

Sledování objektů zájmu pomocí kvadkoptéry

Objects of Interest Tracking Using Quadcopter

Zadání bakalářské práce

Student: **Radek Simkanič**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: **Sledování objektů zájmu pomocí kvadkoptéry**
Objects of Interest Tracking Using Quadcopter

Zásady pro vypracování:

Cílem práce je detekce objektů zájmu v obrazech z kamery umístěné na kvadrikoptéře. Detekci objektů proveďte pomocí příznaků HOG a klasifikátoru SVM nebo jiného.

Ve své práci proveďte:

1. Seznamte se s API pro daný typ kvadrikoptéry a řádně jej popište.
2. Nastudujte metody HOG a SVM pro detekci objektů zájmů (např. lidská tvář).
3. Z kamery získejte obraz a detekujte v něm objekty pomocí nastudovaných metod.
4. Ovládejte kvadrikoptéru tak, aby sledovala zadaný cíl, který se bude pohybovat a udržovala od něj definovaný odstup.
5. Svě závěry z implementace řádně popište.

Seznam doporučené odborné literatury:

- [1] Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection, Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 886-893 (2005)
- [2] Burges, Christopher J. C.: A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery 2:121-167 (1998)

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

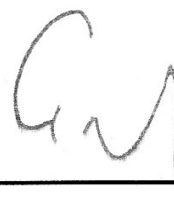
Vedoucí bakalářské práce: **Ing. Jan Gaura**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry



prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 07. 05. 2014

.....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 07. 05. 2014

.....

Rád bych poděkoval mému vedoucímu bakalářské práce Ing. Janu Gaurovi za odbornou pomoc a poskytnutí přístupu ke školnímu serveru, Ing. Radovanu Fuskovi za odbornou pomoc a nasměrování v nerozhodných chvílích. Dále bych chtěl poděkovat za pevné nervy mých přátel, rodičů a mé přítelkyně. V neposlední řadě bych chtěl poděkovat babičce Marcelle Dvořákové za kontrolu pravopisu a Miroslavu Mecovi za diskutování nad různými problémy. Závěrem bych chtěl poděkovat zelenému čaji za udržení mé pozornosti a mému notebooku, který vydržel ne zrovna ukázkové zacházení.

Abstrakt

Tématem této bakalářské práce je detekce a pronásledování objektů zájmu (lidí) zpracováním obrazu získaného z kamery umístěné na kvadrokoptéře. Detekce lidí je prováděna za pomoci SVM detektoru a již implementovaného HOG deskriptoru z knihovny OpenCV. Tyto detekce se ukládají do bufferu a následně jsou zpracovávány algoritmem, který určí jednu detekci zájmu. Teoretická část je zaměřena na všeobecný popis drona a počítačového vidění. Praktická část popisuje samotný program, třídy, algoritmy či řešení. Zdrojové kódy jsou napsány v objektově orientovaném skriptovacím programovacím jazyku Python.

Klíčová slova: ARDrone, kvadrokoptéra, kvadrikoptéra, OpenCV, HOG, SVM, sledování, detekce, Python

Abstract

This work deals with detections and tracking objects of interest (people), processing the image obtained from the camera mounted on quadcopter. SVM detector and HOG descriptor are used for the target detection, both implemented in OpenCV library. These detections are inserted into the buffer and processed by the algorithm selecting the key detection. Theoretical part is focused on the drone description and the computer vision in general. Practical part describes the program itself, its classes, algorithms or solutions. Source codes are written in object orientated scripting programming language Python.

Keywords: ARDrone, Quadcopter, OpenCV, HOG, SVM, tracking, detection, Python

Seznam použitých zkratk a symbolů

CR	– Carriage Return - nový řádek
SVM	– Support Vector Machines - Metoda strojového učení.
HOG	– Histogram of Oriented Gradients
API	– Application Programming Interface - Rozhraní pro programování aplikací
SDK	– Software Development Kit - Sada nástrojů pro vývoj
GUI	– Graphical User Interface - Grafické rozhraní
LED	– Light-Emitting Diode - Dioda emitující světlo
AT	– ATention code
LiPo	– Lithio-polymerova baterie
TCP	– Transmission Control Protocol
UDP	– User Datagram Protocol
HD	– High Definition - Vysoké rozlišení 720p nebo 1080i
ARM	– Advanced RISC Machine - Mikroprocesor založený na filosofii RISC
Wi-Fi	– Označení pro několik standardů IEEE 802.11
QVGA	– Rozlišení 320x240
RPM	– Revolutions Per Minute - Otáčky za minutu
MIPS	– Million Instruction Per Second - Jednotka výkonnosti počítače, která udává počet zpracovaných instrukcí za sekundu
DDR2	– Double-Data-Rate 2 SDRAM - Technologie pro ukládání pracovních dat
DDR3	– Double-Data-Rate 3 SDRAM - Technologie pro ukládání pracovních dat
GPS	– Global Positioning System - Globální polohovací systém
AVR	– Označení pro rodinu 8bitových a některých 32bitových mikročipů typu RISC s harvardskou architekturou od firmy Atmel
RAM	– Random Access Memory - Paměť s přímým přístupem
ESSID	– Extended Service Set Identifier - Identifikátor Wi-Fi sítě
IP	– Internet Protocol
DHCP	– Dynamic Host Configuration Protocol - Používá se pro automatickou konfiguraci počítačů připojených do počítačové sítě.

Obsah

Úvod	6
1 AR.Drone	7
1.1 Technická specifikace	7
1.1.1 Konstrukce	7
1.1.2 Kamera a nahrávání videa	7
1.1.3 Motory a vrtule	9
1.1.4 Senzory, signalizace a elektronika	9
1.2 Princip létání	9
1.3 Komunikace	10
1.4 AT příkazy	11
1.4.1 Syntaxe AT příkazů	12
1.4.2 Seznam příkazů	12
1.5 Navigační data (navdata)	13
2 Počítačové vidění a detekce	15
2.1 OpenCV	15
2.2 HOG a SVM	15
3 Popis navržených algoritmů	17
3.1 Porovnání podobností dvou detekčních obdélníků (Compare Rectangles)	17
3.2 Průměrování detekčních obdélníků (Averaging rectangles)	19
3.3 Průměrování podobných detekčních obdélníků	19
3.4 Odfiltrování nenavazujících detekcí (Filtering incorrect detection)	20
3.5 Detekční obdélník zájmů (Key rectangle)	23
4 Popis vlastního API	26
4.1 API pro ovládání kvadrikopty	26
4.2 API pro jednodušší práci s obrázky a detekcemi	27
4.3 API pro socketovou komunikaci	28
4.4 Kruhový buffer	29
5 Implementace	30
5.1 Klientská část	30
5.2 Serverová část	30
Závěr	33
6 Literatura	34
Reference	34
Přílohy	35

Appendix	35
A Příklad: průměrování podobných detekčních obdélníků	35
B Příklad: detekční obdélník zájmů (Key rectangle)	38
C Přiložené soubory	43
C.1 Bakalářská práce	43
C.2 Modul ardrone_control_lib	43
C.3 Modul detection_worker	43
C.4 Modul socket_server_client	43
C.5 Třída HybridRingBuffer	43
C.6 Hotová serverová část	43
C.7 Hotová klientská část	43
C.8 Vlastní vytváření HOG descriptorů	43
D Popis tříd a modulů	44
D.1 Modul ardrone_control_lib	44
D.1.1 Třída ARDrone2	44
D.2 Modul detection_worker	46
D.2.1 Třída PicturesConvert	46
D.2.2 Třída RectanglesWorker	47
D.2.3 Třída FramesWorker	49
D.3 Modul socket server	50
D.3.1 Třída Message	50
D.3.2 Třída Handler	51
D.3.3 Třída SocketServer	51
D.3.4 Třída Client	51
D.4 Třída HybridRingBuffer	52

Seznam tabulek

1.1	Shrnutí komunikace.	11
1.2	Seznam příkazů s parametry.[5]	13
1.3	Seznam id LED animací pro AT*LED.[6]	14
1.4	Seznam id pohybu pro AT*ANIM.[7]	14
3.1	Demonstrační obdélníky.	18
A.1	Seznam detekčních obdélníků.	36

Seznam obrázků

1.1	Schéma drona s kryty[3]	8
1.2	Indoorový kryt[5]	8
1.3	Outdoorový kryt[5]	8
1.4	Směr otáčení vrtulí[5]	9
1.5	Stoupání[5]	10
1.6	Otáčení doleva[5]	10
2.1	HOG detektory. (a) Ukázka trénování průměrováním gradientů již zpracovaných obrázků. (b) Každý "pixel" zobrazuje maximální pozitivní SVM váhu daného bloku soustředující se na pixel. (c) Stejně tak pro negativní SVM váhu. (d) Zkušební obraz. (e) Vypočítaný R-HOG descriptor. (f, g) SVM pozitivní/negativní váha R-HOG descriptoru. [12]	16
3.1	Grafické vyobrazení čtverců, jejich středů a následné vyobrazení porovnávacího čtverce.	19
3.2	Grafické vyobrazení bufferu s detekčními snímky.	22
5.1	Grafické vyobrazení komunikace mezi dronem, notebookem a serverem Merlin2	31
5.2	Okno s detekcemi	31
5.3	Okno s detekcí zájmu	31
5.4	vyobrazení obou oken (aktuální stream, aktuální detekce) a Tkinter okna pro zachytávání kláves	32
B.1	Grafické vyobrazení bufferu s detekčními snímky.	39

Seznam výpisů zdrojového kódu

1	Syntaxe AT příkazů	12
2	Ukázka dvou AT příkazů	12
3	Zapnutí podrobných navigačních dat	13
4	Ukázka použití modulu ardrone_control_lib: vzletnutí, použití animací diod, zatančení a následné přistání	26
5	Ukázka použití třídy: PicturesConvert - nalezení detekcí lidí a jejich následné zaznamenání v originálním obrázků	27
6	Ukázka použití třídy: RectanglesWorker	27
7	Syntaxe použitá v modulu socket server	28
8	Ukázka syntaxe s parametry	28
9	Ukázka syntaxe bez parametrů	28
10	Ukázka použití třídy Handler	28
11	Ukázka použití třídy SocketServer	29
12	Ukázka použití třídy Client	29

Úvod

Detekce lidí, obličejů či jiných objektů zájmů je počítačovou disciplínou, spadající do počítačového vidění, která má široké využití. Detekce tváře je v současné době již dobře vyřešena. Náročnější je to již s detekcí postav, což zůstává stále velkou výzvou a využívá se mnoha metod.

Detekce má využití ve spoustě odvětvích, jako je medicína, robotika, ochrana osob a majetku, armáda, usnadnění každodenní činnosti atd., ale vše má dvě strany a i tato technologie má svou zápornou tvář. Téměř každý si dovede představit zneužití těchto technologií ve spojení například s robotikou, jak je i často prezentováno nejen ve sci-fi filmech nebo v omezování lidských práv a svobod.

Cílem práce je využít dnes volně dostupné technologie a vyzkoušet si zprovoznění jednoduchého autonomního pronásledování objektu zájmu tak, aby od tohoto cíle byl udržován definovaný odstup, přičemž objekt zájmu bude osoba nacházející se před objektem AR.Drone od firmy Parrot, kde další pozdější detekované osoby nebudou brány v potaz do té doby, až první osoba nezmizí.

Teoretická část je zaměřena na popis částí AR.Drone, z kterých se skládá a jak probíhá komunikace mezi dronem a klientským zařízením. Dále obsahuje základní pojmy počítačového vidění použité v této práci.

V praktické části se budou popisovat navržené algoritmy, API vytvořených tříd, které se budou používat pro detekci či autonomní chování ARDrona a celková implementace. Mezi popisované algoritmy budou patřit: porovnávání podobnosti dvou detekčních obdélníků, odfiltrování nenavazujících detekcí, průměrování obdélníků a získání obdélníku zájmu.

1 AR.Drone

Parrot AR.Drone, kvadrikoptéra nazývaná také kvadrokoptéra, je stroj se 4 vrtulemi v rovině. Lze se setkat i s dalšími názvosloví jako quadrokoptéra, quadkoptéra, quadrotor, apod. V této bakalářské práci se bude používat Parrot AR.Drone 2.0 a názvosloví kvadrikoptéra případně dron.

AR.Drone může být ovládán primárně z mobilních zařízení Apple (iPhone, iPad, iPod) a zařízeními s operačním systémem Android. První generace podporovala i mobilní zařízení s operačním systémem Symbian. Neoficiálně AR.drone lze ovládat z jakéhokoli zařízení podporující komunikaci přes Wi-Fi. [1, 4]

Balení obsahuje kvadrikoptéru AR.Drone, LiPol baterii, nabíječku, kryt pro indoorové létání, kryt pro outdoorové létání, reflexní barevné samolepky a návod k použití. Kryty jsou vyrobené z tvrzeného polystyrénu. Robustní kryt pro indoorové (viz obrázek 1.2) létání zabraňuje kontaktu vrtulí s překážkou, aby se vrtule o danou překážku nepoškodily, případně aby se snížilo nebezpečí poškození předmětů a vybavení místnosti. Menší outdoorový kryt (viz obrázek 1.3) umožňuje snadnější a stabilnější létání a to i při mírně zhoršených povětrnostních podmínkách. Schéma drona s kryty je vyobrazeno na obrázku 1.1. Reflexní samolepky slouží pro hru (AR.Flying Ace) dvou kvadrikoptérů Ar.Drone proti sobě. [2, 4]

1.1 Technická specifikace

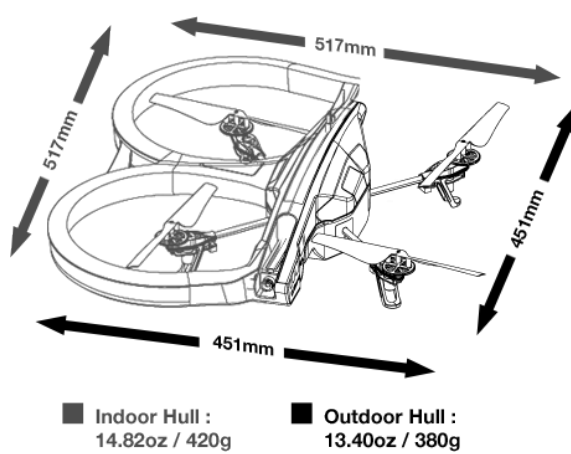
Ovládání a komunikace probíhá přes Wi-Fi. Na straně drona je Wi-Fi Access Point. V přední a spodní části se nachází kamera.

