuje značných nedostatků plynoucích z nedostatků dronu. Dron má senzor pro určení pozice, ovšem tento senzor nevrací spolehlivé data o poloze dronu tudíž není možné implementovat tuto část aplikace spolehlivě.

Kvůli problémům při implementaci modulu pro ovládání dronu nebylo možné aplikace otestovat jako celek v delším běhu. Z tohoto důvodu neexistují data o její spolehlivosti.

Sekundární cílem bylo získání informací o možnostech ovládání více dronů v rámci aplikace a poskytnout názor o proveditelnosti takového řešení.

Ze získaných zkušeností o ovládání dronu není ovládání více dronů z jedné aplikace doporučena, kvůli nedostatku plynoucích z nemožnosti spolehlivě získávat informace o poloze dronu.

Dalším nedostatkem by byla vzájemná komunikace dronů. Dle informací získaných z githubu pyparrot[2] knihovny plyne, že není možné se přihlásit k více dronům z jedné aplikace. Tento fakt tvoří vzájemnou komunikaci téměř nereálnou a bez komunikace mezi drony není opodstatnění jich ovládat více najednou.

Bibliografie

1. *Drone Simulation and Control, Part 1: Setting Up the Control Problem*. YouTube, 2018. Dostupné také z: https://www.youtube.com/watch?v=hGcGPUqB67Q&list=PLy8TVa88QVfoOotVLloGWvK6PMSdjEtSb&index=1&t=65s.
2. *Welcome to pyparrot's documentation!* [Online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://pyparrot.readthedocs.io/en/latest/.
3. PARROT-DEVELOPERS. *Parrot-Developers/RollingSpiderEdu* [online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://github.com/Parrot-Developers/RollingSpiderEdu.
4. *Parrot Minidrones Support from Simulink* [online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://www.mathworks.com/hardware-support/parrot-minidrones.html.
5. *Autonomous Navigation, Part 1: What Is Autonomous Navigation?* YouTube, 2020. Dostupné také z: https://www.youtube.com/watch?v=Fw8JQ5Q-ZwU&t=32s.
6. YUFKA, Alpaslan; PARLAKTUNA, Osman. Performance Comparison of BUG Algorithms for Mobile Robots. In: 2009. Dostupné z DOI: 10.13140/RG.2.1.2043.7920.
7. LIN, Zijie; CASTANO, Lina; XU, Huan. A Fast Obstacle Collision Avoidance Algorithm for Fixed Wing UAS. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2018, s. 559–568. Dostupné z DOI: 10.1109/ICUAS.2018.8453307.
8. GONZÁLEZ, David; PÉREZ, Joshué; MILANÉS, Vicente; NASHASHIBI, Fawzi. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*. 2016, roč. 17, č. 4, s. 1135–1145. Dostupné z DOI: 10.1109/TITS.2015.2498841.
9. KARAMAN, Sertac; WALTER, Matthew R.; PEREZ, Alejandro; FRAZZOLI, Emilio; TELLER, Seth. Anytime Motion Planning using the RRT*. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, s. 1478–1483. Dostupné z DOI: 10.1109/ICRA.2011.5980479.
10. LIU, Yang; XIE, Zongwu; LIU, Hong. An Adaptive and Robust Edge Detection Method Based on Edge Proportion Statistics. *IEEE Transactions on Image Processing*. 2020, roč. 29, s. 5206–5215. Dostupné z DOI: 10.1109/TIP.2020.2980170.
11. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, s. 2980–2988. Dostupné z DOI: 10.1109/ICCV.2017.322.
12. 2021. Dostupné také z: https://opencv.org/.

13. YANG, Ming-Der; BOUBIN, Jayson G.; TSAI, Hui Ping; TSENG, Hsin-Hung; HSU, Yu-Chun; STEWART, Christopher C. Adaptive autonomous UAV scouting for rice lodging assessment using edge computing with deep learning EDANet. *Computers and Electronics in Agriculture*. 2020, roč. 179, s. 105817. ISSN 0168-1699. Dostupné z DOI: `https://doi.org/10.1016/j.compag.2020.105817`.

14. WU, Yuxin; KIRILLOV, Alexander; MASSA, Francisco; LO, Wan-Yen; GIRSHICK, Ross. *Detectron2* [`https://github.com/facebookresearch/detectron2`]. 2019 [cit. 2021-04-24].

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `implementation` ..... zdrojové kódy implementace
    - `controller` ..... zdrojové kódy aplikace běžící na lokálním počítači
    - `server` ..... zdrojové kódy pro část aplikace běžící na serveru
    - `html` ..... zdrojové kódy dokumentace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF