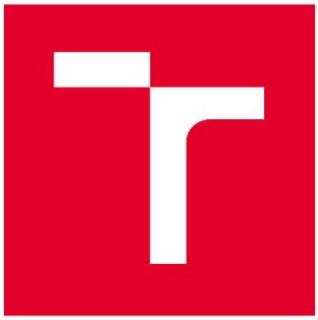
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Vojtěch Hamáček



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

VÝVOJ BEZPILOTNÍHO PROSTŘEDKU PRO AUTONOMNÍ MISE
THE DEVELOPMENT OF AUTONOMOUS UNMANNED AIRCRAFT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Vojtěch Hamáček**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Jílek, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Vojtěch Hamáček ***ID:*** 159649

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Vývoj bezpilotního prostředku pro autonomní mise

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je zprovoznit bezpilotní letadlo M100 s řídicí jednotkou Pixhawk a open-source programovým vybavením s ohledem na autonomní mise.

1. Seznamte se s open-source řídicími programy Arducopter a PX4, vhodný z nich zprovozněte na platformě M100/Pixhawk. Proveďte potřebné hardwarové úpravy platformy.
2. Důkladně se seznamte s možnostmi programového řízení zvoleného řešení a navrhněte programovou architekturu celého řetězce řízení využívající dostupné knihovny a nástroje. Zvolené řešení musí umožňovat integraci do ROS (Robot Operating System).
3. Vyberte a zprovozněte nástroj pro simulaci vysokoúrovňového řízení letu bezpilotního letadla umožňující odladit řídicí algoritmy před samotným letem.
4. Pro dané programové řešení naprogramujte jednoduchou autonomní misi, její funkčnost demonstrujte pomocí simulace.
5. S ohledem na pokyny vedoucího a dostupnost potřebného hardwaru proveďte testování programového řešení na reálném bezpilotním letadle včetně provedení simulované mise.

DOPORUČENÁ LITERATURA:

PYO, YoonSeok, HanCheol CHO, RyuWoon JUNG a TaeHoon LIM. ROS Robot Programming. Republic of Korea: ROBOTIS Co., 2017. ISBN 979-11-962307-1-5.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 17.5.2021

Vedoucí práce: Ing. Tomáš Jílek, Ph.D.
Konzultant: Ing. Petr Gábrlík

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Cílem diplomové práce je upravit komerčně vyráběný dron DJI Matrice 100 a nahradit jeho řídicí jednotku jednotkou Pixhawk a jeho příslušenstvím, které jsou vyvíjeny pod svobodnou licencí. Následně se zabývá výběrem vhodného firmwaru pro Pixhawk, taktéž vyvíjeného pod svobodnou licencí a jeho konfigurací na daném zařízení. Další část práce se věnuje možnostem použití Robotického operačního systému, zkráceně ROS, resp. jeho knihoven MAVROS na palubním počítači Raspberry Pi. S využitím MAVROS zkoumá možnosti programového řízení letu dronu a to jak v prostředí simulace, tak i v prostředí reálném.

Klíčová slova

Dron, DJI, Matrice 100, PX4, Ardupilot, Raspberry Pi, Pixhawk, ROS, MAVROS, MAVLink

Abstract

The aim of this thesis is to modify commercially produced drone DJI Matrice 100 and replace its original control unit by open source Pixhawk and its accessories. Subsequently, it deals with the selection of suitable open source firmware for Pixhawk and its configuration on the device. Another part is dedicated to the possibilities of using the Robotic Operating System (ROS) and its Mavros libraries on the onboard computer Raspberry Pi. By using Mavros, it examines the possibilities of drone flight control, both in the simulation environment and in the real environment.

Keywords

Drone, DJI, Matrice 100, PX4, Ardupilot, Raspberry Pi, Pixhawk, ROS, MAVROS, MAVLink

Bibliografická citace

HAMÁČEK, Vojtěch. *Vývoj bezpilotního prostředku pro autonomní mise.* Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/134780. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Tomáš Jílek.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Vojtěch Hamáček*
VUT ID studenta:	*159649*
Typ práce:	*Diplomová práce*
Akademický rok:	*2020/21*
Téma závěrečné práce:	*Vývoj bezpilotního prostředku pro autonomní mise*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 17. května 2021 ----------

podpis autora

Poděkování

Děkuji Ing. Petru Gábrlíkovi za konzultace, odbornou pomoc a cenné rady při vypracování mé diplomové práce.

V Brně dne: 17. května 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

První část práce se věnuje studiu komerčního dronu Matrice 100 (zkráceně M100) od společnosti DJI. V rámci semestrální práce z předmětu robotika byl dron upraven s využitím řídicí jednotky Pixhawk 1, která je vyvíjena pod svobodnou licencí. Tato práce vychází z původních úprav dronu a využívá novější verzi Pixhawku 2.1. Dále je zde řešena implementace palubního jednodeskového počítače v podobě Raspberry Pi, který bude sloužit k pokročilému řízení dronu.

Další část se zabývá rozborem dvou nejrozšířenějších projektů s otevřeným kódem firmwaru pro Pixhawk. Těmito projekty jsou Ardupilot a PX4. Kapitola srovnává rozdíly mezi nimi a posuzuje vhodnost použití některého z nich pro řízení dronu.

Dále se práce zaměřuje na použití vybraného firmwaru na Pixhawku a jeho konfigurací. Zkoumá možnosti jeho propojení a programového řízení pomocí palubního počítače Raspberry Pi. Věnuje se vytvoření linuxového terminálového skriptu, který zajistí instalaci všech potřebných softwarových balíčků a Robotického operačního systému (ROS) na Raspberry Pi pro testování v reálném prostředí. Dále popisuje obdobný skript pro instalaci na počítači, sloužící pro testování v simulacích.

Věnuje se také praktickému otestování funkčnosti ROSu, v simulovaném prostředí aplikace Gazebo, s využitím testovacích ROS programů pro řízení dronu. Uvádí ukázkové ROS programy v jazyce C++ i v Pythonu a to ve variantě jak pro Ardupilot firmware, tak i pro PX4 firmware. Také porovnává odlišnosti v komunikaci mezi ROSem a oběma firmwary.

Poslední část práce se věnuje tvorbě vlastní rozšiřující knihovny v Pythonu, využívající ROSu. Její funkčnost ověřuje dle pokynů vedoucího práce v simulaci i na reálném dronu. Cílem je úspěšně ověřit jednoduché řízení letu dronu po naprogramované dráze.

Vzhledem k nepříznivé epidemické situaci v ČR a uzavření vysokých škol, způsobené vlivem šíření koronaviru SARS-CoV-2, nebylo možné pracovat na dronu M100, který se nacházel v laboratoři FEKT VUT. Z tohoto důvodu bylo přistoupeno k testování softwarové části práce na soukromém dronu, vybaveném starší verzí Pixhawku 1 s firmwarem Ardupilot, GPS a Raspberry Pi 4B. Stručné shrnutí hardwaru dronu popisuje kapitola 3.1. Řešení z pohledu použitého firmwaru a jeho konfigurace popisuje kapitola 0. Rozborem tohoto dronu se práce zabývá pouze okrajově, protože není součástí zadání. Přesto je možné využít ho díky kompatibilitě zvoleného řešení k otestování funkčnosti programů napsaných pro ROS, které vznikly v rámci práce.

1. HARDWAROVÁ REALIZACE DRONU DJI M100

Kapitola se věnuje popisu komerčního dronu od společnosti DJI Matrice 100 [2], který je dle výrobce určen pro vývojáře, kteří jej mohou modifikovat a použít ve svých aplikacích. Přesto jsou možnosti úprav tohoto výrobku pro účely FEKT VUT v Brně značně omezené. Jedná se především o velmi uzavřený ekosystém jednoho výrobce s omezenými možnostmi zásahu do zdrojového kódu dronu. Další nevýhodou je, že obdobné výrobky společnosti DJI mají na trhu velmi krátký životní cyklus, čímž se komplikuje následná údržba dronu a také jeho podpora ze strany výrobce.

Z těchto důvodů byl proveden průzkum a výběr vhodné řídicí elektroniky pod svobodnou licencí, kterou by mohla být nahrazena originální řídicí elektronika od společnosti DJI. Předcházející semestrální práce [3] se zabývá popisem a srovnáním platformy DJI Matrice 100 s jednou z open source hardwarových platforem – Pixhawk 1. Tato práce vychází z původního rozboru platformy Pixhawk 1 a zabývá se implementací novější verze Pixhawk 2.1 (někdy také označován jako Pixhawk Cube).

1.1 DJI Matrice 100

Jedná se o dron konstrukce typu quadrocopter (viz obrázek 1.1), která se typicky vyznačuje hlavním nosným rámem ve tvaru písmene „X“. Na každém konci nosných ramen se nachází motor s vrtulí. Řídící elektronika a akumulátor se nacházejí ve středu konstrukce. Díky tomuto rozložení je dron stabilní a vyvážený vůči svému středu.

Obrázek 1.1 Dron DJI Matrice 100 s originální elektronikou od výrobce, zdroj: [2]

Tabulka 1.1 popisuje základní parametry DJI Matrice 100, které jsou určeny především použitým pohonným systémem a akumulátorem. V případě nahrazení originální řídící elektroniky za jinou, zůstanou tyto základní parametry zachovány.

Tabulka 1.1 Základní parametry DJI Matrice 100, zdroj: [2]

Základní parametry	
Úhlopříčka ramen	650 mm
Hmotnost s baterií	2355 g
Maximální hmotnost	3600 g
Maximální rychlost stoupání	5 ms^{-1}
Maximální rychlost klesání	4 ms^{-1}
Maximální náklon	35°
Parametry akumulátoru	
Model	TB47D
Kapacita	4500 mAh
Typ	Li-Pol
Počet článků	6
Nominální napětí	22,2 V
Hmotnost	600 g
Minimální pracovní teplota	-10 °C
Maximální pracovní teplota	40 °C
Pohonný systém	
Motory	DJI 3510
Vrtule	DJI 1345s
ESC	DJI E SERIES 620D

1.2 Řídicí jednotka Cube – Pixhawk 2.1

Na trhu existuje velké množství hardwarových řešení pro realizaci nejrůznějších kolových a vrtulových bezpilotních prostředků. Lze zmínit například softwarový projekt LibrePilot [4], který zastřešuje vlastní hardwarové řešení v podobě OpenPilot projektu. Obdobné řešení kombinace svobodného hardwaru a softwaru nabízí projekt Paparazzi UAV [5] a v neposlední řadě existuje také projekt Pixhawk [6].

Tato práce je založena na posledním zmíněném projektu. Konkrétně využívá hardware Cube - Pixhawk 2.1, který je zastřešen pod projektem Dronecode od Dronecode Project, Inc. [7]. Jedná se o velmi obsáhlý ekosystém hardwarových a softwarových řešení pod svobodnou licencí BSD. Tento projekt je vyvíjen ve spolupráci s technologických konsorciem The Linux Foundation. Dále je Dronecode podporován řadou technologických společností, v jejichž čele stojí společnosti 3DR, YUNEEC, NXP, Microsoft a mnoho dalších.

Obrázek 1.2 Řídicí jednotka pro drony, Cube – Pixhawk 2.1, zdroj: [8]

Následující tabulka 1.2 popisuje některé ze základních parametrů použité řídicí elektroniky Pixhawku 2.1. Obsahuje celkem tři měřicí inerciální jednotky, přičemž dvě z nich jsou vybaveny mechanickým tlumením proti vibracím. Třetí není odtlumená a slouží jako referenční/záložní jednotka.

Tabulka 1.2 Vybrané základní parametry Pixhawku 2.1, zdroj: [8]

Procesor	32bit STM32F427 Cortex-M4F® core with FPU
Kmitočet procesoru	168 MHz
RAM	256 kB
Flash	2 MB
PWM výstupů	14
Rozhraní	5× UART, I2C, SPI,2× CAN, PPM, 3.3v ADC, USB
Napájení	4,8 V až 5,4 V
Akcelerometry	3
Gyroskopy	3
Magnetometry	3
Barometry	2

Pixhawk 2.1 podporuje GNSS modul Here2 [9], se kterým komunikuje prostřednictvím sériové linky UART a sběrnice I2C. Modul je schopný přijímat signál až ze tří družicových systémů najednou. Mezi tyto GNSS systémy patří GPS, Galileo, GLONASS a BeiDou. Dále obsahuje vlastní IMU jednotku a barometr. Díky podpoře kinematiky reálného času je možné získat polohu s přesností v řádu centimetrů. Modul dále obsahuje bezpečnostní tlačítko s barevnou indikační kontrolkou, sloužící k aktivaci dronu.

Napájení zajišťuje dodávaný napájecí modul POWER Brick Mini, který je schopný pracovat s až osmi-článkovým Li-Pol akumulátorem.

1.3 Palubní počítač Raspberry Pi 3B+

Jako palubní počítač pro řízení dronu byl zvolen jednodeskový počítač Raspberry Pi 3B+ [10] od společnosti Raspberry Pi Foundation. Lze na něm zprovoznit řadu linuxových distribucí, přičemž hlavní podporovaná distribuce je Raspbian, resp. nově Raspberry Pi OS, která je volně vyvíjena linuxovou komunitou.

Je možné využívat Raspberry i jako plnohodnotný desktopový počítač společně s monitorem, klávesnicí a myší. Nicméně v této práci toho nebude využíváno.

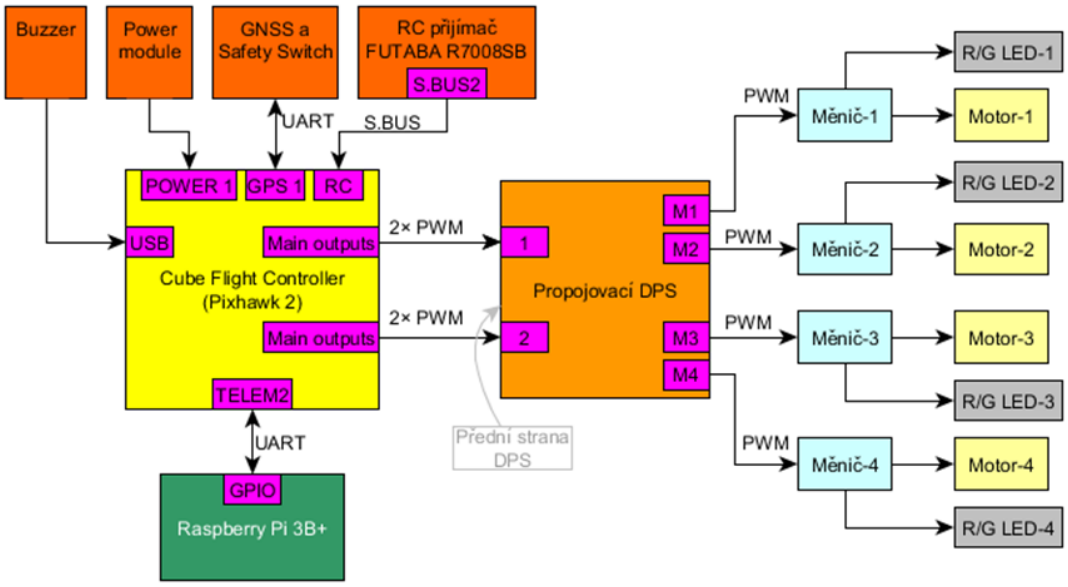
Tabulka 1.3 Základní parametry jednodeskového počítače Raspberry Pi 3B+

Procesor	Broadcom BCM2837B0
Počet jader procesoru	4
Kmitočet procesoru	1,4 GHz
RAM	1 GB
Napájení	5 V/2,5 A
Rozhraní	4× USB 2.0, 40-pin GPIO konektor, HDMI, Gigabitový Ethernet, CSI, DSI, Micro SD port
WiFi	2.4GHz a 5GHz IEEE 802.11.b/g/n/ac
Bluetooth	v4.2

1.4 Modifikace DJI Matrice 100

Jak již bylo zmíněno na začátku kapitoly, původní řídicí elektronika dronu byla nahrazena elektronikou pod svobodnou licencí. Konkrétně se jedná o Pixhawk 2.1 spolu s GNSS modulem Here2 a napájecím modulem POWER Brick Mini. Dále také jednodeskovým počítačem Raspberry Pi 3B+. Možnost dálkového ovládání dronu zajišťuje RC souprava s RC přijímačem FUTABA R7008SB na straně dronu.

