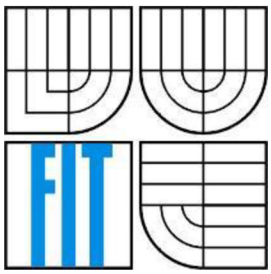
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ROZŠÍŘENÍ ŘÍDICÍHO SYSTÉMU MODELU LETADLA SKYDOG O PODPORU VZDÁLENÉHO A SAMOČINNÉHO ŘÍZENÍ ANDROID APLIKACÍ

EXPANSION OF SKYDOG AIRCRAFT MODEL CONTROL SYSTEM BY REMOTE AND AUTONOMOUS CONTROL BY ANDROID APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. MICHAL BOČEK
AUTHOR

VEDOUCÍ PRÁCE — ING. JOSEF STRNADEL, PH.D.
SUPERVISOR

BRNO 2014

Abstrakt

Tématem práce je návrh a implementace Android aplikace, která bude ovládat autopilota modelu letadla Skydog pomocí bezdrátové komunikace. Aplikace směrem od modelu přijímá data ze senzorů instalovaných v modelu letadla, které zpracuje a odešle zpět instrukce pro autopilota. V případě hrozící srážky modelu s překážkou nebo terénem aplikace odešle autopilotovi instrukce k vyhnutí se překážce. K nalezení bezkolizní trati letu je využito algoritmu RRT. Databázi známých překážek a digitální model terénu aplikace očekává ve formátu XML a GeoTIFF v tomto pořadí.

Abstract

The thesis aims to design and implement an Android application with ability to control the autopilot of the Skydog aircraft model using the wireless telemetry. The application shall receive data from an aircraft model gathered from various installed sensors. These data shall be then processed and corresponding instructions for autopilot shall be sent back. When collision with terrain or obstacle is detected, the application shall send instructions to autopilot to avoid such collision. RRT algorithm is used to find collision-free flight trajectory. Database of known obstacles and digital terrain model are provided to application in formats XML and GeoTIFF respectively.

Klíčová slova

Autopilot, APM, Arduino, Android, bezpilotní letadlo, UAV, UAS, MAVLink, DTM, RRT, GeoTIFF, Protisrážkový systém

Keywords

Autopilot, APM, Arduino, Android, unmanned aerial vehicle, UAV, UAS, MAVLink, DTM, RRT, GeoTIFF, Collision Avoidance System

Citace

BOČEK, Michal. *Rozšíření řídicího systému modelu letadla Skydog o podporu vzdáleného a samočinného řízení Android aplikací*. Brno, 2014. Diplomová práce. FIT VUT v Brně.

Rozšíření řídicího systému modelu letadla Skydog o podporu vzdáleného a samočinného řízení Android aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Josefa Strnadela, Ph.D.
Další informace mi poskytl Ing. Libor Kubečka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Boček
1. 6. 2014

Poděkování

Tímto bych rád poděkoval vedoucímu práce Ing. Josefu Strnadelovi, Ph.D. a Ing. Liboru Kubečkovi za věcné připomínky, cenné rady a konzultaci během zpracování. Dále bych rád poděkoval Bedřichu Saidovi za uskutečnění testovacích letů s využitím jeho modelu letadla Skywalker.

© Michal Boček, 2014
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod ... 7
2 Přehled prostředků a problémů souvisejících s tématem práce ... 8
2.1 Projekt Skydog ... 8
2.2 Autopilot ... 9
2.2.1 Platforma Arduino ... 10
2.2.2 ArduPilot Mega ... 11
2.2.3 Režimy autopilota ... 12
2.3 Bezdrátová komunikace ... 13
2.3.1 Hardware pro komunikaci ... 13
2.3.2 Protokol MAVLink ... 14
2.4 Aplikace Skydog Controller ... 16
2.4.1 Účel aplikace ... 16
2.4.2 Uživatelské rozhraní aplikace ... 16
2.4.3 Univerzálnost aplikace ... 17
2.5 Návrat letadla do oblasti se signálem ... 18
2.6 Protisrážkový systém ... 18
2.6.1 Informace o překážce ... 19
2.6.2 Databáze překážek ... 21
2.7 Algoritmus vyhnutí ... 21
2.8 Legislativní podmínky pro UAV ... 22
3 Implementace Android aplikace ... 24
3.1 Plánování ... 24
3.1.1 Časový plán ... 24
3.1.2 Analýza rizik ... 24
3.1.3 Úložný server ... 25
3.1.4 Verzovací systém ... 25
3.1.5 Vývojové prostředí ... 25
3.1.6 Styl zápisu kódu ... 25
3.2 Detaily aplikace SkyControl ... 26
3.2.1 Omezení Android API ... 26
3.2.2 MAVLink ... 26
3.2.3 Struktura aplikace ... 27
3.3 Reprezentace překážek ... 28
3.3.1 Reprezentace n-bokého hranolu ... 28
3.3.2 Digitální model terénu ... 30
3.4 Protisrážkový systém ... 36
3.4.1 Vincentyho souřadnicové algoritmy ... 37
3.4.2 Výpočet budoucí polohy letadla ... 37
3.4.3 Kontrola překážek před letadlem ... 38

3.4.4 Vyhledání bezkolizní trati letu ... 40
3.5 Samočinné řízení trajektorie letadla ... 45
3.5.1 Mise ... 45
3.5.2 Mód Waypoint Follower ... 46
3.5.3 Mód Flight Director ... 47
3.1 Návrat letadla do oblasti se signálem ... 49
4 Ověření v praxi ... 51
4.1 Testovací lety ... 51
4.2 Simulace letu ... 52
4.3 Zhodnocení realizace ... 54
5 Závěr ... 56
Literatura ... 57
Seznam zkratek ... 59
Seznam obrázků ... 61
Seznam tabulek ... 63
Seznam příloh ... 64

1 Úvod

V nedávné době se rozšířil zájem o bezpilotní systémy (UAS), které se týkají letadel bez lidské posádky, ať lehčích nebo těžších vzduchu, s pevnými či pohyblivými nosnými plochami. UAS sestávají z bezpilotního letadla (UAV) a pozemní stanice s tím, že letadlo je se stanicí propojeno bezdrátově. Jedním z cílů této práce je ulehčit osobám ovládajícím vzdáleně autopilota UAV operování s pozemní stanicí, a to implementací aplikace pozemní stanice pro přenosná Android zařízení. Tato aplikace by měla dokázat měnit trajektorii modelu letadla instruováním autopilota, např. pomocí traťových bodů.

Dalším z cílů této práce je integrovat do Android aplikace protisrážkový systém, schopný poslat autopilotu instrukce k provedení úhybného manévru v případě detekce kolize letadla s překážkou či terénem. Kalkulace úhybných manévrů probíhají v aplikaci, a to nad informacemi o překážkách a terénu, uloženými v souborech formátu XML a GeoTIFF. Žádný z běžně dostupných UAS podobný systém aktuálně nenabízí.

Autor má dále za úkol v rámci této práce prozkoumat možnosti navedení letadla při ztrátě spojení s pozemní stanicí zpět do oblasti se signálem.

Tato diplomová práce navazuje na semestrální práci se stejným titulem. Ze semestrální práce byla převzata celá kapitola 2, zabývající se úvodem do problematiky. Následně byla v rámci diplomové práce sepsána kapitola 3, popisující implementaci Android aplikace. Kapitoly 4 a 5 se věnují ověření aplikace v praxi a zhodnocení projektu.

2 Přehled prostředků a problémů souvisejících s tématem práce

2.1 Projekt Skydog

Skydog je projektem zaměstnanců oddělení Flight Controls firmy Honeywell sloužící jako volnočasová aktivita vhodná pro porozumění návrhu systému řízení letadla. Protože zaměstnanci oddělení Flight Controls řeší v rámci pracovní náplně izolované podproblémy systému řízení velkých letadel, na projektu Skydog mají možnost pochopit komplexitu systému řízení letounu tím, že stojí od počátku u jeho návrhu a také díky tomu, že je takovýto systém mnohem jednodušší. Cílem je zprovoznit platformu pro řízení modelu letounu, která bude vyhodnocovat informace ze senzorů, kterými je letoun osazen, a která bude ovládat motor a řídicí plochy modelu letounu.

V rámci projektu je k dispozici eponymní model letadla na obrázku 2.1. Základní parametry tohoto modelu jsou shrnuté v tabulce 2.1.

Parametr	Hodnota
Rozpětí	2340 mm
Délka	1600 mm
Hmotnost (vč. baterií)	cca 7 kg
Cestovní rychlost	cca 80 km/h
Max. výdrž na plně nabité baterie	cca 25 min

Tabulka 2.1: Základní parametry modelu Skydog. Zdroj: [1]

Obrázek 2.1: Model letadla Skydog. Zdroj: [1]

Model je osazen 3-fázovým motorem na střídavý proud AXI 5320/34 GOLD LINE spolu s vrtulí o průměru 18 palců a stoupáním 12 palců.

Použitou baterií je Hyperion Litestorm VX 25C od firmy APC s celkovou kapacitou 10000 mAh. ESC SPIN Pro 77 je použitý elektronický regulátor otáček (ESC), jehož účelem je řídit rychlost a směr otáčení motoru.

Servomotory použité na modelu Skydog jsou elektromotory, které přenáší mechanický pohyb na:

- křidélka,
- vztlakové klapky,
- směrové kormidlo,
- výškové kormidlo.

Servomotory jsou ovládány pulsně šířkovou modulací (PWM) impulsů z řídící desky modelu. Výchylky a středové polohy ovládacích ploch letadla byly nastaveny na základě chování letadla při letových testech.

Datové spojení s vysokým dosahem je nezbytné pro řízení modelu letadla. Pro radiové spojení byla vybrána RC souprava Spektrum DX7s DSMX 2,4GHz. Vysílač je 7-kanálový.

2.2 Autopilot

Začneme definicí autopilota. Autopilotem je řídicí systém určený k samočinnému řízení trajektorie letadla bez přímé účasti pilota na této činnosti. Autopilot nenahrazuje pilota, pouze pilotovi umožnuje důkladněji se soustředit na ostatní prvky pilotáže, jako například navigaci, sledování počasí, kontrole systémů letadla, atd.

Trajektorii letu zmíněnou výše definujeme jako geometrickou čáru ve vzdušném prostoru, kterou letadlo opisuje za letu. A trať letu, o níž se budeme bavit v dalších kapitolách, definujeme jako průmět trajektorie letadla na povrch Země.

Pro autonomní řízení elektroniky letadla (servomotory, ESC) byla zvolena platforma autopilota ArduPilot Mega verze 2.5, a to z důvodu ceny a dostupnosti zdrojového kódu. ArduPilot Mega je velice rozšířený mezi modeláři a existuje mnoho fór, kde se řeší problémy související s tímto autopilotem. Následující doplňky byly pořízeny ke zkompletování senzorů potřebných pro řízení letu:

- 3DR uBlox GPS – deska osazená 5Hz GPS modulem uBlox LEA-6H a tříosým digitálním kompasem HMC5883L,
- Freescale MPXV7002DP – snímač tlaku z Pitotovy trubice pro získání rychlosti letadla vůči mase vzduchu,
- 3DRobotics 3DR Radio 433 MHz – sada dvou RF modulů pro bezdrátovou komunikaci modelu s pozemní stanicí.

Samotná deska ArduPilot Mega je v základu osazena těmito sensory:

- InvenSense MPU-6000 – šestiosý gyroskop/akcelerometr,
- MEAS MS5611 – barometrický výškoměr.

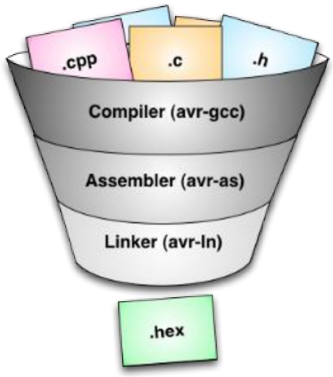
Projekt ArduPilot Mega zahrnuje i firmware autopilota APM:Plane spolu s aplikací pozemní stanice Mission Planner, která umožnuje nastavení hardwarové desky a veškerých parametrů autopilota.

2.2.1 Platforma Arduino

Arduino je univerzální open-source elektronická platforma navržená za účelem co nejvyšší míry zjednodušení použití elektroniky pro uživatele, bez ohledu na jejich obor. Hardware zahrnuje desky plošných spojů (dále jen desky), kterých je nyní na 20 typů, osazené 8 až 32 bitovými mikrokontroléry AVR firmy Atmel.

Projekt zahrnuje i Arduino IDE, což je software, pomocí něhož se programují desky Arduino. IDE zahrnuje překladač (avr-gcc), assembler a linker (GNU Binutils), konfigurované pro procesory AVR.

Avr-gcc podporuje jazyky C a C++. Místo standardní knihovny jazyka C je používána knihovna AVR LibC. Pro jazyk C++ však chybí implementace standardní knihovny libstdc++.



Obrázek 2.2: Kroky k získání HEX souboru pro AVR. Zdroj: [2]

AVR má v paměti sekci nazývanou zavaděč (bootloader), v níž je nahrán kód spouštějící se vždy po restartu desky. Arduino má již předprogramovaný bootloader, který dovoluje přepisovat flash paměť přes sériové rozhraní.

Desky je možno zakoupit už sestavené, ale protože jde o open-source projekt, je možné si stáhnout schéma desky, upravit ho a sestavit si desku svojí. Proto z Arduina vychází mnoho dalších projektů, mj. projekt ArduPilot Mega zmíněný níže.

Obrázek 2.3: Arduino Mega 2560. Zdroj: [3]

2.2.2 ArduPilot Mega

ArduPilot Mega (dále jen APM) je označení pro open-source projekt autopilota modelů dopravních prostředků, jako například letounů, jedno a více rotorových vrtulníků, lodí, atd. Tento projekt zahrnuje jak hardware, tak i software. Nejnovější HW tohoto autopilota vychází z desky Arduino Mega 2560, uvedené na obrázku 2.3.

V začátku projektu v roce 2007 šlo pouze o hardware desku nazvanou ArduPilot osazenou senzory potřebnými pro autonomní řízení letadla se základním firmwarem zpřístupňujícím funkcionalitu desky. Poté, co byla nová verze desky založena na desce Arduino Mega s výkonnějším čipem Atmel ATmega1280, byla deska přejmenována na ArduPilot Mega. Později byl k desce vydán firmware označený APM 1.0, obstarávající základní funkce autopilota. Tento firmware byl neustále rozšiřován o nové funkcionality a od verze firmware APM 2.0 se název rozdělil na ArduPlane, ArduCopter, ArduRover - podle toho, pro který druh pohybujícího se objektu byl autopilot určen. Nejnověji se v roce 2013 sjednotilo pojmenování celé platformy na APM Autopilot Suite, v rámci níž je HW deska nazvána APM a firmware APM:Plane, APM:Copter, APM:Rover.

Nejnovější HW deska s označením APM 2.6 obsahuje mimo jiné čip Atmel ATmega2560 jako hlavní výpočetní jednotku, dále gyroskop, akcelerometr, magnetometr, tlakoměr pro získání nadmořské výšky a GPS modul.

Nejnovější firmware pro letouny, APM:Plane ve verzi 2.76, již zahrnuje i pokročilé funkce autopilota, jako například automatický vzlet a přistání, plně plánovatelné mise s využitím GPS bodů, možnost simulace misí leteckými simulátory pro PC, návrat letounu na místo vzletu, atd.

Obrázek 2.4: Deska ArduPilot Mega 2.5. Zdroj: [4]

V rámci projektu APM je vyvíjen software Mission Planner, což je PC aplikace, tzv. pozemní stanice (GCS), která umožňuje:

- nahrát na APM desku firmware přes USB,
- nastavit parametry autopilota,
- naplánovat misi vytyčením bodů na mapě,
- stáhnout a analyzovat záznamy letu z flash paměti APM,
- komunikovat s deskou bezdrátově, pokud je zapojen externí modul, např. XBee nebo 3DR Radio,
- přijímat informace ze senzorů desky, které graficky zobrazuje, viz obrázek 2.5.

Obrázek 2.5: Grafické rozhraní GCS Mission Planner

2.2.3 Režimy autopilota

APM nabízí množství letových režimů, které staví APM do různých rolí, začínajících u jednoduchého systému pro stabilizaci letu až po sofistikovaného autopilota. APM podporuje následující režimy.

- MANUAL,
- STABILIZE,
- FLY BY WIRE_A (FBWA),
- FLY BY WIRE_B (FBWB),
- AUTOTUNE,
- TRAINING,
- ACRO,
- CRUISE,
- AUTO,
- Return To Launch (RTL),
- LOITER,
- CIRCLE,
- GUIDED,
- TAKEOFF,
- LAND.

Vysvětlení funkce všech režimů by nebylo účelné, protože Android aplikace bude pracovat pouze se třemi z nich, a to s režimy MANUAL, AUTO a RTL.

Režim MANUAL

V tomto režimu je letadlo řízeno vysílačkou a APM nezasahuje do letu, kromě případu ztráty signálu s vysílačkou. V tu chvíli se spustí předdefinovaná akce, která ve výchozím nastavení znamená aktivaci režimu Return To Launch, tedy návrat na základnu.

Režim AUTO

V tomto režimu letadlo vykonává misi, sestávající z jednoho a více příkazů. Nejčastější je použití navigačních příkazů, které mají za úkol dovést letadlo na dané souřadnice. Více o misi a jejích možných příkazech v kapitole 3.5.1.

V režimu AUTO lze stále ovládat výškové a směrové kormidlo a křidélka vysílačkou, která má prioritu, avšak pouze s maximálními výchylkami danými nastavením parametrů APM (např. LIM_ROLL_CD).

Režim RTL

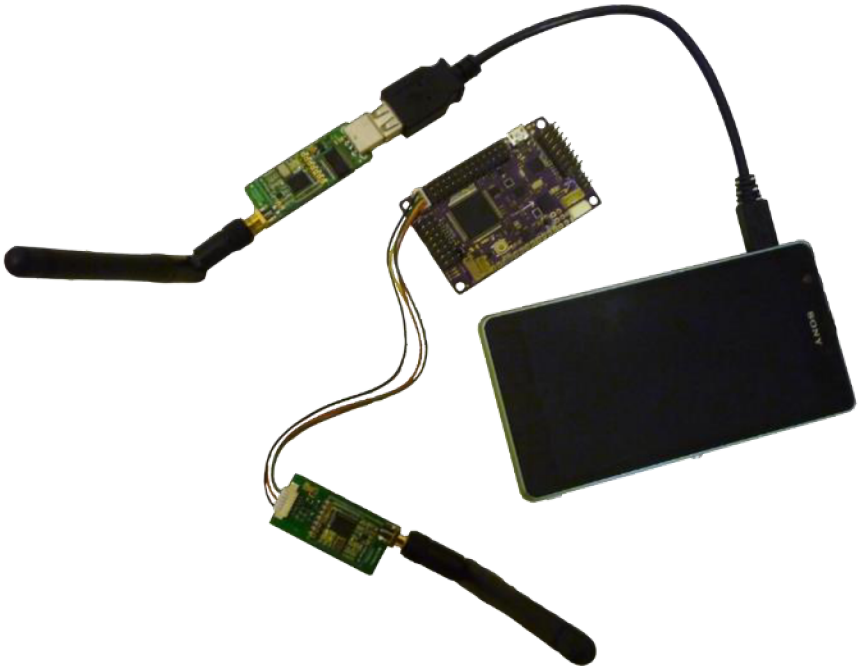
V režimu Return To Launch (RTL) se letadlo vrací do pozice základny. Výchozí pozicí základny je místo, kde GPS modul poprvé získal souřadnice autopilota. Pozici základny je možné kdykoliv změnit z pozemní stanice příkazem mise MAV_CMD_DO_SET_HOME. Nadmořská výška, nad kterou letadlo po dosažení pozice začne kroužit, je dána APM parametrem ALT_HOLD_RTL.

2.3 Bezdrátová komunikace

2.3.1 Hardware pro komunikaci

K obousměrné bezdrátové komunikaci modelu Skydog s pozemní stanicí, tedy Android zařízením, slouží sada 3D Robotics Radio 433Mhz. Přenos dat probíhá na radiové frekvenci 433 MHz – jde o variantu pro Evropu, která má jiná pravidla pro využívání frekvenčního pásma, než USA, kde je používán tento modul na frekvenci 915 Mhz. Evropská verze má dosah přibližně 1,6 km v přímé viditelnosti a přenosová rychlost dosahuje až 250 kb/s. Sada 3DR Radio 433MHz sestává ze dvou modulů s vysílacím výkonem 100 mW.

Většina moderních Android zařízení, ať jde o tablet nebo telefon, podporuje funkci USB host, nebo jinak pojmenované USB OTG (On-The-Go). Pomocí této funkce a USB OTG kabelu lze zapojit modul pro pozemní stanici do telefonu tak, jak je vidět na obrázku 2.6. Druhý modul se zapojuje přes dedikovaný konektor na desce APM.

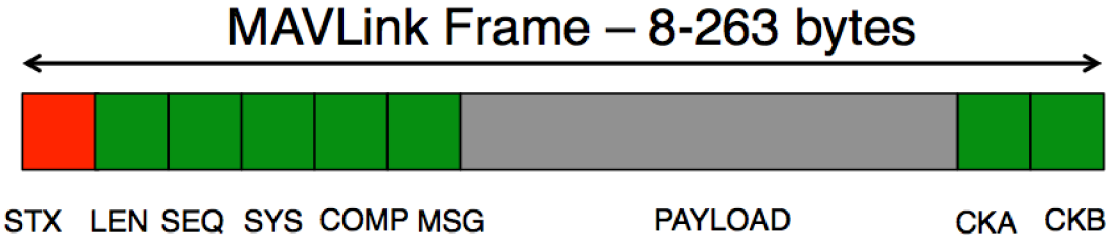


Obrázek 2.6: Propojení APM s Android zařízením pomocí 3DR Radio 433 Mhz

2.3.2 Protokol MAVLink

Komunikaci mezi oběma moduly zajišťuje protokol MAVLink. Jde o protokol vyvinutý v roce 2009 pro přenos parametrů potřebných pro řízení letu bezpilotních letadel (UAV). Nyní je využívaný řadou open-source autopilotů, včetně APM:Plane.

MAVLink je licencován pod licencí Lesser General Public License (LGPL) od Free Software Foundation. To znamená, že aplikace, která je zkompilovaná s přilinkováním knihovny MAVLink, není považována za odvozené dílo od knihovny MAVLink. MAVLink tedy může být použit bez omezení v jakékoli open-source či proprietární aplikaci.



Obrázek 2.7: Formát MAVLink paketu se zprávou. Zdroj: [5]

Bajt	Název	Hodnota	Popis
0	Znak začátku paketu	0xFE	Indikuje začátek nového paketu.
1	Velikost dat zprávy	0 - 255	Indikuje velikost *n* zprávy přenášené v paketu, v bajtech.
2	Sekvenční číslo paketu	0 - 255	Každá komponenta užívá sekvenční číslo odesílaného paketu k detekování ztráty paketu.
3	ID systému	1 - 255	ID systému, který odesílá paket. Umožnuje rozlišit až 255 odesílatelů ve společném vzdušném prostoru.
4	ID komponenty	0 - 255	ID komponenty odesílající paket. Umožňuje rozlišit rozdílné komponenty v rámci jednoho systému, např. IMU a autopilota.
5	ID zprávy	0 - 255	ID zprávy – identifikační číslo určující, co data znamenají a jak mají být dekódována.
6 až (*n*+6)	Data zprávy	(0 - 255) bajtů	Data zprávy.
(*n*+7) až (*n*+8)	CRC	ITU X.25/SAE AS-4 hash počítaný bez znaku začátku paketu.	