1.1.1 Konstrukce

Konstrukce u druhé generace vydrží více než u první. Došlo totiž k vyztužení nejvíce zatěžovaných částí. Nosný kříž je zpevněný karbonem. Celková hmotnost s indoorovým krytem je 420 g a outdoorovým 380 g. Pro snížení vibrací k vnitřním dílům je použita pěna. Na ultrazvukových senzorech se nachází vodoodpudivá nano vrstva. Všechny díly drona jsou plně opravitelné a na stránkách výrobce se nacházejí videa, jak postupovat při opravě nebo výměně poškozených dílů. [2]

1.1.2 Kamera a nahrávání videa

Video stream v reálném čase se během letu zobrazuje na displeji smartphone, tabletu či jiném zařízení. Kvadrikoptéra obsahuje dvě kamery. Hlavní kamera s širokoúhlým objektivem směřující dopředu podporuje rozlišení až 1280 x 720 pixelů (720p) při 30 snímcích za sekundu. Širokoúhlý objektiv snímá v zorném úhlu 92°. Video záznam lze ukládat za letu i do USB Flash disku připojený ke kvadrikoptéře. Vertikální kamera slouží primárně k měření rychlosti. Zaznamenává v rozlišení 320 x 240 pixelů (QVGA) při 60 snímcích za sekundu. [2]



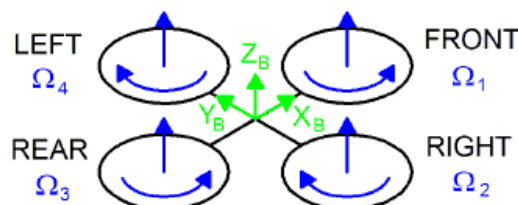
Obrázek 1.1: Schéma drona s kryty[3]



Obrázek 1.2: Indoorový kryt[5]



Obrázek 1.3: Outdoorový kryt[5]



Obrázek 1.4: Směr otáčení vrtulí[5]

1.1.3 Motory a vrtule

Kvadrikoptéra má 4 plastové pružné vrtule, umístěné na konci nosného kříže poháněné výkonnými inrunner motory s příkonem 14,5 W a až 28 500 otáček za minutu (RPM). Pro ovládání každého motoru zvlášť, slouží procesory 8 MIPS AVR. Mezi vrtulemi a motory jsou převodová kola z Nylatronu s převodovým poměrem 1/8,75. Vrtule se otáčí na hřídeli z tvrdé oceli. [2]

1.1.4 Senzory, signalizace a elektronika

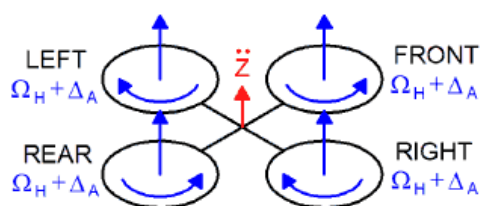
Na každém konci nosného kříže jsou umístěny zelené a červené signalizační diody. Drone je ovládán centrálním 32 bitovým procesorem ARM Cortex A8 o frekvenci 1 GHz a jako operační paměť mu slouží 1 Gb DDR2 RAM 200 MHz. Pro připojení GPS příslušenství nebo USB Flash paměti slouží USB 2.0 port. Wi-Fi podporuje standardy b/g/n.

Drone má v sobě zabudovanou automatickou stabilizaci. Pro určení umístění a výšky slouží následující senzory:[2]

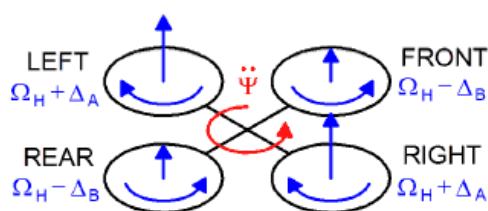
- 3-osý gyroskop s přesností $2000^\circ/\text{s}$,
- 3-osý akcelerometr s přesností $\pm 50 \text{ mg}$,
- 3-osý magnetometr s přesností $\pm 10 \text{ Pa}$ (80 cm nadmořské výšky).
- Ultrazvukový senzor pro měření výšky nad zemí nacházející se ve spodní části kvadrikoptéry.
- Vertikální QVGA kamera.

1.2 Princip létání

Létání kvadrikoptér je velmi specifické a ne zrovna triviální, veškerý pohyb se vytváří na základě změny otáček jednotlivých vrtulí. Dvě vrtule se točí po směru hodinových ručiček a dvě proti směru, tak že protilehlé vrtule se otáčí ve stejném směru viz obrázek 1.4. [5]



Obrázek 1.5: Stoupání[5]



Obrázek 1.6: Otáčení doleva[5]

Úhlová rychlost vrtulí se značí Ω . Zvýšením či snížením otáček u dané vrtule se značí Δ . Pro stoupání se musí zvýšit otáčky o Δ_A (viz obrázek 1.5. Případně pro klesání se musí otáčky snížit o Δ_A .

Pro let vpřed je potřeba zvýšit otáčky o Δ_A u "REAR" a "RIGHT". Pro let vzad je potřeba zvýšit otáčky o Δ_A u "LEFT" a "FRONT". Pro let vpravo je potřeba zvýšit otáčky o Δ_A u "LEFT" a "REAR". Pro let vlevo je potřeba zvýšit otáčky o Δ_A u "RIGHT" a "FRONT".

Je-li potřeba otáčet s dronem na místě, pak pro otáčení doleva je potřeba zvýšit otáčky o Δ_A u "LEFT" a "RIGHT" a snížit otáčky o Δ_A u "REAR" a "FRONT" (viz obrázek 1.6). Pro otáčení doprava se zvýší otáčky o Δ_A u "REAR" a "FRONT" a sníží otáčky o Δ_A u "LEFT" a "RIGHT".

1.3 Komunikace

Kvadríkoptéra může být ovládaná z klientského zařízení za pomoci Wi-Fi. Při zapnutí Drona a automatického spuštění Wi-Fi se vytvoří Wi-Fi síť s ESSID ve tvaru ardrone_xxx případně ardrone2_xxx s lichou IP adresou (většinou 192.168.1.1). Po připojení je klientskému zařízení přiřazená IP adresa za pomoci DHCP serveru (většinou 192.168.1.2). Po té může klientské zařízení posílat a přijímat komunikaci z portů 5554, 5555, 5556 a 5559. Celkové shrnutí je vyobrazeno v tabulce 1.1. [5]

Ovládání AR.Drone se provádí pomocí čtyř komunikačních služeb, přičemž první tři jsou hlavní:

- AT Commands

	AR.Drone 1.0	AR.Drone 2.0
Wi-Fi mode	ad-hoc	AP
SSID	ardrone_xxx	ardrone2_xxx
Navigační data - port	5554	5554
Navigační data - protokol	UDP	UDP
Video stream - port	5555	5555
Video stream - protokol	UDP	TCP
Letové příkazy - port	5556	5556
Letové příkazy - protokol	UDP	UDP
Kritické data - port	5559	5559
Kritické data - protokol	TCP	TCP

Tabulka 1.1: Shrnutí komunikace.

Ovládání a konfigurace drona je umožněno pomocí AT příkazů (*AT commands*) na UDP portu 5556. Tyto příkazy jsou posílány pravidelně - většinou 30 příkazů za sekundu. [5]

- **Navdata**

Navigační informace se nazývají *navdata* a odesílají se klientovi na UDP portu 5554. Přijímají se údaje jako pozice, rychlost, otáčky motoru, stav baterie apod. Dokonce se přijímají informace o detekovaných reflexních tagů, tyto informace mohou být použity pro vytváření hry s augmentovovu realitou. Jsou posílány 15 krát za sekundu v režimu *demo* případně 200 krát za sekundu v režimu *debug* (full). [5]

- **Video stream**

Video stream je vysílán na TCP portu 5555. Vyjímkou je první generace, kde se pro video stream používá protokol UDP. [5]

- **Control port**

Čtvrtý komunikační kanál se nazývá *control port* využívající TCP port 5559 pro přenos důležitých/kritických dat. Přenáší se zejména potvrzování přijatých konfiguračních příkazů, aby se případně poslaly znovu, nedojde-li k potvrzení. [5]

1.4 AT příkazy

AT příkazy (AT commands) slouží k ovládání a konfiguraci drona. Příkazy se posílají UDP protokolem na port 5556, jak je uvedeno v tabulce 1.1. Nepošle-li se žádný příkaz během 2 sekund, pak toto drone vnímá jako ztracený Wi-Fi signál a provede stabilizaci s následným přistáním. Je proto nutné posílat neustále nějaké příkazy. [5]

1.4.1 Syntaxe AT příkazů

Kompletní AT příkaz je řetězec, i čísla v něm obsažené je potřeba převést na řetězce. Řetězec je v kvadrikoptéře dekodován jako sekvence osmi bitových ASCII znaků. Každý příkaz začíná řetězcem `AT*` a je zakončen `CR`, respektive `\r`. [5]

Syntaxe je znázorněná ve výpisu 1. Ve výpisu 2 je ukázka dvou příkazů odeslaných do kvadrikoptéry.

```
AT*[název příkazu]=[číslo sekvence],[parametry příkazu oddělené čárkou]\r
```

Výpis 1: Syntaxe AT příkazů

```
AT*PCMD_MAG=21625,1,0,0,0,0,0,0\rAT*REF=21626,290717696\r
```

Výpis 2: Ukázka dvou AT příkazů

Nesprávné AT příkazy bude drone ignorovat. Sekvenční číslo slouží, aby se dodržela posloupnost příkazů. U každého následujícího příkazu se musí sekvenční číslo navýšit, obvykle o 1, ale není to pravidlem. Obdrží-li dron příkaz s nižším číslem než provedl naposledy, neprovede jej. Tato metoda se využívá na základě využitého UDP protokolu, kde se mohou pakety ztratit, či přijít v jiném pořadí nebo vícekrát. Vyjímkou je však příkaz `COMWDG`, který nuluje číslo sekvence. [5]

1.4.2 Seznam příkazů

Kompletní seznam příkazu je uvedeno níže. K jednotlivým příkazům je popis k čemu slouží. Seznam parametrů příkazů jsou obsaženy v tabulce 1.2.

REF Základní ovládání drona - vzletnutí, přistání, emergency (vypnutí motorů), zrušení emergency.

PCMD Pohyb drona.

PCMD_MAG Pohyb drona s podporou Absolute Control.

FTRIM Flat trims - signal pro označení, že dron leží v rovině. Vynulování hodnot gyroskopu.

CALIB Kalibrace magnetometru.

CONFIG Nastavení konfigurace drona.

CONFIG_IDS Identifikátory pro následující příkaz `AT*CONFIG`.

COMWDG Hlídací pes. Tento příkaz slouží k udržování (aktualizování) komunikace, případně změny číselné sekvence.

LED Spuštění LED diodové animace (za pomoci červených a zelených LED diod). Tabulka 1.3 uvádí seznam id LED animací.

Příkaz	Parametry
REF	číslo operace
PCMD	flag, roll, pitch, gaz, yaw
PCMD_MAG	flag, roll, pitch, gaz, yaw, psi, přesnost psi
FTRIM	-
CALIB	číslo zařízení
CONFIG	klíč, hodnota
CONFIG_IDS	id session, id uživatele, id aplikace
COMWDG	-
LED	id animace, frekvence animace (Hz), vypnutí animace (s)
ANIM	id pohybu, vypnutí daného pohybu (s)

Tabulka 1.2: Seznam příkazů s parametry.[5]

ANIM Spuštění pohybové animace, respektive provádění daného pohybu po určité době. Tabulka 1.4 uvádí seznam id pohybů.

1.5 Navigační data (navdata)

Navigační data jsou posílána přes protokol UDP na portu 5554, jak je uvedeno v tabulce 1.1. Slouží k předávání informací z kvadrikopty o jejím aktuálním stavu. Přijímané informace jsou například: stav baterie, stav drona, úhly pitch, roll a yaw, rychlost v ose X, Y a Z. Pro aktivaci podrobných navigačních dat je potřeba poslat příkaz:[5]

```
AT+CONFIG=123,"general:navdata_demo","TRUE"\r
```

Výpis 3: Zapnutí podrobných navigačních dat

Id	Popis animace
0	Blikání zelená/červená
1	Blikání zelená
2	Blikání červená
3	Blikání oranžová
4	Had zelená/červená
5	Oheň
6	Standard
7	Červená
8	Zelená
9	Červený had
10	Vypnuto
11	Střela vpravo
12	Střela vlevo
13	Dvojitá střela
14	Přední levý zelený, ostatní červené
15	Přední pravý zelený, ostatní červené
16	Zadní pravý zelený, ostatní červené
17	Zadní levý zelený, ostatní červené
18	Levé zelené, pravé červené
19	Levé červené, pravé zelené
20	Standardní blikání

Tabulka 1.3: Seznam id LED animací pro AT*LED.[6]

Id	Popis pohybu
0	Let vpravo pod úhlem 30°
1	Let vlevo pod úhlem 30°
2	Let dozadu pod úhlem 30°
3	Let dopředu pod úhlem 30°
4	Let dopředu pod úhlem 20° a otáčení ve směru hodinových ručiček
5	Let dopředu pod úhlem 20° a otáčení proti směru hodinových ručiček
6	Otočení dokola
7	Otočení dokola a klesání
8	Třesení - Střídavé otáčení vpravo/vlevo
9	Tanec - Otáčení vpravo/vlevo
10	Tanec - Houpání vpravo/vlevo
11	Tanec - Houpání vpřed/vzad
12	Tanec - stoupání/klesání
13	Vlnění
14	Houpání - vpravo/vlevo, vpřed/vzad
15	Dvojité houpání - 2x vpravo/vlevo, 2x vpřed/vzad

Tabulka 1.4: Seznam id pohybu pro AT*ANIM.[7]

2 Počítačové vidění a detekce

Počítačové vidění (Computer Vision) je název odvětví výpočetní techniky a vývoje softwaru, které se zabývá vývojem zařízení schopných získávat informaci ze zachyceného obrazu. Nejčastěji se jedná o rozpoznávání obličejů nebo registračních značek automobilů, ale obecně jde o celou problematiku zpracování videa, obrazu z více zdrojů nebo například obrazové analýzy materiálů. [8]

Obor počítačového vidění je v poslední době na vzestupu, k čemuž dopomáhají jak levnější a stále dokonalejší kamerové systémy, tak i snadná dostupnost potřebného výpočetního výkonu. Zároveň je nutné zmínit i stále snadnější dostupnost různých knihoven a materiálů, zaměřujících se právě na zpracování obrazu. Mezi poslední jmenované činitele patří i OpenCV, který rovněž stojí za růstem aplikací s počítačovým viděním a vývojářům po celém světě umožňuje snadný vstup do problematiky celé systému.