Základní blokové zapojení popisuje obrázek 1.3. Hlavní změnou bylo odpojení originální řídicí jednotky od propojovací DPS – odpojeno od konektorů označených číslicemi 1 a 2. Následně k nim byl připojen Pixhawk 2.1, který je prostřednictvím těchto konektorů propojen s ESC (z angl. Electric speed controller) měniči motorů. Pixhawk nastavuje otáčky jednotlivých motorů prostřednictvím střídy PWM signálů zasílaných ESC měničům. Toto propojení popisuje tabulka 1.4. Zde je zapotřebí brát ohled na skutečnost, že signálová zem ESC měničů (GND) není propojena s výkonovou zemí dronu. Propojovací DPS spojuje signálové země pro motory M1 a M4 a zvlášť pro M2 a M3. Výsledné propojení zemí obou dvojic motorů a výkonové země dronu, je realizováno prostřednictvím Pixhawku 2.1 na portech označených jako *Main outputs*.



Obrázek 1.3 Blokové schéma zapojení dronu s použitím Pixhawku 2.1

Tabulka 1.4 Zapojení pinů propojovacích kabelů Pixhawku 2.1 a propojovací DPS dronu (PWM signály pro ovládání motorů), zdroj: [3]

Kabel číslo (číslo konektoru propojovací DPS)	**Popis**	**Výstupy Pixhawku (Main outputs)**	**Propojovací DPS**
1	**M1**	1S	3 a 5
	M4	4S	6 a 8
	GND	-	4 a 7
2	**M2**	2S	3 a 5
	M3	3S	6 a 8
	GND	-	4 a 7

Dále byl k Pixhawku připojen RC přijímač (viz tabulka 1.5). Propojení mezi Raspberry Pi a Pixhawkem je zajištěno prostřednictvím sériové linky UART (viz tabulka 1.6). Pomocí tohoto propojení bude realizována veškerá komunikace mezi oběma zařízeními.

Tabulka 1.5 Propojení pinů konektorů mezi RC přijímačem a Pixhawkem 2.1, zdroj: [3]

RC přijímač (S.BUS2 - číslováno shora)	**Pixhawk (RC)**
3	-
2	+
1	S

Tabulka 1.6 Propojení pinů konektorů mezi Pixhawkem 2.1 a Raspberry Pi 3B+, zdroj: [3]

Pixhawk 2 (TELEM2)		**RPi 3B+ (GPIO)**	
2	Tx	10	Rx
3	Rx	8	Tx
6	GND	6	GND

2. FIRMWARE PRO ŘÍDICÍ JEDNOTKU PIXHAWK

Zvolený hardware Pixhawk 2.1 je podporován především dvěma firmwary. Jedním z nich je firmware Ardupilot [11], který podporuje širokou řadu bezpilotních prostředků, simulátorů a senzorů. Druhou podobně rozsáhlou platformou je firmware PX4 [12], který je společně s projektem Pixhawk zastřešen pod projektem Dronecode s podporou The Linux Foundation.

Obě řešení jsou spolu často srovnávána, ale vzhledem k tomu, že se jedná o dva stále vyvíjené projekty, je srovnání díky jejich proměnlivosti velmi obtížné. V současné době neexistuje žádné odborné srovnání obou platforem.

Jak PX4, tak i Ardupilot podporují stavbu různorodých bezpilotních prostředků. Mezi ně patří letadla, drony, kolové roboty, ale i lodě nebo ponorky. Jejich výhodou je také skutečnost, že spadají pod svobodné licence a v rámci nich je možné je upravovat a dále šířit. Další výhodou je vysoká vzájemná kompatibilita dílčích řešení obou projektů. Tím je myšlena skutečnost, že například firmware PX4 Dronecodu je schopen skrze komunikační protokol MAVLink spolupracovat s konkurenčním řešením pozemní stanice od Ardupilotu, resp. s Mission Plannerem. Díky jednotnému komunikačnímu protokolu je u obou firmwarů obdobně kompatibilní i použití Robotického operačního systému, který je schopen komunikovat s vybraným firmwarem a povelovat ho.

2.1 Projekt Dronecode a firmware PX4

PX4 je součástí projektu Dronecode (viz obrázek 2.1). Od roku 2009 byl jeho firmware vyvíjen společně s hardwarem Pixhawk v rámci studentského projektu Lorenze Meiera na Spolkové vysoké technické škole v Curychu [6]. Jedná se o otevřený kód pod svobodnou 3-bodovou BSD licencí, který je snadno rozšiřitelný a modifikovatelný pro potřeby vývojářů [13].

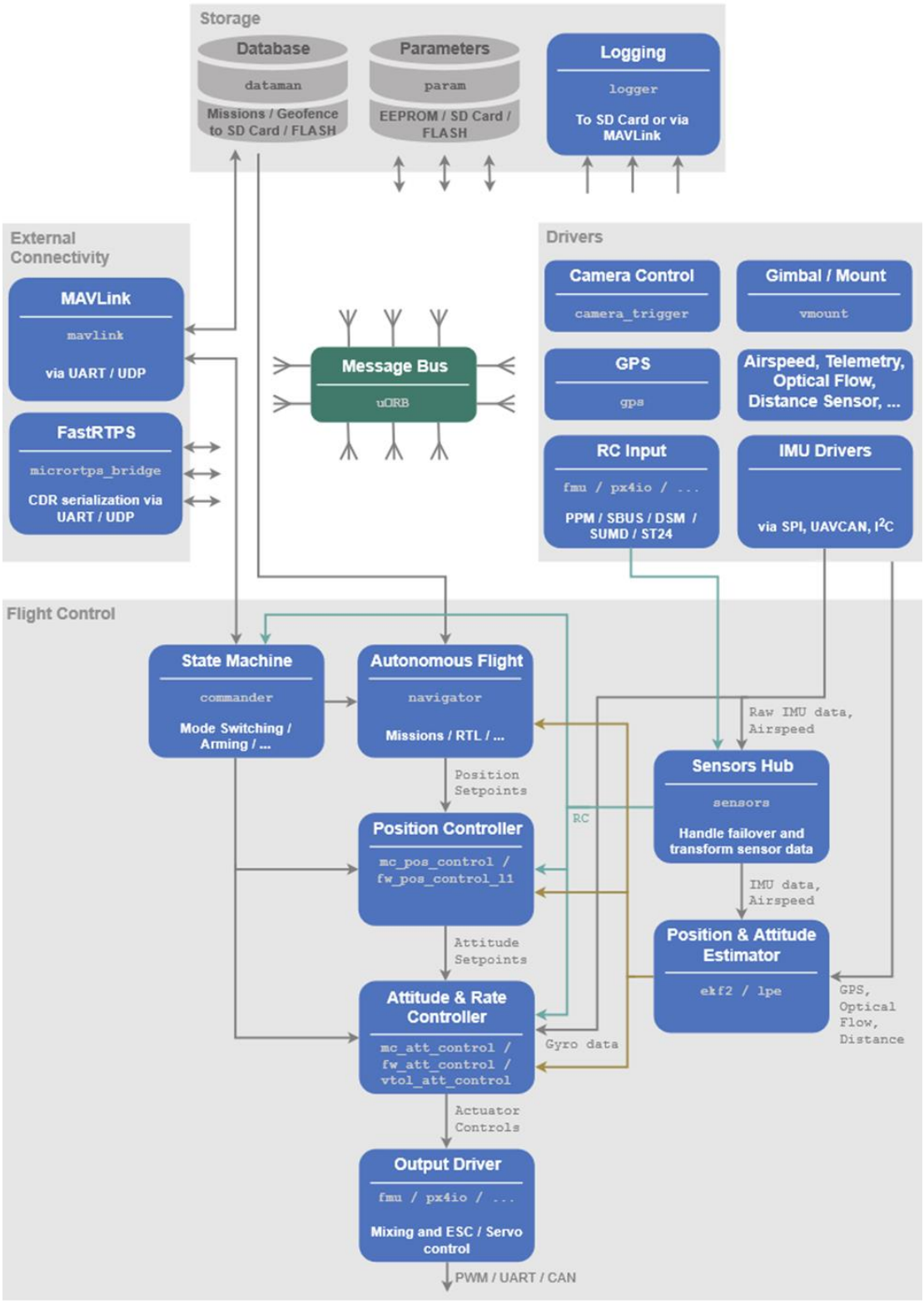
Společně s PX4 je vytvářena aplikace QGroundControl. Jedná se o aplikaci pro pozemní počítač, která umožňuje pokročilou konfiguraci dronu, čtení telemetrie a plánování misí. Dále je zde obsaženo SDK MAVSDK a ROS, což je soubor nástrojů pro tvorbu vlastních softwarových řešení pro řízení dronu. Využívá se zde komunikační protokol MAVlink, sloužící pro komunikaci Pixhawku s palubním počítačem nebo pozemní stanicí. Případně je možné využít protokolu UAVCAN pro komunikaci pomocí sběrnice CAN.

Obrázek 2.1 Přehled projektu Dronecode, zdroj: [13]

2.1.1 Architektura firmwaru PX4

Obrázek 2.2 ukazuje strukturu projektu PX4, kde blok *Storage* zajišťuje obsluhu pamětí. Blok *External Connectivity* má za úkol komunikaci s vnějším světem, především s palubním počítačem nebo pozemní stanicí. Tato komunikace je řešena především prostřednictvím protokolu MAVlink, případně FastRTPS. Poslední blok *Drivers* obsahuje ovladače pro senzory a další periferie. Tyto bloky společně představují kód označovaný jako middleware PX4, pracující na operačním systému reálného času NuttX. Nicméně podporuje i jiné operační systémy jako Linux, macOS nebo QuRT. [14]

Nad middlewarem PX4 se nachází druhá část řešení PX4, která se nazývá *Flight Control*, resp. letový kód. Jeho úkolem je zajistit fúzi dat ze snímačů a na jejich základě zajišťuje regulaci akčních členů. V tomto případě se jedná o tah motorů a o stabilizaci dronu.



Obrázek 2.2 Softwarová architektura firmwaru PX4, zdroj: [14]

2.2 Projekt Ardupilot a firmware Ardupilot - APM

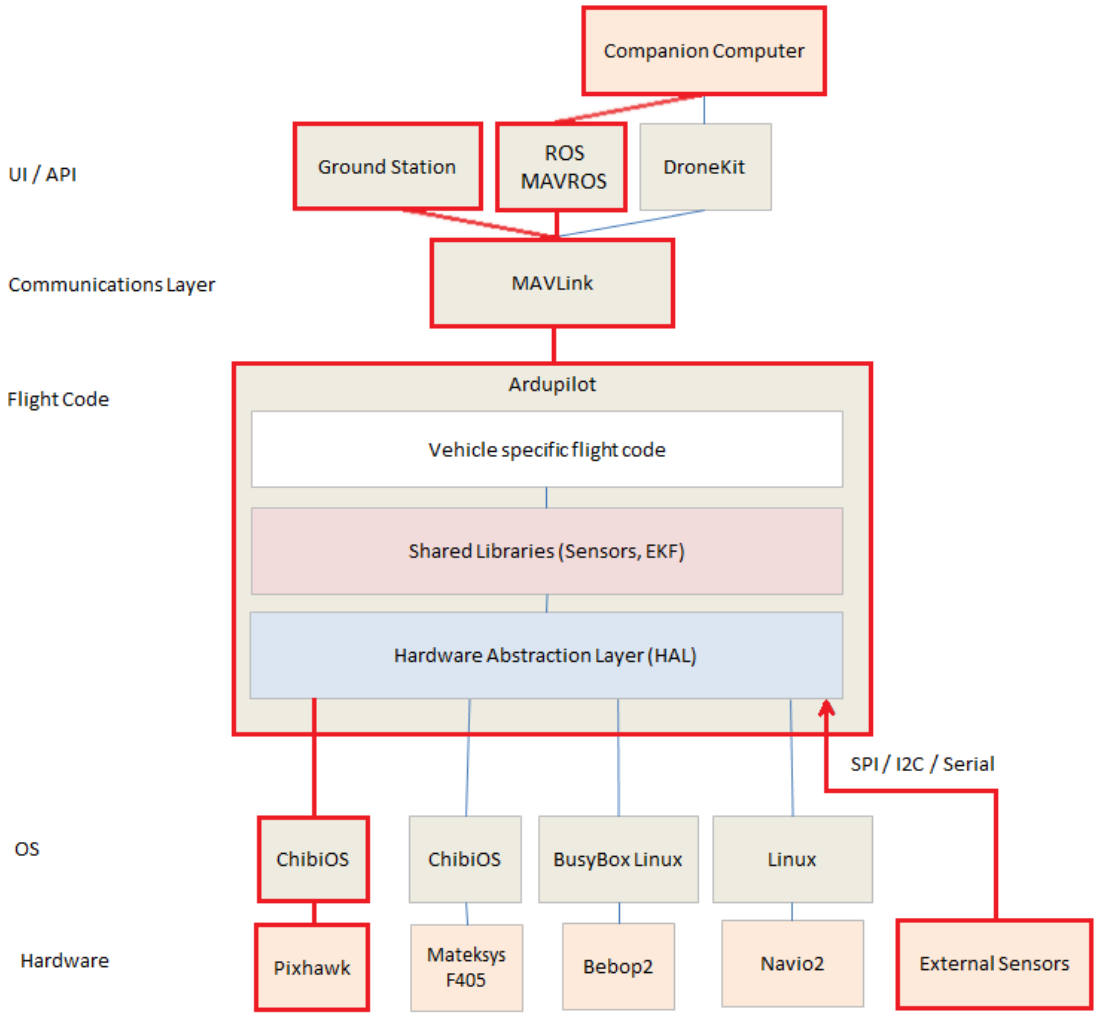
Jedná se o projekt vyvíjený pod svobodnou licencí GPLv3. Začátky projektu sahají do roku 2007 [17]. V té době se jednalo především o software pro zařízení založené na platformě Arduino s mikrokontroléry ATmega. Z této doby pochází i druhý používaný název firmwaru - APM (zkratka z ArduPilotMega). Nicméně v dnešní době není použití APM omezeno pouze na původní platformu, ale i na celou řadu jiných řešení, včetně Pixhawka, který obsahuje 32bitový ARM procesor. Podmnožinou projektu Ardupilot je projekt Arducopter, který se zaměřuje na multikoptéry.

Od roku 2014 až do roku 2016 byl projekt Ardupilot součástí organizace Dronecode a PX4 firmwaru. V jeho rámci zajišťoval úlohu letového kódu realizujícího stabilizaci rotačních pohybů, výšky a samotný let na základě dat ze senzorů dronu. Díky filozofickým neshodám mezi vývojáři Ardupilotu a Dronecodu došlo roku 2016 k jejich oddělení. Jednalo se především o neshody ohledně použité licence, pod kterou bude projekt spravován. Původním autorům také vadilo, že platinoví členové (sponzoři) projektu Dronecode chtěli rozhodovat o směřování celého projektu a chtěli nad ním mít plnou kontrolu. Lidé stojící za projektem Ardupilot chtěli naopak, aby směřování projektu určovala širší komunita uživatelů a přispěvatelů do zdrojových kódů [15].