Tabulka 2.2: Význam jednotlivých bajtů MAVLink paketu. Zdroj: [5]

Autoři protokolu MAVLink zároveň vytvořili knihovnu se zdrojovým kódem v jazyce C++. Její použití v systému Android je možné pomocí Android Native Development Kit (NDK). Nástroj NDK dokáže zkompilovat C/C++ kód a vytvořit dynamickou knihovnu, kterou je možno nahrát a použít za běhu Android aplikace. Kód napsaný v C/C++ se v Androidu nazývá nativní a z Javy se takový kód spouští díky Java Native Interface (JNI).

JNI definuje způsob, jak použít nativní kód v Android aplikaci napsané v Javě. Tento kód se zkompiluje pro konkrétní CPU architekturu pomocí Android NDK s tím, že podporované architektury jsou ARM, x86 a MIPS. Výsledkem kompilace je dynamická knihovna, kterou dokáže Java aplikace za běhu načíst a volat její funkce díky JNI. Takto přeložený nativní kód je následně zpracováván efektivněji, a tedy rychleji, než kdyby byl napsaný v Javě. Nevýhodami jsou nemožnost použití Android framework API a zvýšení složitosti a nepřehlednosti kódu.

V tabulce 2.3 jsou uvedeny operace mezi Javou a C/C++, které umožňuje JNI rozhraní.

Java	C/C++
Zavolání funkce C/C++	Zavolání metody Javy
	Přístup k proměnné Javy
	Konstrukce Java objektu
	Volání výjimky Javy

Tabulka 2.3: Povolené operace mezi rozhraními jazyků Java a C/C++

2.4 Aplikace Skydog Controller

2.4.1 Účel aplikace

V rámci této práce má být navržena a implementována aplikace pro přenosná zařízení s operačním systémem Android, pojmenovaná Skydog Controller.

Aplikace bude fungovat jako odlehčený ekvivalent k GCS Mission Planner a bude umožňovat:

- zapnout/vypnout autopilota,
- zobrazit aktuální polohu letadla na mapě,
- zobrazit údaje o letu (nadmořskou výšku, rychlost letadla, vertikální rychlost, kurz letadla) a nastavit tyto údaje pro samočinné řízení trajektorie letu,
- zadat traťové body,
- indikovat spojení s APM,
- předat modelu letadla pokyny k vyhnutí se překážce.

2.4.2 Uživatelské rozhraní aplikace

Na obrázcích 2.8 a 2.9 je představen návrh možného vzhledu grafického uživatelského rozhraní (UI) obslužné aplikace.

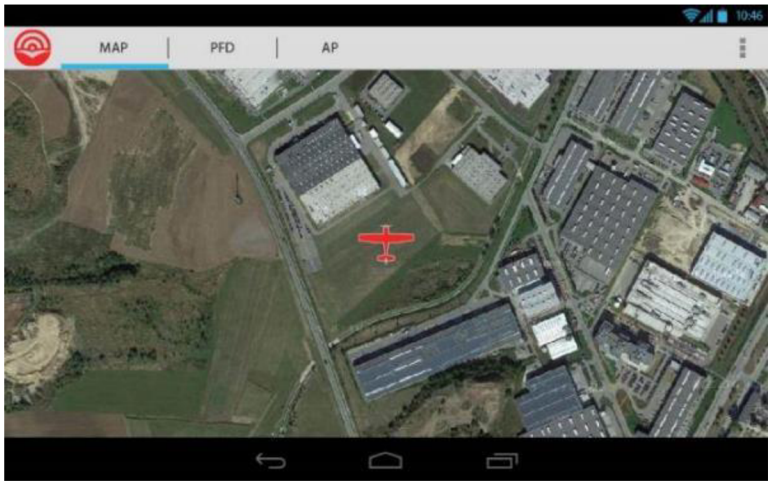
Obrázek 2.8 představuje návrh zobrazení aktuální polohy letadla na základě GPS souřadnic. Po dotyku na mapu bude možné vytvořit traťové body pro autopilota. Ty zůstanou zobrazeny na mapě. Seznam všech zadaných traťových bodů s možností jejich editace bude v kontextové nabídce, vyvolatelné přes tři šedé tečky vpravo nahoře.

V případě návrhu uživatelského rozhraní hlavního displeje (PFD) na obrázku 2.9 jde o snahu o napodobení PFD z letadla Airbus A320, a to hlavně kvůli přehlednosti zobrazovaných veličin:

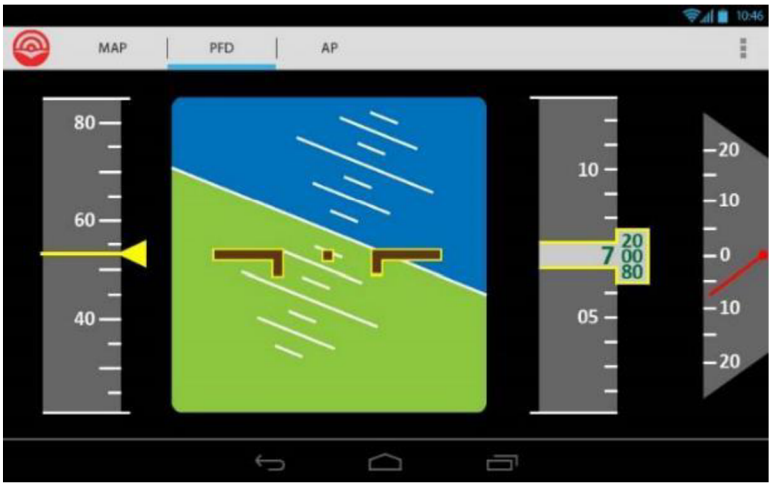
- šedý sloupec nejvíce vlevo indikuje rychlost v uzlech za hodinu,
- druhý šedý sloupec vpravo od umělého horizontu indikuje nadmořskou výšku ve stopách,
- třetí sloupec nejvíce vpravo indikuje vertikální rychlost ve stovkách stop za minutu.

Záložka AP, pro kterou nebyl vytvořen grafický návrh, umožní jednoduché nastavení hodnot pro autopilota, tedy nadmořskou výšku, traťovou rychlost, maximální rychlost stoupání/klesání a kurz. Bude zde rovněž přepínač zapnutí/vypnutí autopilota a výběr z možných režimů autopilota.

Nastavení aplikace bude v kontextové nabídce, vyvolatelné přes tři šedé tečky vpravo na horní liště.



Obrázek 2.8: Návrh UI pro zobrazení GPS polohy letadla



Obrázek 2.9: Návrh UI pro zobrazení parametrů letu

Obrázek 2.10: Ikona aplikace Skydog Controller

2.4.3 Univerzálnost aplikace

Díky použití protokolu MAVLink bude možné použít aplikaci Skydog Controller i k řízení jiných autopilotů. MAVLink protokol podporuje např. autopilot Pixhawk. P. Čamaj z FEKT VUT v Brně začal v roce 2013 v rámci jeho diplomové práce vyvíjet autopilota založeného na platformě Raspberry Pi [1]. Autor této práce plánuje v budoucnu spolupracovat na knihovně MAVLink pro tohoto autopilota a tak ověřit, že Skydog Controller může být použitý k řízení i jiného autopilota, než APM.

2.5 Návrat letadla do oblasti se signálem

Součástí zadání je navrhnout rozšíření, které umožní letadlu jeho návrat při ztrátě spojení s aplikací do oblasti se signálem. Avšak, jak bylo v průběhu práce na semestrálním projektu zjištěno, firmware APM:Plane podobnou funkci nabízí implicitně.

Autopilot se přepne do režimu RTL, který má za úkol navést letadlo nad základnu a začít nad ní kroužit, pokud byla zaznamenána ztráta bezdrátového spojení po více jak 20 vteřin. Tento časový limit je v případě modelu Skydog nedostatečný, protože za 20 vteřin uletí při cestovní rychlosti 80 km/h téměř půl kilometru. Pokud je odesílání povinné MAVLink zprávy pulsu (heartbeat), jejíž přijetí značí zdravé spojení obou bezdrátových modulů, nastaveno na nejvyšší možnou frekvenci 1 Hz, můžeme snížit limit z 20 na 5 vteřin.

Protože pozemní stanice, tedy přenosné Android zařízení, nemusí zůstávat na místě základny, pak na tomto místě, kam se navrací letadlo v režimu RTL, nemusí nutně být signál pro opětovné navázání spojení. Protože bude aplikace pracovat pouze s režimy autopilota AUTO a MANUAL, lze následujícím způsobem rozšířit ve firmware APM:Plane funkci režimu AUTO.

1. Android aplikace v pravidelných režimech posílá letadlu svou pozici a nastavuje jí jako místo vzletu.
2. Při ztrátě bezdrátového spojení je vypočítána aktuální vzdálenost d letadla od místa vzletu.
3. Letadlo zamíří k takovému z následujících traťových bodů, jehož vzdálenost je menší, než d.
4. Pokud nevyhovuje bodu 3 žádný traťový bod, přepne se autopilot do režimu RTL.

Po celou dobu letu je ve všech režimech autopilota možné převzít řízení letadla RC vysílačkou, a to pokud má vysílačka vyhrazený jeden kanál pro přepnutí autopilota do režimu MANUAL.

2.6 Protisrážkový systém

Jedním z cílů této práce je implementovat systém, který má za úkol v případě detekování hrozící srážky s překážkou či terénem vydat autopilotu pokyny k vyhnutí se překážce. K detekování srážky využívá digitálního modelu terénu, databáze překážek a aktuálních letových údajů, na jejichž základě dokáže vyhodnotit budoucí pozici letadla.

V komerčním letectví se běžně používají systémy Terrain Awareness and Warning System (TAWS), mezi něž patří například komerční produkt Enhanced Ground Proximity Warning System (EGPWS), které fungují obdobně, ale v případě detekce hrozící srážky pouze o takové hrozící srážce varují zvukovou zprávou „TERRAIN, TERRAIN, PULL UP“ [6].

Lockheed Martin a USAF plánují v roce 2014 nasadit do běžného provozu stíhací letouny osazené systémem Automatic Ground Collision Avoidance System (AGCAS), který je schopen v pilotovaném letadle automatického vyhnutí se terénu, pokud hrozí srážka [7]. Ve vojenském letectví je takový systém určen především pro situace, kdy je pilot dezorientován po provedení

náročného manévru nebo pokud upadl do bezvědomí při nadměrném přetížení. V případném nasazení v komerčním letectví by systém pomohl v nebezpečných situacích za špatné viditelnosti.

2.6.1 Informace o překážce

První otázkou, kterou bylo třeba se zabývat v rámci protisrážkového systému, je zdroj informace o překážce, které se má letoun vyhnout. Jednou z možností je získávat data v reálném čase ze senzoru, který by byl instalovaný na modelu. Další možností je použít předgenerovanou databázi překážek spolu s digitálním modelem terénu.

Použitelnými technologiemi pro skenování okolí v reálném čase jsou:

- laser,
- radar na bázi Dopplerova efektu,
- počítačové vidění.

Laser

Laserový měřič vzdálenosti, jiným názvem Lidar, měří vzdálenost osvícením cíle laserem a analyzováním odraženého paprsku. Mezi zásadní nevýhody, které Lidar vyřazují, patří:

- zaměření paprsku na jedno konkrétní místo,
- váha zařízení, které by dokázalo změřit vzdálenost >100 m.

Existují laserové měřiče vzdálenosti, které jsou blízko velikostním a váhovým parametrům vhodným pro UAV, jako například Lightware DS00 s váhou 225 g a dosahem 100 m. Nicméně měření vzdálenosti v jednom úhlu vůči modelu letadla je vhodné například pro UAV helikoptéry pro přesné automatické přistání. Při použití v protisrážkovém systému by sice s jeho použitím bylo možné detekovat hrozící srážku za letu, ale ne již nalézt bezkolizní trať letu pro vyhnutí se srážce, protože by chyběly informace o okolí.

Radar na bázi Dopplerova efektu

Technologie radaru je stále příliš robustní k použití na malém UAV. V jedné z aplikací radaru s UAV [8] se podařilo použít radar o váze 304 g s dosahem pouze 15 m a úhlem záběru 15°, což je pro náš model letadla s cestovní rychlostí cca 80 km/h nevyhovující.

Počítačové vidění

Tato metoda získání informací o překážce v reálném čase pomocí zpracování obrazových dat z kamery je velice nadějná. Kamery jsou dostatečně malé a lehké pro použití na modelu letadla. Již proběhly výzkumy na toto téma s různými výsledky, např. [9] a [10], avšak autor není dostatečně obeznámen s problematikou počítačového vidění a řešení by rozsahem odpovídalo spíše disertační práci.

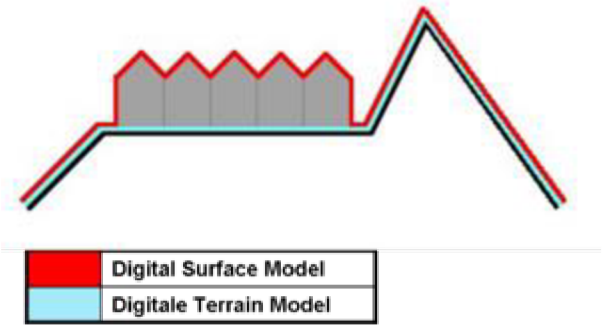
Digitální model terénu

Pro digitální reprezentaci terénu byl zvolen digitální modelu terénu (DTM), jenž bude doplněn databází překážek. Systémy využívající DTM nazývané obecně Terrain awareness and warning system (TAWS) se používají běžně u komerčních letadel. Tyto však poskytují pilotovi pouze doporučení k úhybnému manévru, a to bez automatického zásahu do letu.

Protože APM nemá dostatečnou paměťovou kapacitu pro uložení databáze terénu a také nedisponuje dostatečným výpočetním výkonem pro analýzu dat z databáze překážek a senzorů v reálném čase, byla zvolena varianta využití Android zařízení k výpočtům a uložení dat. Za vstupní informace poslouží Android aplikaci:

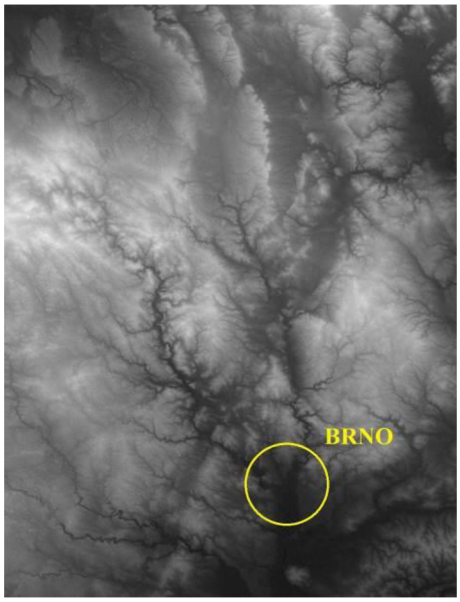
- DTM a databáze překážek,
- data ze senzorů modelu letadla, posílaná bezdrátově do Android zařízení – např. GPS pozice, rychlost vůči zemi, atd.

Ideální variantou by bylo použít model povrchu země (DSM), zahrnující i překážky, jako např. vegetaci, budovy, mosty. Bohužel, takový model je nabízen pouze komerčně firmou Intermap, která data získala přelétnutím většiny pevniny svými speciálně upravenými letadly s laserovým skenerem terénu [11]. Tudíž je potřeba se spokojit s modelem terénu, absentujícím překážky.



Obrázek 2.11: Digital Surface Model (DSM) vs. Digital Terrain Model (DTM). Zdroj: [12]

Nejpodrobnějším a zároveň bez poplatku veřejně dostupným modelem terénu je ASTER GDEM, který je produktem Japonského METI a NASA Spojených států amerických. Rozlišení tohoto modelu je celosvětově 1 úhlová vteřina, odpovídající přibližně 30 m. Referenčním geoidem je World Geodetic System z roku 1984 (WGS84), zpřesněným o odchylky dle Earth Gravitational Model z roku 1996 (EGM96). Dlaždice zahrnující vždy 1 x 1 stupeň zeměpisné šířky/délky jsou distribuovány ve formátu GeoTIFF a ke stažení jsou k dispozici na [13].



Obrázek 2.12: ASTER GDEM dlaždice N49°E16° - N50°E17°. Zdroj: [13]

GeoTIFF je formát vycházející z rozšiřitelného TIFF 6.0 formátu. Umožňuje jednotlivým pixelům obrázku přiřadit nadmořskou výšku. Ze znalosti zeměpisných souřadnic výchozího bodu v levém horním rohu lze vypočítat zeměpisné souřadnice každého pixelu.

2.6.2 Databáze překážek

Pro uložení překážek vyskytujících se nad terénem byl zvolen formát AIXM, který byl vytvořený za účelem sdílení leteckých dat mezi řízeními letového provozu (ATM) v rámci vyvíjených programů SESAR (Evropa) a NextGen (USA). AIXM je XML schéma, které vychází ze schématu Geographical Markup Language (GML). Umožňuje uložení pozice a tvaru objektů důležitých pro letectví, jako např. vzletových a přistávacích drah, majáků a také právě překážek. Konceptuální model AIXM obsahuje entity potřebné pro reprezentaci překážky. Všechny překážky (pevné, pohyblivé, dočasné i stálé) mohou být reprezentovány entitou *VerticalStructure*.

Pro názornost je v příloze 1 uveden příklad popisu jednoduché překážky typu anténa v AIXM XML.

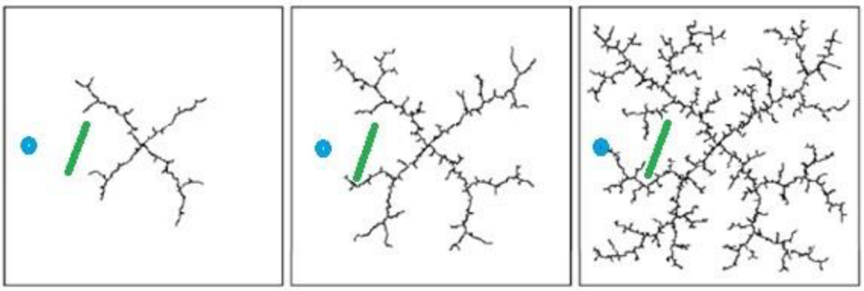
2.7 Algoritmus vyhnutí

Z existujících algoritmů pro hledání trasy v prostoru s překážkami byl vybrán algoritmus RRT*. RRT* [14] vychází z algoritmu Rapidly-exploring random tree (RRT) [15], který prohledává doménu možných bodů trasy. RRT vytváří strom s kořenem v počátečním bodě, typicky aktuální pozici letadla, a následně generuje větve v náhodně vybraných směrech, dokud nedojde k cílovému bodu. Jakmile je cíl nalezen, algoritmus se postupně vrací zpět ke kořenu a získá tím nalezenou trasu.

RRT* je oproti RRT rozšířen tak, že nekončí, nalezne-li trasu k cíli, ale pokračuje hledáním další, optimální trasy, a to dokud neprohledá celý stavový prostor nebo dokud nevyprší časové omezení hledání. I přes rozšíření jde o vcelku jednoduchý algoritmus. Nebere například v potaz rychlost větru, jako to dělá například algoritmus Modified RRT [16]. Cílem ale není použít algoritmus zohledňující co největší množství veličin, jako spíše nalézt jednoduchý a výpočetně nenáročný algoritmus, fungující jako důkaz konceptu.

Pro implementaci algoritmu RRT* v aplikaci Skydog Controller je možné využít knihovnu Open Motion Planning Library (OMPL). Tato knihovna je napsaná v C++ a tudíž je potřeba ji přilinkovat pomocí JNI, podobně jako knihovnu MAVLink, viz kapitola 2.3.2.

Podobný přístup použití algoritmu RRT* a OMPL knihovny je dokumentován v práci [17], jejíž autor nabádá čtenáře k pokračování jeho práce a zlepšení jeho výsledků.



Obrázek 2.13: Větvení stromu náhodnými směry v algoritmu RRT. Střed – výchozí pozice, zelený objekt – překážka, modrý objekt – cíl, ke kterému algoritmus hledá bezkolizní cestu. Zdroj: [17]

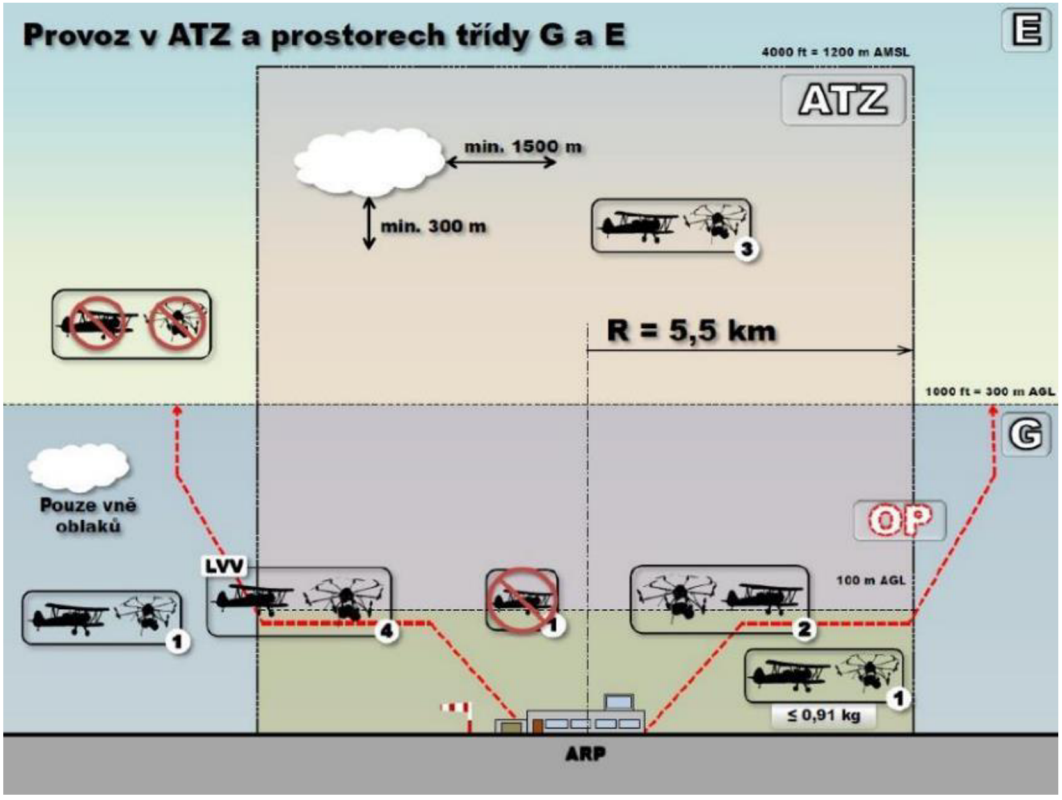
2.8 Legislativní podmínky pro UAV

Provoz UAV v České republice upravuje Doplněk X – Bezpilotní systémy leteckého předpisu L2 – Pravidla létání. Dále je třeba se řídit směrnicí CAA/S-SLS-010-n/2012 - Postupy pro vydání povolení k létání letadla bez pilota vydanou Úřadem pro civilní letectví (ÚCL).