Detekce obličejů, očí, úst (případně úsměvu) není v dnešní době náročná, přičemž detekce lidí je naopak úkol náročný, protože každý člověk vypadá odlišně. Dalším faktem náročnosti detekce lidí způsobuje různé prostředí, ve kterém se nachází. Jeden z možných principů detekovat lidi v obraze lze využít SVM a HOG (viz níže), kde se využívá detekce hran. [12]

2.1 OpenCV

Se svým zaměřením na real-time zpracování, pomáhá OpenCV studentům i profesionálům jednoduše realizovat své dříve obtížně uskutečnitelné projekty. Dříve to bylo realizovatelné jen v několika vyspělých výzkumných laboratořích. [8, 9]

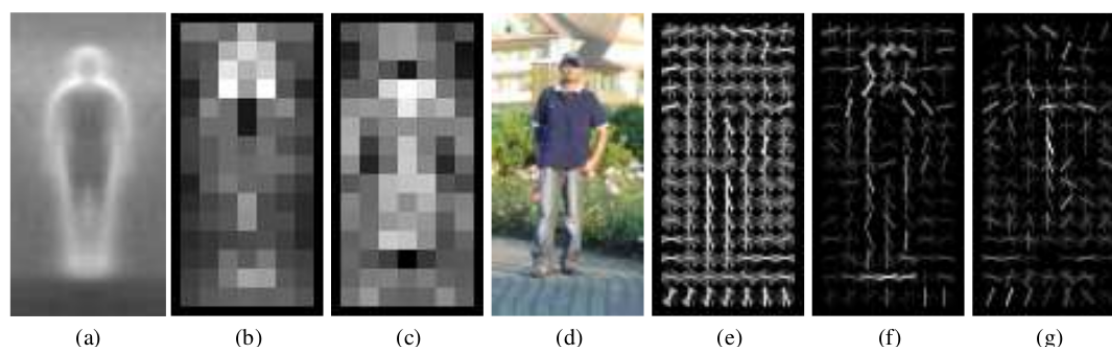
V jediné, navíc volně dostupné knihovně jsou k dispozici funkce počítačového vidění i strojového rozpoznávání předmětů. Obsahuje totiž více než 500 optimalizovaných algoritmů a pod BCD licencí je dostupná zdarma i pro komerční využití na systémech Linux, Mac a Windows. Je napsaná v C/C++ a interface je přístupný pro programovací jazyky: C/C++, Python, Ruby, Matlab, případně dalších. [8, 9]

OpenCV bylo navrženo pro výpočetní efektivitu se silným zaměřením na realtime aplikace. Proto je optimalizována v jazyce C a podporuje vícejádrové procesory. Pro případné další automatické optimalizace na Intel architektuře slouží (placená) knihovna IPP (Integrated Performance Primitives), která se skládá z nízkoúrovňových optimalizovaných rutin v mnoha algoritmických oblastech. OpenCV umí použít odpovídající IPP knihovnu přímo za běhu. [9]

Za zmínku stojí použití OpenCV u vytváření Google Street View. [9] V této práci bude použita právě tato knihovna pro získání detekci osob z video streamu.

2.2 HOG a SVM

Při této technice se využívají sady pozitivních, neutrálních a negativních obrazových snímků, při které se následně vytvoří HOG featury (viz obrázek 2.1), ty se následně zpracují za pomoci SVM. [12]



Obrázek 2.1: HOG detektory. (a) Ukázka trénování průměrováním gradientů již zpracovaných obrázků. (b) Každý "pixel" zobrazuje maximální pozitivní SVM váhu daného bloku soustředující se na pixel. (c) Stejně tak pro negativní SVM váhu. (d) Zkušební obraz. (e) Vypočítaný R-HOG descriptor. (f, g) SVM pozitivní/negativní váha R-HOG descriptoru. [12]

Před započítím detekce je potřeba převést obrázek do stupně šedi a provést filtraci šumu například Gaussovým filtrem. Vyfiltrovaný obrázek se rozčlení na mřížku s překrývajícími se bloky, do kterých se vytvářejí příznakové HOG vektory (HOG feature vectors). Tyto vektory jsou poslány do *linear SVM* pro klasifikaci objektů. [12]

V této práci jsem využil již hotových HOG descriptorů implementovaných v OpenCV. Načal jsem i vlastní vytváření HOG descriptorů (viz příloha C.8) na základě stažených obrázkových setů z INRIA Person Dataset (<http://pascal.inrialpes.fr/data/human/>) a detekce objektů.

3 Popis navržených algoritmů

Popis vlastních navržených algoritmů použitých v této práci, které jsou i následně naprogramovány. Většina algoritmů jsou následně předvedeny na konkrétním příkladu.

3.1 Porovnání podobností dvou detekčních obdélníků (Compare Rectangles)

Tento algoritmus slouží k porovnání dvou detekčních obdélníků, který je označí za podobné pokud na základě udaných parametrů jsou podobně velké a nachází se na podobném místě. Je potřeba předem znát následující vstupní parametry:

- **Size divergence coefficient [SDC]:** Koeficient určující povolený rozdíl velikosti obdélníků. Například $0.25 = 25\%$.
- **Position divergence coefficient [PDC]:** Koeficient určující maximální vzdálenost středů obdélníků na základě velikosti největšího obdélníku. Například $0.25 = 25\%$.

Princip algoritmu bude prezentován na základě příkladu o dvou obdélnících. Vlastnosti obdélníku jsou uvedeny v tabulce 3.1 s následujícími vstupními parametry: $SDC = 0.25$, $PDC = 0.25$. Osnova provádění algoritmu:

1. Porovnání velikosti obdélníků na základě SDC.
2. Zjištění, který obdélník má největší obvod.
3. Vytvoření čtverce/kružnice o stejném obvodu jako největší obdélník.
4. Roznásobení velikosti čtverce/kružnice PDC.
5. Zjištění, zda se druhý střed nachází ve čtverci/kružnici.

Porovná se zvlášť šířka a výška obdélníků. Šířka červeného je menší než šířka modrého obdélníku. Tato větší šířka (120) se označí jako **W**, menší (100) jako **w** a porovná se vzorcem:

$$w \cdot (1 + SDC) \geq W \quad (3.1)$$

Nebude-li tato podmínka platit, algoritmus se ukončí s negativní odpovědí. Po té se provede to samé s výškou. Větší výška (210) se označí **H**, menší (200) jako **h** a porovná se vzorcem:

$$h \cdot (1 + SDC) \geq H \quad (3.2)$$

Opět se navrátí negativní odpověď, nebude-li platit podmínka. Zjištění největšího obvodu se získá sečtením výšky a šířky u obou obdélníků:

- Půl obvod červeného: **HCc** = 300
- Půl obvod modrého: **HCm** = 330

Obdélník	Šířka	Výška	x_1	y_1	x_2	y_2
Červený	100	200	0	0	100	200
Modrý	120	210	20	20	140	230

Tabulka 3.1: Demonstrační obdélníky.

Modrý obdélník je větší. Velikost vzdálenosti středů lze provést porovnáním na základě čtvercové (**PSs** - Position Size square), nebo kružnicové velikosti (**PSc** - Position Size circle). V případě čtvercového porovnání je potřeba zjistit velikost strany dle vzorce 3.3. Následně se tato hodnota roznásobí PDC (viz vzorec 3.4).

$$s = \frac{HCm}{2} = \frac{330}{2} = 165 \quad (3.3)$$

$$PSs = s \cdot PDC = 41,25 \quad (3.4)$$

V případě kružnicového porovnání je potřeba zjistit poloměr kružnice dle vzorce 3.5. Tato hodnota se následně roznásobí PDC (viz vzorec 3.6).

$$r = \frac{HCm}{\pi} \simeq 105 \quad (3.5)$$

$$PSc = r \cdot PDC \simeq 26,26 \quad (3.6)$$

Získání souřadnic středu u červeného obdélníku je znázorněno ve vzorcích 3.7 a 3.8. Podobně se tak získají souřadnice středu modrého obdélníku viz vzorce 3.9 a 3.10.

$$SX_c = x_{1c} + \frac{width_c}{2} = \frac{100}{2} = 50 \quad (3.7)$$

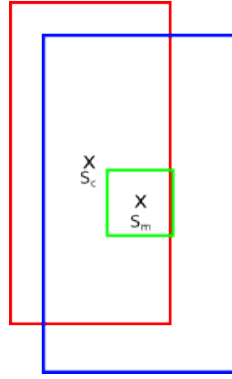
$$SY_c = y_{1c} + \frac{height_c}{2} = \frac{200}{2} = 100 \quad (3.8)$$

$$SX_m = x_{1m} + \frac{width_m}{2} = 20 + \frac{120}{2} = 80 \quad (3.9)$$

$$SY_m = y_{1m} + \frac{height_m}{2} = 20 + \frac{210}{2} = 125 \quad (3.10)$$

V dalším kroku je potřeba porovnat vzdálenost středů obdélníků. Pro čtvercový způsob je potřeba napřed zjistit souřadnice čtverce, viz vzorce 3.11, 3.12, 3.13 a 3.14. Následně se provede porovnání, zda střed z červeného čtverce je v souřadnicích tohoto čtverce. Pro kružnicový způsob je potřeba získat poloměr (viz vzorec 3.15), který se následně porovná se vzdálenosti středů.

$$x_1 = SX_m - \frac{PSs}{2} = 80 - \frac{41}{2} = 59,5 \quad (3.11)$$



Obrázek 3.1: Grafické vyobrazení čtverců, jejich středů a následné vyobrazení porovnávacího čtverce.

$$y_1 = SY_m - \frac{PSs}{2} = 125 - \frac{41}{2} = 104,5 \quad (3.12)$$

$$x_2 = x_1 + PSs = 59,5 + 41 = 100,5 \quad (3.13)$$

$$y_2 = y_1 + PSs = 104,5 - 41 = 145,5 \quad (3.14)$$

$$d = \sqrt{(SX_c - SX_m)^2 + (SY_c - SY_m)^2} \simeq 39 \quad (3.15)$$

V obou případech je odpověď záporná. Grafické vyobrazení je na obrázku 3.1

3.2 Průměrování detekčních obdélníků (Averaging rectangles)

Následující algoritmus slouží k vytvoření jednoho průměrného detekčního obdélníku na základě seznamu obdélníků. Sečtou se všechny souřadnice x_1 , y_1 , x_2 a y_2 . Součet těchto hodnot se podělí počtem detekčních obdélníků v seznamu.

3.3 Průměrování podobných detekčních obdélníků

Díky tomuto algoritmu se získají průměrné detekčních obdélníků, které jsou navzájem podobné. Využívá algoritmu pro porovnání podobnosti dvou obdélníků (Compare rectangles) popisován v kapitole 3.1 a algoritmus pro průměrování detekčních obdélníků viz kapitola 3.2. Detekční obdélníky jsou obsaženy v seznamu.

Jsou dva způsoby průměrování obdélníků v seznamu. První je klasický, jedno průchodový. Druhý je takzvaný bublinkový, který je n-průchodový - viz níže.

Algoritmus obsahuje parametry jen potřebné pro algoritmus Compare rectangles. Osnova provádění algoritmu (klasický způsob):

1. Zjištění, zda není buffer prázdný.

2. Získání detekčního obdélníku k porovnávání.
3. Porovnávání s ostatními detekčními obdélníky.
4. Zprůměrování podobných detekčních obdélníků.

Je-li buffer prázdný, výsledkem nebude žádný průměrný detekční obdélník a navrátí se prázdná hodnota. Bude-li buffer obsahovat jen jeden detekční obdélník, bude navrácen jako výsledek. Bude-li v bufferu více jak jeden detekční obdélník, pokračuje se v algoritmu.

Aktuální detekční obdélník k porovnávání s ostatními je označen jako **AR** (Actual Rectangle). Jednotlivé nalezené detekční obdélníky k zprůměrování se budou zaznamenávat do **AL** (Averaging List). Do AL se zaznamenává i AR. Všechny použité detekční obdélníky se zaznamenávají do **UR** (Used Rectangles).

Jedná-li se o první načtení detekčního obdélníku, nastaví se jako AR a vloží se také do AL. Nejedná-li se o první načtení detekčního obdélníku, zkontroluje se zda již není obsažen v UR. Je-li obsažen v UR přejde se k dalšímu detekčnímu obdélníku a tento proces se opakuje až do té doby, nebude-li obsažen v UR nebo se nenarazí na konec bufferu, jinak se zaznamená do AR, AL a UR. Narazí-li se na konec bufferu, předá se výsledné LAR.

Pro porovnávání s ostatními detekčními obdélníky se načte následující detekční obdélník od místa AR. Zkontroluje se zda není obsažen v UR. Pokud ano, načte se následující detekční obdélník, načítání se opakuje až do té doby, dokud nebude obsažen v UR, nebo se nenarazí na konec bufferu.

Získaný detekční obdélník se porovná s AR za pomoci algoritmu Compare rectangles. Je-li odpověď kladná, zaznamená se do AL a UR. Tento proces se opakuje až do té doby, nedorazí-li se na konec bufferu.

Všechny obsažené detekční obdélníky v AL se zprůměrují za pomoci algoritmu Averaging rectangles. Následně se tento seznam vyprázdní. Získaný průměrný obdélník se zaznamená do seznamu **LAR** (List averaged rectangles). Není-li AR již poslední v seznamu, přejde se k získání detekčního obdélníku k porovnávání. Po ukončení je výsledkem LAR.