Ardupilot následně až do roku 2020 stále využíval middleware PX4, běžící na operačním systému reálného času NuttX a na hardwaru Pixhawk. V průběhu roku 2020 došlo k uvolnění nové verze firmwaru APM, který je již založen na operačním systému reálného času ChibiOS. Tímto krokem došlo k oddělení jedné z posledních společných vrstev obou konkurenčních firmwarů. Co se týče letového kódu, tak obě platformy již od svého rozdělení využívají svá vlastní řešení. Poslední společnou vrstvou zůstává komunikační vrstva MAVLink, umožňující komunikaci s palubním počítačem a pozemní stanicí [16]. Celkový přehled architektury projektu je vidět na obrázku 2.3.

Jak již bylo zmíněno v úvodu kapitoly, totožná komunikační vrstva, i přes značné rozdíly v nižších vrstvách obou firmwarů, zajišťuje téměř plnou kompatibilitu z pohledu komunikace s palubním počítačem a pozemní stanicí. Nad komunikační vrstvou lze tedy volit jak mezi aplikacemi a softwarovými nástroji pro projekt Ardupilot, tak i mezi těmi pro Dronecode.

Jako hlavní aplikaci pozemní stanice využívá projekt Ardupilot aplikaci Mission Planner. Aplikace slouží ke konfiguraci firmwaru nahraného v zařízení, ke čtení telemetrie a k plánování autonomních letových misí. Dále Ardupilot zastřešuje projekt DroneKit, který obsahuje SDK a webové API pro vývoj aplikací pro řízení dronů. Zároveň díky podpoře komunikačního protokolu MAVLink ROSem, je možné použít jej namísto DroneKitu k ovládání dronu, čehož je využito v této práci.



Obrázek 2.3 Základní struktura Ardupilotu (červeně jsou vyznačeny bloky využívané v této práci), obrázek byl upraven, zdroj originálu: [16]

2.2.1 Architektura firmwaru Ardupilot - APM

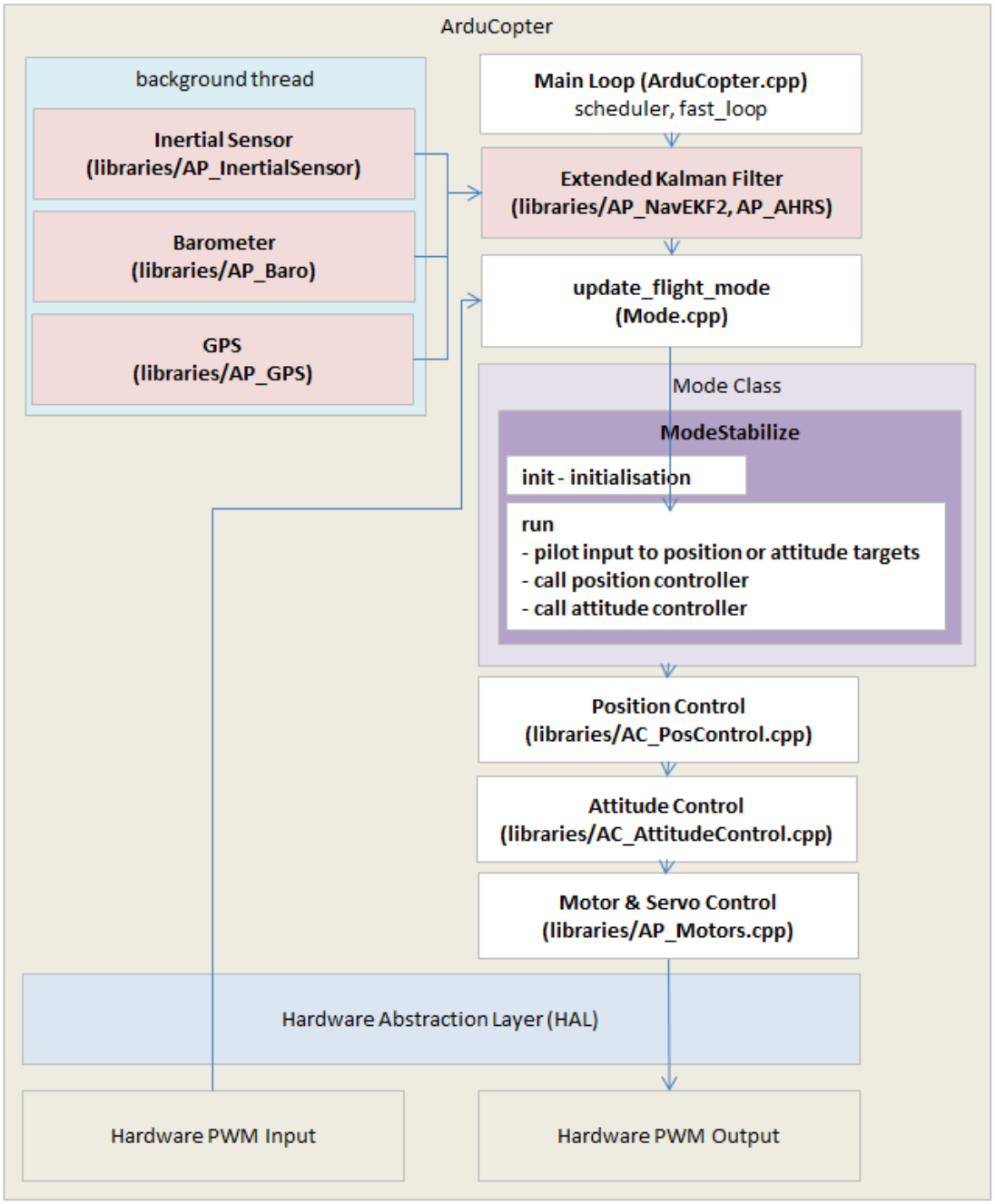
Firmware APM se skládá ze tří hlavních částí, kterými jsou operační systém ChibiOS, letový kód a komunikační vrstva MAVLink. Hlavní vlastnosti firmwaru jsou určeny především letovým kódem. Jak je vidět na obrázku 2.3, skládá se ze tří hlavních součástí, které jsou stručně popsány v následujících odstavcích [16].

Vehicle specific code – jedná se o adresáře nejvyšší úrovně, které definují použitý firmware. Přestože mají různá zařízení mnoho společných prvků, každý se od sebe liší a vyžaduje specifický kód. V současné době projekt Ardupilot podporuje celkem pět typů zařízení - ArduPlane (letadla), ArduCopter (koptéry), APMrover2 (kolové roboty, lodě), ArduSub (ponorky) a AntennaTracker (anténní sledovač).

Shared Libraries – Jedná se o sdílené knihovny, které obsahují ovladače pro senzory, Kalmanův filtr, PID regulátory a další.

Hardware Abstrastion Layer (HAL) – abstrakční vrstva, která obsahuje knihovny pro chod kódu na různém hardwaru. Jedná se především o základní desky a letové kontroléry s Linuxem, jako je například Pixhawk.

Tato práce využívá letového kódu ArduCupter. Jeho podrobnější architekturu zobrazuje obrázek 2.4.



Obrázek 2.4 Architektura letového kódu firmwaru APM, zdroj: [18]

2.3 Výběr firmwaru pro DJI Matrice 100

Z vnějšího pohledu jsou oba srovnávané firmwary velmi podobné a jak již bylo zmíněno, neexistuje žádné jejich odborné srovnání. Hlavní rozdíl spočívá především v uživatelské přívětivosti. PX4 a QGroundControl se z uživatelského pohledu jeví jako přívětivější a nabízí propracovaného průvodce konfigurací dronu.

APM a Mission Planner zase nabízejí kompletní přehled dat ze senzorů v reálném čase. Dále obsahuje funkci *AutoTune*, která umožňuje automatické nastavení parametrů PID regulátorů letového kódu.

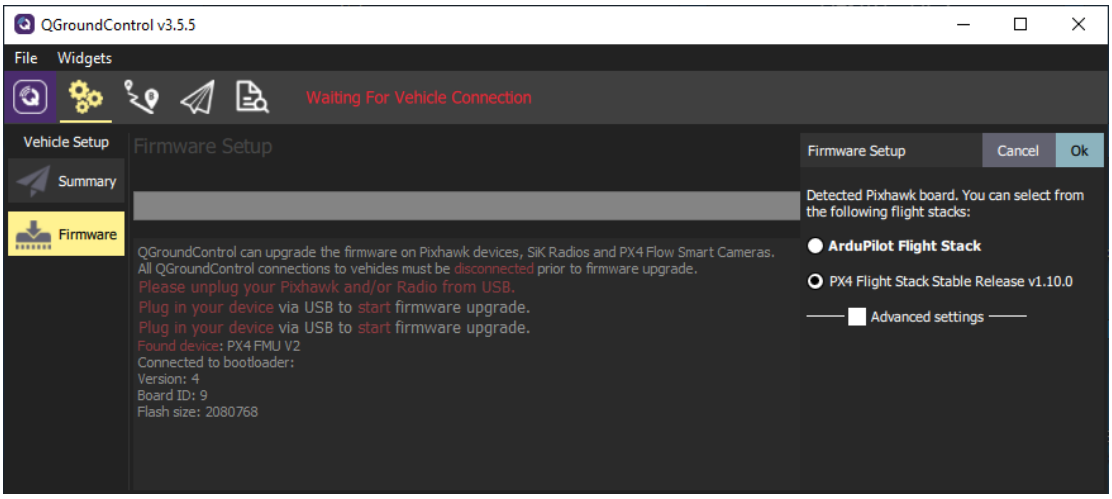
Ve výsledku je tedy na subjektivním rozhodnutí, pro který projekt se rozhodnout. Zároveň, jak již bylo zmiňováno, je možné oba projekty kombinovat a využívat například výhod firmwaru APM a uživatelské přívětivosti QGroundControl.

Práce se nejprve zabývala využitím firmwaru PX4 na dronu DJI M100, což popisují následující podkapitoly. Posléze se autor rozhodl pro přechod na firmware APM a to především z důvodu funkce *AutoTune*. Bohužel, vlivem pandemie koronaviru, již nebylo možno provést tuto změnu na dronu M100.

2.3.1 Zprovoznění a konfigurace PX4

Kompletní průvodce nahráním firmwaru PX4 a jeho nastavením se nachází v jednotlivých kapitolách na webových stránkách projektu [19] a tato práce z něj vychází.

Nejprve je zapotřebí na počítači stáhnout a nainstalovat aplikaci QGroundControl [20]. Po spuštění aplikace je potřeba přejít do záložky pro nahrání firmwaru. Následně se propojí Pixhawk pomocí USB kabelu k počítači a aplikace automaticky rozezná připojené zařízení. Poté nabídne stažení a nahrání poslední verze firmwaru Ardupilot nebo PX4. Viz obrázek 2.5.



Obrázek 2.5 Nahrání firmwaru PX4 do Pixhawku v aplikaci QGroundControl

Po nahrání PX4 dojde k opětovnému připojení Pixhawku k počítači a zobrazení intuitivní nabídky periferií, které je zapotřebí nakonfigurovat nebo zkalibrovat. Zobrazují se v levé nabídce jako červené položky (viz obrázek 2.6). V této nabídce je nyní zapotřebí

vybrat odpovídající druh zařízení. Zvolí se tedy položka *Airframe* a v tomto případě se nastaví *quadrocopter x*, kde je na výběr přímo *DJI Matrice 100*.

Jako další je potřeba provést kalibraci senzorů jako je akcelerometr, gyroskop a magnetometr. Kalibrace se zpřístupní při kliknutí na tlačítko *Sensors* v levé nabídce. Jejich nastavení je v aplikaci velmi intuitivní a průvodce kalibrací obsluze přímo říká co má dělat. Kalibrace senzorů spočívá v natáčení dronu okolo jeho os.

Obdobně intuitivní se jeví nastavení rádiové komunikace pro RC soupravu a namapování letových módů na odpovídající kanály RC ovladače. Toto nastavení je opět dostupné v levé nabídce pod tlačítkem *Radio*.

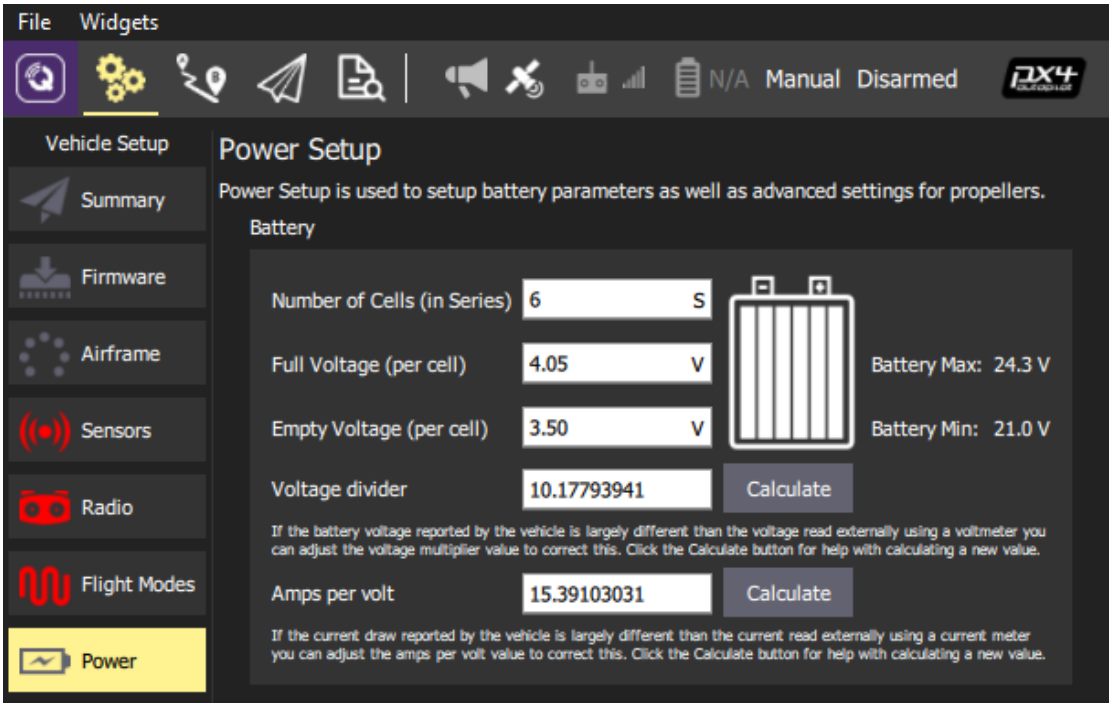
Obrázek 2.6 Průvodce nastavením typu bezpilotního prostředku

Jako poslední musí být nastaven a nakonfigurován napájecí modul, aby byl dron schopen kontrolovat stav nabití akumulátoru. V případě jeho vybití dojde ke spuštění některého z bezpečnostních (failsafe) protokolů, které zajistí bezpečné přistání dronu.

Nastavení napájecího modulu se provádí tak, že se k modulu připojí akumulátor dronu a pomocí multimetru se změří jeho skutečné napětí. Následně se po výběru možnosti *Voltage divider Calculate* zobrazí okno, do kterého je možné vepsat změřené napětí. Na jeho základě se snímač automaticky zkalibruje, viz obrázek 2.7. Dále se zde vyplní počet článků akumulátoru a napětí nabitého a vybitého článku. Běžné je použití Li-Pol akumulátorů, u kterých se napětí nabitého článku pohybuje okolo 4,2 V a u vybitého v rozmezí 3,0 V až 3,3 V. Na základě těchto informací dron vypočte procentuální hodnotu energie zbývající v akumulátoru.