V kontextu legislativního rámce České republiky se za bezpilotní letadla považují všechna bezpilotní letadla s výjimkou modelů letadel s maximální vzletovou hmotností nepřesahující 20 kg. Avšak model letadla je „letadlo, které není schopné nést člověka na palubě, je používané pro soutěžní, sportovní nebo rekreační účely, není vybaveno žádným zařízením umožňujícím automatický let na zvolené místo“. Tudíž je třeba Skydog považovat za bezpilotní letadlo, ne model letadla.

Důležité body pro provádění letů UAV Skydog vycházející z Doplňku X leteckého předpisu L2:

- Dle ust. 7.1 je možné létat v
  - letovém prostoru třídy G tj. do 300 m nad zemí,
  - ATZ a AFIS po předchozí dohodě s majitelem nebo provozovatelem letiště.
- Dle tabulky 1 k ust. 16 Doplňku X je pro výdělečné, experimentální, výzkumné účely použití UAV jakékoliv hmotnosti mj. požadováno
  - vyřízení povolení k létání bezpilotního letadla platné 1 rok,
  - evidence letadla a jeho pilota,
  - doložení provozní příručky UAS.



Obrázek 2.14: Provoz v ATZ a prostorech třídy G a E. Zdroj: [18]

Legenda k obrázku 2.14:

Modely letadel s maximální vzletovou hmotností do 20 kg

Bezpilotní letadla (např. Skydog)

1 Lety bez koordinace

2, 3 Splnění podmínek provozovatele letiště a koordinace s letištní informační službou AFIS

4 Souhlas/povolení ÚCL

3 Implementace Android aplikace

3.1 Plánování

3.1.1 Časový plán

Před začátkem vývoje autor dekomponoval zadání na dílčí úkony, na jejichž základě sestavil časový plán v programu Microsoft Project. Tento původní plán je uveden v příloze 2. Časový odhad jednotlivých kroků však neodpovídal následné době realizace a již po několika krocích se autor dostal časově mimo tento plán. Mnoho z naplánovaných úkolů nebylo ani započato.

3.1.2 Analýza rizik

Protože jde o projekt na několik měsíců, je vhodné se zaměřit na rizikové situace, které se mohou v průběhu projektu vyskytnout. Preventivní řešení bylo nalezeno pro následující rizika.

1. Selhání hardware
   - *Popis*: Komponenty počítače použitého pro vývoj mohou selhat.
   - *Dopad*: Pozastavená možnost pokračovat na vývoji. Hrozí vypracování již hotových částí projektu znovu.
   - *Prevence*: Pravidelné zálohování. Zabezpečení dat pomocí RAID 1 – zrcadlení v diskovém poli, viz kapitola 3.1.3. Využití verzovacího systému, viz kapitola 3.1.4.
2. Nepřehledný zdrojový kód
   - *Popis*: Komentování zdrojového kódu nedostatečným způsobem. Nedodržení jednotného stylu zápisu programu.
   - *Dopad*: Čtenář, včetně samotného autora, má problémy orientovat se v kódu. Je obtížné navázat na dosavadní kód a pokračovat ve vývoji.
   - *Prevence*: Zapisování komentářů ve standardizované formě pro možnost automatického generování dokumentace zdrojového kódu. Vytvoření a dodržování jednotného stylu zápisu kódu, viz kapitola 3.1.6.
3. Podcenění HW vybavení pro vývoj
   - *Popis*: Nedostatečný výkon HW a procesor nepodporující akceleraci Android emulátoru (Android Virtual Device). Aplikaci je sice možné testovat na reálném zařízení, ale pro otestování běhu na různých typech/velikostech Android zařízení je potřeba emulátoru.
   - *Dopad*: Při současném spuštění vývojových nástrojů (Android Development Tools, emulátor Android zařízení, atd.) a prohlížeče internetu je pravděpodobné vyčerpání 4 GB RAM. S procesorem bez akcelerace Android emulátoru je použití emulátoru značně limitované.

- *Prevence*: Použití stroje s minimálně 6 GB RAM a procesorem podporujícím Intel Hardware Accelerated Execution Manager (Intel HAXM).

3.1.3 Úložný server

Pro pravidelné zálohy byl zprovozněn domácí server se dvěma pevnými disky nastavenými pro zápis do pole RAID 1, tzn. na oba disky jsou zapisovány identické informace zároveň. Při selhání jednoho z disků jsou data stále dostupná na disku druhém. Nastaveno je nejen zálohování dat souvisejících s vyvíjenou aplikací, ale i zálohování obrazu operačního systému s nainstalovanými a nastavenými vývojovými nástroji.

3.1.4 Verzovací systém

Použití verzovacího systému při vývoji umožňuje uložení dílčích změn s možností vrátit se k jakékoli z přechozích verzí. Seznam takových změn lze také následně použít k identifikaci původního záměru, pokud je kód nedostatečně komentovaný.

Mezi nejpoužívanější verzovací systémy patří SVN a Git. SVN má obecně více nevýhod, mezi největší patří použití takzvaného repozitáře, což je centralizované úložiště představující zvýšené riziko ztráty dat v případě jeho poškození. Další nevýhodou je citlivost na narušení integrity souboru. Pokud je byť jediná verze souboru v repozitáři poškozena, například kvůli vadnému sektoru na disku, všechny následující verze souboru se stanou nečitelné. Git tyto problémy řeší, avšak autor nakonec zvolil SVN z důvodu důkladné znalosti tohoto systému. Na serveru popsaném v kapitole 3.1.3 byl zprovozněn SVN server, díky čemuž je repozitář uložen na disku v RAID 1 a tím se minimalizují rizika uvedená výše.

3.1.5 Vývojové prostředí

Protože jazykem Android většiny aplikací je Java, existuje mnoho vývojových prostředí, ve kterých lze aplikaci vyvíjet. Google a mnoho návodů však vysvětluje základy programování aplikace v Android Developer Tools (ADT), což je doplněk pro vývojové prostředí Eclipse. Oproti běžným prostředím pro jazyk Java nabízí mimo jiné integrované a přednastavené nástroje z Android SDK, jako například kompilátor, debugger, atd. Před rokem, v květnu 2013, Google představil Android Studio. Jde o nadstavbu IntelliJ IDEA, populárního IDE pro vývoj v Javě. Android Studio má za úkol postupně nahradit ADT v pozici doporučovaného vývojového prostředí pro Android, ovšem nyní je stále ve fázi testování uživateli a chybí mu mnoho funkcí současného ADT. Proto bylo zvoleno pro vývoj aplikace ověřené ADT.

3.1.6 Styl zápisu kódu

Pro přehlednost a jednotnost zdrojového kódu je třeba dodržovat soubor pravidel, nazývaný styl zápisu kódu. Inspirací byl styl povinný pro vývojáře operačního systému Android [19] a také styl pro jazyk Java doporučený firmou Google [20]. Výsledný styl zápisu, kterého bylo dodržováno při implementaci, je popsán v příloze 3.

3.2 Detaily aplikace SkyControl

Níže jsou zmíněny parametry vyvíjené aplikace, definované v souboru *AndroidManifest.xml*, jednom z mála povinných pro jakýkoliv projekt Android aplikace.

- Jméno aplikace: SkyControl
  Původně navrhovaný název Skydog Controller (kapitola 2.4) byl zkrácen z důvodu plánovaného využití aplikace i pro jiné modely letadel, než pouze Skydog.
- Minimální SW požadavky: Android 4.0 a novější
- Minimální HW požadavky: Zařízení podporující USB OTG
- Doporučené rozlišení zařízení: 720 x 1280 pixelů a vyšší

Za licenci aplikace byla zvolena licenci svobodného software *Apache License, Version 2.0*. Ta poskytuje uživateli svobodu v užívání software k jakémukoli účelu, k jeho modifikaci, k distribuci modifikovaných verzí, a bez obav z jakýchkoliv poplatků. Informace o této licenci je uvedena na začátku každého zdrojového souboru.

3.2.1 Omezení Android API

Je důležité zmínit alespoň v krátkosti rozdíly mezi Android API a Java API pro pochopení zvolených řešení popsaných v následujících kapitolách. Většina aplikací pro Android je napsána v jazyce Java, ale mezi Android API a Java API jsou významné rozdíly.

Pro spuštění kódu napsaného v klasické Oracle Javě je takový kód překládán do Java mezikódu (Java bytecode), který je následně spouštěn ve virtuálním stroji nazvaném Java Virtual Machine (JVM). JVM je dodáván se sadou standardních knihoven, která je nazývána Java API.

Android OS však neobsahuje JVM. Java bytecode se nemá kde vykonat. Místo toho, Java bytecode je překompilován do proprietárního formátu, který je spouštěn v Dalvik, speciálním virtuálním stroji vytvořeném pro Android. Dalvik byl navržen tak, aby zabíral co nejméně místa a proto zahrnuje pouze podmnožinu knihoven Java API.

Z výše uvedeného vyplývá, že knihovny třetích stran napsané v Javě mohou, ale nemusí být použitelné i v Android OS. Záleží na tom, jakých balíčků z Java API vývojář knihovny použil.

3.2.2 MAVLink

Knihovna pro komunikaci přes protokol MAVLink

Knihovna v jazyce C++, představená v kapitole 2.3.2, nakonec v aplikaci nebyla využita. Místo ní byla použita knihovna v jazyce Java, převzatá z open-source projektu DroidPlanner, vedeného A. Benemannem [21]. DroidPlanner je Android aplikace s podobnou funkcionalitou požadovanou od této diplomové práce a byla při vývoji aplikace SkyControl důležitým zdrojem inspirace.

Obdobně jako u původní knihovny v C++ je základem pro vygenerování zdrojových souborů knihovny skript napsaný v jazyce Python, který zpracovává XML definice MAVLink zpráv. Místo do C++ tříd je však převádí do Java tříd. Každá jedna MAVLink zpráva tvoří jednu

Java třídu implementující metody *pack()* a *unpack()*. Tyto mají za úkol převést bajty přijaté zprávy do datových typů Javy a naopak, a to podle obsahu dané zprávy definovaného v XML.

Přijímané údaje

Tabulky 3.1 a 3.2 shrnují údaje, které aplikace SkyControl přijímá od připojeného autopilota pomocí protokolu MAVLink. Výchozí frekvencí odesílání údajů z tabulky 3.1 autopilotem je 1 Hz. Tuto frekvenci je možné v nastavení aplikace zvýšit. Údaje z tabulky 3.2 jsou posílány pouze při změně.

Nadmořská výška	Vertikální rychlost
Úhel podélného sklonu (pitch)	Traťová rychlost (GS)
Úhel příčného náklonu (roll)	Indikovaná vzdušná rychlost (IAS)
Úhel vybočení (yaw)	Puls (Heartbeat)
GPS pozice	Traťový bod režimu AUTO, ke kterému autopilot aktuálně naviguje letadlo
Počet zachycených GPS satelitů	

Tabulka 3.1: Údaje přijímané periodicky

Režim autopilota
Autopilot je aktivovaný (Armed)
Autopilot ztratil signál (Failsafe)

Tabulka 3.2: Údaje přijímané při jejich změně

Nadmořskou výšku autopilot APM počítá z barometrické relativní výšky tak, že ji kalibruje dle nadmořské výšky získané z modulu GPS. GPS jako systém poskytuje údaje o nadmořské výšce s přesností v řádu desítek metrů, tudíž i výška, kterou posílá APM, může být velmi nepřesná. APM však interně pro účely navigace v režimu AUTO používá pouze barometrickou výšku.

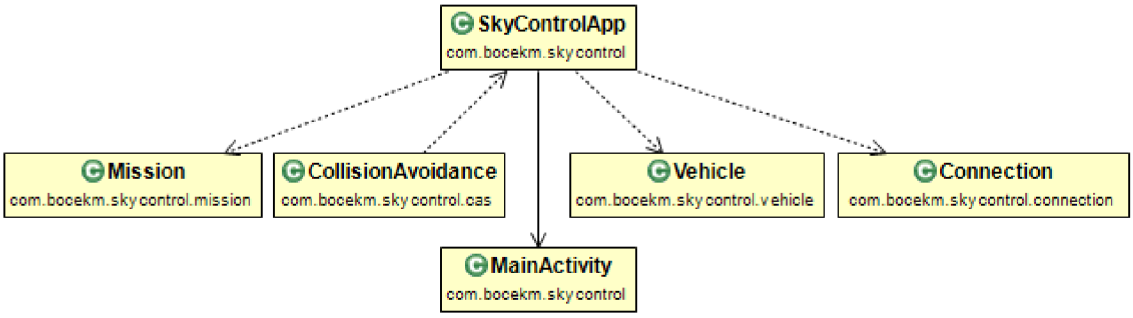
Samotný USB modul 3DR Radio uměle vkládá do sériové komunikace MAVLink pakety obsahující sílu a míru šumu signálu. Ve firmwaru je od výrobce nastaveno vkládání těchto paketů v případě detekce probíhající MAVLink komunikace.

3.2.3 Struktura aplikace

Nejvýše postavenou třídou v hierarchii dědičnosti je *SkyControlApp*, která rozšiřuje Android třídu *Application*. Instance této třídy je vytvořena jako první po spuštění aplikace a má za úkol inicializovat tzv. singletony, tedy třídy fungující dle návrhového vzoru Singleton. Úkolem tohoto vzoru je zajistit právě a pouze jednu instanci třídy za běhu aplikace. Hlavní výhodou singletonů je jejich globální přístupnost – jsou přístupné z jakékoli třídy aplikace. Těmito singletony jsou v aplikaci třídy *Connection*, *Mission*, *Vehicle* a *CollisionAvoidance*.

Účelem singletonu *Vehicle* je udržovat veškeré dostupné údaje o připojeném letadle. Singleton *Mission* spravuje aktuální misi, spustitelnou v módu Waypoint Follower. Singleton *Connection* má na starosti spojení aplikace s letadlem. Po připojení USB radio modulu spustí službu na pozadí, která začne vyhledávat druhý radio modul připojený do APM. Instance singletonu *CollistionAvoidance* obsahuje informace o překážkách a funkce pracující nad těmito informacemi. Když je připojen autopilot, pak kontroluje při každém příjmu pozice letadla, zda mu nehrozí kolize s překážkou. Struktura jednotlivých singletonů je ilustrována v přílohách 5 až 7.

Po inicializaci singletonů přichází na řadu zobrazení uživatelského rozhraní, což má na starosti jediná Android aktivita *MainActivity*, která však inicializuje pouze vrchní lištu aplikace (tzv. Action Bar) a menu aplikace. Zpracování zbylého uživatelského rozhraní deleguje na množství Android fragmentů. Struktura třídy *MainActivity* je ilustrována v příloze 4.



Obrázek 3.1: Hlavní třídy aplikace

3.3 Reprezentace překážek

Protisrážkový systém pracuje s následujícími typy překážek.

- Překážky nad terénem definované kolmými n-bokými hranoly.
- Digitální model terénu samotného.

3.3.1 Reprezentace n-bokého hranolu

AIXM

Původní idea využití formátu AIXM XML, jak byla popsána v kapitole 2.6.2, nebyla nakonec využita. Nebyla totiž nalezena knihovna v jazyce Java, která by dokázala zpracovat XML podle AIXM schématu. Sice existuje Java knihovna AIXM-J, která však byla opuštěna již ve stadiu začátku vývoje v roce 2009. AIXM schéma obsahuje stovky entit konceptuálního modelu. Knihovna AIXM-J jich dokáže z XML přečíst sotva pár desítek a překážky mezi nimi nejsou.

GML

Co se týká reprezentace překážek v AIXM, tak jde pouze o nadstavbu GML. GML je formát umožňující definovat složité geometrické tvary v rovině (ne však v třírozměrném prostoru)

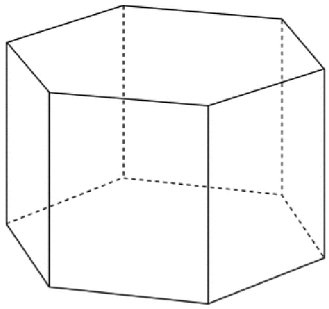
s projekcí do souřadnicového systému Země a je využívaný zejména v kartografické oblasti. AIXM dokáže těmto geometrickým tvarům (n-úhelník, kružnice) přiřadit pro účely reprezentace překážky nadmořskou výšku. Autor tedy vytvořil jednoduché schéma, vycházející z původního AIXM schématu [22], kde ponechal pouze element *ElevatedSurface*, který rozšiřuje GML element *Surface* o nadmořskou výšku. Podle takového schématu je možné zpracovat XML použitím Java knihovny GeoTools tak, že lze následně jednoduše pracovat s geometrickými tvary definovanými v GML jako s objekty. Avšak zmíněná knihovna GeoTools bohužel není využitelná pro OS Android z důvodu popsaném v kapitole 3.2.1, proto bylo přistoupeno použití vlastního formátu XML pro definování překážky.

XML

Pro definování překážky byl nakonec použit vlastní formát XML. Každá překážka je definovaná v XML:

- mnohoúhelníkem, jenž představuje ortogonální projekci trojrozměrné překážky na zemský povrch, a
- nadmořskou výškou překážky.

Tyto údaje tvoří z každé překážky n-boký hranol. Obrázek 3.3 ilustruje tvar 6-bokého hranolu.



Obrázek 3.3: Příklad kolmého 6-bokého hranolu. Zdroj: [23]

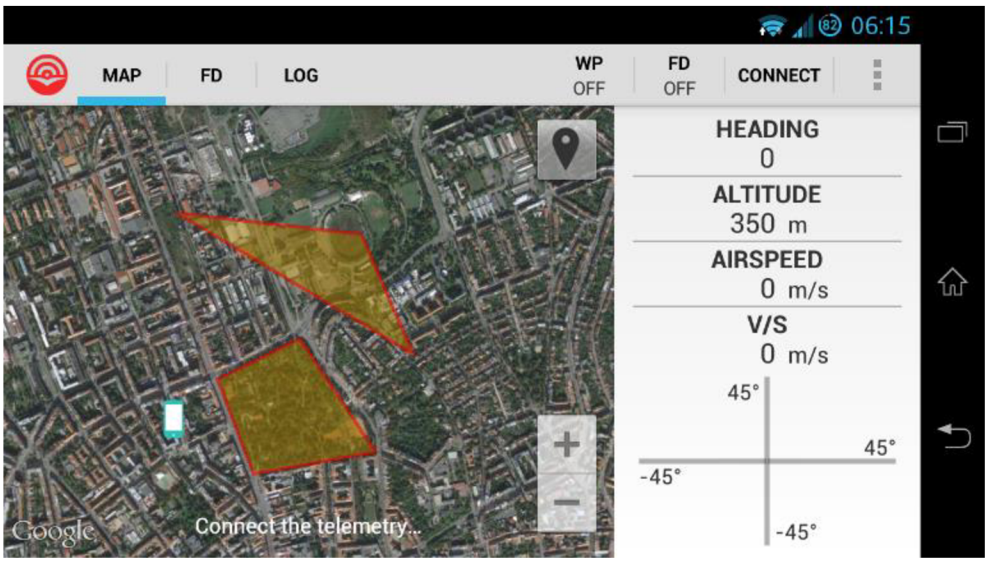
```
<?xml version="1.0" encoding="utf-8"?>
<obstacles>
    <obstacle>
        <coords>
             49.207568, 16.604625
             49.205009, 16.613187
             49.204224, 16.606578
             ...
        </coords>
        <elevation>400</elevation>
    </obstacle>
    ...
</obstacles>
```

Příklad 3.1: Jednoduché XML s definovanou jedinou překážkou, reprezentovanou 3-bokým hranolem s nejvyšším vrcholem ležícím v nadmořské výšce 400 m

Formát XML lze popsat na příkladu 3.1. Kořenový element *<obstacles>* obsahuje jednotlivé překážky v elementech *<obstacle>*. Celkový počet překážek definovaných elementem *<obstacle>* v XML je neomezený.

Do elementu *<coodrs>* je zadáván seznam dvojic *zeměpisná šířka, zeměpisná délka* v systému WGS84, oddělených od sebe bílým znakem. Tyto dvojice reprezentují zeměpisné souřadnice vrcholů mnohoúhelníku. Povinné jsou tři a více zadaných souřadnic. Takto definovaný mnohoúhelník je v aplikaci převeden do objektu třídy *Polygon* knihovny JTS.

Hodnota elementu *<elevation>* představuje nadmořskou výšku nejvyššího bodu překážky.



Obrázek 3.4: Ukázka zobrazení překážek v aplikaci SkyControl

Načítání překážek z XML je v aplikaci implementováno metodou *parseObstaclesFromXml* třídy *XmlParser* z balíčku *com.bocekm.skycontrol.cas*.

3.3.2 Digitální model terénu

Jak již bylo zmíněno v kapitole 2.6.1, pro reprezentaci modelu terénu byl zvolen formát GeoTIFF. Naneštěstí, Java knihovny, které umí pracovat s GeoTIFF, konkrétně GeoTools a Apache Commons Imaging, není možné pro Android aplikaci použít (viz kapitola 3.2.1). Tudíž bylo potřeba implementovat vlastní kód pro zpracování GeoTIFF dat.

Uložení modelu terénu v aplikaci

Soubory, které nesouvisí přímo s během Android aplikace, jako jsou obrázky, video, nebo XML, je potřeba uložit do složky *assets/* Android projektu. Po zkompilování se Android projekt balí do instalačního balíčku ve formátu Android Package File (APK).

Ještě před zabalením se většina souborů projektu zkomprimuje, mimo ty, které jsou již zkomprimovány (JPEG, ZIP, atd.), aby byl výsledný balíček co nejmenší. Testovací soubor

ASTER GDEM dlaždice N49°E16°-N50°E17° je v projektu uložen jako *assets/digital_elevation_model.tif* s velikostí cca 26 MB a po kompresi zabírá v APK cca 10 MB.

Abychom mohli z GeoTIFF na požádání získat nadmořskou výšku na daných souřadnicích, je potřeba mít možnost přistupovat ke konkrétní pozici v souboru bez čtení celého souboru vždy znovu. Držet za běhu aplikace celý obsah souboru v paměti by sice bylo proveditelné, ale zároveň pomalé a neefektivní.

Kvůli kompresi lze za běhu aplikace číst soubory z *assets/* pouze jednosměrně sekvenčně pomocí Java třídy *InputStream*. Tedy, k přečtení *n*. bajtu je potřeba nejdříve přečíst předchozích *n-1* bajtů a poté se již nelze vrátit zpět k bajtům na pozici $<n$. K překonání tohoto omezení byl zvolen přístup zkopírování *digital_elevation_model.tif* v dekomprimované podobě na externí úložiště Android zařízení, tzn. buď na emulovanou, nebo na skutečnou SD kartu, odkud se již dá k souborům přistupovat pomocí Java třídy *RandomAccessFile*. Ta umožnuje číst z libovolné pozice v souboru.