Při bublinkovém způsobu se používá klasický způsob, který se opakuje dokud bude výstupní seznam jiný než vstupní. Při dokončení algoritmu se výstupní seznam použije vždy jako vstupní. Příklad je pro jeho délku prezentován v příloze A.

3.4 Odfiltrování nenavazujících detekcí (Filtering incorrect detection)

Následující popisovaný algoritmus využívá algoritmu pro porovnání podobnosti dvou obdélníků (Compare Rectangles) popisován v kapitole 3.1. Algoritmus je vhodný použit například u získaných detekcí na základě video sekvence tak, aby se předešlo náhodným detekcím a jejich následného odfiltrování.

Algoritmus obsahuje buffer o určité velikosti, který uchovává jednotlivé snímky s detekčními obdélníky. Pro tento snímek se používá i název frame.

Před použitím tohoto algoritmu je vhodné použít algoritmus zprůměrování podobných detekčních obdélníků (Averaging appropriate rectangles nebo Bubble averaging appro-

priate rectangles) pro jednotlivé snímky, popisované v kapitole 3.3. K níže uvedeným parametrům přibývají i parametry algoritmu Compare Rectangles.

- **Buffer size [BS]:** Velikost bufferu, který slouží k zachovávání jednotlivých framů, které obsahují detekční obdélníky.
- **Find in buffer [FB]:** Minimální počet framů, ve kterých musí být nalezené podobné detekční obdélníky. Maximální hodnota:

$$BS - 1 \geq FB \quad (3.16)$$

- **Step coefficient [SC]:** O kolik se má v každém následujícím framu navýšit PDC, pokud nebyla nalezená detekce v aktuálním framu.
- **Maximum step coefficient [MSC]:** Nepovinný parametr sloužící k určení maximálního navyšování PDC koeficientem SC.

Porovnávání probíhá na principu vlnění hada, to znamená, že nesmí být jednotlivé detekční obdélníky v následujícím framu příliš odlišné, na příliš odlišném místě. Obdélníky s příliš odlišnými parametry budou umazány a považovány jako šum. Osnova provádění algoritmů:

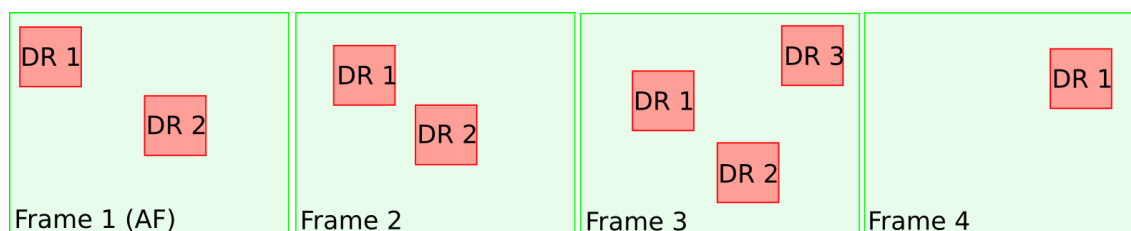
1. Zjištění, zda je buffer naplněn.
2. Získání prvního detekčního snímku s detekčními obdélníky.
3. Porovnání s následujícím detekčním snímkem.
4. Zhodnocení detekčního obdélníku.

Ad 1. Napřed se musí zjistit, zda je buffer naplněn na maximální hodnotu BS. Pokud ano, může algoritmus dále pokračovat, jinak výsledkem bude prázdná hodnota. Buffer může být například kruhový.

Ad 2. Získání prvního framu s detekcemi slouží, aby se mohly prohledávat následující framy s jejich uloženými detekčními obdélníky. Aktuální detekční snímek se značí jako **AF** (Actual Frame). Aktuální detekční obdélník se značí jako **ADR** (Actual Detection Rectangle). ADR je proměnlivý (hadovitý) - nenabývá neustále detekčních obdélníků z AF, obsahuje je jen na začátku.

Je-li první detekční snímek prázdný (bez detekčních obdélníků), pak se přechází k dalším detekčním snímkům tak dlouho, dokud nebude obsahovat minimálně jeden detekční obdélník. Nalezený detekční snímek bude považován za AF, pokud bude obsahovat minimálně jeden detekční obdélník.

Ad 3. Získaný ADR se porovná s následujícími detekčními obdélníky v detekčních snímcích za využití algoritmu Compare Rectangles. Při první pozitivní odpovědi algoritmu Compare Rectangles se ukončí další porovnávání detekčních obdélníků.



Obrázek 3.2: Grafické vyobrazení bufferu s detekčními snímky.

Tento pozitivní nález se zaeviduje do FC (Find Counter), respektive bude navýšen o jedničku. Následně se aktualizuje ADR tímto novým pozitivním detekčním obdélníkem (je možné tento detekční obdélník i smazat nechceme-li, aby se s ním dále mohly spárovat jiné detekční obdélníky).

Není-li v následujícím detekčním snímku žádný detekční obdélník nebo se nepodaří nalézt pozitivní detekční obdélník, pak se FC ani ADR neaktualizuje. Navýší se však PDC o daný SC. Tento bod se opakuje tak dlouho dokud FC nedosáhne počtu FB, nebo dokud se nedojde na konec bufferu.

Ad 4. Je-li FC rovno FB pak se eviduje konkrétní detekční obdélník z AF do PDRB (Positive Detection Rectangle Buffer). Není-li FC rovno FB pak se nic neeviduje.

V obou případech se z AF vezme další detekční obdélník, zaznamenaná se do ADR a následně se přejde opět k operaci popisované v bodě **Ad 3**. Toto se vykonává tak dlouho, dokud jsou v AF detekční obdélníky. Bylo-li provedeno poslední zhodnocení detekčního obdélníku z AF, pak se navrátí PDRB, které obsahuje vyfiltrované pozitivní detekční obdélníky.

Příklad bude prezentovat princip algoritmu na základě obrázku B.1 se vstupními parametry $BS = 4$, $FB = 3$, $SC = 0.25$, $SDC = 3$, $PDC = 3$. Na obrázku B.1 zkratka DR označuje detekční obdélník. Postup jednotlivých kroků algoritmu:

- DR1 z AF se přiřadí do ADR
- ADR se porovná s DR1 ve Frame 2, kde Compare Rectangles vrátí pozitivní odpověď. Na základě toho se DR1 z Frame 2 přiřadí do ADR a FC se navýší o jedničku.
- ADR se porovná s DR1 ve Frame 3, kde opět Compare rectangles vrátí pozitivní odpověď. Opět se DR1 z Frame 3 přiřadí do ADR a FC se navýší o jedničku.
- ADR se porovná s DR1 ve Frame 4, kde Compare rectangles vrátí negativní odpověď. Tím, že se právě algoritmus nachází v posledním detekčním snímku, nemusí se navyšovat PDC o SC a ani se nezaeviduje do PDRB. FC se vynuluje.
- ADR se přepíše DR2 z Frame 1.
- ADR se porovná s DR1 z Frame 2, kde Compare Rectangles vrátí negativní odpověď.

- ADR se porovná s DR2 z Frame 2, kde Compare Rectangles vrátí pozitivní odpověď. Na základě toho se DR2 z Frame 2 přiřadí do ADR a FC se navýší o jedničku.
- ADR se porovná s DR1 z Frame 3, kde Compare Rectangles vrátí negativní odpověď.
- ADR se porovná s DR2 z Frame 3, kde Compare Rectangles vrátí pozitivní odpověď. Na základě toho se DR2 z Frame 3 přiřadí do ADR a FC se navýší o jedničku.
- ADR se porovná s DR1 z Frame 4, kde Compare Rectangles vrátí pozitivní odpověď. Nyní FC nabývá hodnoty 3, která je shodná s FB, tudíž se DR1 z Frame 1 zaeviduje do PDRB.
- Není již co porovnávat, předá se hotový PDRB obsahující pouze DR2 z AF.

3.5 Detekční obdélník zájmů (Key rectangle)

Compare Rectangles popisován v kapitole 3.1 je používán i v tomto algoritmu podobně jako v algoritmu Filtering Incorrect Detection z kapitoly 3.4. Algoritmus je vhodné použít při získání zobrazení jen jedné detekce objektu z několika, který se z velké pravděpodobnosti vyskytuje nejdelší dobu před objektivem.

Algoritmus obsahuje buffer o určité velikosti, který uchovává jednotlivé snímky s detekčními obdélníky. Tento snímek se nazývá frame.

Před použitím tohoto algoritmu je vhodné použít algoritmus průměrování podobných detekčních obdélníků (Averaging appropriate rectangles nebo Bubble averaging appropriate rectangles) pro jednotlivé snímky, popisované v kapitole 3.3. Vhodné je použít i Filtering incorrect rectangles popisované v kapitole 3.4. K níže uváděným parametrům přibývají i parametry algoritmu Compare Rectangles.

- **Find in buffer [FB]:** Minimální počet detekčních snímků, ve kterých musí být nalezené podobné detekční obdélníky. Maximální hodnota:

$$BS - 1 \geq FB \quad (3.17)$$

- **Step coefficient [SC]:** O kolik se má v každém následujícím detekčním snímku navýšit PDC, pokud nebyla nalezená detekce.
- **Maximum step coefficient [MSC]:** Nepovinný parametr sloužící k určení maximálního navyšování PDC koeficientem SC.
- **Number of steps to predict [NSP]:** Počet detekčních snímků, které mohou být na začátku bufferu prázdné či odlišné, respektive kolik detekčních snímků se má sloučit do jednoho.

$$NSP + 1 \leq FB \quad (3.18)$$

Porovnávání probíhá na principu vlnění hada, to znamená, že nesmí být jednotlivé detekční obdélníky v následujícím detekčním snímku příliš odlišné, na příliš odlišném místě. Osnova provádění algoritmů:

1. Zjištění, zda buffer obsahuje minimální počet detekčních snímků.
2. Vytvoření kandidátů (detekčních obdélníků) na základě sloučení prvních NSP detekčních snímků.
3. Hledání v bufferu nejrelevantnějších detekčních obdélníků.
4. Porovnání nálezů a zvolení nejvhodnějšího detekčního obdélníku.

Ad 1. Není-li buffer větší než hodnota FB, pak se běh algoritmu zastaví a navrátí prázdnou hodnotu.

Ad 2. Vytvoří se fiktivní frame nazývaný jako **CDR** (Candidates Detection Rectangles), který bude obsahovat kandidáty detekčních obdélníků dle následujících detekčních snímků v počtu NSP. V prvním kroku se zkontroluje, zda aktuální detekční snímek obsahuje nějaké detekční obdélníky. Pokud ne, vezme se další detekční snímek, dokud se nenarazí na neprázdný nebo dokud nebyl překročen počet NSP.

Byl-li překročen počet NSP a všechny detekční snímky v těchto skocích byly prázdné, pak se algoritmus ukončí a předá prázdný snímek. Byl-li nalezený neprázdný frame zaevidují se všechny jeho detekční obdélníky do CDR. Je-li neprázdný CDR a nepřekročený počet skoku NSP, pak se porovnávají ostatní detekční snímky, dokud nebude překročen NSP a hledají se tak další kandidáti do CDR a to následovně:

1. Z CDR se vytvoří kopie nazývaná jako **ACDR** (Actual Candidates Detection Rectangles).
2. V následujícím detekčním snímku se porovnávají všechny detekční obdélníky, které obsahuje s všemi ACDR za pomoci Compare Rectangles. Ty, které měly negativní odpověď, uloží se do CDR a ACDR, v opačném případě se nahradí v ACDR.
3. Zkontroluje se, zda nebyl překročen počet skoků NSP. Pokud ano, pokračuje se v další části algoritmu s předaným CDR. Pokud ne, načte se další detekční snímek a přejde se k bodu 2.

Ad 3. Hledání v bufferu nejrelevantnějších detekčních obdélníků se provádí v následujících bodech:

1. Z CDR se vytáhnou postupně všechny detekční obdélníky, které se porovnávají s ostatními detekčními obdélníky ostatních detekčních snímků.
2. Aktuální detekční obdélník se značí **ADR** (Actual Detection Rectangle). ADR je proměnlivý (hadovitý) - nenabývá neustále detekčních obdélníků z CDR, obsahuje je jen ze začátků.
3. Získaný ADR se porovná s následujícími detekčními obdélníky pomocí Compare Rectangles.

4. Je-li následující detekční snímek prázdný nebo se nepodaří vyhledat žádný pozitivní nález, pak se FC ani ADR neaktualizuje. Navýší se však PDC o daný SC a **NM** (No Match) o jedničku, což je počet detekčních snímků, ve kterých nebyl žádný nález.
5. Při první pozitivní detekci, za pomoci Compare rectangles se ukončí další porovnávání a **FC** (Find Counter) se navýší o jedničku. Následně se aktualizuje ADR tímto novým detekčním obdélníkem a zaeviduje se růstový koeficient **CP** (Candidate Price) daného detekčního obdélníku z CDR. Hodnota CP se počítá následovně:

$$CP = (CP + FC + SB) - (AP + NM) \quad (3.19)$$

Přičemž **AP** (Actual Position) je aktuální pozice v bufferu a **SB** (Size Buffer) je velikost bufferu. Následně se NM vynuluje.

6. V obou případech se celý proces od bodu 3 opakuje tak dlouho, dokud se nenarazí na konec bufferu. Narazí-li se na konec bufferu, pak se od CP odečte NM. CP se zaeviduje k danému detekčnímu obdélníku v CDR a načte se další detekční obdélník z CDR do ADR, vynuluje se FC, NM, AP, CP a přejde se již k zmiňovanému třetímu bodu.
 7. Jakmile je zkontrolovaný poslední detekční obdélník v CDR, přejde se do další části algoritmu.
- Ad 4.** Na základě hodnot CP se porovnají jednotlivé detekční obdélníky z CDR. Jestliže nejvyšší hodnota má minimálně hodnotu FB, předá se jako obdélník zájmu. V opačném případě je předaná prázdná hodnota.

Vyhledávání detekčního obdélníku zájmu je prezentován na příkladu. Tento příklad je pro jeho délku prezentován v příloze B.