Dále lze zadat hodnotu *Amps per volt*. Z té je pak určeno skutečné napětí akumulátoru i při jeho zátěži, kdy modul díky úbytkům napětí na vnitřním odporu akumulátoru a na vedení detekuje nižší než skutečné napětí. Tomu je potřeba zabránit, protože je pak

chybně detekováno vyšší vybití akumulátoru, než skutečně je. Tato hodnota se opět určuje experimentálně připojením ampérmetru a libovolného rezistoru do série na výstup napájecího modulu. Odečtený proud se opět zadá do zobrazeného okna po stisknutí tlačítka *Calculate*. Změřený proud musí být zadán před rozpojením měřicího řetězce.



Obrázek 2.7 Nastavení napájecího modulu dronu

Aby bylo možné komunikovat s doprovodným počítačem, musí se v záložce *Parameters* nastavit následující parametry [27]:

```
MAV_1_CONFIG      = TELEM 2
MAV_1_MODE        = Onboard
SER_TEL2_BAUD     = 921600
```

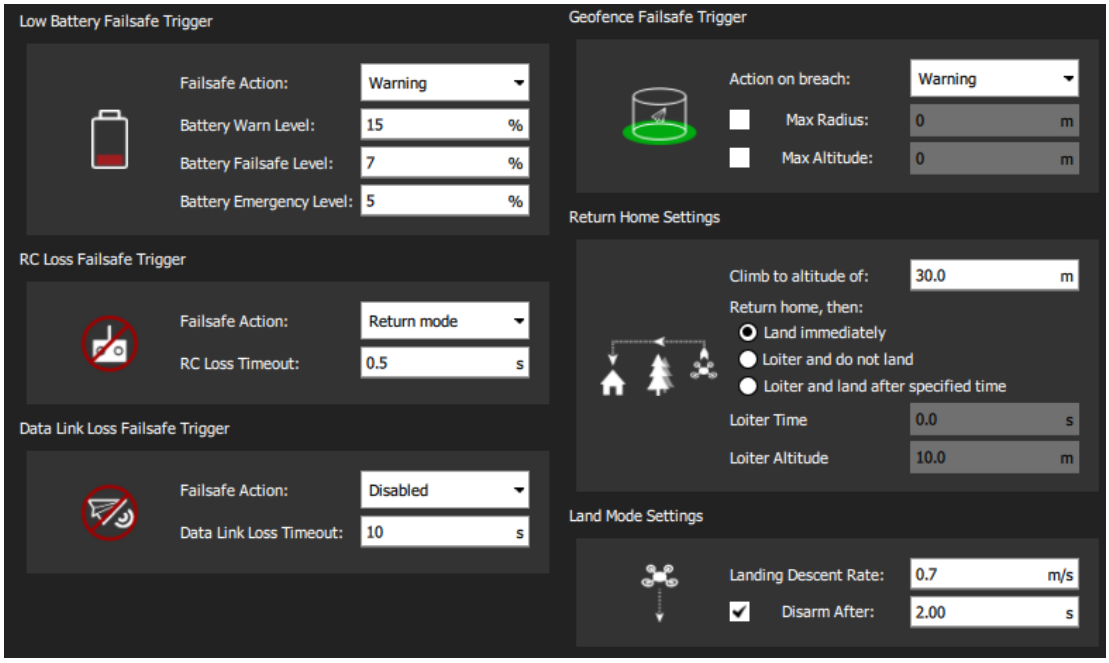
První parametr nastavuje port, na kterém bude realizováno spojení. Druhý parametr udává, že se jedná o spojení s palubním počítačem. Poslední nastavený parametr je komunikační rychlost pro sériovou linku UART, prostřednictvím které je realizována komunikace. Rychlost lze nastavit i vyšší, než je uvedeno výše. Nižší rychlosti nejsou doporučovány.

2.3.2 Bezpečnostní (failsafe) mechanismy

PX4 je vybaven řadou bezpečnostních opatření, která mají za úkol zabránit nehodám a úrazům způsobených drony. Dron je vybaven fyzickým tlačítkem, po jehož stisknutí dojde k inicializaci motorů dronu. Po následném nastavení pravé páčky RC vysílačky do spodní polohy a zároveň levé páčky do pravé spodní polohy na pět sekund, dron přejde z neaktivního stavu *Disarmed* do aktivního stavu *Armed*. Dojde k roztočení motorů na

nejnižší výkon. V tomto stavu je dron schopen vzlétnout. Obdobně je řešen přechod ze stavu *Armed* do *Disarmed*, kdy se pravá páčka nastaví do spodní pozice a levá vlevo dolů po dobu pěti sekund.

Další bezpečnostní akce lze nastavit v QGroundControl v sekci *Safety*, kde je možné nastavit rozdílné akce pro vybitý akumulátor, pro ztrátu signálu od RC vysílačky nebo při překročení vytyčeného perimetru v okolí startovní pozice. Většinou je zde možnost vybrat mezi akcí varování, návratu na výchozí pozici a okamžitým přistáním.



Obrázek 2.8 Průvodce nastavením bezpečnostních (failsafe) mechanismů dronu

2.3.3 Letové módy

Tabulka 2.1 zobrazuje stručný přehled letových módů, kterými platforma PX4 disponuje. Přesnější popis funkcí letových módů lze nalézt v dokumentaci projektu [22]. Zde je proveden pouze základní rozbor některých z nich.

Nejběžněji používané módy jsou *Position, Altitude* a *Manual/Stabilized*. Jedná se o módy řízení pomocí RC soupravy a v případě *Position* se jedná o mód, kdy je možné nastavovat rychlost letu v osách pitch a roll a rychlost stoupání. Jakmile dojde k puštění ovládacích pák na RC vysílači, zůstane dron díky GNSS zafixován na daných souřadnicích. Obdobně funguje mód *Altitude*, který vzhledem k absenci GNSS udržuje pouze výšku, ale může se volně pohybovat v horizontálních směrech. To může být zapříčiněno například vlivem větru. V *Manual/Stabilized* módu již nedochází k žádnému fixování polohy ani výšky.

Dále je zde nabídka několika plně automatických režimů, ze kterých jsou nejdůležitější režimy *Return* pro navrácení na startovní pozici a mód *Land* pro přistání na aktuálních souřadnicích. Pro tuto práci je stěžejní poslední mód, označovaný jako

Offboard. V tomto režimu je možné řídit Pixhawk, resp. dron pomocí palubního počítače na jeho palubě. Pixhawk v tu chvíli přijímá MAVLink příkazy přes sériovou linku a vykonává je. Přesto je stále možné převzít ruční řízení pomocí RC soupravy a pilot má stále plnou kontrolu nad chováním dronu.

Tabulka 2.1 Zjednodušená tabulka letových módů PX4, zdroj: [22]

Mód	**Roll a Pitch**	**Yaw**	**Tah**	**Pozice**
Position	Asistence autopilota			GNSS, IMU, barometr, …
Altitude	Asistence autopilota			Bez GNSS, pouze IMU, barometr, …
Manual/Stabilized	Asistence autopilota		Manuální	-
Acro	Asistence autopilota		Manuální	-
Orbit	-	-	-	-
Takeoff	Auto			GNSS
Land	Auto			GNSS
Hold	Auto			GNSS
Return	Auto			GNSS
Mission	Auto			GNSS
Follow Me	Auto			GNSS
Offboard	Auto			GNSS

3. TESTOVACÍ DRON

Z důvodu nedostupnosti dronu M100 vlivem vládních restrikcí v období pandemie koronaviru SARS-CoV-2, byl pro účely testování postaven a použit soukromý dron. Jeho konstrukce není součástí zadání diplomové práce, proto kapitola uvádí pouze jeho stručný přehled.

3.1 Hardware

Dron byl navržen autorem práce a vytištěn na 3D tiskárně. Návrh dronu je možno vidět na obrázku 3.1, zatímco na obrázku 3.2 lze vidět již vytisknutý a složený dron. Jedná se o čtyřvrtulový dron, který je vybaven pohonným systémem DJI E310, akumulátorem o kapacitě 6500 mAh, Pixhawkem 1, GPS modulem NEO M8N a palubním počítačem Raspberry Pi 4B v konfiguraci se 4 GB paměti RAM. Hmotnost dronu je 1,5 kg. Principiální zapojení elektroniky odpovídá zapojení M100 podle blokového schématu na obrázku 1.3.

Po stránce hardwaru oproti M100 obsahuje starší, ale plně kompatibilní verzi řídicí jednotky Pixhawk. Ta je oproti M100 doplněna novější verzí palubního počítače Raspberry Pi.

Obrázek 3.1 Model testovacího dronu

Obrázek 3.2 Sestavený testovací dron

3.2 Firmware a software

Pixhawk dronu obsahuje firmware APM. Konkrétně se jedná o verzi *ArduCopter V4.0.7*, který byl nakonfigurován obdobným způsobem jako dron M100 v kapitole 2.3.

Rozdíl v použitém firmwaru pro Pixhawk je způsoben tím, že práce na dronu M100 probíhaly ještě před pandemií koronaviru. V té době se jevil jako výhodnější firmware PX4, což také popisuje kapitola 2.3. Nicméně, při dlouhodobější práci s oběma platformami se autorovi práce jeví firmware APM jako vhodnější. Především díky funkci *AutoTune*.

Co se týče palubního počítače Raspberry Pi 4B, tak ten je opatřen operačním systémem *Raspberry Pi OS* s kernelem *5.10.11-v7l+ #1399 SMP*. Následně byla provedena instalace a konfigurace všech potřebných nástrojů podle kapitoly 4.1.1.

4. SOFTWARE PRO PALUBNÍ A SIMULAČNÍ POČÍTAČ

Kapitola se v první části věnuje instalaci a konfiguraci veškerého nezbytného softwaru na palubním počítači Raspberry Pi. Ve druhé části popisuje zprovoznění softwaru na simulačním počítači. Pro tyto účely byly v rámci práce vytvořeny dva instalační skripty, které jsou popsány v následujících podkapitolách.

4.1 Palubní počítač Raspberry Pi

Jako palubní počítač dronu M100 byl zvolen jednodeskový počítač Raspberry Pi 3B+ s Linuxovým operačním systémem Raspberry Pi OS, dříve známý jako Raspbian. V případě potřeby je možné použít i jiný počítač s libovolnou linuxovou distribucí.

Raspberry Pi OS je ke stažení přímo na webových stránkách projektu Raspberry Pi [23]. Je vhodné použít plnou verzi systému, protože v průběhu experimentování bylo zjištěno, že verzi *Lite* chybí některé důležité softwarové nástroje a instalace některých balíčků končí chybovým hlášením. Systém je distribuován jako obraz disku, který je nutné přenést na micro SDHC kartu, ze které bude následně na Raspberry Pi spouštěn. K tomu lze využít například program Win 32 Disk Imager [24].

Před prvotním spuštění je potřeba povolit připojení pomocí SSH. Toho lze docílit umístěním souboru s názvem *ssh* bez přípony do *boot* oddílu na SDHC kartě. Nyní je možné komunikovat s Raspberry pomocí ethernetového kabelu nebo pomocí WiFi. V případě použití WiFi je nutné přidat informace o síti do souboru */etc/wpa_supplicant/wpa_supplicant.conf*. Tyto informace mohou vypadat například následovně:

```
network=    {
            ssid="nazev_site"
            psk="heslo"
            }
```

Další možnosti nastavení WiFi připojení je možné dohledat na webových stránkách projektu Raspberry [25].

Pro připojení počítače k Raspberry Pi je v prostředí Windows vhodné použít například software Putty. Druhou možností je připojení klávesnice, myši a monitoru k Raspbperry a provést jeho konfiguraci v grafickém prostředí desktopu. Tato varianta je mnohem komfortnější, protože při prvním spuštění dojde k zobrazení intuitivního grafického průvodce konfigurací. Ta zahrnuje nastavení sítě, WiFi, výchozího jazyka a mnohé další.

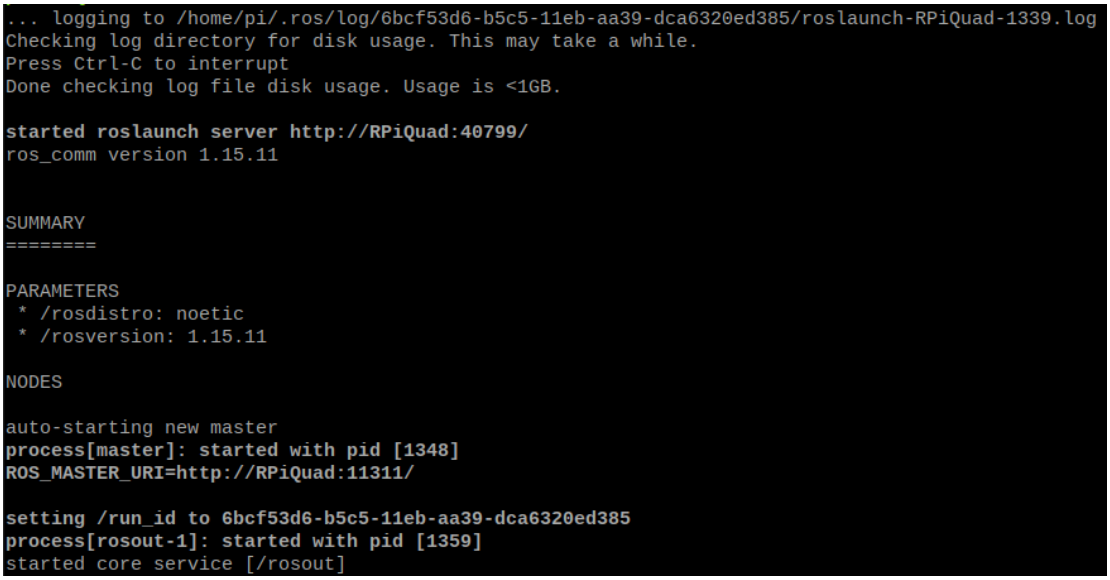
4.1.1 Instalační skript pro palubní počítač Raspberry Pi

Pro snadné a automatické zprovoznění potřebného softwaru na Raspberry Pi byl napsán skript pro linuxový terminál, který je k nalezení na přiloženém CD/DVD jako *Příloha A – setup_rpi.sh*. Zjednodušený vývojový diagram skriptu popisuje obrázek 4.2.