Formát GeoTIFF

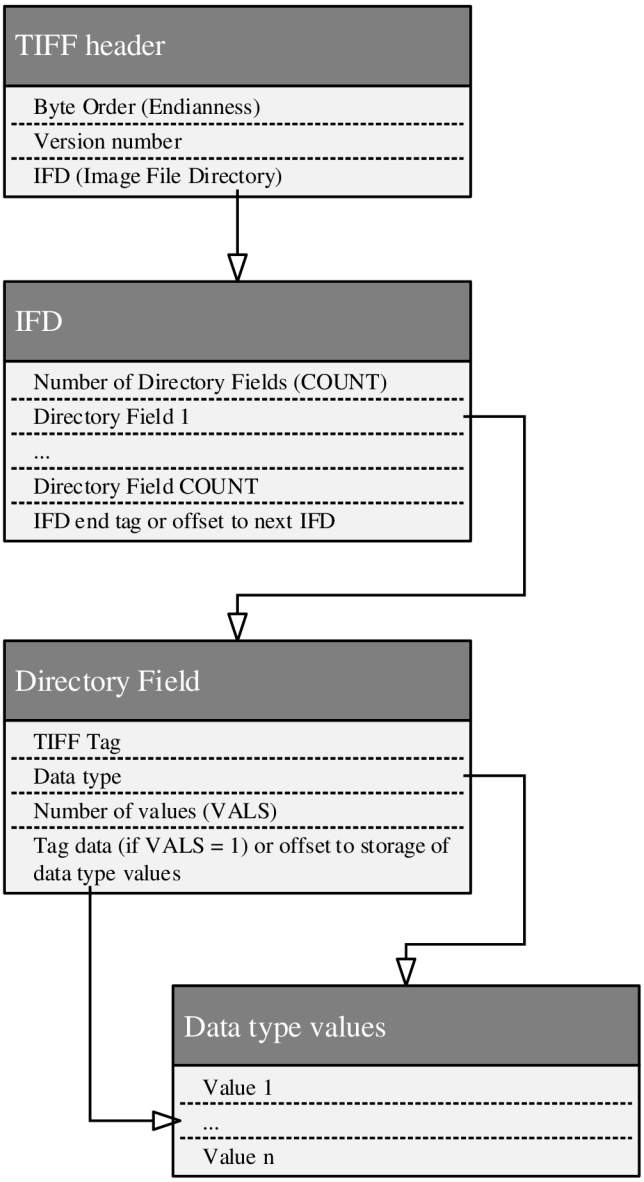
GeoTIFF je rozšíření formátu Tagged Image File Format (TIFF), který patří díky své rozšiřitelnosti k nejuniverzálnějším grafickým formátům. Toto rozšíření spočívá v použití speciálních tagů registrovaných pro GeoTIFF, několika z mnoha TIFF tagů uložitelných v metadatech obrázku. Tyto tagy, popsané detailněji v následujících kapitolách, definují mimo jiné:

- použitý souřadnicový systém,
- souřadnice počátku, tedy levého horního rohu obrázku,
- jednotku, ve které jsou uloženy souřadnice počátku.

GeoTIFF je zpětně kompatibilní s TIFF a je tedy zobrazitelný v jakémkoli prohlížeči obrázků. Konkrétně ASTER GDEM GeoTIFF se zobrazuje jako obrázek ve stupních šedi. Je to tím, že každý pixel má přiřazenou dvoubajtovou hodnotu reprezentující nadmořskou výšku (0 až 65,535), kde 0 je reprezentována jako černá barva a nejvyšší hodnota v daném souboru je reprezentována jako bílá, viz obrázek 2.12.

TIFF metadata

Pro získání nadmořské výšky konkrétních souřadnic, je prvně třeba přečíst metadata souboru. Na obrázku 3.5 je vidět struktura metadat formátu TIFF 6.0, což je aktuální verze, představená v roce 1992. TIFF hlavička (header) začíná na prvním bajtu souboru. Z těchto metadat jsou důležité TIFF tagy uvedené v tabulce 3.3.



Obrázek 3.5: Metadata formátu TIFF

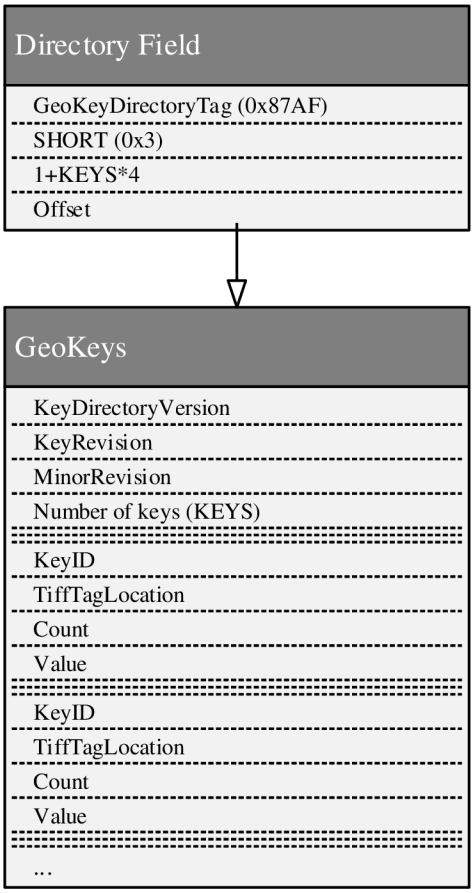
TIFF Tag	Popis hodnoty	Hodnota pro ASTER GDEM
0x0100	Počet pixelů na šířku	3601
0x0101	Počet pixelů na výšku	3601
0x0102	Počet bajtů jedné hodnoty	2
0x0111	Offsety pruhů	Offset na seznam 3601 offsetů
0x0115	Počet hodnot na pixel	1
0x0116	Počet řad pixelů v jednom pruhu	1

TIFF Tag	Popis hodnoty	Hodnota pro ASTER GDEM
0x0117	Počet bajtů v jednom pruhu	7202
0x87AF	GeoKeyDirectoryTag	Obsah tagu popsán níže
0x830E	Měřítko pixelu, tedy kolik stupňů zeměpisné šířky, délky představuje jeden pixel	$(2,\overline{77} * 10^{-4}, 2,\overline{77} * 10^{-4})$
0x8482	Souřadnice počátku – levý horní roh obrázku	např. (50.00013888888889, 15.99986111111111)

Tabulka 3.3: Seznam důležitých TIFF tagů

GeoTIFF metadata

Po přečtení TIFF metadat je třeba přečíst data klíčů, tzv. GeoKeys, uložených pod TIFF tagem GeoKeyDirectoryTag, který má strukturu popsanou UML diagramem na obrázku 3.6. Na začátku offsetu, uloženého v GeoKeyDirectoryTag, je hlavička, která mimo jiné obsahuje počet uložených GeoKeys („Number of keys“). Hned za hlavičkou již leží jednotlivé klíče.

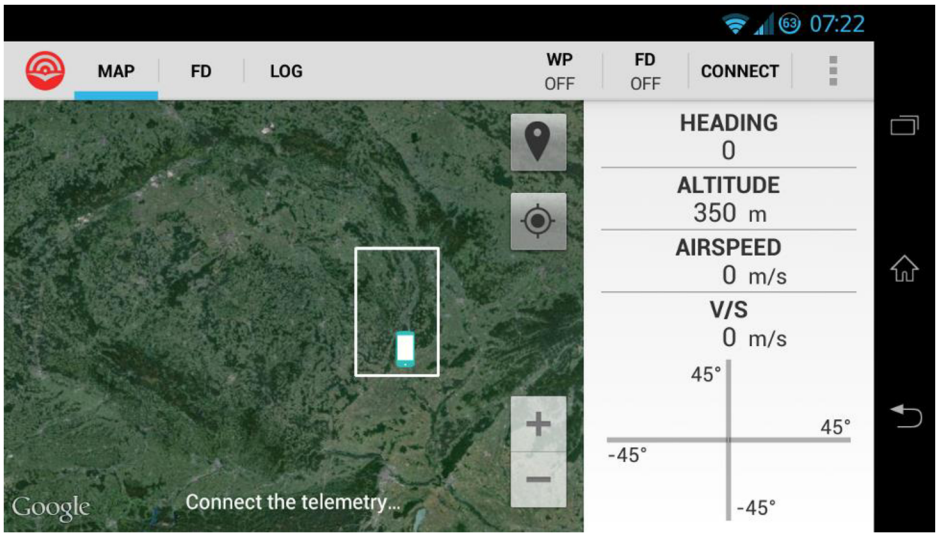


Obrázek 3.6: Metadata formátu GeoTIFF

Jednotlivých klíčů GeoKeys existují desítky. Pro naše účely jsou důležité klíče uvedené v tabulce 3.4.

KeyID	Popis hodnoty	Hodnota pro ASTER GDEM
0x0401	Reprezentace jednoho pixelu (plocha nebo bod)	0x0001 (pixel reprezentuje plochu)
0x0800	Souřadnicový systém	0x10E6 (WGS84)
0x0806	Jednotka, ve které jsou uloženy souřadnice počátku	0x238E (stupeň)

Tabulka 3.4: Popis několika důležitých klíčů GeoKeys

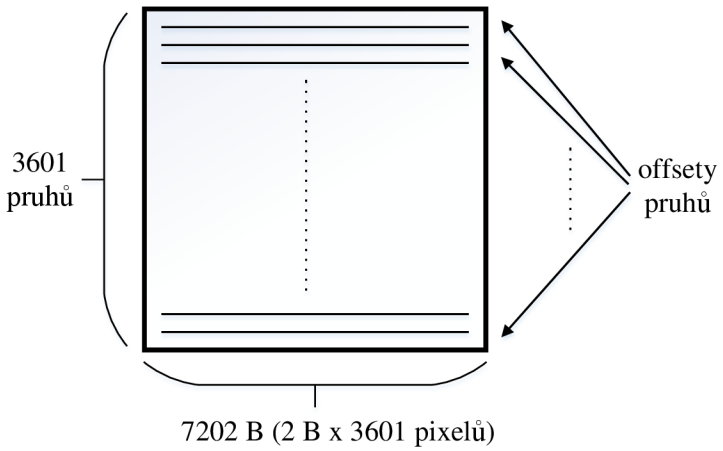


Obrázek 3.7: Zobrazení krajních souřadnic GeoTIFF na základě přečtených metadat

TIFF pruhy v ASTER GDEM GeoTIFF

Pro získání hodnoty jednotlivých pixelů je zapotřebí pochopit koncept TIFF pruhů. Celý obrázek je horizontálně rozdělen na pruhy. Pro každý obrázek TIFF může být nastavení pruhů jiné, avšak pro ASTER GDEM GeoTIFF platí, že každá jedna řada pixelů obrázku odpovídá jednomu pruhu. Ukazatele na začátek jednotlivých pruhů (offsety pruhů) jsme přečetli z metadat. Protože obrázek má na výšku 3601 pixelů, máme také 3601 offsetů. Jde o pozice prvního bajtu pruhu počítané od začátku souboru.

Protože hodnota jednoho pixelu, tedy nadmořská výška, je uložená na dvou bajtech, je velikost jednoho pruhu 7202 bajtů.



Obrázek 3.8: Offsety pruhů

Nadmořská výška na souřadnicích

Při dotazu na nadmořskou výšku na konkrétních zeměpisných souřadnicích je třeba zjistit pozici správného pixelu v obrázku. Z důvodu rychlosti aplikace při spuštění jsou čteny pouze metadata obrázku a ne samotné hodnoty pixelů. Největší položkou z těchto metadat jsou offsety pruhů, což je pole 4B čísel o velikosti 3601, které tedy zabírají v paměti cca 14 KiB. Dle algoritmu 3.1 jsou při dotazu na nadmořskou výšku přečteny ze souboru pouze dva další bajty jakožto hodnota konkrétního pixelu.

Vstup: zeměpisná šířka (*lat*), zeměpisná délka (*lng*)
Známé konstanty:
měřítko zeměpisné šířky/délky (*latScale*/*lngScale*), zeměpisná šířka nejjižnějšího bodu obrázku (*minLat*), zeměpisná délka nejzápadnějšího bodu obrázku (*minLng*), počet pixelů obrázku na výšku (*h*), počet pixelů obrázku na šířku (*w*), počet bajtů pro jeden pixel (*bytes*), pole offsetů pruhů (*stripOffsets*), soubor obrázku (*TIFF*)
Výstup: nadmořská výška v metrech (*elev*)
Postup:

1. $latScaled = \frac{(lat - minLat)}{latScale}$
2. $lngScaled = \frac{(lng - minLng)}{lngScale}$
3. $stripOffsetIndex = h - ceil(latScaled)$
4. $stripOffset = stripOffsets[stripOffsetIndex]$
5. $offsetInStrip = floor(lngScaled) * bytes$
6. $elev = readBytes(TIFF, bytes, stripOffset + offsetInStrip)$

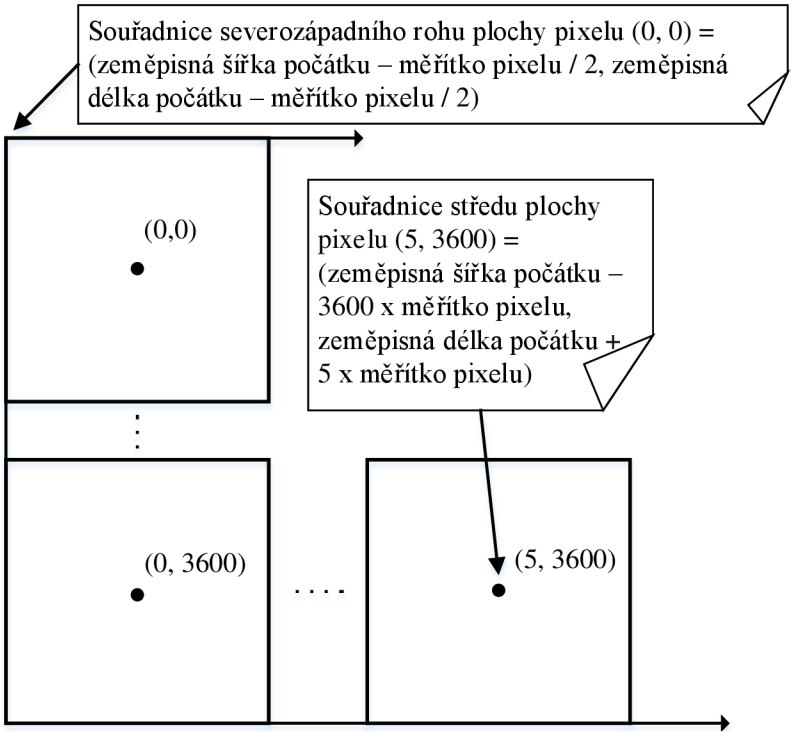
Poznámky:

- *ceil* a *floor* jsou matematické funkce zaokrouhlení nahoru a dolů
- *readBytes(file,bytes,offset)* je funkce pro přečtení *bytes* bajtů na pozici *offset* ze souboru *file*

Algoritmus 3.1: Nadmořská výška na souřadnicích

V GeoTIFF obrázku leží počátek souřadnic v levém horním rohu obrázku a proto se hodnota zeměpisné šířky snižuje směrem na jih k rovníku. To znamená, že nejnižší stupeň zeměpisné šířky leží na nejspodnějším pruhu obrázku. Tento předpoklad omezuje použití algoritmu 3.1 pouze pro ASTER GDEM GeoTIFF s body ležícími na severní polokouli Země.

Protože jeden pixel ASTER GDEM GeoTIFF reprezentuje plochu, a ne bod, hodnota jednoho pixelu obrázku představuje nadmořskou výšku plochy o rozměrech $měřítko\ pixelu * měřítko\ pixelu$, tedy $2,\overline{77} * 10^{-4}$ x $2,\overline{77} * 10^{-4}$ stupňů. To je důvodem použití matematických funkcí zaokrouhlení *ceil* a *floor*. Bez nich by při výpočtu celočíselného indexu pozice pixelu spadal bod v polovině případů do nesprávných sousedních ploch.



Obrázek 3.9: Pixel jako plocha

Implementace algoritmu 3.1 v aplikaci je součástí metody *getElevation* třídy *TiffParser* z balíčku *com.bocekm.skycontrol.cas*.

3.4 Protisrážkový systém

Protisrážkový systém slouží k zábraně srážky letadla s překážkou, ať již s terénem nebo překážkou definovanou hranolem. Protože je použitým jazykem pro pojmenování a komentáře v aplikaci angličtina, je tento systém, představený v kapitole 2.6 pojmenován jako Collision Avoidance System (CAS).

Pokud je v aplikaci CAS zapnutý, pak je při každém příjmu nové polohy letadla přes MAVLink provedena kontrola, zda přepokládaná budoucí poloha letadla nekoliduje s překážkou.

Pokud koliduje, je vyslán signál *CollisionEvent.DANGER_OF_COLLISION* zainteresovaným posluchačům, což jsou třídy zajišťující módy Flight Director a Waypoint Follower. Tyto třídy již samy zařídí úhybný manévr letadla.

3.4.1 Vincentyho souřadnicové algoritmy

V následujících kapitolách bude třeba provádět výpočty nad zeměpisnými souřadnicemi. Nyní si představíme algoritmy, kterých je použito v aplikaci právě pro takové výpočty.

Protože lze matematicky obtížně definovat geoid, který by odpovídal fyzikálnímu tvaru Země, využil T. Vincenty v roce 1975 [24] pro výpočty nad zeměpisnými souřadnicemi rotační elipsoid. Ten je geometrickým tělesem, jehož tvar je blízký tvaru geoidu. Vincentyho algoritmy dokáží získat:

- vzdálenost a azimut pro dvě zadané zeměpisné souřadnice,
- souřadnice druhého bodu pro zadaný azimut a vzdálenost od první bodu.

Inspirací pro implementaci těchto algoritmů v jazyce Java byla verze v jazyce Javascript z [25]. V aplikaci jsou algoritmy implementovány metodami třídy *SkyControlUtils*. Použitým rotačním elipsoidem pro výpočty je referenční elipsoid souřadnicového systému WGS84.

$$vzdálenost\ v\ metrech = distVincenty(souřadnice\ prvního\ bodu, souřadnice\ druhého\ bodu)$$

Funkce 3.1: Vzdálenost mezi dvěma zeměpisnými body

$$azimut\ ve\ stupních = azimVincenty(souřadnice\ prvního\ bodu, souřadnice\ druhého\ bodu)$$

Funkce 3.2: Azimut dvou bodů

$$souřadnice\ nového\ bodu = destVincenty(souřadnice\ původního\ bodu, azimut, vzdálenost\ v\ m)$$

Funkce 3.3: Souřadnice bodu v dané vzdálenosti a s daným azimutem

3.4.2 Výpočet budoucí polohy letadla

Použitý algoritmus pro výpočet budoucí polohy letadla je zcela elementární. Vychází z aktuální polohy, kurzu a rychlosti letadla a získá tak bod před letadlem. V nastavení aplikace je možné zvolit, v jaké vzdálenosti od letadla se má tento bod nacházet. Nastavitelný údaj vzdálenosti je ve vteřinách a výchozí hodnotou je 5 s. Vypočet vzdálenosti v metrech je uveden ve vzorci 3.1.

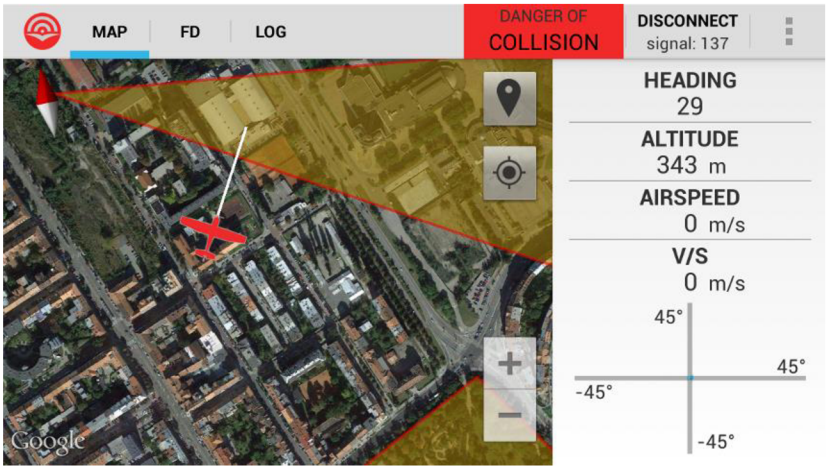
$$vzdálenost\ v\ metrech = vzdálenost\ ve\ vteřinách * aktuální\ traťová\ rychlost$$

Vzorec 3.1: Vzdálenost bodu před letadlem

Pro získání souřadnic budoucí polohy letadla využijeme funkce 3.3, která počítá cílovou souřadnici na základě výchozí souřadnice, kurzu letadla a vzdálenosti.

$$budoucí\ poloha = destVincenty(současná\ poloha, kurz\ letadla, vzdálenost\ v\ metrech)$$

Vzorec 3.2: Budoucí poloha letadla v určité vzdálenosti



Obrázek 3.10: Budoucí poloha letadla je v aplikaci znázorněna koncem bílé čáry před letadlem

3.4.3 Kontrola překážek před letadlem

Jak bylo zmíněno v úvodu kapitoly, pokud je v aplikaci CAS zapnutý, pak při každém příjmu nové polohy letadla aplikace provede kontrolu, zda přepokládaná budoucí poloha letadla nekoliduje s překážkou či terénem. K tomu slouží následující dvě funkce.

Kontrola trati letu

Funkce $zkontrolujTrať$ vrací logickou hodnotu *pravda*, pokud je *nadmořská výška* nižší nebo rovna nadmořské výšce:

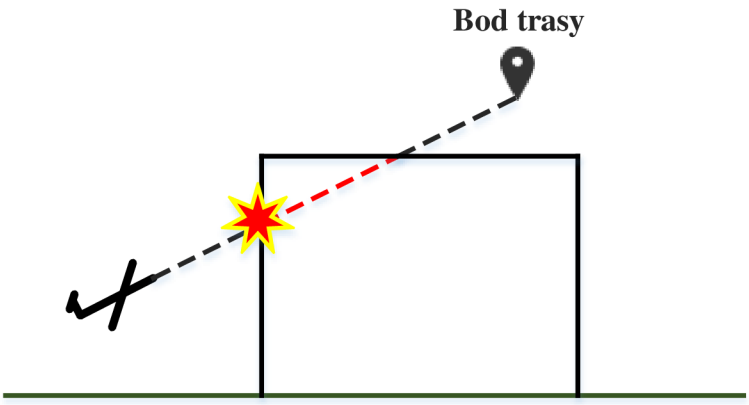
- překážky v případě, kdy je mnohoúhelník, reprezentující tuto překážku, proťat přímkou danou dvěma body, nebo
- překážky v případě, že přímka daná dvěma body leží uvnitř mnohoúhelníku, reprezentující překážku, nebo
- terénu na souřadnicích druhého bodu,

jinak vrací logickou hodnotu *nepravda*.

$$koliduje = zkontrolujTrať(souřadnice\ prvního\ bodu, souřadice\ druhého\ bodu, nadmořská\ výška)$$

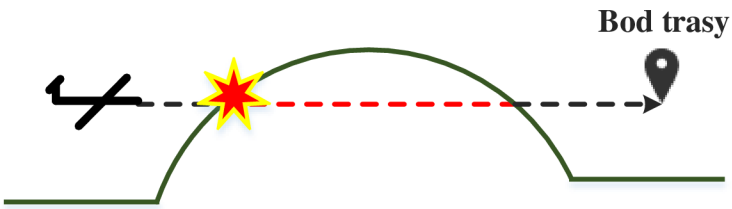
Funkce 3.4: Kontrola, zda je trať letu daná dvěma body bezkolizní

V případě, kdy přímka protíná mnohoúhelník reprezentující překážku, je kontrolována výška pouze konce přímky, zda není nižší či rovna výšce překážky. Rozšířením funkce by mohlo být získání souřadnic protnutí přímky s mnohoúhelníkem a odhadnutí výšky letadla v tomto bodě.



Obrázek 3.11: Nedetekovaná kolize s překážkou

Další možné rozšíření funkce 3.4 představuje přidání kontroly kolize s terénem pro celou délku přímky, a ne pouze jejího koncového bodu.