4 Popis vlastního API

Vlastní API obsahuje několik modulů tříd a modulů, které obsahují různé algoritmy nebo ulehčení. Vše je implementováno v Pythonu. Všechny třídy obsahují českou dokumentaci (v komentářích). U modulu `ardrone_control.lib` jsou komentáře i v angličtině. Popis jednotlivých metod daných tříd se nachází v příloze D.

4.1 API pro ovládání kvadrikopty

Pro ovládání kvadrikopty slouží modul `ardrone_control.lib` obsahující třídu `ARDrone2`, statické třídy `Config`, `Led` a `Anim` a funkci `f2i`. Zdrojové kódy jsou umístěny na CD viz C.2.

Funkce `f2i` slouží k převodu čísel s desetinou čárkou/tečkou na integer ve standardu IEEE-754. Statické třídy slouží pro jednodušší použití třídy `ARDrone2`:

- **Config** Seznam konfiguračních příznaků.
- **Led** Seznam Led animací.
- **Anim** Seznam letových akrobacií.

Třída `ARDrone2` byla inspirována Pythonovou knihovnou od Bastian Venthur. Třída `ARDrone2` slouží ke komunikaci a ovládání kvadrikopty `ARDrone 2.0` a přijímání video streamu. Obsahuje vysokoúrovňové i nízkoúrovňové ovládání pohybu drona. Popis jednotlivých metod třídy se vyskytuje v příloze D.1.1. Na základě síťové komunikace využívá třída v pozadí 3 vlákna (+ hlavní vlákno):

- Příjem navigačních dat.
- Zachytávání video streamu.
- Watchdog odesílající pravidelně co 0.2 s `AT*COMWDG`, který nastavuje sekvenční číslo na kvadrikoptěře.

Sada abstraktních metod slouží pro modifikování vlastní zaevidování již získaných údajů `navdata` a video streamu. U abstraktních metod `navdataCycle` a `videoBufferCycle` je potřeba, aby jejich vykonávání netrvalo příliš dlouho, brzdila by se tím komunikace s kvadrikoptérou. Je vhodné případné dlouhotrvající zpracování provádět například v jiném vlákne či procesu. Ukázka použití modulu:

```

1 drone = ARDrone2()
2 drone.takeoff()
3 drone.atLed( Led.SNAKE_GREEN_RED, 1.1, 2 )
4 time.sleep(2)
5 drone.atAnim( Anim.YAW_DANCE, 2)
6 time.sleep(2)
7 drone.land()

```

Výpis 4: Ukázka použití modulu `ardrone_control.lib`: vzletnutí, použití animací diod, zatančení a následné přistání

4.2 API pro jednodušší práci s obrázky a detekcemi

Pro jednodušší práci s obrázky a detekcemi slouží modul `detection_worker` s třídy `PicturesConvert`, `RectanglesWorker` a `FramesWorker`. Zdrojové kódy jsou umístěny na CD viz C.3.

Třída **PicturesConvert** slouží pro jednodušší práci s obrázky. Obsahuje také různé detekce z obrázků (například detekce obličejů, lidí apod.). Podporuje například převod obrázku do odstínu šedi, gausovou filtraci či Cannyho hranový detektor. Popis jednotlivých metod třídy se vyskytuje v příloze D.2.1. Ukázka použití třídy `PicturesConvert`:

```

1 pc = PicturesConvert( image )
2 pc.getGauss()
3 r = pc.getPeopleDetection()
4 pc.reset()
5 image = pc.drawRectangles( r )
6 cv2.imshow( "detections", image )

```

Výpis 5: Ukázka použití třídy: `PicturesConvert` - nalezení detekcí lidí a jejich následné zaznamenání v originálním obrázků

Třída **RectanglesWorker** slouží pro práci se seznamem detekčních obdélníků. Obsahuje algoritmy jako je například zprůměrování podobných obdélníků nebo porovnání dvou obdélníků (popisováno v kapitolách 3.3 a 3.1). Popis jednotlivých metod třídy se vyskytuje v příloze D.2.2. Ukázka použití třídy `RectanglesWorker`:

```

1 pc = PicturesConvert( image )
2 pc.getGauss()
3 r = pc.getPeopleDetection()
4
5 rw = RectanglesWorker( r )
6 rw.correctionRectangles()
7 params = {
8     "size_divergence" : 0.3,
9     "position_divergence" : 0.3,
10    "buble_mode" : True
11 }
12 r = rw.averagingAppropriateRectangles( **params )
13
14 pc.reset()
15 image = pc.drawRectangles( r )
16 cv2.imshow( "detections", image )

```

Výpis 6: Ukázka použití třídy: `RectanglesWorker`

Třída **FramesWorker** slouží pro práci se seznamem detekčních snímků obsahující detekční obdélníky. Obsahuje algoritmy jako jsou zprůměrování podobných obdélníků na podobném místě (pro každý detekční snímek zvlášť), odfiltrování nenavazujících detekcí, získání detekčního obdélníků zájmů, sloučení detekčních snímků, apod (popisováno v kapitolách 3.3, 3.4 a 3.5). Popis jednotlivých metod třídy se vyskytuje v příloze D.2.3.

4.3 API pro socketovou komunikaci

Pro socketovou komunikaci lze použít modul `socket_server_client` obsahující třídy `Message`, `Handler`, `SocketServer` a `Client`. Zdrojové kódy jsou umístěny na CD viz C.4.

Zprávy se posílají a přijímají ve formátu uvedený ve výpisu 7. První příklad (viz výpis 8 představuje odeslání zprávy s parametry a druhý příklad (viz výpis 9) představuje odeslání zprávy bez parametrů.

```
<start>[název příkazu][parametry oddělené <param>]<end>
```

Výpis 7: Syntaxe použitá v modulu `socket server`

```
<start>setTime<param>23<param>00<param>00<end>
```

Výpis 8: Ukázka syntaxe s parametry

```
<start>startStopwatch<end>
```

Výpis 9: Ukázka syntaxe bez parametrů

Modul předpokládá vstup pouze ASCII znaků. Není vhodné používat znaky `<` a `>`, mohlo by se stát, že by sestavovaná zpráva mohla mít jeden z těchto řetězců: `<start>`, `<param>` a `<end>`, které by mohly zapříčinit degradaci zprávy. Jiné znaky než čísla, je vhodné převést do datového formátu Base64. Za pomoci Base64 je možný přenos jakýchkoliv dat.

Třída **Message** slouží k sestavování a rozparsování zpráv s následným spuštěním dané funkce či metody. Neslouží však k příjmu ani odesílání zpráv. Je jen neviditelným základním kamenem tříd `Handler` a `Client`. Popis jednotlivých metod třídy se vyskytuje v příloze D.3.1.

Třída **Handler** slouží k udržování spojení s daným připojeným klientem či serverem s následnou případnou obsluhou. Popis jednotlivých metod třídy se vyskytuje v příloze D.3.2. Příklad použití třídy `Handler`:

```
1 class MyHandler( Handler ):
2     # rewrite
3     def runFunction( self , params ):
4         f = params[0]
5         p = []
6         if len( params ) > 1:
7             p = params[ 1 : ]
8         if f == "detectPeoples" and len(p) == 3:
9             r = self.detectionPeoples( *p )
10            # poslaní zprávy klientovi s návratovými daty provedené funkce
11            self.sendMessage( "returnDetectPeoples", r )
12        # add
13        def detectionPeoples(self, width, height, data):
14            pass # nějaká definice
```

Výpis 10: Ukázka použití třídy `Handler`

Třída **SocketServer** slouží k jednoduchému vytvoření základního serveru. Popis jednotlivých metod třídy se vyskytuje v příloze D.3.3. Ukázka použití třídy `SocketServer`:

```

1 class MyServer( SocketServer ):
2     # rewrite
3     def clientHandler( self, client ):
4         client_handler = MyHandler( client )
5
6 if __name__ == "__main__":
7     ms = MyServer( 12345 )
8     ms.run()

```

Výpis 11: Ukázka použití třídy SocketServer

Třída **Client** slouží k navázání a udržování spojení s daným serverem s následnou případnou obsluhou. Popis jednotlivých metod třídy se vyskytuje v příloze D.3.4. Ukázka použití třídy Client:

```

1 class MyClient( Client ):
2     # add
3     def returnDetectionPeoples( self, width, height, data ):
4         if int( width ) == 0:
5             return []
6         # rekonstrukce numpy
7         np_data = np.fromstring( base64.b64decode( data ), dtype='int' )
8         # rekonstrukce správného rozlišení
9         np_data = np_data.reshape( ( int( width ), int( height ) ) )
10        return np_data
11
12    # add
13    def detectionPeoples( self, image, compress = False ):
14        function = "detectPeoples"
15        string = image.tostring()
16        base = base64.b64encode( string )
17        param = [ str( image.shape[0] ), str( image.shape[1] ), base ]
18        r = self .sendMessageWaiting( function, param, "returnDetectPeoples" )
19        return self .returnDetectionPeoples( *r[ 1 : ] )

```

Výpis 12: Ukázka použití třídy Client

4.4 Kruhový buffer

Třída **HybridRingBuffer** je jednoduchý hybridní kruhový buffer. Není to čistý kruhový buffer, lze získat prvky na jakékoliv pozici. Má jednotnou maximální velikost. Při přetečení se přepisují starší hodnoty. Popis jednotlivých metod třídy se vyskytuje v příloze D.4. Zdrojový kód je umístěn na CD viz C.5.

5 Implementace

Pro implementaci ukázky pronásledování osob byly použity výše popisované třídy v kapitole 4. Celková implementace je rozdělená na klientskou (prostředník) a serverovou část (viz obrázek 5.1). Obě aplikace se spouštějí přes terminál.

5.1 Klientská část

Klientská část se připojuje na kvadrikoptéru za pomoci Wi-Fi a do ethernetové sítě k serverové části a tvoří tak prostředníka. V klientské části se provádí veškeré operace a logika ovládání drona na základě získaných obrázků (video streamu) a detekcí v nich obsažené.

Byl použit notebook DELL Latitude E6410 s CPU Intel® Core™ i7 CPU M 620 2.67GHz a DDR3 RAM o velikosti 4 GB (3,9 GiB) na operačním systému GNU Linux Ubuntu 12.04 (32 bit) s jádrem Linux 3.8.0-38-generic. Odeslání obrázků ve stupni šedi na server a následnými přijatými detekcemi trvá přibližně 0.2 s.

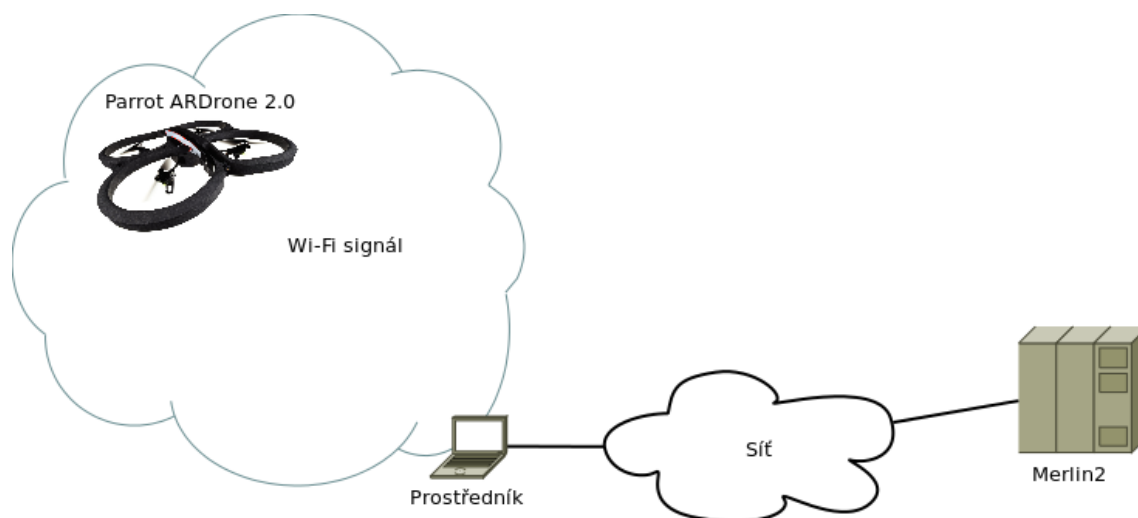
Pro vytvoření klientského programu se využívají výše zmiňované moduly `ardrone_control_lib`, `detection_worker`, `socket_server_client` a třídu `HybridRingBuffer`. Nad modulem `ardrone_control_lib` je vytvořena další vrstva `drone_fly`, která slouží k ovládání drona za pomoci klávesnice a přepínání mezi manuálním ovládáním a autonomním chováním. Pro uskutečnění ovládání přes klávesnici jsou vytvořeny třídy v adresáři `./core/key_mapper`. Využívá se v nich Tkinter, terminál, OpenCV, případně lze doimplementovat i další možnosti pro získání detekce stisknutých kláves, například Pygame. Zdrojové kódy jsou umístěny na CD viz C.7.

Je-li spuštěné manuální ovládání drona, zobrazuje se pouze jedno okno s aktuálním video streamem. Spustí-li se autonomní chování, otevře se druhé okno s nalezenými detekcemi, které jsou označené červeně (viz obrázek 5.2). Jakmile je rozpoznána detekce zájmu je vyobrazená ještě zeleným obdélníkem (viz obrázky 5.3 a 5.4). V případě využití Tkinteru k zachytávání kláves je otevřené další (šedé) okno (viz obrázek 5.4).

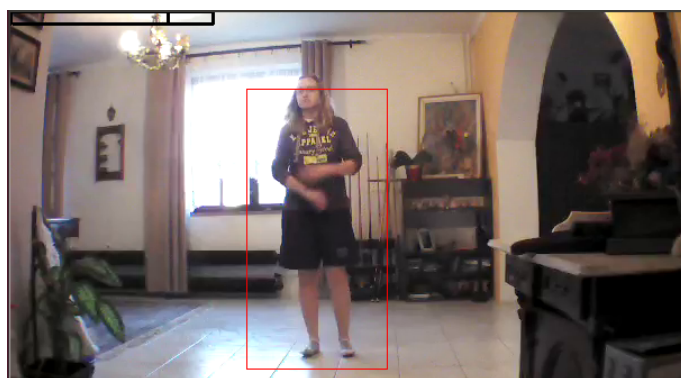
5.2 Serverová část

Serverová část slouží pro rychlé detekce osob z příchozího černobílého obrázku a následné odeslání do klientské části přes síť. Byl použit školní server Merlin2. Vygenerování detekcí z obrázků trvá přibližně 0.03 s - provádí se dvojnásobná detekce.