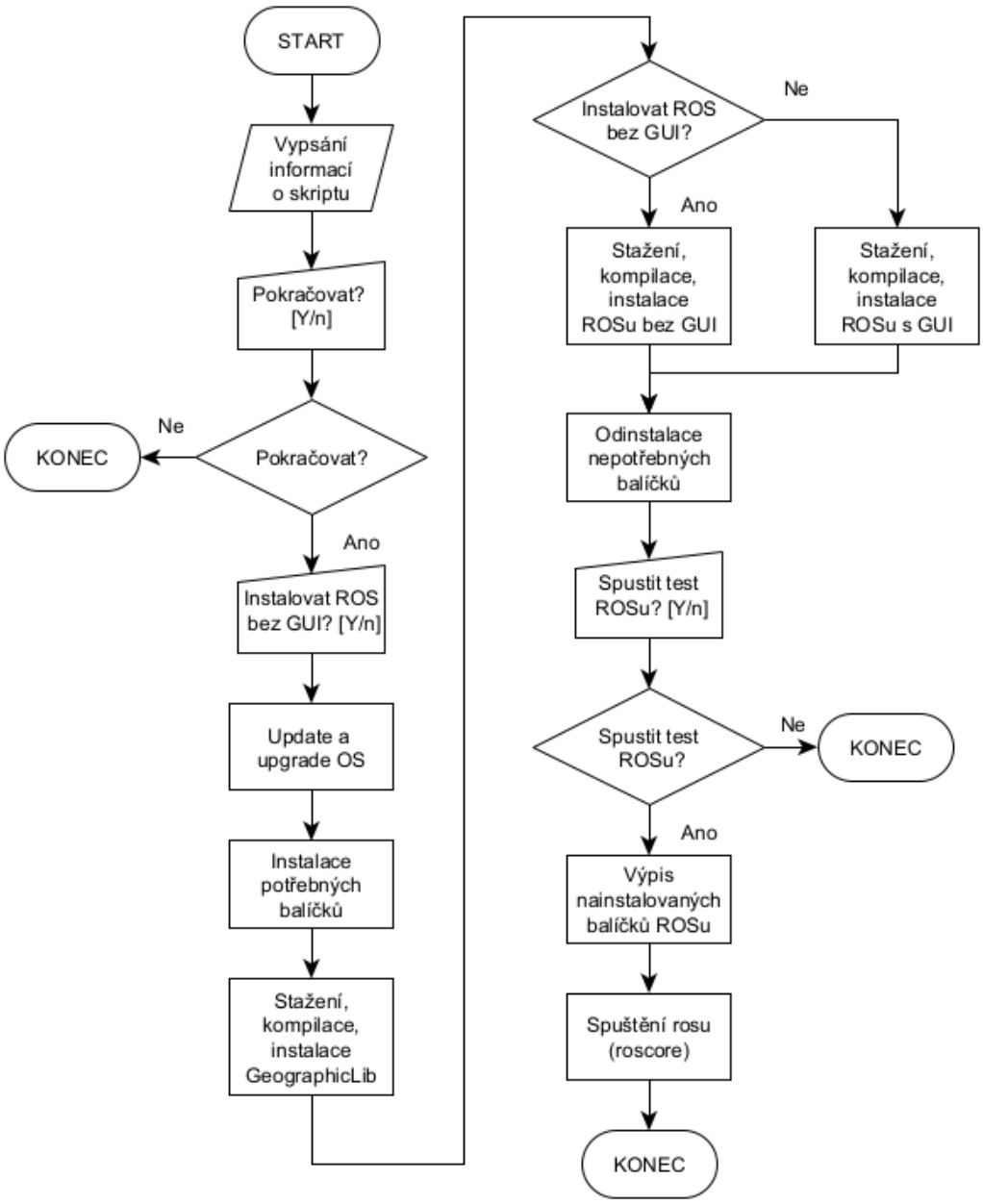
Skript po spuštění vypíše základní údaje, kterými je seznam balíčků, které budou nainstalovány a které budou naopak odstraněny. Dále vypíše konfiguraci, na které byl testován, což je v tomto případě testovací dron s Raspberry Pi 4B se 4 GB paměti RAM a s Raspberry Pi OS s kernelem *5.10.11-v7l+ #1399 SMP*. Dále se skript zeptá, zda chce uživatel opravdu pokračovat. V případě, že ano, zeptá se, jestli chce uživatel nainstalovat plnou verzi ROSu s GUI, nebo pouze základní verzi bez GUI aplikací.

Poté dojde k aktualizaci repozitářů a aktualizaci všech balíčků a operačního systému. Dále se nainstalují ostatní balíčky, které jsou nezbytné pro zprovoznění ROSu. Mezi tyto balíčky patří například balíček GeographicLib [29], který slouží ke konverzi souřadnicových systémů a je vyžadován knihovnou MAVROS. Dalším krokem je samotná instalace zvoleného ROSu s GUI či bez, která vychází z návodu na jeho webových stránkách [27]. Návod byl upraven tak, aby pracoval s nejaktuálnější verzí ROSu, kterou je v současné době ROS Noetic. Zároveň byly k ROSu přidány balíčky navíc, kterými jsou MAVROS a MAVROS extras. Vzhledem k tomu, že není dostupný zkompilovaný balíček v rámci repozitářů OS, je potřeba provést kompilaci na Raspberry. Tato část skriptu může zabrat několik hodin. Záleží, jestli byla zvolena verze s GUI nebo bez. V případě verze s GUI je nutné použít Raspberry s větší pamětí RAM. Experimentálně bylo zjištěno, že na verzi s 1 GB RAM skončí kompilace chybovou hláškou. Verze bez GUI neobsahuje některé aplikace, jako jsou například grafické aplikace rqt, RVIZ pro zobrazování dat a další. Po nainstalování ROSu dojde k odinstalaci nepotřebných balíčků, které plná verze Raspberry Pi OS obsahuje.

V posledním kroku se skript uživatele zeptá, zda chce provést test ROSu. Ten spočívá ve vypsání všech nainstalovaných balíčků v rámci ROSu. Následně se pokusí spustit *ROS core*. Obrázek 4.1 ukazuje, jak by měl vypadat výsledek. Pokud se vše podařilo, je možné ukončit skript stisknutím kláves *CTR+C*



Obrázek 4.1 Test funkčnosti ROSu po dokončení skriptu *setup_rpi.sh*



Obrázek 4.2 Zjednodušený vývojový diagram skriptu *setup_rpi.sh*

4.2 Simulační počítač

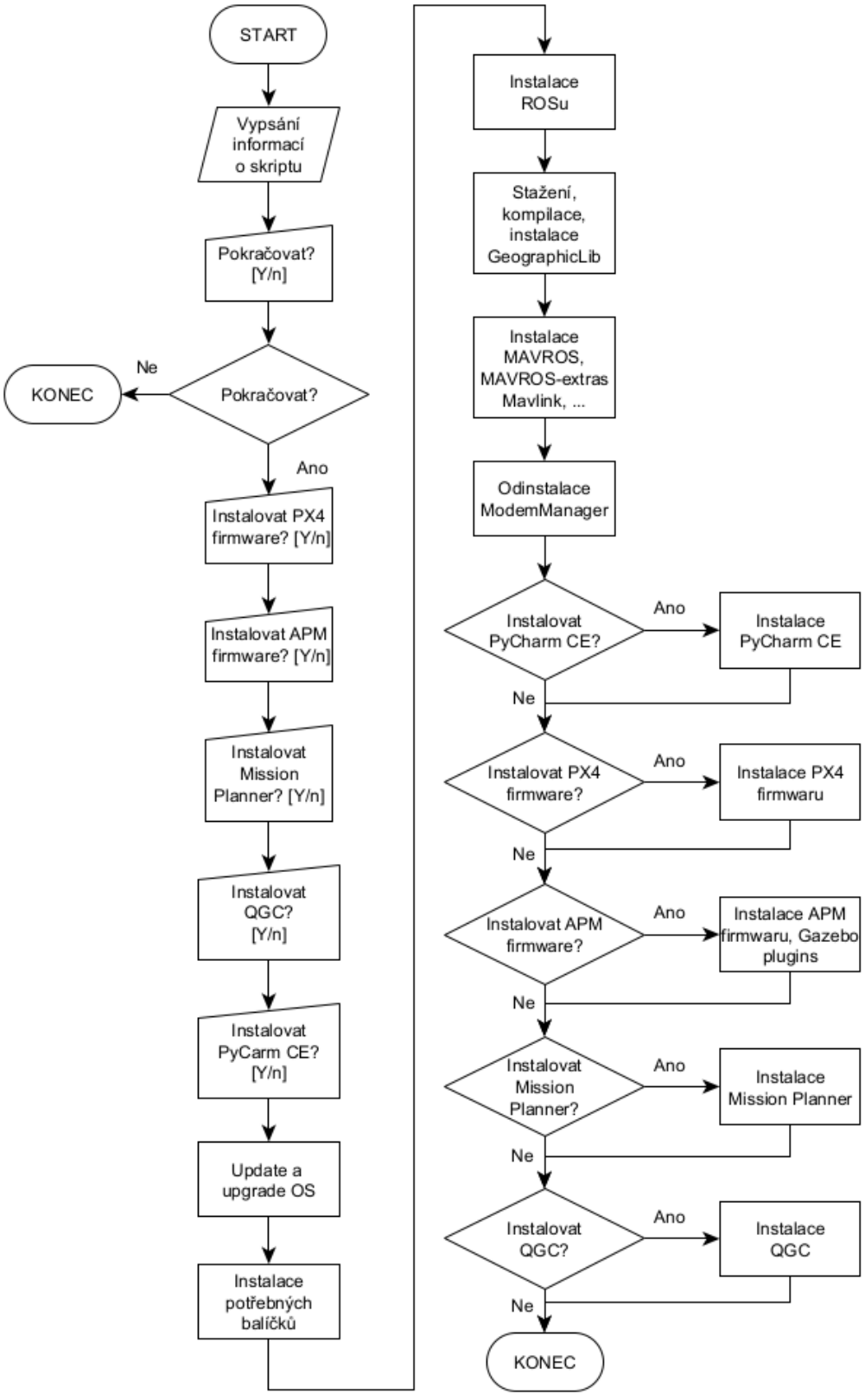
Simulační počítač je obvykle stolním počítačem v klasickém slova smyslu, na kterém je nainstalována některá z běžných linuxových distribucí. V tomto případě bylo pracováno s distribucí Ubuntu.

Účelem počítače je umožňovat spouštění simulace fyzikálního modelu dronu se zvoleným firmwarem. Jako simulační software je použita aplikace Gazebo, která je součástí instalačního balíčku plné verze ROSu [28].

4.2.1 Instalační skript pro simulační počítač

Skript slouží pro snadnou instalaci potřebného simulačního softwaru. Skript vznikl v rámci této práce a je opět k nalezení na přiloženém CD/DVD jako *Příloha B – setup_pc.sh*. Ihned po spuštění vypíše základní informace včetně verze OS, na kterém byl testován. V tomto případě byl skript testován na *Ubuntu 20.04 LTS*. Dále se uživatele zeptá, zda chce pokračovat dále.

V dalším kroku dojde k dotazování, jestli chce uživatel nainstalovat jednotlivé části, kterými jsou firmware PX4 [31], firmware APM [30], aplikace pozemní stanice Mission Planner, aplikace pozemní stanice QGroundControl (zkráceně QGC) a jako poslední se dotazuje na instalaci PyCharm CE, což je vývojové prostředí, které je vhodné pro psaní kódu v C++ a v Pythonu. Následuje aktualizace operačního systému a instalace potřebných balíčků, instalace plné verze ROSu včetně RVIZ, rqt, Gazebo atd. V případě Ubuntu je plná verze ROSu dostupná v repozitářích a lze postupovat dle návodu [28]. V neposlední řadě proběhne instalace GeographicLib [29] a balíčků MAVROS a MAVROS extras [33]. Pro správný chod pozemní stanice je nutné odinstalovat aplikaci ModemManager. Tento krok je vyžadován při instalaci pozemních stanic. Jako poslední se vykoná instalace vybraných součástí, na které se skript dotazoval na počátku. V případě instalace firmwaru PX4 je nutné nainstalovat balíček *ignition-common3-graphics* podle návodu [32], protože se nenachází v repozitářích Ubuntu. V opačném případě bude budoucí kompilace firmwaru PX4 vracet chybu.



Obrázek 4.3 Zjednodušený vývojový diagram skriptu *setup_pc.sh*

5. SIMULACE

Kapitola se věnuje popisu simulace dronu. Hlavní výhodou simulací je jejich bezpečnost a rychlost ladění kódu v nich. Další nespornou výhodou je že není nutné mít u sebe neustále těžký a objemný dron.

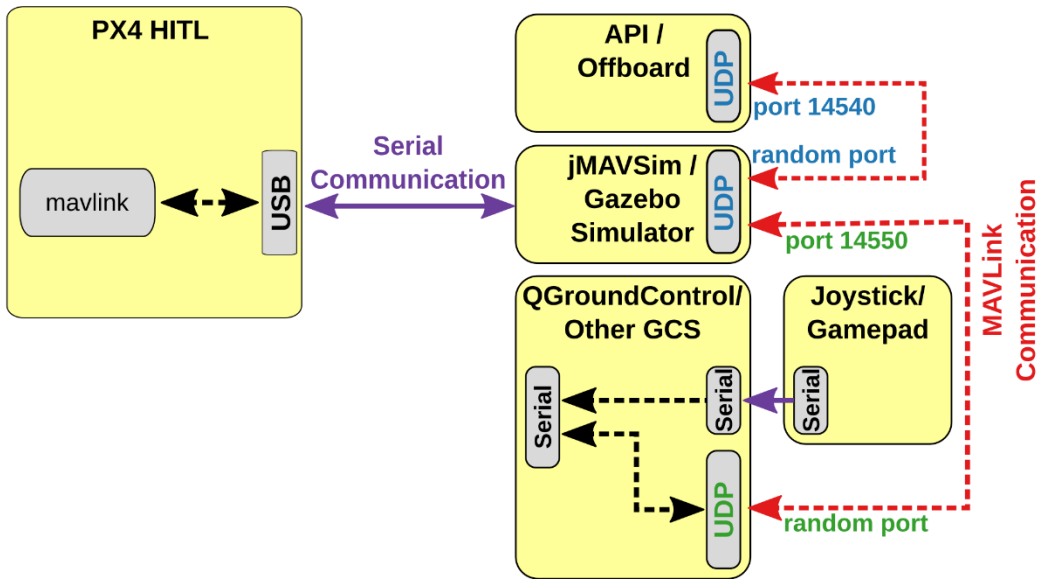
Simulace existuje dvojího typu. Prvním typem je SITL (z anglického Software In The Loop) a druhým je HITL (z anglického Hardware In The Loop). Simulace typu SITL probíhá celá v rámci počítače, zatímco HITL využívá k simulaci reálný hardware dronu, například v podobě Pixhawku. Z toho vyplývá, že SITL vyžaduje spuštění firmwaru dronu na počítači, zatímco HITL pracuje s firmwarem, který je nahraný v Pixhawku a pouze mu dodává vstupní data a vyčítá výstupní data. V rámci této práce je simulace prováděna na fyzikálním modelu dronu v rámci aplikace Gazebo, která je součástí instalace ROSu.

V této práci je využívána především simulace typu SITL.

5.1.1 Simulace HITL (hardware in the loop)

Obrázek 5.1 ukazuje blokové schéma PX4 HITL simulace. Obdobně lze provést i APM HITL simulaci. Tento typ simulace vyžaduje připojení letového kontroléru k počítači. Dále je na něm nutné nastavit pomocí pozemní stanice *Airframe* typu HIL [39].

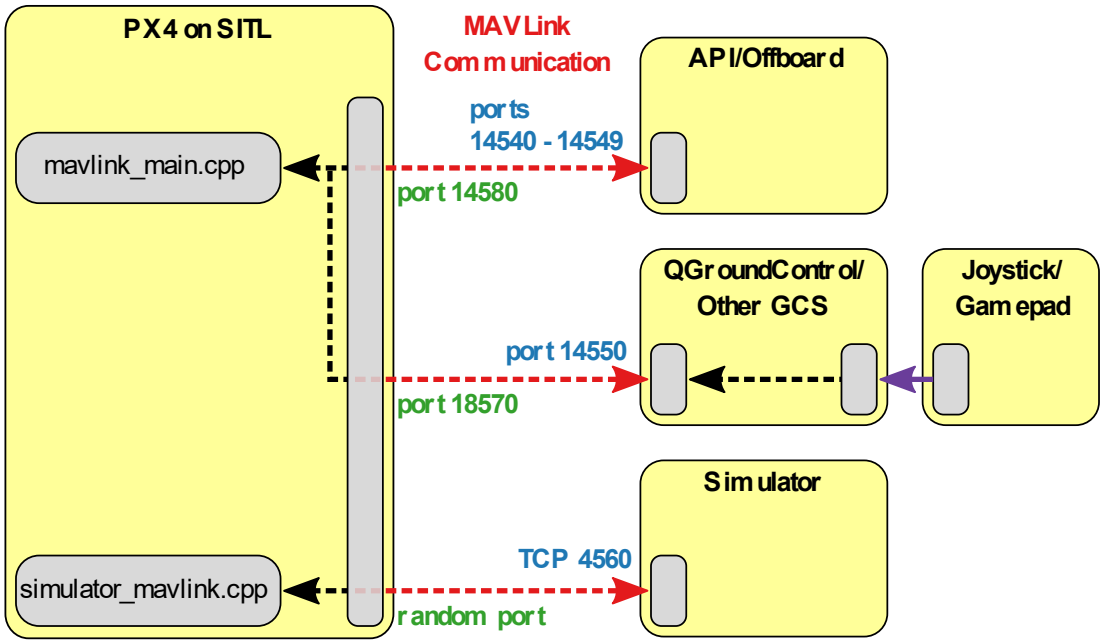
Na počítači je potom potřeba spustit Gazebo s fyzikálním modelem a nastavit jeho komunikaci na odpovídajícím USB portu. Nakonec je možno spustit uživatelskou aplikaci a simulovat odezvu na ni.