Obrázek 3.12: Funkce 3.4 kontroluje v případě terénu pouze konečný bod přímky a nedokáže detekovat situaci na obrázku

Funkce 3.4 odpovídá v kódu aplikace metodě *checkForObstacleCollision* třídy *CollisionAvoidance* z balíčku *com.bocekm.skycontrol.cas*.

Kontrola souřadnic

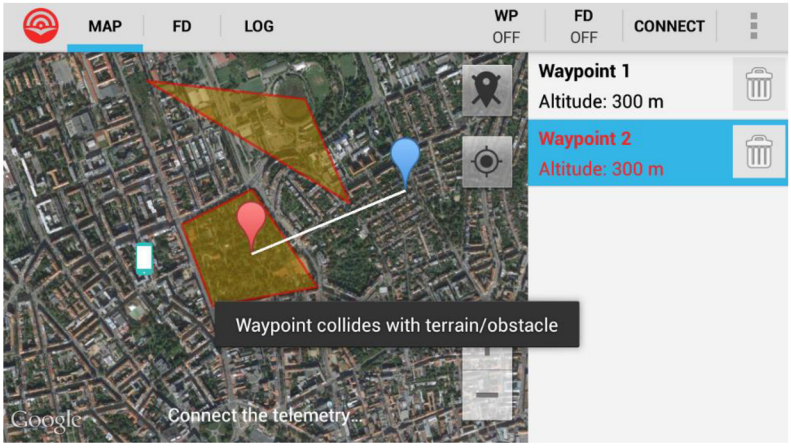
Funkce $zkontrolujBod$ je použita při přidání nového traťového bodu do mise, čímž mj. umožnuje upozornit uživatele na případnou kolizi. Uživatel následně upraví výšku nebo pozici traťového bodu. Vrací logickou hodnotu *pravda*, pokud bod leží:

- pod úrovní terénu, nebo
- pokud leží uvnitř překážky,

jinak vrací logickou hodnotu *nepravda*.

$$koliduje = zkontrolujBod(souřadnice\ prvního\ bodu, nadmořská\ výška)$$

Funkce 3.5: Kontrola, zda bod neleží uvnitř překážky, či pod terénem



Obrázek 3.13: Příklad využití funkce 3.5 při přidávání traťového bodu

Funkce 3.5 odpovídá v kódu aplikace metodě *checkForObstacleCollision* třídy *CollisionAvoidance* z balíčku *com.bocekm.skycontrol.cas*.

Knihovna JTS

Obě funkce uvedené v předchozích kapitolách využívají pro kontrolu kolize s překážkou Java knihovnu Java Topology Suite (JTS). API JTS obsahuje kompletní sadu algoritmů pro Euklidovskou geometrii. Tedy například pro objekty kružnice a přímky umí zodpovědět, jestli mají tyto objekty společné body a jaké to jsou.

Z XML máme mnohoúhelníky, reprezentující překážky, uložené v instancích třídy JTS *Polygon*. Pro zjištění, zda v případě funkce 3.4 přímka, uložená v instanci třídy JTS *LineString*, protíná mnohoúhelník, zavoláme na objektu mnohoúhelníku metodu *intersects(přímka)*. Pro zjištění, zda v případě funkce 3.5 bod, uložený v instanci třídy JTS *Point*, leží v mnohoúhelníku, zavoláme na objektu *Point* metodu *within(mnohoúhelník)*.

3.4.4 Vyhledání bezkolizní trati letu

Pro nalezení bezkolizní trati letu v případě hrozící srážky byl použit algoritmus RRT [15] s jednoduchou optimalizací nalezené bezkolizní trati. Algoritmus RRT se snaží najít z výchozího bodu bezkolizní trať letu k zadanému cílovému bodu, a to v prostoru s definovanými překážkami. Algoritmus byl implementován do dvourozměrného prostoru.

Aktivní vyhnutí se srážce, a tedy využití RRT, je implementováno pouze pro režim autopilota AUTO, tedy v módech Waypoint Follower a Flight Director. V režimu MANUAL je zobrazeno varování o hrozící srážce, ale do letu aplikace nezasahuje.

Konfigurační prostor

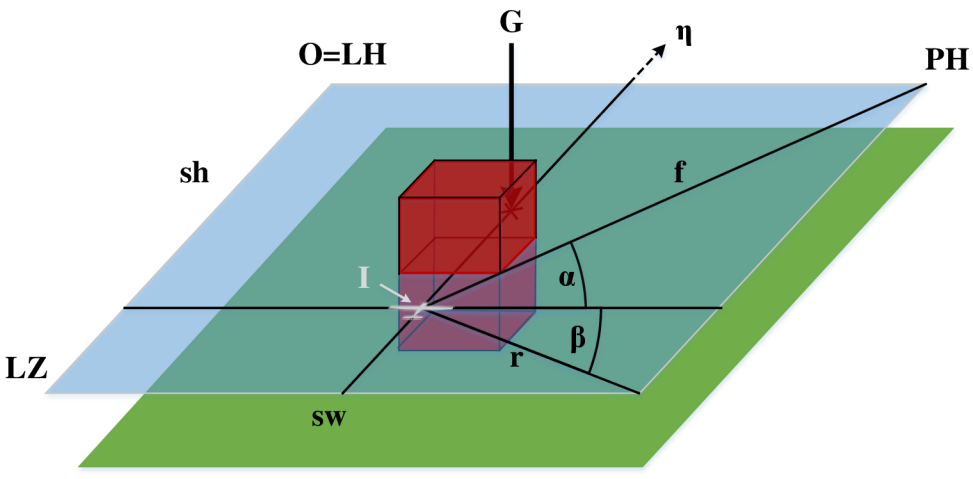
Algoritmus RRT musí mít specifikovaný prostor, ve kterém má hledat bezkolizní trať letu. Pro účely definování tohoto prostoru budeme letadlo považovat za systém, který se v každém okamžiku nachází v určité konfiguraci. Konfigurací letadla je jeho pozice v prostoru, který

budeme nazývat konfigurační prostor C. Tímto prostorem nechť je rotační elipsoid kopírující povrch Země v nadmořské výšce, odpovídající výšce letadla ve chvíli detekce hrozící srážky. Metrika tohoto prostoru ρ, vyjadřující vzdálenost mezi dvěma body v tomto prostoru, je totožná s funkcí 3.1, tedy $\rho(p,q) = distVincenty(p,q)$.

Konfigurační prostor reprezentuje oblast kolem letadla, ve které bude vyhledávána bezkolizní trať letu k cíli. Tuto oblast definujeme dle algoritmu 3.2. Konstanty ovlivňující velikost prostoru jsou v aplikaci nastaveny empiricky na $\alpha = 45°, \beta = \alpha/2$ a $n = 3$.

Vstup: souřadnice výchozího bodu (I), souřadnice cílového bodu (G)
Známé konstanty:
úhel k hornímu levému i pravému rohu prostoru (α), úhel k zadnímu levému i pravému rohu prostoru (β), kurz letadla (η), násobitel vzdálenosti cílového bodu (n)
Výstup:
souřadnice počátku (a zároveň levého horního rohu) prostoru (O), šířka prostoru ve stupních (sw), výška prostoru ve stupních (sh)
Postup:
1. $f = n * distVincenty(I, D)$
2. $O = destVincenty(I, \eta - \alpha, f)$
3. $PH = destVincenty(I, \eta + \alpha, f)$
4. $r = n * distVincenty(I, D) * cos(\alpha) * cos(\beta)$
5. $LZ = destVincenty(I, \eta + 90° + \beta, r)$
6. $sh = distVincenty(O, LZ)$
7. $sw = distVincenty(O, PH)$

Algoritmus 3.2: Získání počátku a rozměrů konfiguračního prostoru



Obrázek 3.14: Ilustrace k algoritmu 3.2

Implementace algoritmu 3.2 je v aplikaci součástí metody *computeRrtSpace* třídy *RrtSearch*.

Generování náhodné konfigurace v C

Za běhu algoritmu RRT je potřeba generovat v konfiguračním prostoru C náhodné konfigurace. V aplikaci bylo využito generátoru náhodných čísel, který dodává náhodná čísla s plovoucí řádovou čárkou v intervalu $\langle 0, 1)$. Tyto hodnoty lze zasadit do prostoru dle algoritmu 3.3.

Vstup: souřadnice počátku prostoru (O), šířka prostoru ve stupních (sw), výška prostoru ve stupních (sh), kurz letadla (η)

Výstup: konfigurace s náhodnými souřadnicemi (q_{rand})

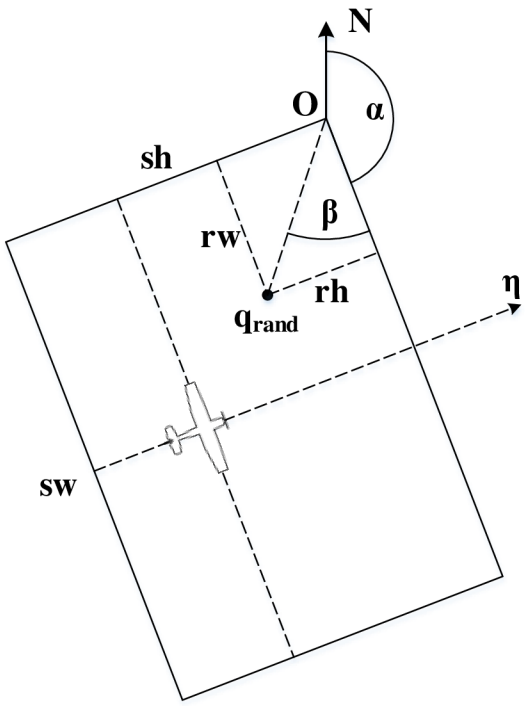
Postup:

1. $rh = sh * rand()$
2. $rw = sw * rand()$
3. $randDist = \sqrt{rh^2 + rw^2}$
4. $\alpha = \eta + 90°$
5. $\beta = \cos^{-1}(\frac{rh}{randDist})$
6. $q_{rand} = destVincenty(O, \alpha + \beta, randDist)$

Poznámky:

- *rand* je funkce vracející náhodné číslo s plovoucí řádovou čárkou v intervalu $\langle 0, 1)$

Algoritmus 3.3: Generování náhodné konfigurace v konfiguračním prostoru C



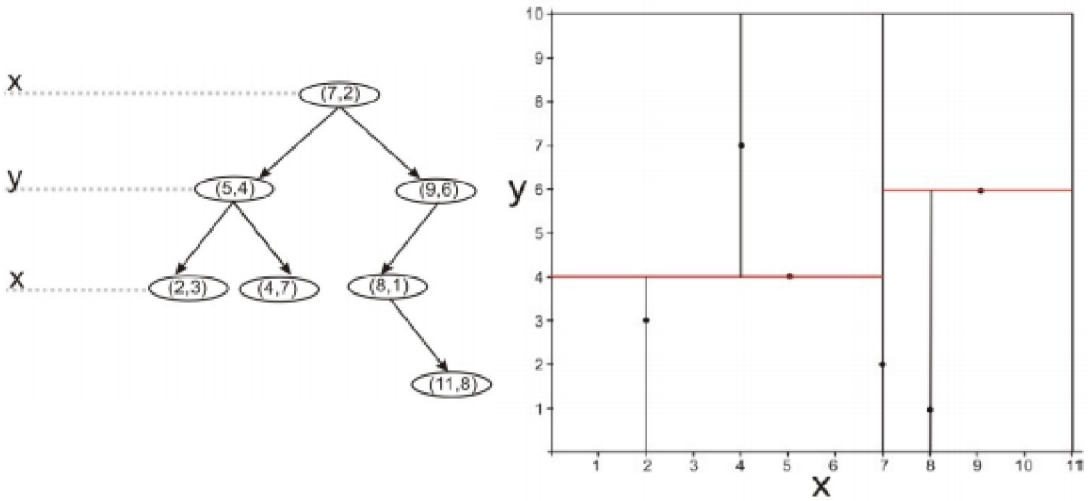
Obrázek 3.15: Ilustrace k algoritmu 3.3

Implementace algoritmu 3.3 v aplikaci je součástí metody *getRandomPoint* třídy *RrtSearch* z balíčku *com.bocekm.skycontrol.rrt*.

K-dimensionální strom

Jak název algoritmu Rapidly-exploring Random Tree (RRT) napovídá, při vyhledávání bezkolizní trati letu je budován strom, jehož uzly představují náhodně generované body v konfiguračním prostoru *C*. Pro reprezentaci tohoto stromu byl využit k-dimensionální strom (K-D stromu), který je často volený pro RRT díky své efektivitě vyhledávání souřadnic v prostoru.

K-D strom je binární vyhledávací strom, jehož uzly představují souřadnice bodu v k-dimensionálním prostoru. Každý vnitřní uzel tohoto stromu rozděluje prostor na dva podprostory (ilustrace na obrázku 3.16). Díky tomu je strom efektivní v hledání nejbližšího souseda (NNS), kterého využijeme v RRT algoritmu. Problém hledání nejbližšího souseda spočívá v nalezení vzdálenostně nejbližšího uzlu stromu k zadanému libovolnému bodu v prostoru. V K-D stromu je problém NNS řešen nalezením nejmenšího podprostoru, ve kterém zadaný bod leží.



Obrázek 3.16: Ilustrace dělení 2D prostoru na podprostory dle K-D stromu. Zdroj: [26]

Třída *RrtTree* reprezentující v aplikaci K-D strom využívá existující implementace K-D stromu z Java knihovny libssrckdtree-j [27]. Vzdálenostní funkce NNS odpovídá metrice ρ prostoru *C*.

RRT

Rozdíl implementovaného algoritmu oproti původnímu algoritmu RRT je v přidání určité pravděpodobnosti zvolení cílové konfigurace místo generování konfigurace náhodné (krok 2 v algoritmu 3.4). To má za následek růst grafu směrem k cílové konfiguraci q_{goal}, oproti růstu grafu do zcela náhodných směrů bez implementace této pravděpodobnosti. Pravděpodobnost byla v aplikaci nastavena empiricky na 30%, tedy $prob = 30$. Inspirací k této úpravě byl algoritmus Modified RRT z [8].

Vstup: počáteční konfigurace (q_{init}), cílová konfigurace (q_{goal}), konfigurační prostor C s metrikou ρ, prázdný K-D strom ($tree$), vzdálenost hrany dvou uzlů stromu v metrech (len), pravděpodobnost zvolení q_{goal} namísto generování q_{rand} v % ($prob$)

Výstup: $tree$ s cestou od kořene q_{init} k uzlu q_{goal}

Postup:

1. Vlož do $tree$ konfiguraci q_{goal}.
2. Vygeneruj q_{rand} v C. S pravděpodobností $prob$ polož $q_{rand} = q_{goal}$.
3. Nalezni pomocí metriky ρ v $tree$ takový uzel q_{near}, který je nejblíže konfiguraci q_{rand} (NNS).
4. Vytvoř konfiguraci q_{extend} ve vzdálenosti len a směru $q_{near} \rightarrow q_{rand}$. Pokud je q_{rand} identický q_{goal}, pak polož $q_{extend} = q_{goal}$.
5. Použij funkci $zkontrolujTrať(q_{near}, q_{extend})$ pro zjištění, zda trať letu mezi q_{near} a q_{extend} nekoliduje s překážkou.
6. Pokud je v kroku 5 zjištěna kolize, pokračuj krokem 2.
7. Pokud v kroku 5 není zjištěna kolize, vlož q_{extend} do $tree$.
8. Pokud je q_{extend} identický q_{goal}, vrať $tree$, jinak pokračuj krokem 2.

Algoritmus 3.4: Modifikovaný RRT algoritmus

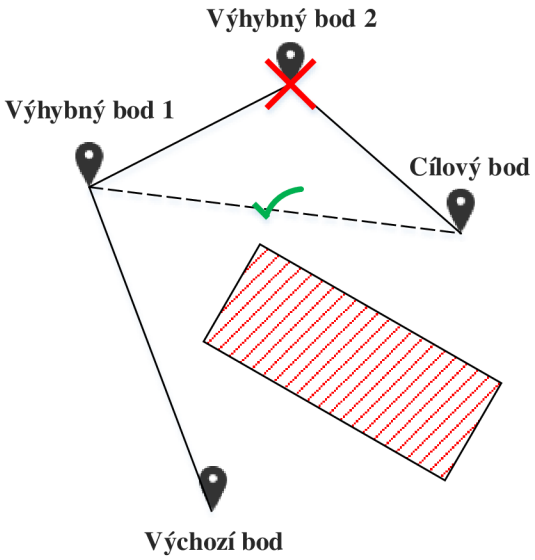
Výstupem algoritmu 3.4 je K-D strom obsahující kromě cest nevedoucích k cíli i cestu od q_{init} k q_{goal}. Seznam traťových bodů získáme průchodem stromu směrem od uzlu q_{goal} ke kořenu. Samotný q_{goal} získáme použitím funkce vyhledání nejbližšího souseda ve stromu pro q_{goal}.

Implementace algoritmu 3.4 v aplikaci je součástí metody *runSearch* třídy *RrtSearch*.

Optimalizace nalezené bezkolizní trati letu

Protože algoritmus 3.4 volí traťové body pro vyhnutí se překážce v rámci konfiguračního prostoru *C* náhodně, mohou tyto body být v jistých případech nadbytečné. Tyto nadbytečné body je možné identifikovat tak, že se načtou postupně tři po sobě jdoucí body vygenerované trati a pokud je trať mezi prvním a třetím bezkolizní, pak je druhý traťový bod odstraněn. Pro kontrolu volnosti trati letu mezi prvním a třetím bodem je využito funkce 3.4.

Optimalizace je v aplikaci implementována metodou *optimizeWaypoints* třídy *RrtSearch*.



Obrázek 3.17: Odstranění nadbytečného traťového bodu vygenerované algoritmem 3.4

3.5 Samočinné řízení trajektorie letadla

3.5.1 Mise

Základním pojmem samočinného řízení trajektorie letadla je mise. Mise je seznam po sobě jdoucích příkazů pro autopilota, které autopilot sekvenčně vykonává při přepnutí do režimu AUTO. Příkazy se dělí do následujících tří kategorií:

- navigační – MAV_CMD_NAV_* – obsahují poziční údaje,
- podmínečné – MAV_CMD_CONDITION_* – mění postup provádění mise na základě podmínky,
- vykonávající – MAV_CMD_DO_* – popisují okamžité akce.

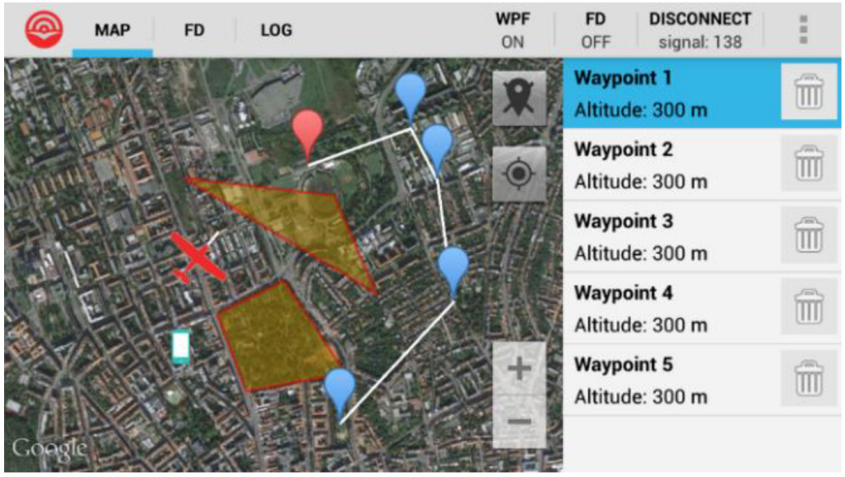
Příkazů mise existují desítky typů (dle MAVLink výčtu MAV_CMD). SkyControl však používá pouze dva typy příkazů v misích:

- příkaz definující traťový bod (MAV_CMD_NAV_WAYPOINT) a
- příkaz nastavující rychlost letu (MAV_CMD_DO_CHANGE_SPEED).

Aplikace SkyControl implementuje dva módy, ve kterých je samočinně řízena trajektorie letadla. Jsou nazvány Flight Director a Waypoint Follower. Oba pracují v režimu autopilota AUTO, tedy posílají do autopilota k vykonání právě mise. Popis jejich chování je obsahem následujících kapitol.

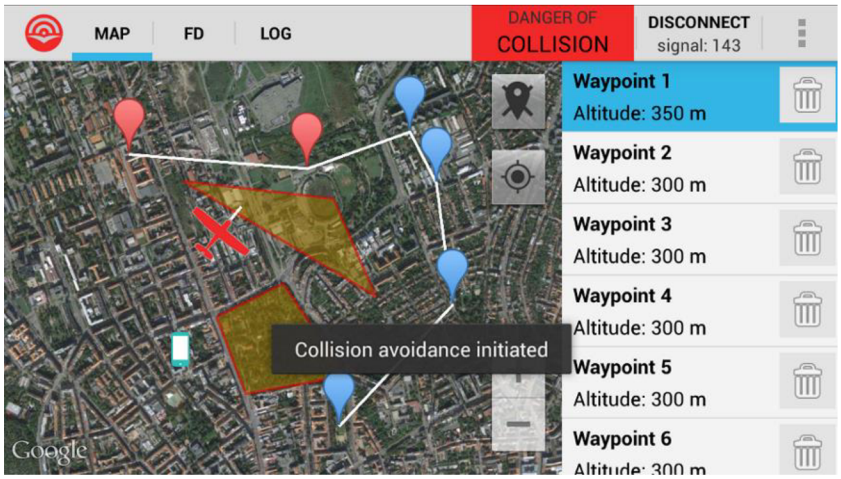
3.5.2 Mód Waypoint Follower

Mód Waypoint Follower (WPF) je v podstatě pouze spouštěč mise, kterou uživatel nadefinuje pomocí traťových bodů (waypoints). Traťový bod lze vytvořit dlouhým stiskem místa na mapě. Bodu je kromě souřadnic přiřazena nadmořská výška, a to dle nastavené výchozí výšky traťových bodů v možnostech aplikace. Bod lze přesunout jeho dlouhým podržením a tažením. Pro změnu nadmořské výšky bodu je třeba bod dlouze podržet v seznamu traťových bodů v postranní liště.



Obrázek 3.18: Zapnutý mód WPF s pěti traťovými body

Pokud je za letu detekována hrozící kolize s překážkou, je vygenerována bezkolizní trať letu pomocí algoritmu 3.4, a to od letadla k traťovému bodu, ke kterému autopilot aktuálně míří. Mise s bezkolizní tratí je autopilotu ihned odeslána k vykonání.



Obrázek 3.19: Vygenerování výhybného traťového bodu

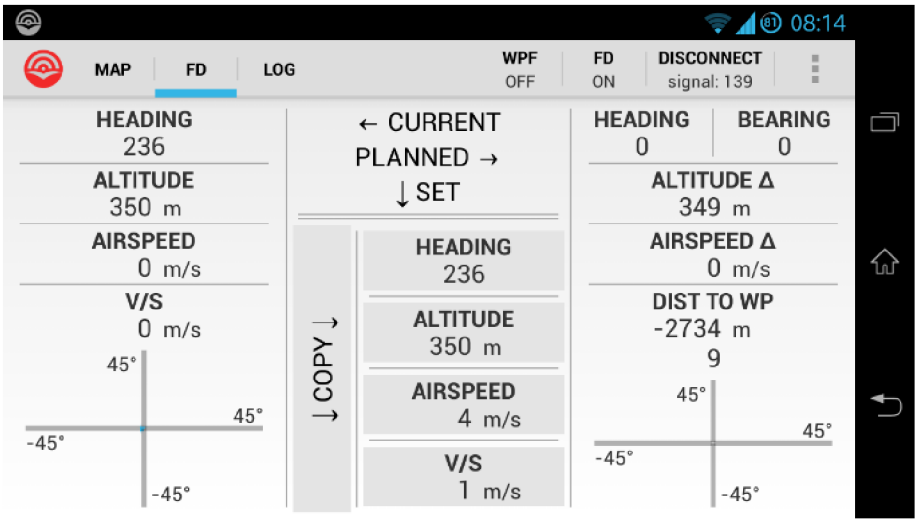
Mód Waypoint Follower je v aplikaci implementován třídou *WaypointFollower* v balíčku *com.bocekm.skycontrol.mission*.