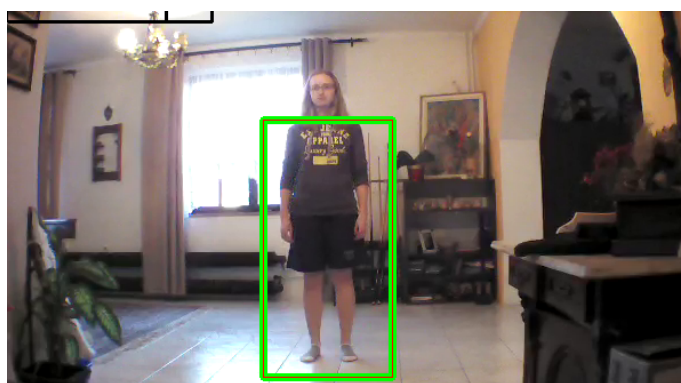
Pro vytvoření programu se využívají výše zmiňované moduly `detection_worker` a `socket_server_client`. Zdrojové kódy jsou umístěny na CD viz C.6.



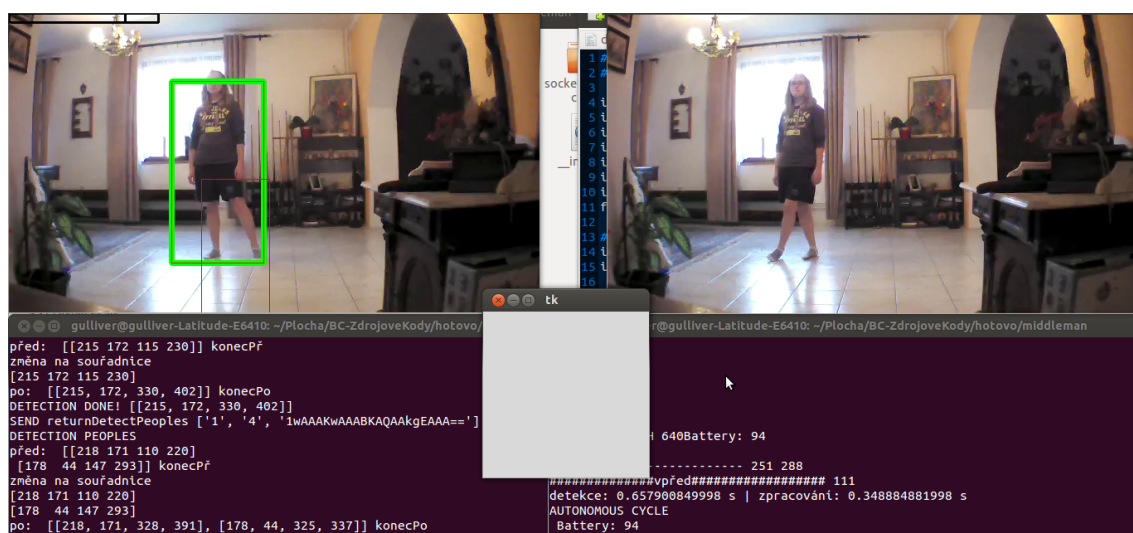
Obrázek 5.1: Grafické vyobrazení komunikace mezi dronem, notebookem a serverem Merlin2



Obrázek 5.2: Okno s detekcemi



Obrázek 5.3: Okno s detekcí zájmu



Obrázek 5.4: vyobrazení obou oken (aktuální stream, aktuální detekce) a Tkinter okna pro zachytávání kláves

Závěr

Cílem této bakalářské práce bylo popsat a seznámit se s API kvadrikopty, nastudovat metody HOG a SVM pro detekci objektů zájmů a pomocí těchto metod detekovat ze získaného obrazu objekty. Následně ovládat kvadrikopty tak, aby sledovala zadaný cíl a dodržovala od něj zadaný odstup. Zvolil jsem kvadrikopty Parrot AR.Drone 2.0. Popsal jsem její technické specifikace a jak se dá ovládat za pomoci AT příkazu přes Wi-Fi.

Pro celkovou implementaci jsem si vybral objektově orientovaný programovací jazyk Python. Pro Python není oficiální knihovna od výrobce Parrot. Nalezl jsem knihovnu od Bastiana Venthury. Avšak v této knihovně mi nefungovalo zachytávání video streamu z kvadrikopty. Knihovnu jsem několikrát upravoval pro své potřeby, až jsem dospěl k napsání své vlastní, názvem `ardrone_control.lib`. Ta obsahuje některé části knihovny původní a pro zachycení video streamu byla použita knihovna OpenCV. Vlastní knihovnu jsem také v této práci popsal.

Ze zachycených video snímků jsem získal detekce za pomoci HOG deskriptoru a SVM klasifikátoru, již implementovaných v OpenCV knihovně. Pro jejich nerealtimové detekce na mém notebooku byl použit školní server Merlin2. Detekce tak byly rychlejší, ale jejich získání do klientského notebooku přes síť trvalo stále v nerealtimovém čase, respektive dlouho trvalo odeslání obrázků na server.

Další problém, se kterým jsem se tu setkal, byla špatná detekce lidí. Dostatečně zřetelná osoba není detekována a získává se také mnoho falešných detekcí. Díky těmto problémům jsem se snažil navrhnout vlastní algoritmy pro získání relevantnějších detekcí. Také jsem navrhl algoritmus k získání detekce zájmu. Pod detekcí zájmu se rozumí takové detekce, které se objevují dostatečně dlouho na video sekvenci a z nichž se vybere ta co je tam nejdéle. Tyto algoritmy jsem následně naprogramoval. Započal jsem i vlastní vytváření HOG feature pro lepší pochopení této problematiky a případné budoucí lepší detekce za pomoci pozitivních a negativních obrázků.

Za použití vlastních algoritmů a knihovny jsem se snažil vytvořit program pro jednoduché autonomní chování. Ten však nefunguje zrovna ideálně. Bylo by potřeba do něj ještě zakomponovat práci s údaji z gyroskopu pro lepší otáčení kvadrikopty. Detekuje-li se objekt zájmu příliš vpravo, natočí se doprava a naopak. Je-li objekt zájmu příliš blízko - a je přitom stále detekován - vzdálí se, v opačném případě se přiblíží. Zdrojové kódy a algoritmy mohou být dále rozšířeny v navazující práci dalšího studia.

Radek Simkanič

6 Literatura

- [1] Kompatibilita. PARROT. *Parrot Ardrone 2 Cz* [online]. ©2012 [cit. 2014-04-17]. Dostupné z: <http://ardrone2.parrot-ardrone.cz/podpora/kompatibilita>
- [2] Technická specifikace. PARROT. *Parrot Ardrone 2 Cz* [online]. ©2012 [cit. 2014-04-17]. Dostupné z: <http://ardrone2.parrot-ardrone.cz/specifications/>
- [3] Technical specifications. PARROT. *AR.Drone 2.0 Parrot new wi-fi quadricopter* [online]. ©2012 [cit. 2014-04-17]. Dostupné z: <http://ardrone2.parrot.com/ardrone-2/specifications/>
- [4] HRMA, Jiří. Recenze AR.Drone 2.0: Kvadroptéra prošpikovaná technikou. In: *SmartMania.cz* [online]. 16. 7. 2012 [cit. 2014-04-17]. Dostupné z: <http://smartmania.cz/recenze/recenze-ardrone-20-kvadroptera-prospikovana-technikou-2908>
- [5] PISKORSKI, BRULEZ, ELINE a D'HAEYER. PARROT. *AR.Drone Developer Guide* [online]. SDK 2.0. December 11, 2012, 124 s. [cit. 17.4.2014]. Dostupné z: <https://projects.ardrone.org/>
- [6] SANDLER, Anna. How to Control AR.Drone LEDs. In: *Robot App Store* [online]. © 2010-2014 [cit. 2014-04-20]. Dostupné z: <http://www.robotappstore.com/Knowledge-Base/How-to-Control-ARDrone-LEDs/100.html>
- [7] How to run AT*ANIM command?. In: *ARDrone API* [online]. 10-05-2011 01:57 [cit. 2014-04-20]. Dostupné z: <https://projects.ardrone.org/boards/1/topics/show/3009>
- [8] OpenCV – Počítačové vidění pro každého. In: *Pandatron.cz: Elektrotechnický magazín* [online]. Pandatron, 16. března 2011 - 9:27 [cit. 2014-04-19]. Dostupné z: http://pandatron.cz/?2612&opencv_%96_pocitacove_videni_pro_kazdeho
- [9] BRADSKI, Gary, Adrian KAEHLER, Mike LOUKIDES, Robert ROMANO. *Learning OpenCV*. Gravenstein Highway North, Sebastopol: O'Reilly, 2008. ISBN 978-0-596-51613-0.
- [10] *OpenCV* [online]. © 2014 [cit. 2014-04-19]. Dostupné z: <http://opencv.org/>
- [11] BURGESS, Christopher J.C. *A Tutorial on Support Vector Machines for Pattern Recognition*. Data Mining and Knowledge Discovery 2, 1998. 121-167. Dostupné z: <http://research.microsoft.com/pubs/67119/svmtutorial.pdf>
- [12] DALAL, Navneet a Bill TRIGGS. *Histograms of Oriented Gradients for Human Detection*. Conference on Computer Vision and Pattern Recognition, 2005. DOI : 10.1109/CVPR.2005.177:886-893. Dostupné z: http://hal.archives-ouvertes.fr/docs/00/54/85/12/PDF/hog_cvpr2005.pdf

A Příklad: průměrování podobných detekčních obdélníků

Princip algoritmu bude prezentován na základě tabulky viz A.1 a vstupních parametrů:

- $SDC = 0.25$
- $PDC = 0.3$

Počáteční hodnoty:

- $AL = []$
- $UR = []$
- $LAR = []$

Postup:

- Buffer obsahuje 5 detekčních obdélníků. Pokračuje se.
- Načtení DR1 pro zaznamenání do AR. DR1 se zaznamená do AR, AL a UR. Aktuální hodnoty:
 $AR = DR1$
 $AL = [DR1]$
 $UR = [UR]$
- Načtení DR2 pro porovnání s AR (DR1). DR2 se nenachází v UR. Compare rectangles vrátí kladnou hodnotu při porovnání DR2 a AR. Zaznamená se do AL a UR. Aktuální hodnoty:
 $AL = [DR1, DR2]$
 $UR = [DR1, DR2]$
- Načtení DR3 pro porovnání s AR (DR1). DR3 se nenachází v UR. Compare rectangles vrátí zápornou hodnotu při porovnání DR3 a AR. Načte se další detekční obdélník.
- Načtení DR4 pro porovnání s AR (DR1). DR4 se nenachází v UR. Compare rectangles vrátí zápornou hodnotu při porovnání DR4 a AR. Načte se další detekční obdélník.
- Načtení DR5 pro porovnání s AR (DR1). DR5 se nenachází v UR. Compare rectangles vrátí zápornou hodnotu při porovnání DR5 a AR. Načte se další detekční obdélník.
- Zprůměrují se detekční obdélníky v AL za pomocí Averaging rectangles. AR obsahuje DR1 a DR2. Zprůměrovaný obdélník se zaznamená do LAR. AL se vyprázdní. Aktuální hodnoty:
 $AL = []$
 $UR = [DR1, DR2]$
 $LAR = [[5, 5, 110, 210]]$
- Načtení DR2 pro zaznamenání do AR. DR2 je však obsažen v UR, nemůže se použít.

Obdélník	Šířka	Výška	x_1	y_1	x_2	y_2
DR1	100	200	0	0	100	200
DR2	110	210	10	10	120	220
DR3	120	210	20	20	140	230
DR4	100	180	100	50	200	230
DR5	118	208	22	22	140	230

Tabulka A.1: Seznam detekčních obdélníků.

- Načtení DR3 pro zaznamenání do AR. DR3 se zaznamená do AR, AL a UR. Aktuální hodnoty:
 $AR = DR3$
 $AL = [DR3]$
 $UR = [DR1, DR2, DR3]$
- Načtení DR4 pro porovnání s AR (DR3). DR4 se nenachází v UR. Compare rectangles vrátí zápornou hodnotu při porovnání DR4 a AR. Načte se další detekční obdélník.
- Načtení DR5 pro porovnání s AR (DR3). DR5 se nenachází v UR. Compare rectangles vrátí kladnou hodnotu při porovnání DR5 a AR. Zaznamená se do AL a UR. Aktuální hodnoty:
 $AL = [DR3, DR5]$
 $UR = [DR1, DR2, DR3, DR5]$
- Zprůměrují se detekční obdélníky v AL za pomoci Averaging rectangles. AR obsahuje DR3 a DR5. Zprůměrovaný obdélník se zaznamená do LAR. AL se vyprázdní. Aktuální hodnoty:
 $AL = []$
 $UR = [DR1, DR2, DR3, DR5]$
 $LAR = [[5, 5, 110, 210], [21, 21, 140, 230]]$
- Načtení DR4 pro zaznamenání do AR. DR4 se zaznamená do AR, AL a UR. Aktuální hodnoty:
 $AR = DR4$
 $AL = [DR4]$
 $UR = [DR1, DR2, DR3, DR5, DR4]$
- Načtení DR5 pro porovnání s AR (DR4). DR4 se nachází v UR. Načte se další detekční obdélník.
- V dalším kroku je konec bufferu, DR4 se zaznamená do LAR. LAR se předá v následující podobě: $LAR = [[5, 5, 110, 210], [21, 21, 140, 230], [100, 50, 200, 230]]$

Pro bublinkový způsob se přijatý LAR použije jako vstupní seznam detekčních obdélníků. V tomto případě by druhý průchod vypadal následovně:

- Buffer obsahuje 3 detekční obdélníky. Pokračuje se.