Obrázek 5.1 Blokové schéma PX4 HITL simulace, zdroj: [39]

5.1.2 Simulace SITL (software in the loop)

Simulace typu SITL se odehrává celá v počítači. Vyžaduje stažení a spuštění simulovaného PX4 nebo APM firmwaru. Blokové schéma SITL je vidět na obrázku 5.2. Opět je možné provádět simulaci obdobně i pro APM firmware. Simulace opět vyžaduje fyzikální model dronu, který zajistí aplikace Gazebo. Simulaci je také možno propojit s pozemní stanicí nebo s jinou uživatelskou aplikací, určenou k řízení dronu. [40]



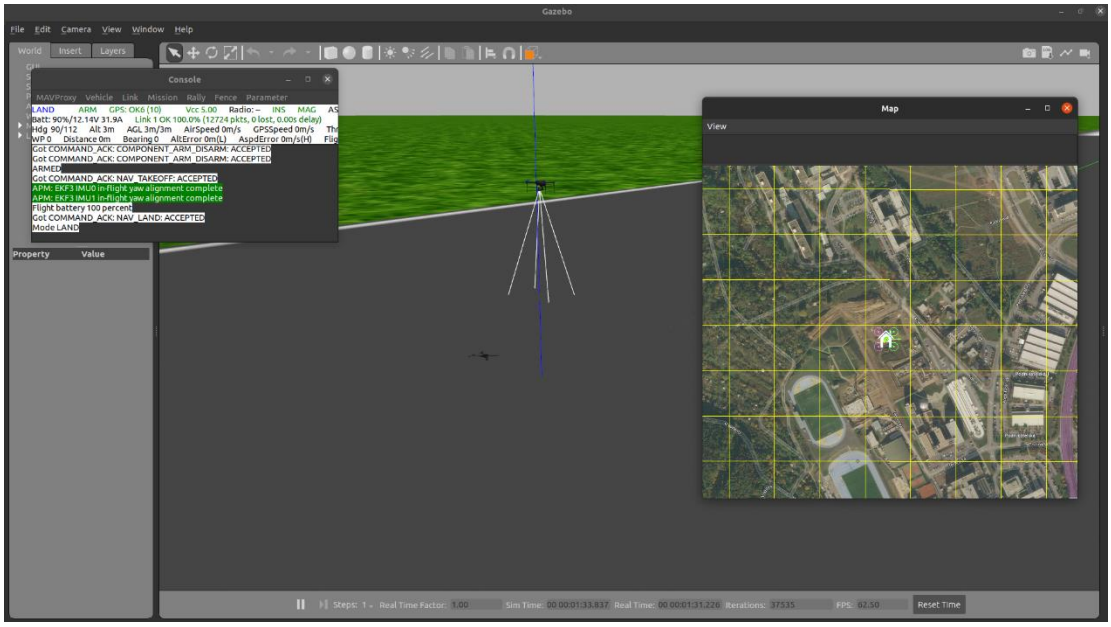
Obrázek 5.2 Blokové schéma PX4 SITL simulace, zdroj: [40]

5.1.3 Spouštěcí skripty simulací

Za účelem snadného spuštění SITL simulace PX4, byl v rámci práce vytvořen skript *setup_sitl_px4.sh*, který je dostupný v *příloze C – ros_ws.zip*. Skript vychází z návodu [40].

Obdobně byl vytvořen skript *setup_sitl_apm.sh* pro simulaci s APM firmwarem, který se opět nachází v *příloze C*. Skript byl vytvořen podle zdroje [41].

Skripty spouštějí buďto PX4 nebo APM firmware. Spolu s ním je spouštěno Gazebo s fyzikálním modelem dronu. Zpravidla se jedná o model *Iris – quadrotor*. V případě potřeby je možno doplnit spouštěný model o pluginy, které zajistí, že se dron v simulaci nachází v různých vizuálních prostředích. Je možné namodelovat si i vlastní prostředí s různými překážkami. Na obrázku 5.3 je vidět ukázka simulace letícího dronu v aplikaci Gazebo.



Obrázek 5.3 Ukázka simulace letu dronu v aplikaci Gazebo

6. ROBOTICKÝ OPERAČNÍ SYSTÉM (ROS)

První část kapitoly se věnuje teoretickému úvodu do problematiky fungování Robotického operačního systému – ROS. Dále přechází do stručného popisu tvorby ROS nodů. Také uvádí různé možnosti spouštění těchto nodů.

Druhá polovina popisuje praktickou tvorbu testovacích nodů, které jsou určeny pro rychlé otestování funkčnosti simulovaného prostředí, příp. funkčnosti dronu a všech softwarových nástrojů instalovaných na něm. Poslední část se zaměřuje na popis testovací mise, která vznikla jako hlavní výstup této práce.

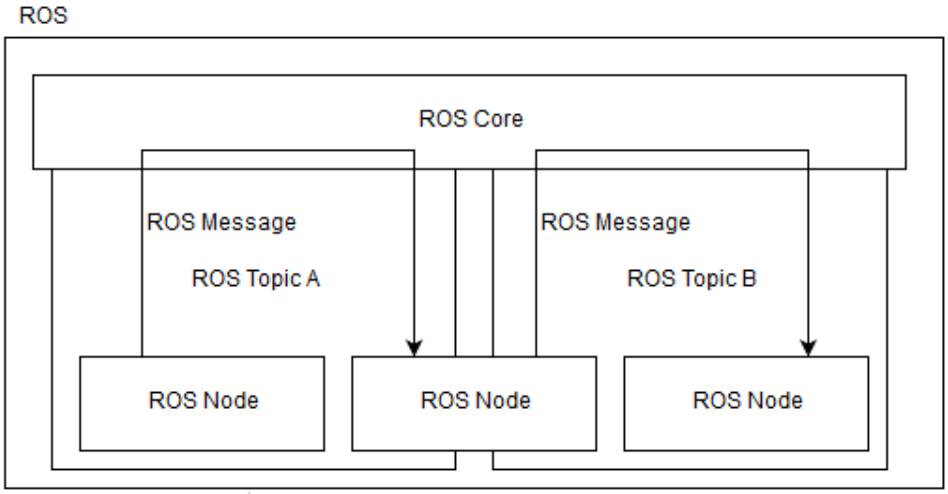
6.1 Princip fungování ROSu

Nejedná se o skutečný operační systém, ale o middleware určený pro Unixové systémy. ROS je určen pro tvorbu aplikací v oblasti robotiky. Obsahuje širokou škálu balíčků předdefinovaných funkcí a ovladačů pro hardware. V současné době se projekt ROS rozděluje na ROS 1 a ROS 2, které se liší např. v tom, že ROS 2 je již schopen fungovat v reálném čase. Nicméně tato práce je založena na ROS 1 v jeho poslední verzi Noetic a proto se kapitola dále zabývá pouze jím.

Jádro *ROS Core* je licencováno pod svobodnou licencí BSD [34]. Většina ostatních knihoven je licencována pod více svobodnými licencemi. Tento projekt pracuje nejvíce s knihovnou MAVROS, která spadá pod licence GPLv3, LGPLv3 a BSD [35].

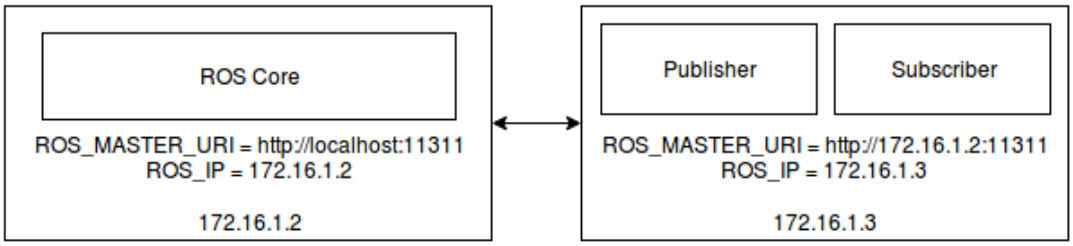
ROS uživateli nabízí prostředí, pro tvorbu obsáhlých projektů se snadnou přenositelností mezi platformami. Zároveň je nezávislý na programovacím jazyce. Tím je myšleno, že ROS funguje jako spojovací článek pro programy psané různých programovacích jazycích, kterými mohou být Python, C++, Lisp a experimentálně i jazyky Java a Lua [36].

Zjednodušená topologie ROSu sestává z řady uzlů *node*, což jsou jednotlivé programy. Jádro *core* plní úlohu serveru, který realizuje komunikaci mezi uzly skrze domény nazývané *topic*. Jednotlivé zprávy *message* jsou rozesílány mezi uzly v rámci jejich domén prostřednictvím jádra. Komunikace v rámci ROSu je zprostředkována protokolem UDP. Komunikaci ilustruje obrázek 6.1.



Obrázek 6.1 Zjednodušené schéma komunikace v rámci ROSu, zdroj: [37]

Další zásadní výhodou ROSu je skutečnost, že je možné směrovat jeho provoz na místní síť. Obrázek 6.2 popisuje potřebnou konfiguraci v rámci dvou počítačů ve společné síti. Pokud nejsou použity speciální knihovny ROSu, může *ROS Core* běžet pouze na jednom počítači v síti. Jeho úkolem je řídit síťový provoz mezi ním a ostatními počítači a realizovat mezi nimi propojení jednotlivých nodů.



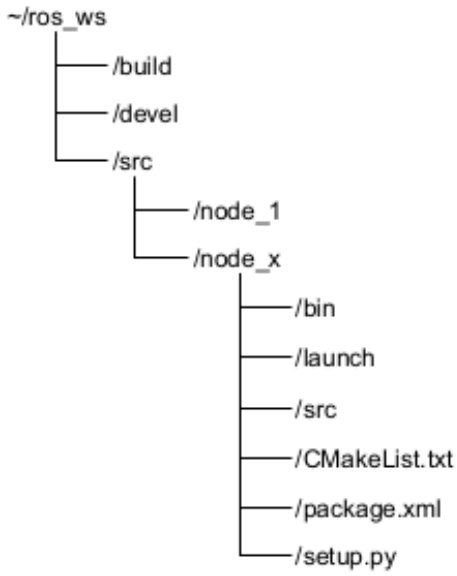
Obrázek 6.2 Schematické znázornění ROSu distribuovaného na jiný počítač v rámci místní sítě, zdroj: [37]

6.2 Seznámení s tvorbou ROS nodu

Kapitola vychází z oficiálních ROS tutoriálů [38] a stručně shrnuje základní tvorbu nodů v jazycích C++ a Python Následně shrnuje účel a tvorbu takzvaných launch skriptů, které jsou taktéž popsány v příslušné kapitole zmíněných tutoriálů.

Nody je vhodné tvořit v pracovním prostoru ROSu, který se zpravidla umisťuje do domovského adresáře uživatele. Jeho struktura je vidět na obrázku 6.3. Adresáře *build* a *devel* obsahují zkompilovaný kód. Pro samotnou tvorbu nodů je stěžejní adresář *src*, ve kterém se nacházejí zdrojové kódy nodů, resp. programů. Adresáře samotných nodů se generují pomocí příslušného terminálového příkazu. Blíže jsou popsány v následujících podkapitolách. Ilustrační node s názvem *node_x* má v sobě vygenerovány adresář *bin*

(typicky obsahuje program psaný v Pythonu) a adresář *src* (pro program v C++). Adresář *launch* se běžně automaticky negeneruje, ale je vhodné počítat s ním a vytvořit ho. Soubory *CMakeList.txt* a *package.py* potom obsahují nastavení pro kompilaci. Soubor *setup.py* je možné vytvořit a použít v případě nodu psaného v Pythonu. Obsahuje popis a cestu ke spouštěcímu souboru nodu.



Obrázek 6.3 Typická struktura pracovního adresáře ROSu

6.2.1 Tvorba nodu v C++

Nejprve je potřeba v adresáři *src* vygenerovat balíček nodu, ve kterém bude psán jeho vlastní kód. Toho se docílí pomocí příkazu:

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Místo *package_name* se doplní název vytvářeného balíčku, například *node_1*. *Depend* se nahradí knihovnami ROSu, které budou v kódu nodu importovány a používány. Příkladem takových knihoven jsou *std_msgs, roscpp* (základní C++ knihovna ROSu), atd. Následně je třeba vytvořit soubor s kódem nodu a umístit ho následujícím způsobem - *node_1/src/node_1_node.cpp*. V automaticky vytvořeném souboru *package.xml* je vhodné doplnit verzi nodu, e-mail tvůrce, licenci aj. Uvnitř nově vzniklého souboru *CMakeList.xml*, je potřeba zajistit, aby byly odkomentovány následující řádky, což zajistí spustitelnost zkompilovaného kódu:

```
add_executable(${PROJECT_NAME}_node src/node_1_node.cpp)

target_link_libraries(${PROJECT_NAME}_node
   ${catkin_LIBRARIES}
)
```

6.2.2 Tvorba nodu v Pythonu

Tvorba je podobná jako u C++ nodu. Nejprve se vygeneruje balíček nodu stejně jako pro C++, ale knihovna C++ se nahradí základní knihovnou ROSu pro Python - *rospy*.

Dále je možné vytvořit soubor s kódem nodu v Pythonu. Pro příklad je tento node pojmenován jako *node_x*, kterému se musí v operačním systému nastavit práva ke spuštění. Tato práva je potřeba nastavit i pro všechny ostatní pythonovské soubory, jako je třeba setup.py. Příklad nastavení práv ke spuštění souboru v linuxovém terminálu:

```
chmod -x node_x/bin/node_x_node.py
```

V automaticky vytvořeném souboru *package.xml* je opět vhodné doplnit všechny náležitosti jako tomu bylo u C++ nodu. Uvnitř *CMakeList.xml*, se odkomentuje řádek:

```
catkin_python_setup()
```

Dále se v adresáři nodu vytvoří soubor *setup.py* s následujícím obsahem:

```
setup_args=generate_distutils_setup(
    packages=['node_x'],
    scripts=['bin/node_x_node.py'],
    package_dir={'': 'src'}
)
```

6.2.3 Tvorba launch skriptu

Za předpokladu, že se v pracovním adresáři ROSu vykoná příkaz pro kompilaci nodů a aktualizuje se prostředí ROSu:

```
catkin_make
source ~/ros_ws/devel/setup.bash
```

Potom je možné jednotlivé nody spustit. Toto spuštění lze provádět postupně pro každý node zvlášť. K tomu slouží příkaz *rosrun*. Zároveň je nutné v jiném terminálu spustit jádro ROSu příkazem *roscore*. Tento způsob samostatného spouštění není příliš

vhodný v případě nutnosti spouštět více nodů najednou. Zároveň je potřeba spouštět pro každý node nový terminál.