3.5.3 Mód Flight Director

Mód Flight Director (FD) má v aplikaci SkyControl za úkol stabilizovat:

- vzdušnou rychlost,
- nadmořskou výšku,
- kurz letadla,

a to dle nastavených hodnot na záložce s nápisem FD.



Obrázek 3.20: Záložka FD umožňující nastavení hodnot pro mód Flight Director

FD v podstatě kombinuje dohromady funkci tří módů autopilota, známých ze světa dopravních letadel jako:

- Track Select – autopilot řídí příčný náklon pro dosažení nastaveného směru letu (kurz letadla opravený o snos větru),
- Vertical Speed – autopilot řídí podélný sklon pro dosažení nastavené nadmořské výšky současně s dodržováním nastavené vertikální rychlosti,
- Autothrottle – autopilot řídí výkon motoru za účelem dosažení nastavené vzdušné rychlosti.

Omezení FD spočívá v tom, že výše uvedené módy nelze vykonávat odděleně. Pokud je FD zapnutý, využívá k řízení trajektorie letadla všechny čtyři nastavitelné údaje, tedy kurz, nadmořskou výšku, vzdušnou rychlost a vertikální rychlost. Nelze tedy nastavit například pouze udržování určité nadmořské výšky a vše ostatní řídit RC vysílačkou. APM nenabízí žádný režim vyhovující takovému kritériu.

Vertikální rychlost (V/S) nastavitelná v aplikaci je vždy kladná. Jde o absolutní hodnotu, tzn., pokud je letadlo výše, než je nastavená nadmořská výška, letadlo bude k požadované výšce klesat s nastavenou vertikální rychlostí. Jde o odklon od praxe nastavování V/S v dopravních letadlech, kde je pro klesání letadla potřeba mít zadanou zápornou hodnotu.

Pro nastavení trajektorie, kterou má autopilot udržovat, posílá FD autopilotu mise vždy se třemi příkazy. Tyto tři příkazy sestávají z:

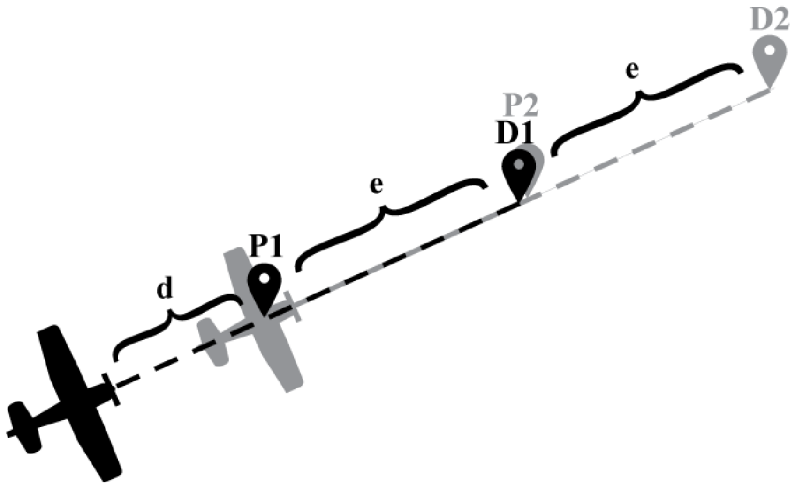
- jednoho příkazu nastavení požadované rychlosti, označme ho S,
- dvou navigačních příkazů *P* a *D*, definujících traťové body v požadovaném kurzu (HEADING) a výšce (ALTITUDE).

Vzdálenost *d* mezi letadlem a bodem *P* je počítána dle vzorce 3.3. Vzdálenost *e* mezi body *P* a *D* je empiricky nastavena na 15 vteřin a na metry je přepočítávána dle aktuální traťové rychlosti.

$$d = \frac{|výška\ letadla - požadovaná\ výška|}{vertikální\ rychlost} * vzdušná\ rychlost$$

Vzorec 3.3: Vzdálenost letadla od bodu P

První z traťových bodů označme *P1* a následující *D1*. Autopilot posílá pravidelně přes MAVLink identifikaci traťového bodu, ke kterému aktuálně naviguje letadlo. Díky tomu zjistíme, že letadlo dosáhlo bodu *P1*, a v tu chvíli aplikace vypočte a pošle autopilotu novou misi s body *P2* a *D2*, kde *P2* = *D1* a D2 leží opět v požadované trajektorii a ve vzdálenosti *e* od *P2*. Při změně jakéhokoliv ze čtyř zadaných údajů se traťové body vygenerují znovu. Takto se stále generují mise pro autopilota, dokud není mód FD vypnut.

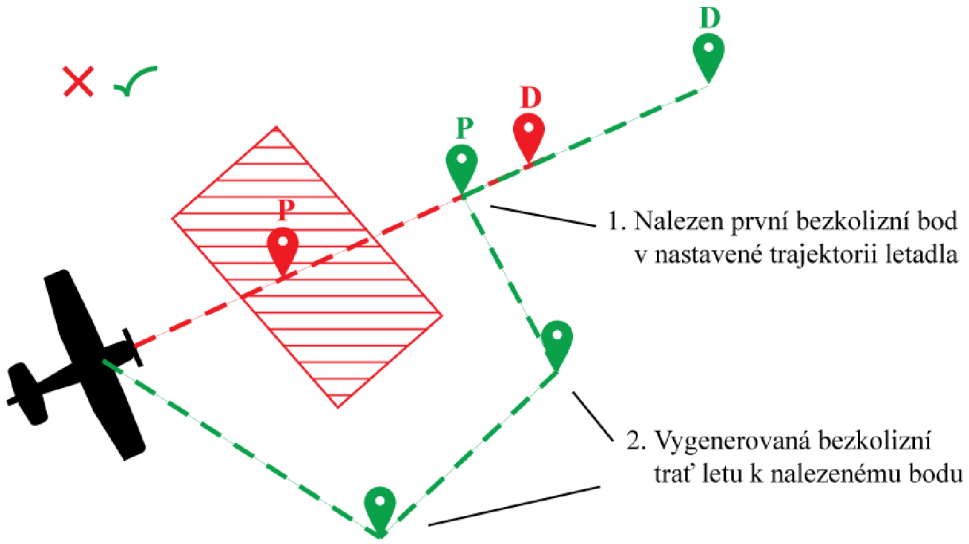


Obrázek 3.21: Udržování trajektorie letadla vytvářením traťových bodu *P* a *D*

Pokud některý z vypočítaných traťových bodů leží v překážce, pak je prodloužena vzdálenost kolidujícího bodu směrem od letadla, v případě neúspěchu opakovaně, s opětovnou kontrolou na kolizi s překážkou. Jakmile je nalezen bezkolizní bod, je mezi ním a letadlem vypočítána bezkolizní trať letu pomocí algoritmu 3.4. V tom případě jsou výše uvedené tři příkazy *S*, *P* a *D* doplněny traťovými body nalezené bezkolizní trati letu.

Může se stát, že uživatel nastaví trajektorii letadla směřující do země. V tom případě není nalezen bezkolizní bod, ke kterému by autopilot měl mířit, a mód FD je deaktivován. Řešením

tohoto omezení by bylo rozšíření algoritmu 3.4 tak, aby hledal bezkolizní trať letu v třírozměrném prostoru, jak je popsáno v kapitole 4.3.



Obrázek 3.22: Vyhnutí se překážce v módu FD

Mód Flight Director je v aplikaci implementován třídou *FlightDirector* v balíčku *com.bocekm.skycontrol.mission*.

3.1 Návrat letadla do oblasti se signálem

Pro návrat letadla do oblasti se signálem při ztrátě spojení bylo využito tzv. „failsafe“ funkce autopilota, která má za úkol provést určitou akci při vypršení časového limitu, který je resetován vždy při splnění určité podmínky. Mezi podmínky patří:

- přijetí paketu se zprávou pulsu od pozemní stanice,
- dostatečná kapacita baterie,
- posílání signálu vysílačkou pro dostatečný výkon motoru.

Nás zajímá podmínka přijetí paketu se zprávou pulsu od pozemní stanice, pro jejíž kontrolování je potřeba mít nastavený APM parametr FS_GCS_ENABL na hodnotu 1.

Časové limity, po jejichž vypršení je provedena akce, jsou dva, a to krátký (FS_SHORT_ACTN) a dlouhý (FS_LONG_ACTN). Ve výchozím nastavení je:

- krátký časový limit nastaven na 1,5 vteřiny, po nichž následuje akce (FS_SHORT_ACTN) přepnutí autopilota do režimu CIRCLE,
- dlouhý časový limit nastaven na 20 vteřin, po nichž následuje akce (FS_LONG_ACTN) přepnutí autopilota do režimu RTL.

Jako vhodné se jevilo zrušení akce při vypršení krátkého časového limitu nastavením FS_SHORT_ACTN na 0, pokud bude dostatečně krátký dlouhý časový limit. Aktivní failsafe akcí tedy zůstalo přepnutí autopilota do režimu RTL při vypršení dlouhého časového limitu. Ten aplikace nastavuje v autopilotu ihned po připojení letadla, a to na polovinu časové vzdálenosti, na kterou je kontrolována případná kolize s překážkou (viz kapitola 3.4.2). Tento limit v řádu jednotek vteřin by měl být bezpečný i v případě výpadku spojení letadla před překážkou. Což však platí za předpokladu, že taková překážka neleží v trati letu, po které letadlo poletí při návratu k základně. V opačném případě letadlo do překážky narazí, aplikace v tu chvíli nemá možnost zasáhnout.

Po vypršení časového limitu je letadlo naváděno na základnu čistě algoritmy autopilota, bez jakékoliv intervence aplikace SkyControl, a to dokud není spojení obnoveno. Režim RTL znamená v podstatě vykonávání mise s jedním traťovým bodem. APM dokáže kompenzovat snos větru a tak mu navedení letadla nad pozici základny nečiní problémy. Po obnovení spojení s aplikací je možné autopilota přepnout zpět do režimu MANUAL nebo AUTO.

Z bezpečnostního hlediska má pilot modelu letadla kromě funkce failsafe možnost převzít řízení letadla vysílačkou. Typicky je jeden z kanálů vysílačky vyhrazen pro možnost přepnutí autopilota do režimu MANUAL.

Navrhované řešení úpravy firmwaru autopilota APM pro zajištění specifického chování při ztrátě spojení s pozemní stanicí (viz kapitola 2.5) nebylo realizováno.

Nastavení failsafe parametrů autopilota APM po jeho připojení je realizováno v aplikaci ve třídě *VehicleParameters* z balíčku *com.bocekm.skycontrol.vehicle*.

4 Ověření v praxi

4.1 Testovací lety

Model Skydog nebyl pro testovací lety nakonec dostupný, a proto byl využit model letadla Skywalker patřící Bedřichu Saidovi, který ho také při testech pilotoval. Model byl osazený stejným autopilotem APM a jeho doplňky, které byly plánované pro použití s modelem Skydog. Na otestování funkcionality aplikace nemělo použití jiného modelu letadla vliv.

První testovací let na modelářském letišti v Brně - Medlánkách odhalil nízko nastavený časový limit pro zprávy pulsu od autopilota. Tento limit byl nastaven na 3 vteřiny a po nich bylo spojení s letadlem považováno aplikací za ztracené. APM posílá zprávy pulsu každou jednu vteřinu, a protože v terénu je míra ztráty paketů vysoká, tak stačilo, aby se ztratily dva po sobě jdoucí pakety se zprávou pulsu a spojení s letadlem bylo považováno za ztracené.

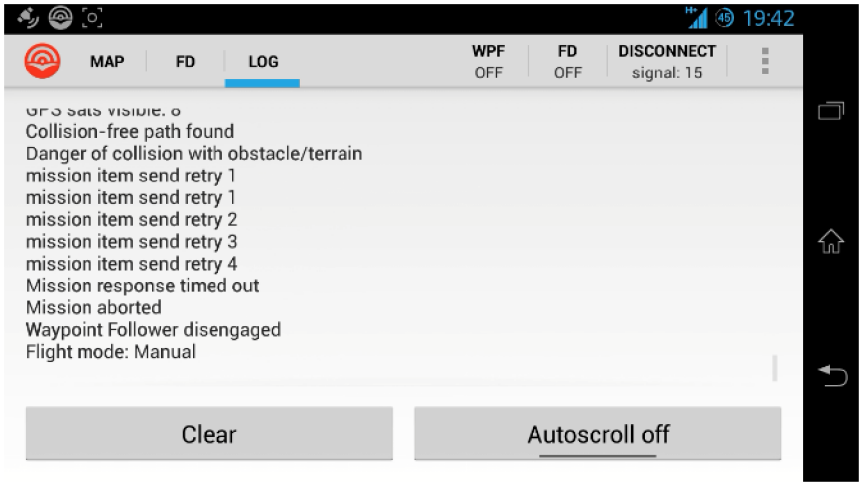
Pro účely ověření vyhnutí se překážce byla definovaná v XML jedna pomyslná překážka poblíž vzletové a přistávací dráhy modelářského letiště. I přes výpadky spojení se podařilo při letu směrem k této překážce zapnout mód Waypoint Follower s traťovým bodem nastaveným ve směru letu za překážkou. Při detekci překážky aplikace vygenerovala jeden výhybný traťový bod, na který letadlo následně zamířilo.

Podařilo se zapnout i mód Flight Director, u kterého ale nešlo ověřit, zda průběžně generuje traťové body pro udržení letadla na nastavené trajektorii. Misi s traťovými body, které by udržely letadlo na zadané trajektorii, se totiž nepodařilo kvůli ztrátám signálu do autopilota zapsat.

Druhý testovací let potvrdil enormní ztrátovost paketů. Časový limit pro příchod zprávy pulsu byl sice zvýšen na 15 vteřin a tak nebylo letadlo považováno za ztracené při ztrátě několika těchto zpráv, ale nebyla tím vyřešena příčina - nekvalitní bezdrátové spojení. Problém se ztrácejícími se pakety se projevil při odesílání misí do letadla. Anténa rádiového modulu na modelu letadla Skywalker byla instalována vně trupu pro co nejnižší míru blokování radiových vln materiálem, viz obrázek 4.1, což stejně výrazně nepomohlo kvalitě spojení.

Při letu v módu Waypoint Follower byla vždy správně detekována překážka a vygenerovány traťové body pro vyhnutí, ale mise s těmito body se již nedařilo do letadla nahrát. Pouze jednou se výhybná trať letu dokázala zapsat do APM, avšak až po několika vteřinách, za které letadlo již stačilo pomyslnou překážkou proletět.

Obrázek 4.1: Umístění instalace antény v modelu Skywalker



Obrázek 4.2: Záznam neúspěšného nahrání mise s výhybným manévrem do autopilota

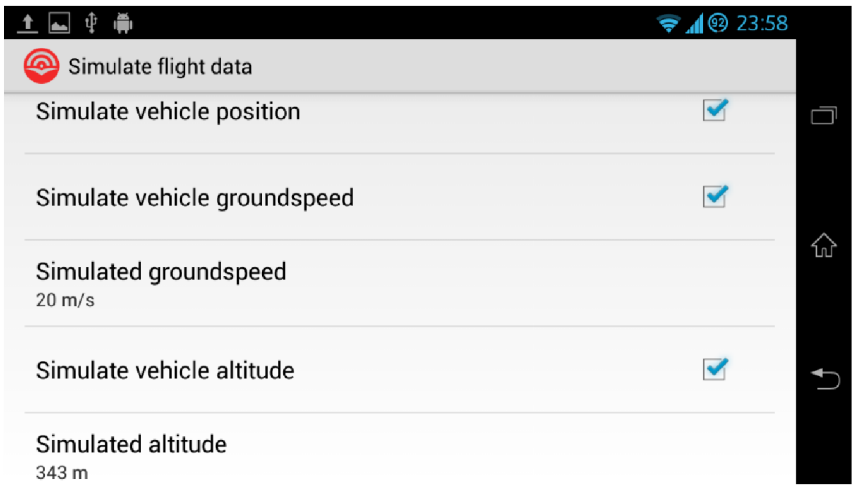
Při druhém testovacím letu byla ověřena funkce návratu letadla na základnu při ztrátě spojení. Před odstartováním letadla byla nastavena pozice základny pomocí položky *Set home* v menu aplikace. Parametr povolující „failsafe“ akci při ztrátě spojení *FS_GCS_ENABL* byl nastaven na 1, tedy povoleno. Časový limit pro přechod do režimu RTL byl nastaven parametrem autopilota *FS_LONG_TIMEOUT* na 7 vteřin. Po vypnutí MAVLink komunikace tlačítkem *DISCONNECT* se opravdu za 7 vteřin letadlo letící směrem od pozice základny otočilo, vrátilo se nad základnu a začalo nad ní kroužit. Spojení aplikace s autopilotem bylo následně bez problému znovu navázáno tlačítkem *CONNECT*.

4.2 Simulace letu

Z důvodu nekvalitního bezdrátového spojení za letu byla v aplikaci implementována jednoduchá simulace letu za účelem ověření správné funkce módu Flight Director. Správná funkce módu Waypoint Follower byla ověřena již při prvním testovacím letu, viz předchozí kapitola.

V možnostech aplikace lze povolit simulovanou pozici a nastavit simulovanou rychlost i nadmořskou výšku letadla. Výchozí simulovanou pozicí je pozice Android zařízení a od ní se letadlo pohybuje nastavenou rychlostí směrem daným magnetometrem APM. Směr tedy není simulovaný a je možné ho měnit manuálním natáčením desky APM.

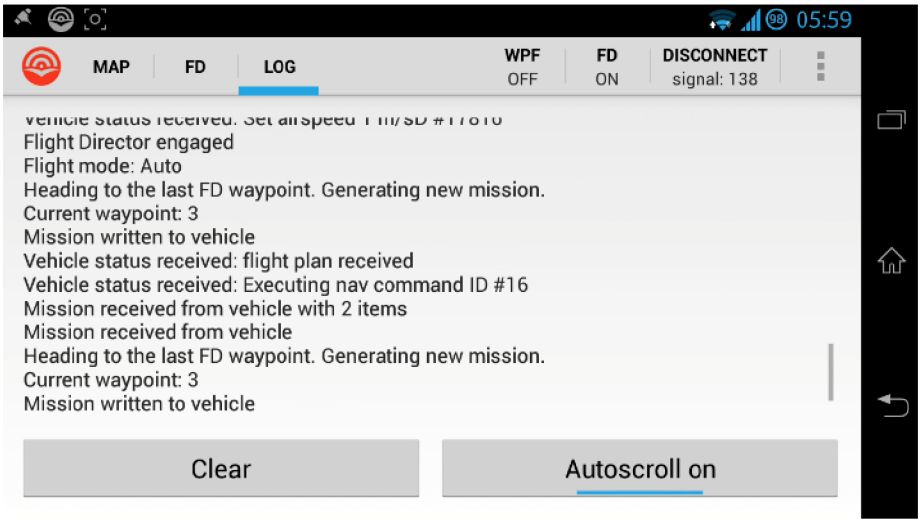
Simulované je i posílání zpráv s aktuálního traťovým bodem, ke kterému v rámci mise autopilot míří. To je zapotřebí k otestování funkce FD, který na tyto zprávy spoléhá. Taková zpráva je nasimulována v případě, že je simulovaná poloha letadla v okruhu 30 m od některého z traťových bodů, které FD vygeneroval.



Obrázek 4.3: Nastavení letových údajů pro simulaci letu

Po spuštění módu FD v testu byl první traťový bod *P1* dosažen téměř ihned a následovalo tedy odeslání nové dvojice bodů *P2* a *D2*. S deskou APM bylo natáčeno tak, aby letadlo letělo k novému bodu *P2*. Po jeho dosažení byla opět vygenerována nová dvojice bodů. Tedy mód pracoval podle předpokladu. Na obrázku 4.4 je snímek obrazovky se záznamem postupného generování misí při dosažení bodů *P*.

Snímek obrazovky s mapou by nic kromě polohy letadla nevypověděl, protože mise generované v módu FD jsou zcela odděleny od misí módu WPF a pouze traťové body mise WPF jsou vykreslovány na mapě. Takové chování je záměrné. Umožňuje uživateli mít přednastavenu komplexní misi pro mód WPF a pouštět mód FD, aniž by taková mise byla přepsána.



Obrázek 4.4: Snímek obrazovky se záznamem samočinného generování misí v módu FD

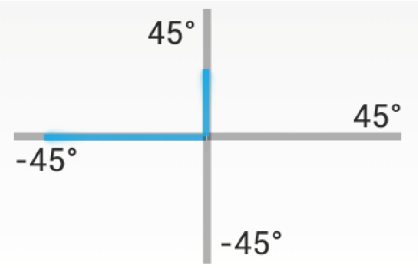
4.3 Zhodnocení realizace

Funkčnost algoritmů a funkcí popsaných v kapitole 3 byla ověřena v kapitolách 4.1 a 4.2. Pro uvedení aplikace do běžného provozu s reálnými překážkami je však vhodné zvážit následující návrhy na úpravy a vylepšení.

1. Rozšířit funkci 3.5 kontrolující kolizi zeměpisného bodu s překážkou tak, aby z důvodu možné nepřesnosti definovaných překážek kontrolovala kolizi nejen v jednom bodě daným souřadnicemi, ale v prostoru kolem tohoto bodu, například v kruhu s daným poloměrem. Obdobně pro funkci 3.4, kde by byla kontrolována kolize s překážkou nejen na traťových bodech, ale i v blízkém okolí trati.
2. Rozšířit vyhledávání bezkolizní tratě letu o třetí rozměr. Aktuálně je bezkolizní trať vyhledávána v rovině rovnoběžné s povrchem Země ve výšce detekce hrozící kolize. Rozšíření by vyžadovalo zavedení omezení rychlosti stoupání od vyhledaného nejbližšího uzlu stromu q_{near} k náhodně vygenerovanému bodu q_{rand}.
3. Přidat možnost vyhýbání se překážkám i v režimu autopilota MANUAL. To však s sebou přináší riziko ztráty kontroly pilota nad letadlem. Aktuálně je protisrážkový systém implementován pouze pro módy Flight Director a Waypoint Follower. Ty pracují v režimu autopilota AUTO a pilot je schopný tento režim (či jakýkoliv jiný) změnit za letu na režim MANUAL pomocí RC vysílačky pokud usoudí, že samočinné řízení autopilotem není bezpečné. V režimu MANUAL má pak plnou kontrolu nad letadlem. Avšak pokud by bylo implementováno vyhýbání se překážkám i v režimu MANUAL, autopilot by měl možnost soustavně zasahovat do řízení takovým způsobem, že by se letadlo stalo neřiditelným.
4. Přidat navigační příkaz MAV_CMD_NAV_RETURN_TO_LAUNCH jako poslední příkaz všech misí, které aplikace vytváří. Ten zajistí, že se letadlo vrátí na základnu

v případě dokončení mise. Vhodné toto chování může být v případě, kdy se v režimu Flight Director nepodaří po dosažení bodu *P* zapsat do autopilota pokračující misi s novými body *P* a *D*. Autopilot v tomto případě aktuálně dokončí misi při dosažení bodu *D*, zůstane v režimu AUTO a letadlo je následně v podstatě neřízeno.