- Načtení DR1 pro zaznamenání do AR. DR1 se zaznamená do AR, AL a UR. Aktuální hodnoty:
 $AR = DR1$
 $AL = [DR1]$
 $UR = [UR]$
- Načtení DR2 pro porovnání s AR (DR1). DR2 se nenachází v UR. Compare rectangles vrátí kladnou hodnotu při porovnání DR2 a AR. Zaznamená se do AL a UR. Aktuální hodnoty:
 $AL = [DR1, DR2]$
 $UR = [DR1, DR2]$
- Načtení DR3 pro porovnání s AR (DR1). DR3 se nenachází v UR. Compare rectangles vrátí zápornou hodnotu při porovnání DR3 a AR. Načte se další detekční obdélník.
- Zprůměrují se detekční obdélníky v AL za pomoci Averaging rectangles. AR obsahuje DR1 a DR2. Zprůměrovaný obdélník se zaznamená do LAR. AL se vyprázdní. Aktuální hodnoty:
 $AL = []$
 $UR = [DR1, DR2]$
 $LAR = [[13, 13, 125, 220]]$
- Načtení DR2 pro zaznamenání do AR. DR2 je však obsažen v UR, nemůže se použít.
- Načtení DR3 pro zaznamenání do AR. DR3 se zaznamená do AR, AL a UR. Aktuální hodnoty:
 $AR = DR3$
 $AL = [DR3]$
 $UR = [DR1, DR2, DR3]$
- V dalším kroku je konec bufferu, DR3 se zaznamená do LAR. LAR se předá v následující podobě: $LAR = [[13, 13, 125, 220], [100, 50, 200, 230]]$

V třetím průchodu již hodnoty budou stejné, tudíž u bublinkového principu bude konečná hodnota: $[[13, 13, 125, 220], [100, 50, 200, 230]]$

B Příklad: detekční obdélník zájmů (Key rectangle)

Ukázkový příklad je založen na obrázku B.1 a vstupních parametrech:

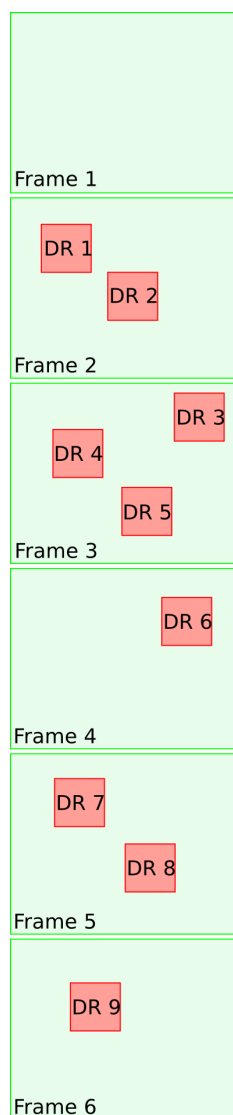
- $FB = 4$
- $SC = 0.25$
- $PDC = 3$
- $NSP = 3$

Další hodnoty:

- $SB = 6$
- $CDR = []$
- $ACDR = []$
- $CP = []$

Na obrázku zkratka DR označuje detekční obdélník. Operace nad PDC s SC budou vynechány. Postup:

- Buffer (SB) je větší než hodnota FB.
- První detekční snímek neobsahuje žádný detekční obdélník. Přejde se k dalšímu detekčnímu obdélníku
- Druhý detekční snímek dva detekční obdélníky. Zaevidují se do CDR i ACDR.
 $CDR = [DR1, DR2]$
 $ACDR = [DR1, DR2]$
 Přejde se k dalšímu snímku.
- Frame 3 obsahuje tři detekční obdélníky. DR4 (podobný s DR1) a DR5 (podobný s DR2) mají pozitivní odpověď z algoritmu Compare rectangles a DR3 má negativní odpověď. DR3 se zaeviduje do CDR a ACDR, DR1 se nahradí za DR4 a DR2 za DR5 v ACDR.
 $CDR = [DR1, DR2, DR3]$
 $ACDR = [DR4, DR5, DR3]$
 K dalšímu detekčnímu snímku se nepokračuje, protože je již dosažen NSP (průchod třemi detekčními snímky).
- Načte se první detekční obdélník z CDR do ADR (DR1).
- Načte se Frame 1. Ten neobsahuje žádné detekční obdélníky. Zaznamenává se:
 $NM = NM + 1 = 1$
 $AP = AP + 1 = 2$



Obrázek B.1: Grafické vyobrazení bufferu s detekčními snímky.

- Načte se Frame 2. Obsahuje dva detekční obdélníky. DR1 je shodný s ADR (DR1).
Zaznamenává se:

$$FC_1 = FC_1 + 1 = 1$$

$$ADR = DR_1$$

$$CP_1 = CP_1 + (FC + SB) - (AP + NM) = 0 + (1 + 6) - (2 + 1) = 4$$

$$NM = 0$$

$$AP = AP + 1 = 3$$
- Načte se Frame 3, který obsahuje 3 detekční obdélníky. DR4 je shodný s ADR (DR1).
Zaznamenává se:

$$FC_1 = FC_1 + 1 = 2$$

$$ADR = DR_4$$

$$CP_1 = CP_1 + (FC + SB) - (AP + NM) = 4 + (2 + 6) - (3 + 0) = 9$$

$$NM = 0$$

$$AP = AP + 1 = 4$$
- Načte se Frame 4, který obsahuje jeden detekční obdélník. DR6 není shodný s ADR (DR4). Zaznamenává se:

$$NM = NM + 1 = 1$$

$$AP = AP + 1 = 5$$
- Načte se Frame 5, který obsahuje 2 detekční obdélníky. DR7 je shodný s ADR (DR4).
Zaznamenává se:

$$FC_1 = FC_1 + 1 = 3$$

$$ADR = DR_7$$

$$CP_1 = CP_1 + (FC + SB) - (AP + NM) = 9 + (3 + 6) - (5 + 1) = 12$$

$$NM = 0$$

$$AP = AP + 1 = 6$$
- Načte se Frame 6, který obsahuje jeden detekční obdélník. DR9 je shodný s ADR (DR7). Zaznamenává se:

$$FC_1 = FC_1 + 1 = 4$$

$$ADR = DR_9$$

$$CP_1 = CP_1 + (FC + SB) - (AP + NM) = 12 + (4 + 6) - (6 + 0) = 16$$

$$NM = 0$$

$$AP = AP + 1 = 7$$
- Již další detekční snímek v bufferu není, do ADR se načte DR2 z CDR a nastaví se výchozí hodnoty: $NM = 0$

$$AP = 1$$
- Načte se Frame 1. Ten neobsahuje žádné detekční obdélníky. Zaznamenává se:

$$NM = NM + 1 = 1$$

$$AP = AP + 1 = 2$$
- Načte se Frame 2. Obsahuje dva detekční obdélníky. DR2 je shodný s ADR (DR2).
Zaznamenává se:

$$FC_2 = FC_2 + 1 = 1$$

$$ADR = DR_2$$

$$CP_2 = CP_2 + (FC + SB) - (AP + NM) = 0 + (1 + 6) - (2 + 1) = 4$$

$$NM = 0$$

$$AP = AP + 1 = 3$$

- Načte se Frame 3, který obsahuje 3 detekční obdélníky. DR5 je shodný s ADR (DR2).

Zaznamenává se:

$$FC_2 = FC_2 + 1 = 2$$

$$ADR = DR_5$$

$$CP_2 = CP_2 + (FC + SB) - (AP + NM) = 4 + (2 + 6) - (3 + 0) = 9$$

$$NM = 0$$

$$AP = AP + 1 = 4$$

- Načte se Frame 4, který obsahuje jeden detekční obdélník. DR6 není shodný s ADR (DR5). Zaznamenává se:

$$NM = NM + 1 = 1$$

$$AP = AP + 1 = 5$$

- Načte se Frame 5, který obsahuje 2 detekční obdélníky. DR8 je shodný s ADR (DR5).

Zaznamenává se:

$$FC_2 = FC_2 + 1 = 3$$

$$ADR = DR_8$$

$$CP_2 = CP_2 + (FC + SB) - (AP + NM) = 9 + (3 + 6) - (5 + 1) = 12$$

$$NM = 0$$

$$AP = AP + 1 = 6$$

- Načte se Frame 6, který obsahuje jeden detekční obdélník. DR9 není shodný s ADR (DR8). Zaznamenává se:

$$NM = NM + 1 = 1$$

$$AP = AP + 1 = 7$$

- Již další detekční snímek v bufferu není, do ADR se načte DR3 z CDR a nastaví se výchozí hodnoty: $CP_2 = CP_2 - NM = 12 - 1 = 11$

$$NM = 0$$

$$AP = 1$$

- Načte se Frame 1. Ten neobsahuje žádné detekční obdélníky. Zaznamenává se:

$$NM = NM + 1 = 1$$

$$AP = AP + 1 = 2$$

- Načte se Frame 2. Obsahuje dva detekční obdélníky. Žádný není shodný s ADR (DR3). Zaznamenává se:

$$NM = NM + 1 = 2$$

$$AP = AP + 1 = 3$$

- Načte se Frame 3, který obsahuje 3 detekční obdélníky. DR3 je shodný s ADR (DR3). Zaznamenává se:
 $FC_3 = FC_3 + 1 = 1$
 $ADR = DR_3$
 $CP_3 = CP_3 + (FC + SB) - (AP + NM) = 0 + (1 + 6) - (3 + 2) = 2$
 $NM = 0$
 $AP = AP + 1 = 4$
- Načte se Frame 4, který obsahuje jeden detekční obdélník. DR6 je shodný s ADR (DR3). Zaznamenává se:
 $FC_3 = FC_3 + 1 = 2$
 $ADR = DR_6$
 $CP_3 = CP_3 + (FC + SB) - (AP + NM) = 2 + (2 + 6) - (4 + 0) = 6$
 $NM = 0$
 $AP = AP + 1 = 5$
- Načte se Frame 5, který obsahuje 2 detekční obdélníky. Žádný není shodný s ADR (DR6). Zaznamenává se:
 $NM = NM + 1 = 1$
 $AP = AP + 1 = 6$
- Načte se Frame 6, který obsahuje jeden detekční obdélník. DR9 není shodný s ADR (DR6). Zaznamenává se:
 $NM = NM + 1 = 2$
 $AP = AP + 1 = 7$
- Již další detekční snímek v bufferu není. V CDR také již není žádný detekční obdélník, nastaví se pouze: $CP_3 = CP_3 - NM = 6 - 2 = 4$
- Konečné hodnoty jsou:
 $CP = [16, 11, 4]$
 $FC = [4, 3, 2]$
 $CDR = [DR1, DR2, DR3]$
 Klíčovým detekčním obdélníkem je DR1, protože má nejvyšší CP (16) a splňuje podmínku $FC \geq FB$.

C Přiložené soubory

C.1 Bakalářská práce

Bakalářská práce ve formátu PDF se nachází v rootu CD s názvem `SIM0094.pdf`.

C.2 Modul `ardrone_control_lib`

Zdrojové kódy modulu `ardrone_control_lib` obsahující třídy pro komunikaci a ovládání kvadrikoptéry se nachází v adresáři `ardrone_control_lib`.

C.3 Modul `detection_worker`

Zdrojové kódy modulu `detection_worker` obsahující třídy pro vytváření a manipulaci s detekcemi se nachází v adresáři `detection_worker`.

C.4 Modul `socket_server_client`

Zdrojové kódy modulu `socket_server_client` obsahující třídy pro komunikaci přes sockety se nachází v adresáři `socket_server_client`.

C.5 Třída `HybridRingBuffer`

Třída kruhového bufferu se nachází v adresáři `class`.

C.6 Hotová serverová část

Zdrojové kódy hotového programu běžící na serveru Merlin se nachází v adresáři `merlin_server`.

C.7 Hotová klientská část

Zdrojové kódy hotového programu běžící na klientské části (prostředník) se nachází v adresáři `middleman`.

C.8 Vlastní vytváření HOG descriptorů

Zdrojové kódy pro vytváření HOG descriptorů se nachází v adresáři `features`.

D Popis tříd a modulů

D.1 Modul `ardrone_control_lib`

Modul obsahuje třídu `ARDrone2`, statickou třídu `Config` se seznamem konfiguračních příznaků, statickou třídu `Led` se seznamem LED animací, statickou třídu `Anim` se seznamem letových akrobacií, funkci `f2i` k převodu čísel s desetinou čárkou/tečkou na integer ve standardu IEEE-754.

Samozřejmě obsahuje i základní ukázkou použití třídy `ARDrone2`.

D.1.1 Třída `ARDrone2`

Pro ovládání slouží následující metody:

`navdataDemo` Inicializace drona.

`setSpeed` Nastavení rychlosti pro následující vysokoúrovňové příkazy. Má pouze jeden parametr přijímající float hodnoty od 0 po 1.

`trim` Flat trims - signal pro označení, že dron leží v rovině. Vynulování hodnot gyroskopu.

`takeoff` Vzlétnutí

`land` Přistání

`hover` Vznášení na místě.

`moveLeft` Nahnutí vlevo - let vlevo.

`moveRight` Nahnutí vpravo - let vpravo.

`moveBackward` Nahnutí vzad - let vzad.

`moveForward` Nahnutí vpřed - let vpřed.

`moveDown` Klesání

`moveUp` Stoupání

`turnLeft` Otočení vlevo.

`turnRight` Otočení vpravo.

`emergency` Vypnutí motorů.

`readyToStart` Vypnutí emergency módu a připravit opět k vzlétnutí.

`halt` Vypnutí drona.

Výše zmíněné metody využívají metody s nízkoúrovňovými příkazy:

sendAt Sestaví a odešle AT příkaz dronovi.

atRef Sestavení a odeslání příkazu AT*REF. Základní ovládání drona - vzletnutí, přistání, emergency (vypnutí motorů), zrušení emergency.

atPcmd Sestavení a odeslání příkazu AT*PCMD. Pohyb drona.

atPcmdMag Sestavení a odeslání příkazu AT*PCMD_MAG. Pohyb drona.

atFtrim Sestavení a odeslání příkazu AT*FTRIM. Flat trims - signal pro označení, že dron leží v rovině. Vynulování hodnot gyroskopu.

atCalib Sestavení a odeslání příkazu AT*CALIB. Kalibrace magnetometru.

atConfig Sestavení a odeslání příkazu AT*CONFIG. Nastavení konfigurace drona.

atConfigIds Sestavení a odeslání příkazu AT*CONFIG_IDS. Identifikátory pro následující příkaz AT*CONFIG.

atComwdg Sestavení a odeslání příkazu AT*COMWDG. Hlídací pes. Tento příkaz slouží k udržování komunikace.

atLed Spuštění LED diodové animace (za pomoci červených a zelených LED diod).

atAnim Spuštění pohybové animace, respektive provádění daného pohybu po určitou dobu.