Tento problém řeší soubory, psané ve značkovacím jazyce XML, s příponou *launch*. Pro tyto soubory je vhodné vytvořit adresář */launch* uvnitř adresáře nodu (viz obrázek 6.3). Následující kód ukazuje jednoduchý launch soubor:

```
<?xml version="1.0"?>
<launch>
      <include file="$(dirname)/other.launch" />
        <arg name="(name)" value="(value)" />
      </include>
    <node name="(name)" pkg="(pkg)" type="(type)" output="screen"/>
</launch>
```

Řádek uvozený značkou *include* slouží pro volání jiného launch souboru. Jiných launch souborů je možné volat i více než jeden nebo žádný. Za *dirname* se dosadí adresář, kde se nachází a za *other* se dosadí jeho název. Řádek uvozený značkou *arg* potom obsahuje argumenty, se kterými se má tento další launch soubor spouštět. Za *(name)* se dosadí název argumentu a za *(value)* jeho hodnota. Launch soubor může být volán i s více argumenty. Předposlední řádek, který je uvozený značkou node, spouští samotný node. Za *(name)* se dosadí název, za *(pkg)* název celého balíčku a za *(type)* jeho typ. *Output* potom určuje, kam bude vypisován výstup nodu. Nodů lze takto volat vícero jejich přidáním do souboru.

Aby bylo možné propojit ROS s Pixhawkem, je nutné spustit odpovídající MAVROS node. Existuje MAVROS pro PX4 i pro APM firmware. Jejich spuštění je vhodné realizovat pomocí launch skriptu. Do spouštěcího souboru stačí přidat následující řádky [34]:

```
<include file="$(find mavros)/launch/apm.launch">
   <arg name="fcu_url" value="/dev/ttyS0:921600" />
   <arg name="gcs_url" value="udp://:14555@192.168.0.108:14550" />
</include>
```

Tento kód zajišťuje spuštění MAVROS APM nodu, který se snaží komunikovat s Pixhawkem prostřednictvím sériové linky *ttyS0* na Raspberry Pi. Rychlost sériové linky je v tomto případě 921 600 Bd. Obdobně je možné realizovat propojení i přes USB. Parametr *gcs_url* udává IP adresu počítače, na kterém je spuštěna pozemní stanice, na kterou se posílají telemetrická data.

Obdobně lze spustit i MAVROS node pro PX4. Stačí ve skriptu nahradit *apm.launch* za *px4.launch*.

Pro případ spouštění MAVROSu v SITL simulaci, je nutné pro APM přiřadit *fcu_url* řetězec *udp://127.0.0.1:14551@14555*. V případě PX4 se *fcu_url* úplně vynechá. Má ho nastaven jako výchozí.

6.3 Testovací ROS nody

Za účelem otestování správného fungování ROSu a simulačního softwaru na počítači, ale i k otestování ROSu na Raspberry a jeho správného fungování s Pixhawkem, byly vytvořeny následující čtyři balíčky testovacích nodů.

- test_px4_cpp
  - test_px4_cpp_sim.launch
  - test_px4_cpp.launch
- test_px4_py
  - test_px4_py_sim.launch
  - test_px4_py.launch
- test_apm_cpp
  - test_apm_cpp_sim.launch
  - test_apm_cpp.launch
- test_apm_py
  - test_apm_py_sim.launch
  - test_apm_py.launch

Všechny tyto balíčky se nacházejí na přiloženém CD/DVD v *příloze C – ros_ws.zip*, v adresáři */src*. Každý balíček obsahuje dva spouštěcí launch soubory. První z nich je určen pro spuštění nodu v simulaci, zatímco druhů je určen ke spuštění nodu na skutečném dronu. Cílem všech čtyř nodů je přikázat dronu, aby vzlétl do výšky 2 m. Nody se liší v tom, zda jsou psané v jazyce C++ nebo v Pythonu a liší se také v tom, pro který firmware Pixhawku jsou napsané. Přestože firmware APM i PX4 využívají jednotný komunikační protokol MAVLink, liší se v některých příkazech. Jednou z těchto odlišností je i název letového módu, ve kterém je možné ovládat let palubním počítačem namísto RC soupravou. V případě PX4 se tento mód nazývá *OFFBOARD*, zatímco u APM je to mód *GUIDED*.

Balíček *test_px4_cpp* obsahuje ukázkový node pro firmware PX4, který je psaný v jazyce C++. Kód je převzat ze zdroje [42].

Balíček *test_px4_py* obsahuje ukázkový node pro firmware PX4, který je psaný v jazyce Python. Kód je převzat ze zdroje [43].

Balíček *test_apm_cpp* obsahuje ukázkový node pro firmware APM, který je psaný v jazyce C++. Jedná se o upravený kód *test_px4_cpp*, který byl upraven pro APM.

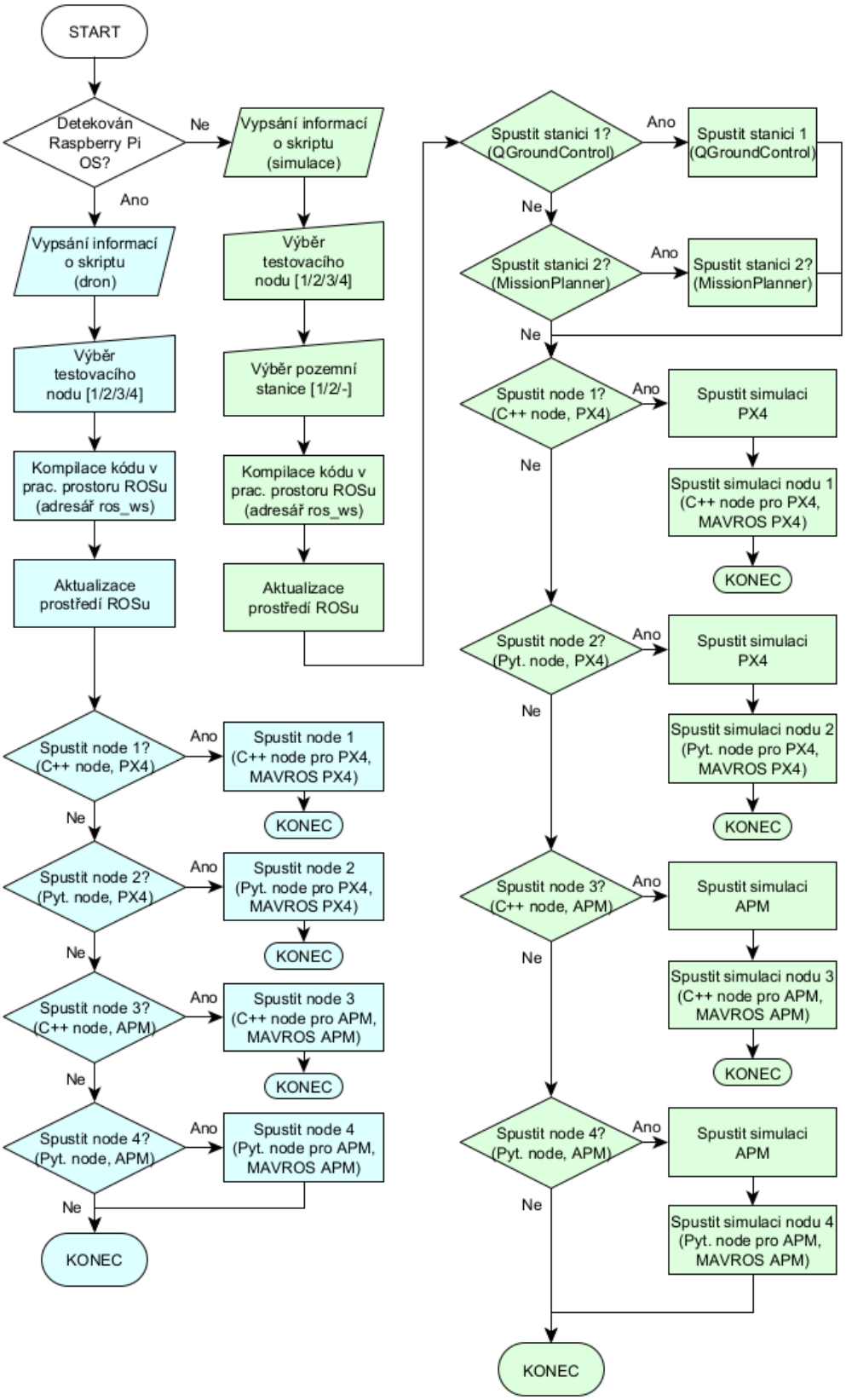
Balíček *test_apm_py* obsahuje ukázkový node pro firmware APM, který je psaný v jazyce Python. Jedná se o upravený kód *test_px4_py*, který byl upraven pro APM.

6.3.1 Spouštěcí skript pro testovací ROS nody

V rámci této práce byl napsán skript *launch_test.sh*, který se nachází v adresáři */ros_ws*. Lze ho nalézt na přiloženém CD/DVD v *příloze C – ros_ws.zip*. Jeho činnost je popsána vývojovým diagramem na obrázku 6.4.

Skript ihned po spuštění detekuje, zda byl spuštěn na Raspberry Pi OS, což je určující pro režim, ve kterém bude dál pokračovat. V případě, že je skutečně spuštěn na Raspberry Pi OS, pokračuje v režimu dronu. To znamená, že nody nebudou spouštěny v simulaci, ale budou se snažit propojit se skutečným Pixhawkem. Tento režim je ve vývojovém diagramu na obrázku 6.4 naznačen světle modrou barvou. Nejprve dojde k vypsání informací o skriptu a k dotázání, který node se má spustit. K tomu slouží volba číslice 1 až 4. Po zvolení nodu proběhne kompilace všech balíčků v pracovním prostoru a aktualizaci dostupných balíčků v prostředí ROSu. Následně dojde k zavolání odpovídajícího launch skriptu, který spustí MAVROS node pro daný firmware a také samotný požadovaný node.

Pokud se skript spustí na jiném operačním systému, předpokládá se, že se nejedná o dron, ale o simulační počítat. V tomto případě se vykoná světle zelená část vývojového diagramu z obrázku 6.4. Opět dojde k vypsání informací o skriptu a k dotázání na node, který se má spustit. Navíc dojde k dotázání, zda chce uživatel spustit některou z pozemních stanic. Další část kódu je obdobná jako v prvním případě. Rozdíl je pouze v tom, že se node spustí v režimu simulace a zároveň dojde ke spuštění simulace fyzikálního modelu dronu v aplikaci Gazebo. Dále se spustí simulace firmwaru PX4 nebo APM v závislosti na zvoleném nodu.



Obrázek 6.4 Zjednodušený vývojový diagram skriptu *launch_test.sh*

6.4 Ukázková mise pro Ardupilot

Hlavním výstupem této práce je vytvoření vlastního ukázkového nodu *mission_1_node.py*. Balíček nodu *mission_1* se nachází v *příloze C – ros_ws.zip*, adresář */src*. Jedná se o node, který má realizovat jednoduchou leteckou misi. Node je určen pro dron s firmwarem APM. Balíček nodu obsahuje následující spouštěcí soubory:

- mission_1_node
  - mission_1_sim_lite.launch
  - mission_1_sim.launch
  - mission_1.launch

První uvedený spouštěcí launch soubor slouží pro spuštění simulace s firmwarem APM, ale *samotný mission_1_node.py* se nespustí. Tato volba umožňuje spustit node v libovolném vývojovém prostředí a provádět jeho ladění. Launch soubor *mission_1_sim.launch* potom slouží pro klasické spuštění simulace včetně samotného nodu. Poslední spouštěcí soubor *mission_1.launch* slouží ke spuštění APM MAVROS a nodu *mission_1_node.py* na skutečném dronu.

Node má za cíl vzlétnout s dronem do výšky 2 m, kde setrvá po sobu jedné sekundy. Potom se přesune na lokální souřadnice (4; 4; 4) m vůči výchozímu bodu. Zde vyčká dvě sekundy a přesune se na globální souřadnice 49,2287°N 16.5731°E a vystoupá do výšky 4 m. Za další dvě sekundy se přesune na lokální souřadnice (-5; -5; 2) m. Předposlední souřadnice, na které dron poletí jsou 49.2288 °N 16.5732°E, přičemž setrvá ve výšce 2 m nad zemí. Nakonec se po dvou sekundách přesune na lokální souřadnice (0; 0; 5) m, což je jeho výchozí poloha. Po dalších dvou sekundách přistane a vypne motory.

Vzhledem k tomu, že node využívá globálního pozicování na konkrétní zeměpisné souřadnice, lze s ním reálně létat pouze na těchto souřadnicích, které se nacházejí poblíž FEKT VUT v Brně.

6.4.1 Knihovna mavros_lib.py

Tato knihovna vznikla v rámci diplomové práce a je součástí nodu *mission_1*. Knihovna využívá MAVROS node a vychází ze zdroje [35]. Nachází se v ní třída *MavrosLib*, která zajišťuje kontrolu připojení ROSu k Pixhawku, příp. k simulovanému firmwaru. Dále testuje dostupnost služeb MAVROSu a poskytuje řadu metod určených pro řízení dronu. Tato podkapitola popisuje nejdůležitější metody.

```
start_takeoff(altitude, mode)
```

Metoda *start_takeoff* má za úkol provést řadu operací, které mají zajistit nastavení dronu do požadovaného módu argumentem *mode*, spuštění motorů a vzlet do výšky *altitude* udané v metrem. Oba argumenty jsou povinné.

```
land(timeout)
```

Metoda *land* slouží k přistání dronu na aktuální pozici a vypnutí motorů.

```
local_pos(x, y, z, incremental, timeout)
```

Metoda *local_pos* slouží k letu s dronem na souřadnice vůči výchozí pozici. Souřadnice jsou dány argumenty (*x; y; z*) a jednotkou je metr. Argumenty *x* a *y* jsou povinné. Pokud není výška z zadána, dron setrvá v původní výšce. Argument *incremental* je typu boolean a může tedy nabývat pouze hodnot *True* a *False*. V případě volby *incremental=True*, bude dron přičítat nově zadané souřadnice k těm, které byly zadány dříve a bude je tedy vztahovat ke své aktuální poloze. Pokud bude *incremental=False,* tak vždy bude zadané souřadnice vztahovat ke startovní pozici.

```
global_pos(latitude, longitude, altitude, timeout)
```

Metoda *global_pos* je určena pro let na zadané zeměpisné souřadnice. Argument *latitude* udává zeměpisnou šířku a *longitude* udává zeměpisnou délku. Oba argumenty jsou povinné a jejich hodnoty se zadávají ve stupních. Nepovinný argument *altitude* udává výšku. Pokud není výška zadána, dron setrvá v původní výšce.

Většina metod je opatřena nepovinným argumentem *timeout*, který nastavuje maximální dobu, za jakou musí dojít k vykonání dané metody. Jeho výchozí hodnota je nastavena na 60 sekund. Metody jsou také ošetřeny proti chybějícím povinným argumentům a proti zadání příliš vysoké letové hladiny, která nesmí přesáhnout 100 m.

V rámci knihovny *mavros_lib.py* byl vytvořen ještě jeden soubor jmenující se *mavros_cons.py*. Ten obsahuje pouze konstanty, třídy typu *Enum* a texty.