5. Implementovat čitelnou grafickou podobu umělého horizontu. Aktuální implementace indikátoru příčného náklonu a podélného sklonu letadla není dostatečně čitelná.



Obrázek 4.5: Nízká čitelnost indikátoru sklonu/náklonu letadla

5. Použít hardware pro bezdrátovou komunikaci s vyšším dosahem. Dosah použité sady 3DR Radio 433 MHz nebyl dostatečný, viz kapitola 4.1.
6. V kapitole 3.4.3 jsou zmíněna významná omezení funkce 3.4 a jejich možná řešení.

5 Závěr

Cílů, kterých bylo vymezeno v zadání této práce, se podařilo úspěšně dosáhnout. Implementovaná Android aplikace umožňuje instruovat autopilota ArduPilot Mega k samočinnému řízení trajektorie letadla, při kterém aplikace dokáže detekovat blížící se překážku a docílit vyhnutí se srážce.

Samočinného řízení trajektorie letadla aplikace dosahuje dvěma módy, které má možnost uživatel separátně aktivovat. Těmito módy jsou Waypoint Follower a Flight Director. Mód Waypoint Follower posílá autopilotu k vykonání misi, kterou nadefinoval uživatel. Tyto mise sestávají ze sekvence traťových bodů, které se autopilot snaží obletět. Mód Flight Director dokáže udržovat trajektorii danou čtyřmi letovými údaji, které nastavuje uživatel, a těmi jsou kurz, nadmořská výška, vzdušná rychlost a vertikální rychlost.

Pro oba módy, WPF i FD, je v aplikaci možné zapnout protisrážkový systém. Ten pracuje nad informacemi o terénu a známých překážkách. Překážky jsou definované v XML a informace o terénu jsou získávány ze souboru formátu GeoTIFF, který obsahuje nadmořskou výšku terénu s cca 30m rozestupem jednotlivých bodů. V případě detekované srážky buď s terénem, nebo s překážkou, je pomocí algoritmu RRT proveden pokus o nalezené bezkolizní trati letu. Tato trať je následně přidána do mise, kterou zrovna autopilot vykonává.

Aplikace nastavuje v autopilotu po jeho připojení časový limit pro ztrátu spojení s pozemní stanicí v řádu jednotek sekund. Po vypršení tohoto limitu, kdy autopilot nedostává zprávy od aplikace, přechází do režimu RTL, tedy návratu na základnu.

Výše zmíněné vlastnosti fungují podle očekávání v laboratorních podmínkách, ale při testovacích letech bylo zjištěno, že kvůli nedostatečně silnému bezdrátovému spojení autopilota s aplikací se ztrácí vysoké množství paketů a to znemožňuje správnou funkci ovládání letadla aplikací. Autor proto částečně ověřoval funkčnost módů aplikace simulováním údajů o letu. Řešením problému s bezdrátovou komunikací by mohlo být použití radio modulů s vyšším dosahem. Např. radio moduly XBee-PRO 868 mají udávaný dosah 80 km, oproti dosahu 1,6 km udávaném pro použité radio moduly 3DR Radio.

Významným omezením aplikace je vyhledávání bezkolizní trati letu pouze v rovině místo v třírozměrném prostoru. Pokud letadlo letí směrem do země v režimu Flight Director, není pro tento případ nalezena trať pro vyhnutí. Další omezení implementace a jejich možná řešení jsou shrnuty v kapitole 4.3.

Jak bylo zmíněno v kapitole 2.4.3, student P. Čamaj vytvořil tento rok v rámci diplomové práce hardware autopilota založeného na platformě Raspberry Pi s OS RASBIAN. V návaznosti na tuto diplomovou práci autor plánuje implementaci knihovny protokolu MAVLink pro tento operační systém a tedy ověření funkčnosti vyvíjené aplikace s jiným autopilotem, než APM.

Dále je plánováno rozšiřování funkcionality aplikace v rámci interních projektů oddělení Flight Controls firmy Honeywell, která poskytla většinu vybavení souvisejícího s touto prací.

Literatura

[1] ČAMAJ, Peter. *Stabilizační a autopilotní systém pro RC model letadla*. Brno, 2013. Semestrální práce. FEKT VUT v Brně.

[2] ALEXANDER, Brandon. The AVR And The Arduino. In: *Alexander Robotics* [online]. 2011 [cit. 2014-01-14]. Dostupné z: http://alexanderrobotics.com/blog/2011/10/avr-and-the-arduino/

[3] Vývojový kit ARDUINO Arduino MEGA2560 | GM Electronic. [online]. [cit. 2014-06-04]. Dostupné z: http://www.gme.cz/vyvojovy-kit-arduino-arduino-mega2560-p772-006

[4] GUINAULT, Pierre. Carte ArduPilot et capteurs - Iut Si Polytech QuadriCoptère - Forge Clermont Université. [online]. 2013 [cit. 2014-06-04]. Dostupné z: http://forge.clermont-universite.fr/projects/quadri/wiki/Carte_ArduPilot_et_capteurs

[5] MEIER, Lorenz. MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS. QGroundControl GCS [online]. 2009 [cit. 2014-01-14]. Dostupné z: http://qgroundcontrol.org/mavlink/start

[6] SKYbrary - Terrain Avoidance and Warning System (TAWS). [online]. 2014 [cit. 2014-06-04]. Dostupné z: http://www.skybrary.aero/index.php/Terrain_Avoidance_and_Warning_System_(TAWS)

[7] MAJUMDAR, Dave. USAF to field F-16s with auto-GCAS in 2014 - 9/12/2013 - Flight Global. [online]. 2013 [cit. 2014-06-04]. Dostupné z: http://www.flightglobal.com/news/articles/usaf-to-field-f-16s-with-auto-gcas-in-2014-390453/

[8] VIQUERAT, Andrew, Lachlan BLACKHALL, Alistair Smyth REID, Salah SUKKARIEH a Graham M. BROOKER. Reactive Collision Avoidance for Unmanned Aerial Vehicles Using Doppler Radar. *Springer Tracts in Advanced Robotics*. Springer-Verlag Berlin Heidelberg, 2008, roč. 2007, Results of the 6th International Conference.

[9] WATANABE, Yoko, Anthony J. CALISE a Eric N. JOHNSON. Vision-Based Obstacle Avoidance for UAVs. In: AIAA Guidance, Navigation and Control Conference and Exhibit. Hilton Head, South Carolina: American Institute of Aeronautics and Astronautics, Inc., 2007.

[10] MCGEE, Timothy G., Raja SENGUPTA a Karl HEDRICK. Obstacle Detection for Small Autonomous Aircraft Using Sky Segmentation. In: Robotics and Automation, 2005. ICRA 2005. 4679,4684.

[11] TIGHE, M. Lorraine a Gregory BUCKMAN. IFSAR AND LIDAR HIGH RESOLUTION ELEVATION DATA FOR MINING APPLICATIO NS: ADVANTAGES AND DISADVANTEGES OF COMP LIMENTARY TECHNOLOGIES [online]. 2008 Geospatial Conference, 2008 [cit. 2014-06-04]. Dostupné z: http://www.tips.osmre.gov/newsroom/conferences/2008Geospatial/pdf/presentation/L.Tighe.pdf. Prezentace. Intermap Technologies.

[12] ADLib Glossary (D) - Digital terrain model. UNIVERSITY OF HERTFORDSHIRE. ADLib [online]. Agricultural Document Library, 2011 [cit. 2014-01-14]. Dostupné z:

http://adlib.everysite.co.uk/adlib/defra/content.aspx?id=2RRVTHNXTS.96RYU2KJZWFLU

[13] Reverb | ECHO. NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. Reverb | ECHO [online]. [cit. 2014-01-14]. Dostupné z: http://reverb.echo.nasa.gov/reverb/

[14] KARAMAN, Sertac a Emilio FRAZZOLI. Sampling-based algorithms for optimal motion planning. In: The International Journal of Robotics Research. 2011, s. 846-894.

[15] LAVALLE, Steven M. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Department of Computer Science. Iowa State University, 1998. Dostupné z: http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

[16] SAUNDERS, Jeffery B., Andrew CURTIS, Randal W. BEARD a Timothy W. MCLAIN. Static and Dynamic Obstacle Avoidance in Miniature Air Vehicles. Proceedings of the Infotech@Aerospace Conference, 2005. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.4282&rep=rep1&type=pdf. American Institute of Aeronautics and Astronautics, Brigham Young University.

[17] DUTOIT, Ryan, Matt HOLT, Megan LYLE a Saad BIAZ. AV Collision Avoidance Using RRT* and LOS Maximization. Technical Report #CSSE12 - 03. 2012.

[18] Doplněk X – Bezpilotní systémy. In: Letecký předpis L2. 2010. Dostupné z: http://lis.rlp.cz/predpisy/predpisy/dokumenty/L/L-2/data/effective/doplX.pdf

[19] Code Style Guidelines for Contributors | Android Developers. [online]. [cit. 2014-06-04]. Dostupné z: https://source.android.com/source/code-style.html

[20] Google Java Style. [online]. 2014 [cit. 2014-06-04]. Dostupné z: http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

[21] BENEMANN, Arthur. DroidPlanner/droidplanner - Github. [online]. [cit. 2014-06-04]. Dostupné z: https://github.com/DroidPlanner/droidplanner

[22] Index of AIXM schema/5.1. [online]. [cit. 2014-06-04]. Dostupné z: http://www.aixm.aero/gallery/content/public/schema/5.1/

[23] Hexagonal Prism | ClipArt ETC. FLORIDA CENTER FOR INSTRUCTIONAL TECHNOLOGY. [online]. [cit. 2014-06-04]. Dostupné z: http://etc.usf.edu/clipart/43100/43131/prism-hex7_43131.htm

[24] VINCENTY, T. Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations. urvey Review. 1975, 88–93. XXII. Dostupné z: http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf

[25] Movable Type - Information Design & Management. MOVABLE TYPE LTD. [online]. [cit. 2014-06-04]. Dostupné z: http://movable-type.co.uk/

[26] BEZDĚK, Josef. Problém hledání nejbližšího souseda a indexační algoritmy. 2009. Dostupné z: http://gis.zcu.cz/studium/pdb/referaty/2008/Bezdek_ProstoroveIndexy/Bezdek_ProstoroveIndexy.pdf. Fakulta aplikovaných věd, Západočeská univerzita v Plzni.

[27] Libssrckdtree-j Generic k-d tree Java library. [online]. [cit. 2014-06-04]. Dostupné z: http://www.savarese.com/software/libssrckdtree-j/

Seznam zkratek

3DR	3D Robotics
ADT	Android Development Tools
AGCAS	Automatic Ground Collision Avoidance System
AIXM	Aeronautical Information Exchange Model
AP	Autopilot
APK	Android Package File
API	Application Programming Interface
APM	ArduPilot Mega
AFIS	Aerodrome Flight Information Service
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
ATM	Air Traffic Management
ATZ	Aerodrome Traffic Zone
CAS	Collision Avoidance System
CPU	Central Processing Unit
DSM	Digital Surface Model
DTM	Digital Terrain Model
EGM	Earth Gravitational Model 1996
EGPWS	Enhanced Ground Proximity Warning System
ESC	Electronic Speed Control
FD	Flight Director
GA	General Aviation
GCC	GNU Compiler Collection
GCS	Ground Control Station
GDEM	Global Digital Elevation Model
GML	Geographical Markup Language
GPS	Global Positioning System
GPWS	Ground Proximity Warning System
GS	Groundspeed
HAXM	Hardware Accelerated Execution Manager
HW	Hardware
IAS	Indicated Air Speed
IFD	Image File Directory
ID	Identifikace
IDE	Integrated Development Environment
IMU	Inertial Measurement Units
JNI	Java Native Interface
JTS	Java Topology Suite
JVM	Java Virtual Machine
K-D	K-Dimensionální
KiB	Kibibajt

LGPL	Lesser General Public License
Li-Pol	Lithium-Polymer
LVV	Letecké Veřejné Vystoupení
METI	Ministry of Economy, Trade, and Industry
NASA	National Aeronautics and Space Administration
NDK	Native Development Kit
NNS	Nearest Neighbor Search
OMPL	Open Motion Planning Library
OS	Operační systém
OTG	On-The-Go
PC	Personal Computer
PFD	Primary Flight Display
PWM	Pulse Width Modulation
RAID	Redundant Array of Inexpensive/Independent Disks
RAM	Random Access Memory
RC	Remote Control
RF	Radiová Frekvence
RRT	Rapidly-exploring Random Tree
RTL	Return To Launch
SD	Secure Digital
SVN	Subversion
SW	Software
TIFF	Tagged Image File Format
UAS	Unmanned Aircraft Systems
UAV	Unmanned Aerial Vehicle
UI	User Interface
UML	Unified Modeling Language
USA	United States of America
USAF	United States Air Force
USB	Universal Serial Bus
ÚCL	Úřad Pro Civilní Letectví
ust.	Ustanovení
V/S	Vertical Speed
WGS84	World Geodetic System 1984
WPF	Waypoint Follower
XML	Extensible Markup Language

Seznam obrázků

Obrázek 2.1: Model letadla Skydog. Zdroj: [1]
Obrázek 2.2: Kroky k získání HEX souboru pro AVR. Zdroj: [2]
Obrázek 2.3: Arduino Mega 2560. Zdroj: [3]
Obrázek 2.4: Deska ArduPilot Mega 2.5. Zdroj: [4]
Obrázek 2.5: Grafické rozhraní GCS Mission Planner
Obrázek 2.6: Propojení APM s Android zařízením pomocí 3DR Radio 433 Mhz
Obrázek 2.7: Formát MAVLink paketu se zprávou. Zdroj: [5]
Obrázek 2.8: Návrh UI pro zobrazení GPS polohy letadla
Obrázek 2.9: Návrh UI pro zobrazení parametrů letu
Obrázek 2.10: Ikona aplikace Skydog Controller
Obrázek 2.11: Digital Surface Model (DSM) vs. Digital Terrain Model (DTM). Zdroj: [12]
Obrázek 2.12: ASTER GDEM dlaždice N49°E16° - N50°E17°. Zdroj: [13]
Obrázek 2.13: Větvení stromu náhodnými směry v algoritmu RRT. Střed – výchozí pozice, zelený objekt – překážka, modrý objekt – cíl, ke kterému algoritmus hledá bezkolizní cestu. Zdroj: [17]
Obrázek 2.14: Provoz v ATZ a prostorech třídy G a E. Zdroj: [18]
Obrázek 3.1: Hlavní třídy aplikace
Obrázek 3.3: Příklad kolmého 6-bokého hranolu. Zdroj: [23]
Obrázek 3.4: Ukázka zobrazení překážek v aplikaci SkyControl
Obrázek 3.5: Metadata formátu TIFF
Obrázek 3.6: Metadata formátu GeoTIFF
Obrázek 3.7: Zobrazení krajních souřadnic GeoTIFF na základě přečtených metadat
Obrázek 3.8: Offsety pruhů
Obrázek 3.9: Pixel jako plocha
Obrázek 3.10: Budoucí poloha letadla je v aplikaci znázorněna koncem bílé čáry před letadlem
Obrázek 3.11: Nedetekovaná kolize s překážkou
Obrázek 3.12: Funkce 3.4 kontroluje v případě terénu pouze konečný bod přímky a nedokáže detekovat situaci na obrázku
Obrázek 3.13: Příklad využití funkce 3.5 při přidávání traťového bodu
Obrázek 3.14: Ilustrace k algoritmu 3.2
Obrázek 3.15: Ilustrace k algoritmu 3.3
Obrázek 3.16: Ilustrace dělení 2D prostoru na podprostory dle K-D stromu. Zdroj: [26]
Obrázek 3.17: Odstranění nadbytečného traťového bodu vygenerované algoritmem 3.4
Obrázek 3.18: Zapnutý mód WPF s pěti traťovými body
Obrázek 3.19: Vygenerování výhybného traťového bodu
Obrázek 3.20: Záložka FD umožňující nastavení hodnot pro mód Flight Director
Obrázek 3.21: Udržování trajektorie letadla vytvářením traťových bodu P a D
Obrázek 3.22: Vyhnutí se překážce v módu FD

Obrázek 4.1: Umístění instalace antény v modelu Skywalker
Obrázek 4.2: Záznam neúspěšného nahrání mise s výhybným manévrem do autopilota
Obrázek 4.3: Nastavení letových údajů pro simulaci letu
Obrázek 4.4: Snímek obrazovky se záznamem samočinného generování misí v módu FD
Obrázek 4.5: Nízká čitelnost indikátoru sklonu/náklonu letadla

Seznam tabulek

Tabulka 2.1: Základní parametry modelu Skydog. Zdroj: [1]
Tabulka 2.2: Význam jednotlivých bajtů MAVLink paketu. Zdroj: [5]
Tabulka 2.3: Povolené operace mezi rozhraními jazyků Java a C/C++
Tabulka 3.1: Údaje přijímané periodicky
Tabulka 3.2: Údaje přijímané při jejich změně
Tabulka 3.3: Seznam důležitých TIFF tagů
Tabulka 3.4: Popis několika důležitých klíčů GeoKeys
Tabulka 1: Obsah přiloženého CD/DVD

Seznam příloh

Příloha 1. Příklad AIXM XML
Příloha 2. Časový plán vývoje
Příloha 3. Styl zápisu kódu
Příloha 4. Třída *MainActivity*
Příloha 5. Třída *Mission*
Příloha 6. Třída *Vehicle*
Příloha 7. Třída CollisionAvoidance
Příloha 8. Třída *Connection*
Příloha 9. CD/DVD s následujícím obsahem:

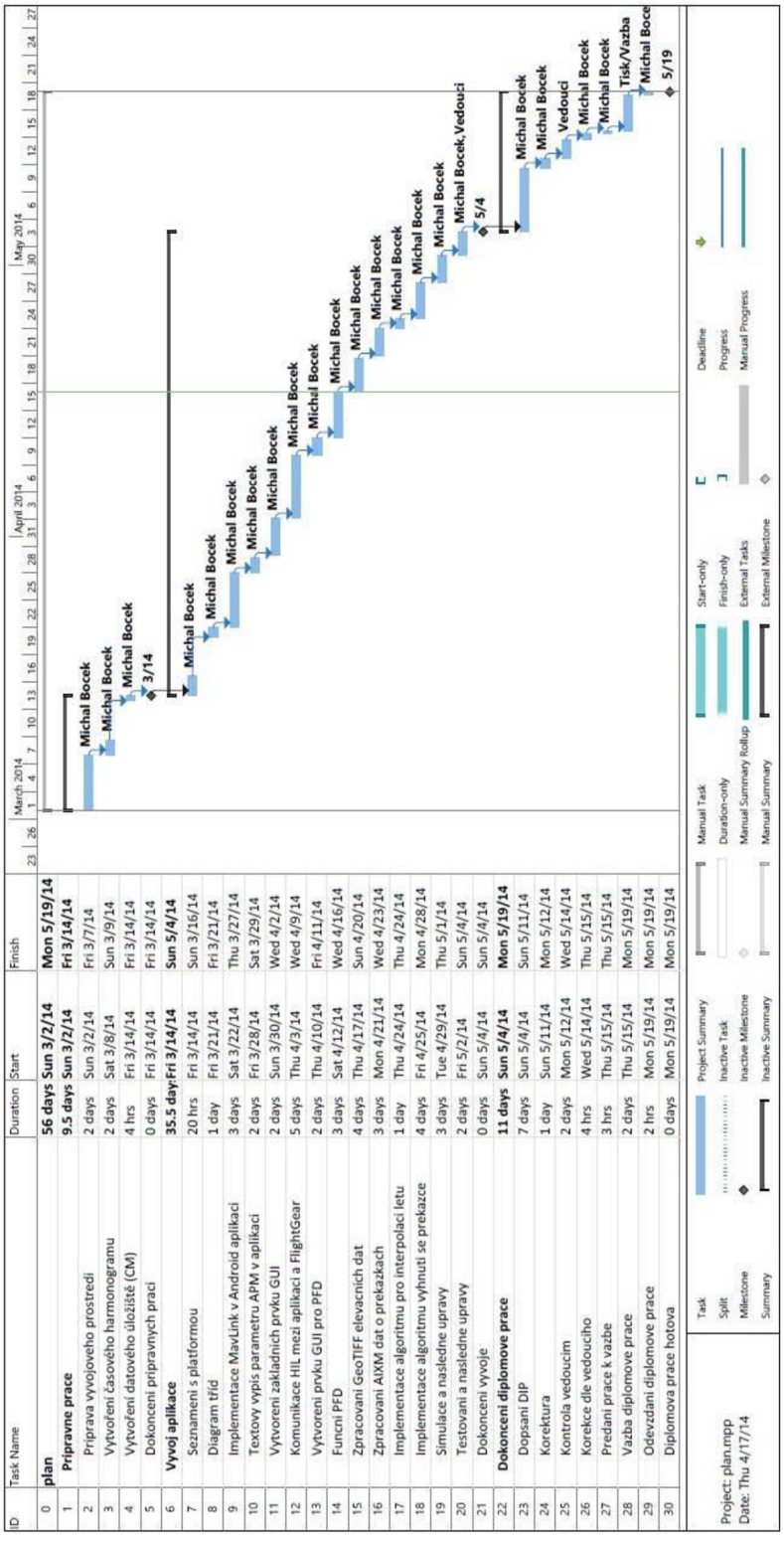
Složka	Popis obsahu složky
dokumentace_zdrojoveho_kodu	Autogenerovaná dokumentace ke zdrojovému kódu aplikace
instalacni_soubor	Instalační soubor APK
pisemna_zprava	Písemná zpráva ve formátech Office Open XML (.docx) a PDF
zdrojovy_kod	Zdrojový kód aplikace

Tabulka 1: Obsah přiloženého CD/DVD

Příloha 1. Příklad AIXM XML

```
<?xml version="1.0" encoding="UTF-8"?>
<message:AIXMBasicMessage gml:id="uniqueid">
  <message:hasMember>
    <aixm:VerticalStructure gml:id="urn.uuid.5f68d835-828c-4ccd-91b7">
      <gml:identifier codeSpace="urn:uuid:">5f68d835-828c</gml:identifier>
      <aixm:timeSlice>
        <aixm:VerticalStructureTimeSlice gml:id="VSRUCTTS1121">
          <aixm:type>ANTENNA</aixm:type>
          <aixm:group>NO</aixm:group>
          <aixm:part>
            <aixm:VerticalStructurePart gml:id="V-af4ee6d6">
              <aixm:horizontalProjection_location>
                <aixm:ElevatedPoint gml:id="p1121">
                  <gml:pos>52.36171388888889 -32.03756666666666</gml:pos>
                  <aixm:elevation uom="M">93</aixm:elevation>
                </aixm:ElevatedPoint>
              </aixm:horizontalProjection_location>
            </aixm:VerticalStructurePart>
          </aixm:part>
        </aixm:VerticalStructureTimeSlice>
      </aixm:timeSlice>
    </aixm:VerticalStructure>
  </message:hasMember>
</message:AIXMBasicMessage>
```