Metody video streamu:

getActualFrame Předá aktuální snímek z video streamu drona.

getActualResolution Předá v tuple rozlišení aktuálního video streamu z drona.

Metody pro získání různých statusů/informací z drona:

statusBattery Procentuální stav baterie (0 - vybita, 100 - plně nabitá)

statusSetSpeed Aktuálně nastavená rychlost prováděných ovládacích metod.

statusTheta Aktuální úhel θ .

statusPhi Aktuální úhel ϕ .

statusPsi Aktuální úhel ψ .

statusAltitude Aktuální nadmořská výška.

statusVX Aktuální rychlost v ose X (viz obrázek 1.6).

statusVY Aktuální rychlost v ose Y (viz obrázek 1.6).

statusVZ Aktuální rychlost v ose Z (viz obrázek 1.6).

statusFrames Aktuální počet video snímku.

Abstraktní metody:

navdataInit Metoda spouštějící se při inicializaci přijímání navigačních dat (navdata). Slouží například pro vytvoření souborů či bufferu, do kterého se budou zaznamenávat navigační data.

navdataCycle Metoda spouštějící při každém příjmu navigačních dat. Slouží například pro zaznamenávání navigačních dat do souboru či bufferu.

navdataDead Metoda spouštějící se při ukončení přijímání navigačních dat. Slouží například pro ukončení bufferu.

videoBufferInit Metoda spouštějící se při inicializaci přijímání video streamu. Slouží například pro vytvoření bufferu video snímků.

videoBufferCycle Metoda spouštějící při každém příjmu video framu. Slouží například pro zobrazení video záznamu.

videoBufferDead Metoda spouštějící se při ukončení přijímání video streamu. Slouží například pro uložení videa do souboru.

D.2 Modul `detection_worker`

D.2.1 Třída `PicturesConvert`

Při vytváření instance je možné zadat obrázek a změnu velikosti obrázku.

Třída obsahuje následující metody:

resize Změna velikosti obrázku.

Obsahuje jen jeden parametr **scale**, který určuje procentuální změnu velikosti obrázku.

getActualPicture Vráť aktuální obrázek se všemi úpravami.

getOriginalActualPicture Vráť aktuální originální obrázek - bez jakýchkoliv aplikovaných úprav.

setPicture Nastaví jiný obrázek k zpracování (je brán jako originální).

Obsahuje jen jeden parametr **image**, který přijímá obrázek (numpy).

reset Smaže veškeré změny provedené nad posledním vloženým obrázkem.

getGray Převod obrázku do odstínu šedi.

getGauss Gausová filtrace včetně převodu do odstínu šedi.

getCanny Cannyho hranový detektor, včetně převodu do odstínu šedi.

Parametry:

low_threshold Velikost spodního prahu pro eliminaci nevýznamných hran. Výchozí hodnota je 50.

up_threshold Velikost horního prahu pro eliminaci nevýznamných hran. Výchozí hodnota je 50.

detection Detekce za pomoci kaskády.

Obsahuje jen jeden parametr **cascade**, který přijímá detekční kaskádu.

drawRectangles Vykreslení obdélníků do obrázku.

Parametry:

rectangles Seznam (list) obdélníků, které mají hodnoty: $[x, y, x2, y2]$.

color Barva obdélníků. 3 hodnoty (tuple) v pořadí: modrá, zelená, červená. Výchozí hodnota je $(0, 255, 0)$.

border Šířka čáry obdélníků. Výchozí hodnota je 2.

getFaceDetection Detekce obličejů.

getEyeDetection Detekce očí.

getFaceAllDetection Detekce obličejů, očí a brýlí.

getObjectsDetection Detekce podle seznamu kaskád.

Obsahuje jen jeden parametr **cascades**, který přijímá seznam (list) detekčních kaskád.

getPeopleDetection - Detekce lidí za pomoci HOG deskriptoru z OpenCV.

Parametry:

win_stride Čtvercové rozdělení obrázku na sít' stejně velkých bloků. Výchozí hodnota je 8

padding Rozměry detekčního okna. Výchozí hodnota je 32

scale Koeficient nárůstu okna detekce. Výchozí hodnota je 1.15.

sizesToCoordinates Změna na souřadnicové obdélníky. Z $[x, y, w, h]$ se vytvoří $[x, y, x2, y2]$.

Obsahuje jen jeden parametr **rectangles**, který přijímá seznam (list) obdélníků.

D.2.2 Třída RectanglesWorker

Při vytváření instance je možné zadat seznam (list nebo numpy) detekčních obdélníků.

Třída obsahuje následující metody:

setRectangles Nastavení nového seznamu obdélníků.

Obsahuje jen jeden parametr **rectangles** - seznam obdélníků.

getRectangles Vrácení aktuální sestavy obdélníků.

correctionRectangles Změna rozměrů detekčních obdélníků.

Parametry:

width Šířka - je-li None nebo 0, nebude se měnit. Výchozí hodnota je 0.75.

height Výška - je-li None nebo 0, nebude se měnit. Výchozí hodnota je 0.75.

getFindMaxWidthRectangle Najde nejširší obdélník.

getWidthRectangle Šířka jednoho obdélníku, není-li žádný vrátí 0.

Obsahuje jen jeden parametr **rectangle** - jeden obdélník.

getHeightRectangle Výška jednoho obdélníku, není-li žádný vrátí 0

Obsahuje jen jeden parametr **rectangle** - jeden obdélník.

getFilteredMaxSizeRectangles Vyfiltrování obdélníků, které přesahují zadanou šířku nebo výšku, respektive návratovou hodnotou budou obdélníky přesahující šířku nebo výšku.

Parametry:

width Maximální šířka obdélníků (bude-li None - nebude se brát v potaz). Výchozí hodnota je None.

height Maximální výška obdélníků (bude-li None - nebude se brát v potaz). Výchozí hodnota je None.

compareRectangles Porovnání, zda jsou oba obdélníky na podobném místě v podobné velikosti (viz algoritmus v kapitole 3.1).

Parametry:

rectangle_1 Obdélník k porovnání.

rectangle_2 Obdélník k porovnání.

size_divergence Koeficient (procenta 1 = 100 %, 0.5 = 50 %, 0 = 0 % [stejně velké]) o kolik mohou být obdélníky velikostně rozdílné. Viz SDC v kapitole 3.1. Výchozí hodnota je 0.25.

position_divergence Viz PDC v kapitole 3.1.

mode_position Zda se má určovat střed dle čtverce či kružnice (True - čtverec, False - kružnice). Výchozí hodnota je True.

averagingRectangles Vytvoření jednoho zprůměrovaného obdélníku na základě seznamu obdélníků

averagingAppropriateRectangles Zprůměrování podobných obdélníků na podobném místě. Využívá se algoritmu, vysvětleno viz kapitola 3.3.

Parametry:

size_divergence Viz SDC. Výchozí hodnota je 0.25.

position_divergence Viz PDC.

mode_position Viz mode_position. Výchozí hodnota je True.

buble_mode Je-li zapnut (`True`) bublinkový mód, tak se provádí ověřování obdélníků do té doby, dokud je vždy nějaký podobný obdélník nalezen. Je-li vypnut (`False`), provede se jen jednou. Výchozí hodnota je `False`.

filterEmptyRectangles Smazat prázdné obdélníky bez souřadnic.

D.2.3 Třída `FramesWorker`

Při vytváření instance je možné zadat seznam (list nebo numpy) framů.

Třída obsahuje následující metody:

setFrames Nastavení nového seznamu framů.

Obsahuje jen jeden parametr **frames** - seznam obdélníků.

getFrames Vrácení aktuální sestavy framů.

getCount Počet framů v seznamu.

averagingAppropriateRectangles Zprůměrování podobných obdélníků na podobném místě v daném framu. Aplikuje se pro každý frame zvlášť. Využívá se algoritmu vysvětleno v kapitole 3.3.

Parametry:

size_divergence Viz SDC. Výchozí hodnota je 0.25.

position_divergence Viz PDC.

mode_position Viz `mode_position`. Výchozí hodnota je `True`.

buble_mode Viz (`buble_mode`). Výchozí hodnota je `False`.

filteringIncorrectRectangles Algoritmus na odfiltrování nenavazujících detekcí na základě hadovitého hledání (viz popis algoritmu v kapitole 3.4).

Parametry:

buffer_size Velikost bufferu.

find_in_buffer Minimální počet framů, ve kterých musí být nalezené podobné detekce (maximální velikost: `buffer_size - 1`).

step_coefficient O kolik se má v každém následujícím framu navýšit `position_divergence`, pokud nebyla nalezená detekce v aktuálním framu. Výchozí hodnota je 0.25.

maximum_step Kolikrát se může navýšit `position_divergence` o `step_coefficient` (-1 - vypnuto - neomezeněkrát; 0 - nenavýšovat). Výchozí hodnota je -1.

size_divergence Viz SDC. Výchozí hodnota je 0.25.

position_divergence Viz PDC.

mode_position viz `mode_position`. Výchozí hodnota je `True`.

keyRectangle Algoritmus, který předá frame s jedním nebo žádným detekčním obdélníkem zájmu za pomoci hadovitého hledání (viz popis algoritmu kapitola 3.5).

Parametry:

find_in_buffer Velikost bufferu.

number_steps_predict Počet framů k vytvoření predikčního framu CDR (Candidates Detection Rectangles).

step_coefficient Viz `step_coefficient`. Výchozí hodnota 0.25.

maximum_step viz `maximum_step`. Výchozí hodnota je -1.

size_divergence viz SDC. Výchozí hodnota je 0.25.

position_divergence Viz PDC. Výchozí hodnota je 0.25.

mode_position jak je uvedeno výše. Výchozí hodnota je `True`.

merging Sloučení framů do jednoho framu.

mergingWithAveragingAppropriateRectangles Sloučení framů do jednoho framu, přičemž se podobné obdélníky budou průměrovat.

Parametry:

size_divergence Viz SDC. Výchozí hodnota je 0.25.

position_divergence Viz PDC. Výchozí hodnota je 0.25.

mode_position jak je uvedeno výše. Výchozí hodnota je `True`.

buble_mode jak je uvedeno výše. Výchozí hodnota je `True`.

D.3 Modul socket server

Modul obsahuje třídy: `Message`, `Handler`, `SocketServer`, `Client` pro socket server-klient komunikaci.

D.3.1 Třída Message

Třída obsahuje následující metody:

parseMessage Rozparsování bufferu a spuštění daných funkcí na základě abstraktní metody `runFunction`.

buildMessage Sestavení zprávy. Metoda vrátí seznam (list) fragmentů (řetězců) sestavené zprávy. Každý fragment má délku maximálně 4096 znaků.

Parametry:

function Název funkce, nebo-li datový/návratový typ či specifikace zprávy.

params Seznam (list) přidaných/specifikujících parametrů.

addData Zaevidování dat do bufferu.

Abstraktní metody:

runFunction Abstraktní metoda, která bude volat následně metody dle předávaného parametru `param`.

D.3.2 Třída Handler

Při vytváření instance je potřeba vložit objekt socketu, případně nastavit velikost bufferu (výchozí hodnota je jinak 10000000).

Třída obsahuje následující metody:

run Čekání na příjem dat od klienta/serveru, které se zaevidují do bufferu, rozparsují a na základě toho se spustí funkce s případnou odpovědí.

sendMessage Odeslání zprávy klientovi/serveru.

Parametry:

function Název funkce, nebo-li datový/návratový typ či specifikace zprávy.

params Seznam (list) přidanych/specifikujících parametrů.

D.3.3 Třída SocketServer

Při vytváření instance je potřeba zadat port na kterém má naslouchat.

Třída obsahuje následující metodu **run**, která spustí server. Dále obsahuje abstraktní metodu **clientHandler**, která se spustí při navázání komunikace s klientem. Obsahuje parametr `client` obsahující objekt připojeného socketu.

D.3.4 Třída Client

Při vytváření instance je potřeba zadat adresu hostitelského serveru (parametr **host**), port hostitelského serveru (parametr **port**), případně velikost bufferu (parametr **buffer.limit**), který jinak má výchozí hodnotu 10000000.

Třída obsahuje 2 následující metody:

sendMessage Odeslání zprávy serveru. Neočekává se žádná odpověď (jednosměrná komunikace)

Parametry:

function Název funkce, nebo-li datový/návratový typ či specifikace zprávy.

params Seznam (list) přidanych/specifikujících parametrů.

sendMessageWaiting Odeslání zprávy na server s následným čekáním na odpověď. Obousměrná komunikace.

Parametry:

function Název funkce, nebo-li datový/návratový typ či specifikace zprávy.

params Seznam (list) přidanych/specifikujících parametrů.

reply_function Název funkce návratové zprávy, respektive očekávaný název odpovědi.

identification_param Číselná hodnota identifikace zprávy (musí být jako první parametr - hned po názvu funkce). Výchozí hodnota je `None` - neočekává se žádná číselná identifikace zprávy.

D.4 Třída `HybridRingBuffer`

Při vytvoření instance je potřeba zadat maximální velikost bufferu, nezadá-li se, bude velikost rovná 10 (a již nepůjde velikost změnit).

Třída obsahuje následující metody:

add Přidání dalšího prvku do bufferu.

Obsahuje jen jeden parametr **item** - přidávaný prvek.

getAllItems Vrátí obsah (list) celého bufferu.

getFirstItem Vrátí nejaktuálnější/nejčerstvější záznam.

getSecondItem Vrátí druhý nejčerstvější záznam.

getLastItem Vrátí nejstarší záznam nacházejícího se v bufferu.

getItemInPosition Vrátí záznam na dané pozici v bufferu.

Obsahuje jen jeden parametr **i** - pozice v bufferu kde 0 je nejčerstvější záznam. Při zadání neexistující pozice se vrátí `None`.

getSize Aktuální velikost bufferu.