Příloha 2. Časový plán vývoje



ID	Task Name	Duration	Start	Finish
0	**plan**	**56 days**	**Sun 3/2/14**	**Mon 5/19/14**
1	**Pripravne prace**	**9.5 days**	**Sun 3/2/14**	**Fri 3/14/14**
2	Priprava vyvojoveho prostredi	2 days	Sun 3/2/14	Fri 3/7/14
3	Vytvoření časového harmonogramu	2 days	Sat 3/8/14	Sun 3/9/14
4	Vytvoření datového úložiště (CM)	4 hrs	Fri 3/14/14	Fri 3/14/14
5	Dokonceni pripravnych praci	0 days	Fri 3/14/14	Fri 3/14/14
6	**Vyvoj aplikace**	**35.5 days**	**Fri 3/14/14**	**Sun 5/4/14**
7	Seznameni s platformou	20 hrs	Fri 3/14/14	Sun 3/16/14
8	Diagram tříd	1 day	Fri 3/21/14	Fri 3/21/14
9	Implementace MavLink v Android aplikaci	3 days	Sat 3/22/14	Thu 3/27/14
10	Textovy vypis parametru APM v aplikaci	2 days	Fri 3/28/14	Sat 3/29/14
11	Vytvoření základních prvku GUI	2 days	Sun 3/30/14	Wed 4/2/14
12	Komunikace HIL mezi aplikaci a FlightGear	5 days	Thu 4/3/14	Wed 4/9/14
13	Vytvoření prvku GUI pro PFD	2 days	Thu 4/10/14	Fri 4/11/14
14	Funcni PFD	3 days	Sat 4/12/14	Wed 4/16/14
15	Zpracovani GeoTIFF elevacnich dat	4 days	Thu 4/17/14	Sun 4/20/14
16	Zpracovani AIXM dat o prekazkach	3 days	Mon 4/21/14	Wed 4/23/14
17	Implementace algoritmu pro interpolaci letu	1 day	Thu 4/24/14	Thu 4/24/14
18	Implementace algoritmu vyhnuti se prekazce	4 days	Fri 4/25/14	Mon 4/28/14
19	Simulace a nasledne upravy	3 days	Tue 4/29/14	Thu 5/1/14
20	Testovani a nasledne upravy	2 days	Fri 5/2/14	Sun 5/4/14
21	Dokonceni vyvoje	0 days	Sun 5/4/14	Sun 5/4/14
22	**Dokonceni diplomove prace**	**11 days**	**Sun 5/4/14**	**Mon 5/19/14**
23	Dopsani DIP	7 days	Sun 5/4/14	Sun 5/11/14
24	Korektura	1 day	Sun 5/11/14	Mon 5/12/14
25	Kontrola vedoucim	2 days	Mon 5/12/14	Wed 5/14/14
26	Korekce dle vedouciho	4 hrs	Wed 5/14/14	Thu 5/15/14
27	Predani prace k vazbe	3 hrs	Thu 5/15/14	Thu 5/15/14
28	Vazba diplomove prace	2 days	Thu 5/15/14	Mon 5/19/14
29	Odevzdani diplomove prace	2 hrs	Mon 5/19/14	Mon 5/19/14
30	Diplomova prace hotova	0 days	Mon 5/19/14	Mon 5/19/14

Příloha 3. Styl zápisu kódu

- **Pojmenování proměnných tříd**
  Nestatické a neveřejné proměnné třídy začínají na *m*.
  Statické proměnné třídy začínají na *s*.
  Veřejné statické *final* proměnné třídy (konstanty) jsou pojmenovány VELKYMI_PISMENY_S_PODTRZITKY.
  Ostatní proměnné třídy začínají malým písmenem.

- **Pojmenování lokálních proměnných**
  Lokální proměnné jsou pojmenovány stylem lowerCamelCase. Jednopísmenných proměnných s výjimkou dočasných proměnných a proměnných pro smyčku nepoužívá.

- **Pojmenování metod**
  Metody jsou pojmenovány stylem lowerCamelCase. Jména metod jsou typicky slovesa, například *sendMessage* nebo *stop*.

- **Pojmenování tříd**
  Třídy jsou pojmenovány stylem UpperCamelCase. Jména metod jsou typicky podstatná jména, například *Character* nebo *ImmutableList*.

- **Pojmenování ID Android zdrojů**
  ID Android zdrojů jsou pojmenovány malými písmeny s podtržítky. ID jsou generována při kompilaci mimo jiné i z názvu souborů jednotlivých zdrojů, tudíž se toto pravidlo pojmenování vztahuje i na názvy souborů zdrojů, tedy například *main_activity_fragment.xml* nebo *ic_launcher.png*.

- **Veřejné proměnné třídy**
  Veřejné proměnné třídy se s výjimkou konstant nepoužívají. Proměnné třídy jsou buď soukromé nebo *protected*. Kde nejsou soukromé nebo *protected* proměnné viditelné, použijí se k přístupu k nim metody typu *getter* a *setter*.

- **Zkratky**
  Zkratky se v případě lowerCamelCase a UpperCamelCase stylů pojmenování upravují tak, aby obsahovaly nejvýše jedno velké písmeno. Příklad pro UpperCamelCase:

Původní forma	XML HTTP request
Správné zkrácení	*XmlHttpRequest*
Nesprávné zkrácení	*XMLHTTPRequest*

- **Použití statických metod**
  Foo aFoo = ...;
  Foo.aStaticMethod(); // správně
  aFoo.aStaticMethod(); // špatně

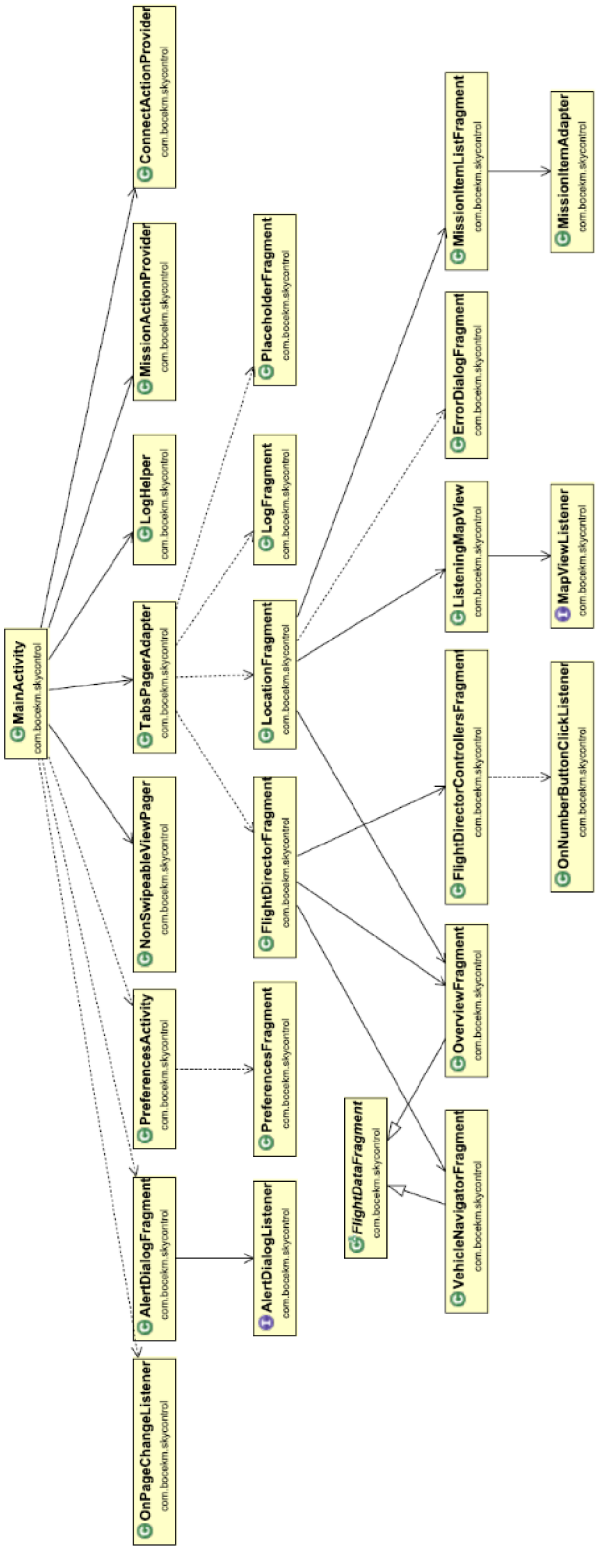
- **Javadoc**
  Javadoc jsou speciálně formátované komentáře, jejichž strojovým zpracováním lze získat dokumentaci kódu. Minimálně je Javadoc komentář použit pro každou veřejnou třídu, pro každou veřejnou metodu a pro každý *protected* prvek třídy.

- **Deklarace polí**
  Pole se deklaruje odlišně od stylu jazyka C. Hranaté závorky jsou součástí typu, ne proměnné. Tedy například *String[] args* a ne *String args[]*.

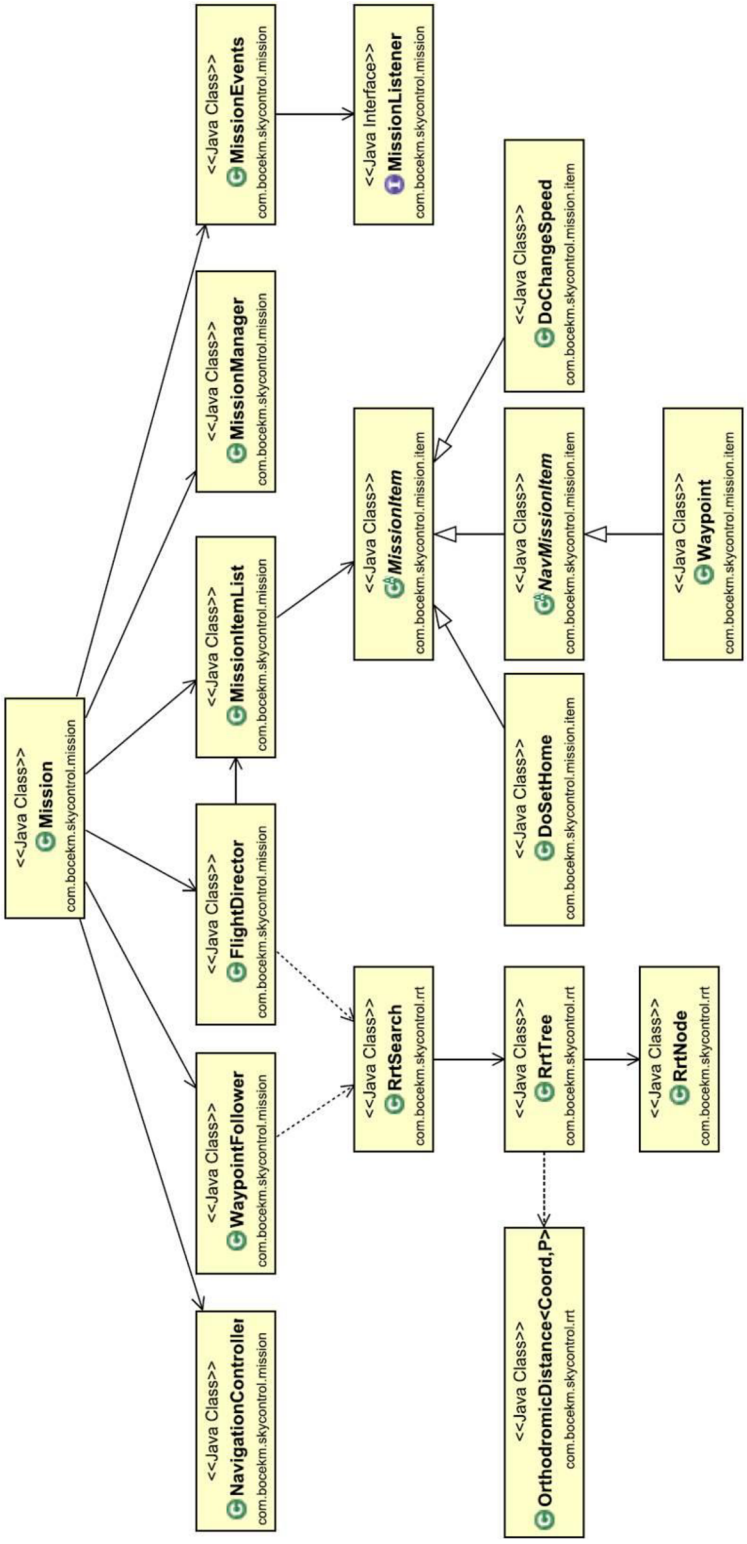
- **Licenční ujednání pro Java soubory**
  Každý zdrojový soubor jazyka Java obsahuje následující hlavičku.

```
/*
 * Copyright (c) 2014, Michal Bocek, All rights reserved
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not use this file except
 * in compliance with the License. You may obtain a copy of the License
at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
distributed under the License
 * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
OF ANY KIND, either express
 * or implied. See the License for the specific language governing
permissions and limitations under
 * the License.
 */
```

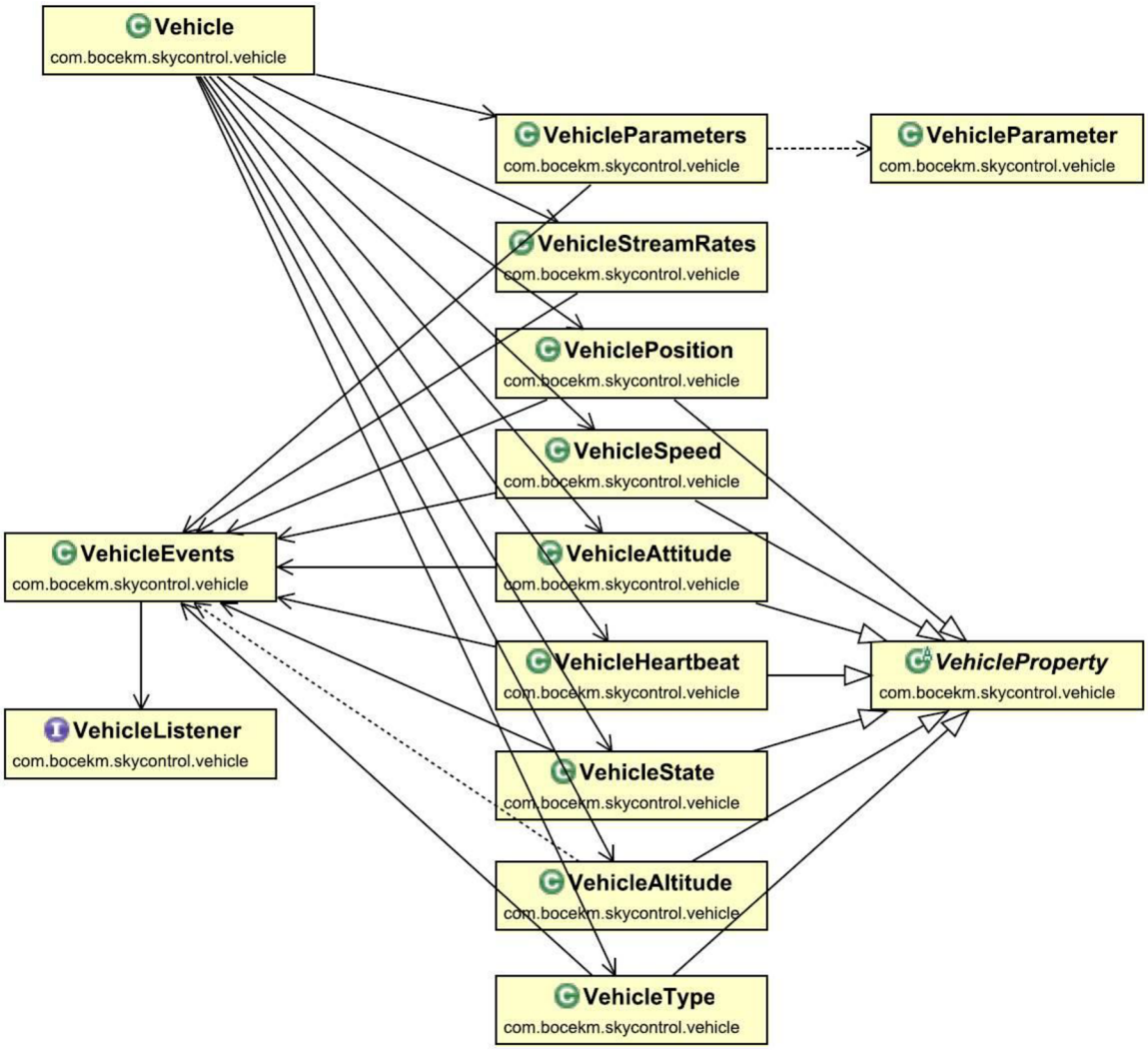
Příloha 4. Třída *MainActivity*



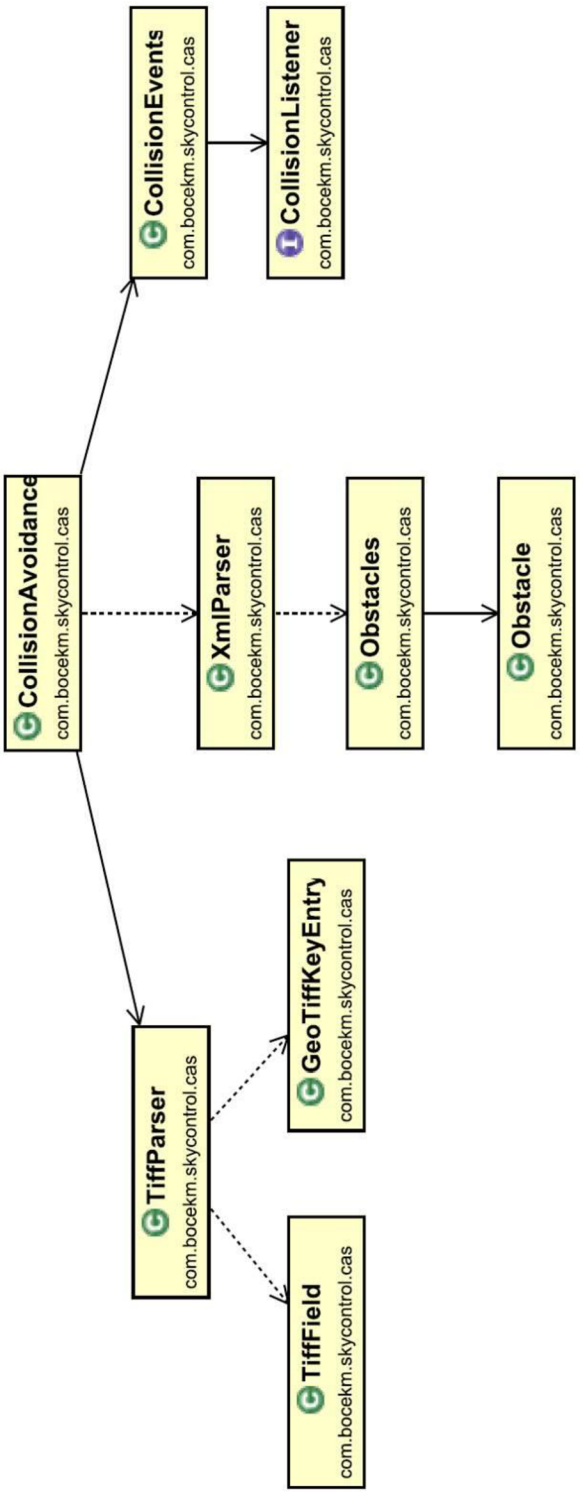
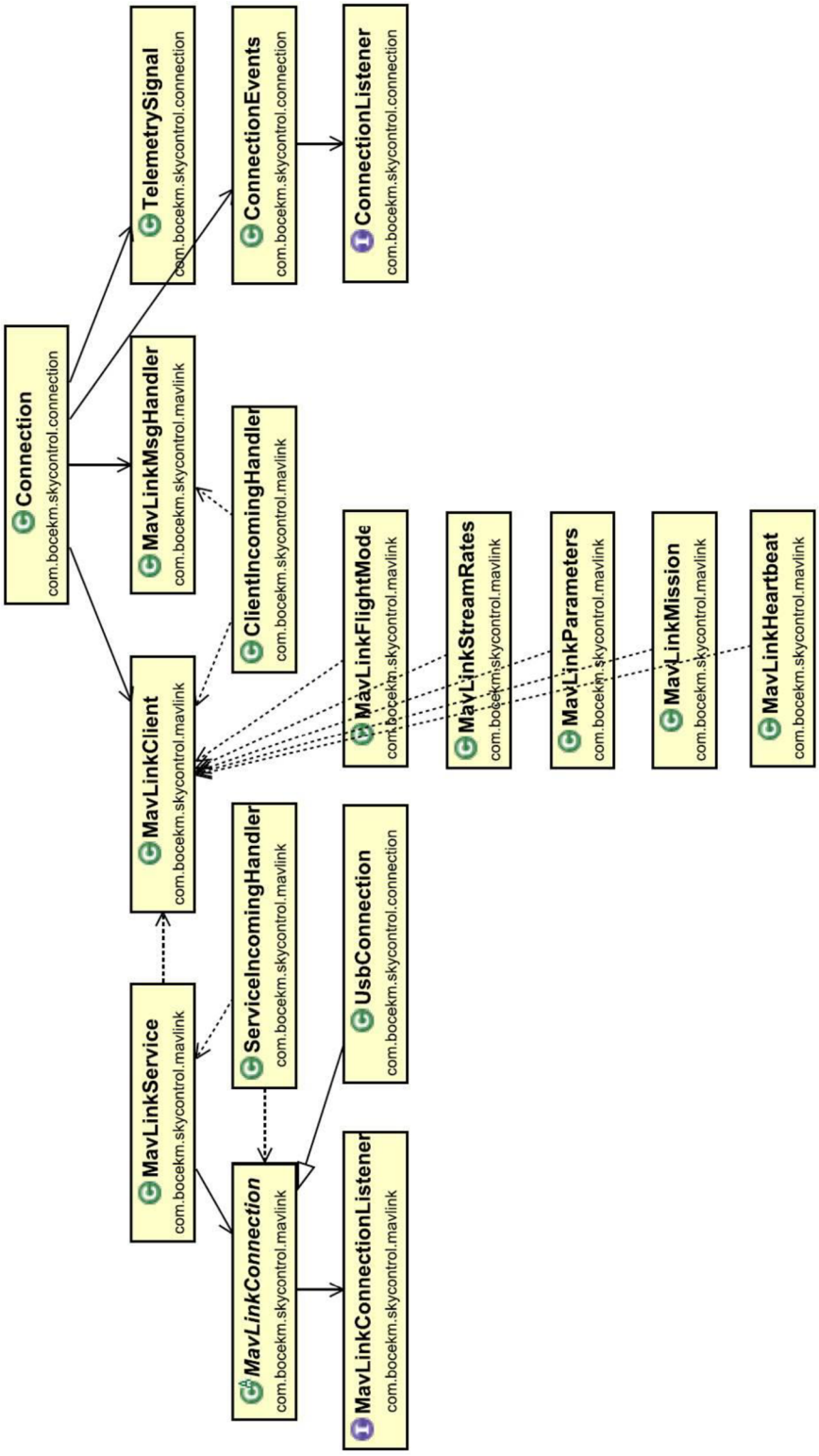
Příloha 5. Třída *Mission*



Příloha 6. Třída *Vehicle*



Příloha 7. Třída CollisionAvoidance



Příloha 8. Třída *Connection*