6.4.2 Spouštěcí skript pro ukázkovou misi

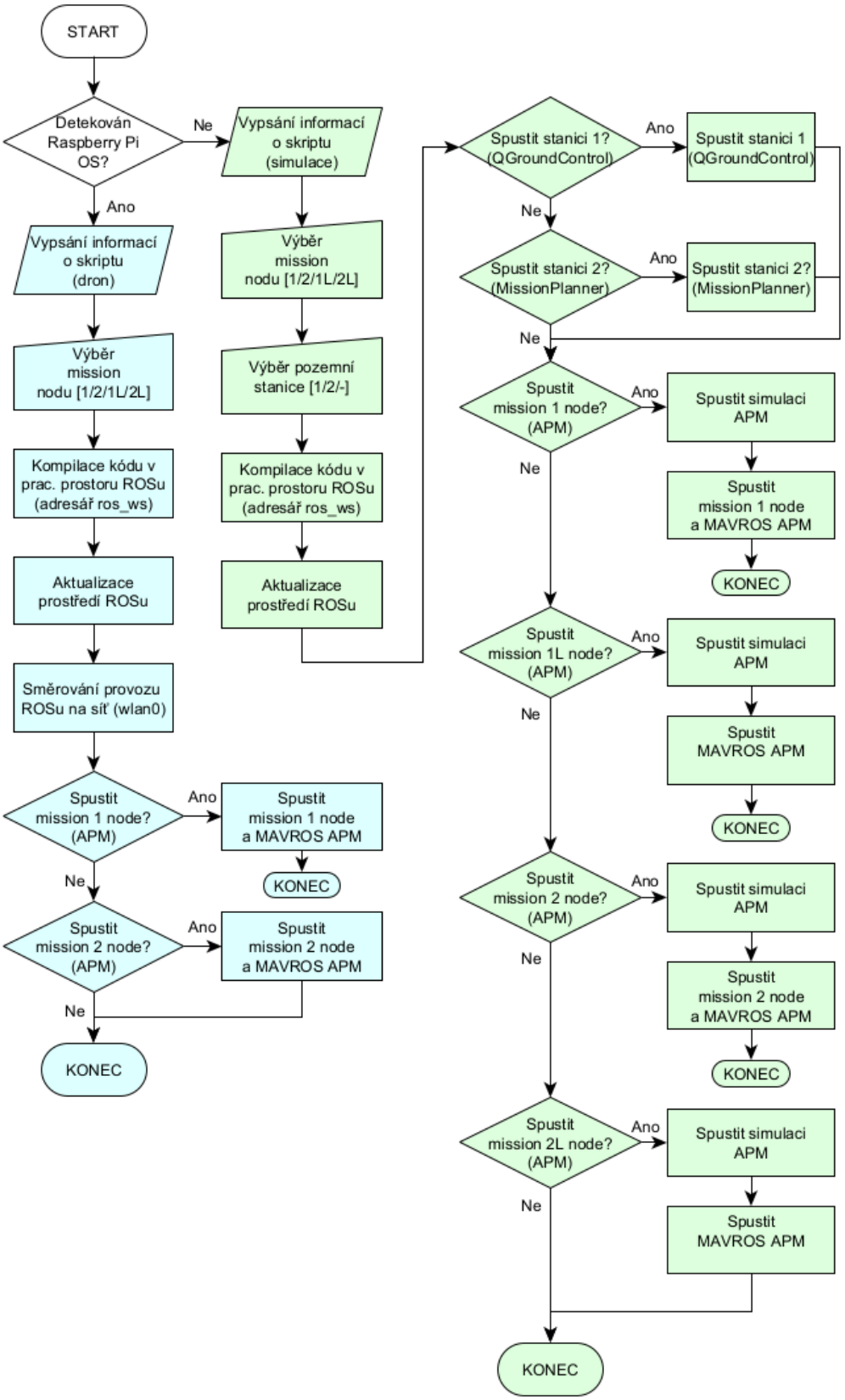
Ke spuštění mise byl v rámci práce napsán skript *launch_mission.sh*, který se opět nachází na přiloženém CD/DVD v rámci *přílohy C – ros_ws.zip*, adresář */ros_ws.*

Jeho činnost je popsána vývojovým diagramem na obrázku 6.5. Princip jeho funkce je obdobný jako v případě kapitoly 6.3.1. Skript je opět založen na detekci, zda je spuštěn

na dronu či na simulačním počítači. Podle toho se vykonají modře nebo zeleně vyznačené bloky.

V případě spuštění na dronu dojde k vypsání základních informací. Potom se dotáže uživatele, který node chce spustit. Skript má předpřipraven výběr mezi nodem *mission_1* a *mission_2*. Druhý uvedený není prozatím implementován. Tato možnost je tu předchystána pro případné rozšíření skriptu o další mise. Skript dále nastaví směrování ROSu na místní síť, ke které je Raspberry připojeno pomocí WiFi. To umožňuje spouštět další nody v rámci sítě nebo sledovat provoz ROSu na Raspberry. Následně se provede kompilace balíčků v pracovním prostoru ROSu a spustí se požadovaný node spolu s MAVROS APM nodem.

V případě spuštění skriptu na simulačním počítači, dojde ke spuštění požadovaného nodu v simulaci a případně se zároveň spustí požadovaná pozemní stanice. V případě výběru volby *1L* nebo *2L* dojde pouze ke spuštění simulace a MAVROS APM nodu, ale nespustí se samotný node mise. Ten je potom možno spustit v rámci vývojového prostředí a provádět jeho ladění.



Obrázek 6.5 Zjednodušený vývojový diagram skriptu *launch_mission.sh*

7. ZÁVĚR

V první části semestrální práce byl uveden přehled použitého open source hardwaru, kterým byl vybaven komerční dron DJI Matrice 100. Tyto úpravy zahrnovaly nahrazení originální řídicí elektroniky za řídicí jednotku Pixhawk 2.1, který je propojený s jednodeskovým počítačem Raspberry Pi 3B+. Toto propojení je realizováno pomocí sériové linky UART.

Další část práce se zabývá srovnáním dvou nejrozšířenějších firmwarů pro jednotku Pixhawk. Jedná se o firmwary APM a PX4. Oba projekty jsou si v mnohém podobné a do velké míry jsou vzájemně kompatibilní. V současnosti neexistuje žádné objektivní srovnání obou projektů. Srovnáváno obou platforem je také ztíženo skutečností, že oba projekty jsou velmi živé a okolo obou se odehrává velmi bouřlivý vývoj. Proto nelze říci, který z nich je vhodnější použít. Výběr tedy záleží pouze na subjektivním rozhodnutí autora. V této práci byl na dronu Matrice 100 nejprve použit, nakonfigurován a otestován firmware PX4. Odpovídající kapitola shrnuje postup a možnosti této konfigurace firmwaru.

Vlivem vládních opatření z důvodu šíření koronaviru SARS-CoV-2, nebylo možné dále pracovat na dronu M100, který se nacházel v laboratoři FEKT VUT. Z tohoto důvodu bylo další praktické testování prováděno na obdobném náhradním dronu. Tento dron byl již opatřen firmwarem APM.

Druhá polovina práce se věnuje instalaci softwaru na palubní počítač dronu a simulačního softwaru na stolní počítač. K tomu účelu v rámci diplomové práce vznikly a byly úspěšně otestovány dva instalační skripty. Prvním z nich je skript *setup_rpi.sh*, který zajistí instalaci nezbytného softwaru na palubní počítač Raspberry Pi. Druhý skript *setup_rpi.sh* slouží k instalaci softwaru na stolní počítač, který je určen k provádění simulací. Do softwaru instalovaného na obě zařízení se řadí především Robotický operační systém, zkráceně ROS, v případě simulačního PC doplněný o simulátor Gazebo.

Dále se práce zabývá možnostmi simulací a principem fungování ROSu. Shrnuje postup tvorby a spouštění ROS nodů. Následovalo zprovoznění ukázkových ROS nodů, které byly převzaty z internetových zdrojů. Ukázkové nody pro firmwary PX4 i APM jsou napsány v programovacím jazyce C++ i v Pythonu. Pro jejich snadné otestování byl autorem diplomové práce vytvořen skript *launch_test.sh*, který detekuje, zda je spuštěn na reálném dronu nebo na stolním počítači. Podle toho spustí vybraný ukázkový node na reálném dronu nebo v simulaci. Všechny ukázkové ROS nody mají za úkol vzlétnout s dronem do výšky 2 m. Všechny byly úspěšně otestovány v simulaci a node napsaný v jazyce Python pro dron s firmwarem APM, byl úspěšně otestován i na skutečném dronu.

Poslední část práce se věnuje tvorbě vlastní letecké mise, která je realizována prostřednictvím ROS balíčku pojmenovaného *mission_1*. Jedná se o hlavní výstup diplomové práce. Cílem této mise je vzlétnou s dronem a letět na několik blízkých souřadnic v okolí. Následně se má vrátit na výchozí souřadnice a přistát. Uvedený node

je napsán v jazyce Python pro APM firmware. V jeho rámci byla autorem vytvořena obecná knihovna *mavros_lib.py*, Ta obsahuje metody s ošetřenými příkazy pro let s dronem. Jedná se především o metody realizující funkce pro vzlet, přistání a pro let na konkrétní lokální souřadnice nebo globální zeměpisné souřadnice. Pro snadné spouštění v simulaci i na skutečném dronu, byl vytvořen další spouštěcí skript *launch_mission.sh*. Skript opět detekuje, zda je spuštěn na Raspberry na dronu, nebo na běžném počítači kde spustí simulaci. Funkčnost nodu *mission_1* byla úspěšně otestována v simulaci. V reálném prostředí byl proveden pouze jeden test, který selhal. Více testů nebylo možno provést, protože vlivem pandemie nebylo možné podstatnou část semestru cestovat mimo zastavěné území obcí nebo nebyly vhodné teplotní či povětrnostní podmínky pro testování.

Práce by se v budoucnu mohla zaměřit na zjištění důvodu, proč node *mission_1* v rámci simulace funguje, ale na skutečném dronu selhává. Dále by bylo vhodné zaměřit se na možnosti použití pokročilejšího ROS 2 namísto současného ROS 1.

LITERATURA

[1] HAMÁČEK, Vojtěch. *Vývoj bezpilotního prostředku pro autonomní mise.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/123211. Semestrální práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Luděk Žalud.

[2] *DJI: MATRICE 100* [online]. Shenzhen, 2019 [cit. 2020-01-02]. Dostupné z: https://www.dji.com/cz/matrice100

[3] HAMÁČEK, Vojtěch, Filip JECH, Vratislav FIKAR a Antonín ŠKAPA. *M17. Seznámení s UAV platformou* DJI *M100* [online]. 24.4.2019, 47 [cit. 2019-12-29]. Dostupné z: https://sites.google.com/site/vutrobotika/projekty/2019

[4] *LibrePilot* [online]. 2016 [cit. 2020-01-02]. Dostupné z: https://www.librepilot.org/site/index.html

[5] *Paparazzi UAV* [online]. 2018 [cit. 2020-01-02]. Dostupné z: http://wiki.paparazziuav.org/wiki/Main_Page

[6] *Pixhawk* [online]. Zurich: Meier [cit. 2020-01-02]. Dostupné z: https://pixhawk.org/

[7] *Dronecode Project, Inc.* [online]. San Francisco, 2019 [cit. 2020-01-02]. Dostupné z: https://www.dronecode.org/

[8] Cube Flight Controller. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-03]. Dostupné z: http://docs.px4.io/master/en/flight_controller/pixhawk-2.html

[9] HEX/ProfiCNC Here2 RTK GPS. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-03]. Dostupné z: https://docs.px4.io/v1.9.0/en/gps_compass/rtk_gps_hex_here2.html

[10] *Raspberry Pi Foundantion* [online]. Cambridge, 2020 [cit. 2020-01-02]. Dostupné z: https://www.raspberrypi.org/

[11] *ArduPilot* [online]. 2019 [cit. 2020-01-03]. Dostupné z: https://ardupilot.org/ardupilot/

[12] *PX4* [online]. Dronecode, 2019 [cit. 2020-01-03]. Dostupné z: https://px4.io/

[13] Ecosystem. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-03]. Dostupné z: https://px4.io/ecosystem/

[14] PX4 Architectural Overview. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-03]. Dostupné z: https://dev.px4.io/v1.9.0/en/concept/architecture.html

[15] TRIDGELL, Andrew. ArduPilot and DroneCode. *ArduPilot* [online]. 2019, 2016 [cit. 2020-01-03]. Dostupné z: https://discuss.ardupilot.org/t/ardupilot-and-dronecode/11295

[16] Learning ArduPilot — Introduction. *Ardupilot* [online]. c2020 [cit. 2021-5-13]. Dostupné z: https://ardupilot.org/dev/docs/learning-ardupilot-introduction.html

[17] History of ArduPilot. *Ardupilot* [online]. c2020 [cit. 2021-5-14]. Dostupné z: https://ardupilot.org/planner2/docs/common-history-of-ardupilot.html

[18] Code Overview (Copter). *ArduPilot* [online]. © 2020 [cit. 2021-5-15]. Dostupné z: https://ardupilot.org/dev/docs/apmcopter-code-overview.html
[19] Basic Configuration. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-04]. Dostupné z: https://docs.px4.io/v1.9.0/en/config/
[20] *QGroundControl* [online]. Dronecode, 2019 [cit. 2020-01-04]. Dostupné z: http://qgroundcontrol.com/
[21] Companion Computer for Pixhawk Series. *PX4* [online]. Dronecode, 2019 [cit. 2020-01-04]. Dostupné z: https://dev.px4.io/v1.9.0/en/companion_computer/pixhawk_companion.html
[22] Flight Modes. *PX4* [online]. Dronecode, 4/21/2021 [cit. 2021-5-15]. Dostupné z: http://docs.px4.io/master/en/flight_modes/
[23] Downloads. *Raspberry Pi Foundantion* [online]. Cambridge, 2020 [cit. 2020-01-04]. Dostupné z: https://www.raspberrypi.org/downloads/
[24] *Win 32 Disk Imager: Download* [online]. 2020 [cit. 2020-01-04]. Dostupné z: https://win32diskimager.download/
[25] Setting WiFi up via the command line. *Raspberry Pi Foundantion* [online]. Cambridge, 2020 [cit. 2020-01-04]. Dostupné z: https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md
[26] ROSberryPi/Installing ROS Melodic on the Raspberry Pi - ROS Wiki. *Documentation - ROS Wiki* [online]. 2020 [cit. 2021-5-15]. Dostupné z: http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Melodic%20on%20the%20Raspberry%20Pi
[27] Communicating with Raspberry Pi via MAVLink. *ArduPilot* [online]. 2019 [cit. 2020-01-05]. Dostupné z: https://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html
[28] Noetic/Installation/Ubuntu - ROS Wiki. Documentation - ROS Wiki [online]. Open Robotics, 2020, 2020-03-25 [cit. 2021-5-16]. Dostupné z: http://wiki.ros.org/noetic/Installation/Ubuntu
[29] GeographicLib: Installing GeographicLib. *GeographicLib* [online]. 2020 [cit. 2021-5-16]. Dostupné z: https://geographiclib.sourceforge.io/html/install.html
[30] Setting up the Build Environment (Linux/Ubuntu). *Ardupilot* [online]. c2020 [cit. 2021-5-16]. Dostupné z: https://ardupilot.org/dev/docs/building-setup-linux.html#building-setup-linux
[31] Ubuntu Development Environment | PX4 User Guide. *PX4* [online]. Dronecode, 2021 [cit. 2021-5-16]. Dostupné z: https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html#gazebo-jmavsim-and-nuttx-pixhawk-targets
[32] Ignition Rendering: Installation. *Ignition Robotics* [online]. [cit. 2021-5-16]. Dostupné z: https://ignitionrobotics.org/api/rendering/4.1/installation.html

[33] ROS with MAVROS Installation Guide | PX4 User Guide. *PX4* [online]. Dronecode, 2021 [cit. 2021-5-16]. Dostupné z: https://docs.px4.io/master/en/ros/mavros_installation.html
[34] *ROS* [online]. Open Robotics [cit. 2020-01-05]. Dostupné z: http://wiki.ros.org/
[35] Mavros - ROS Wiki. *Documentation - ROS Wiki* [online]. Open Robotics, 2020, 2018-03-03 [cit. 2021-5-17]. Dostupné z: http://wiki.ros.org/mavros
[36] ROS/Introduction - ROS Wiki. *Documentation - ROS Wiki* [online]. Open Robotics, 2020, 2018-08-08 [cit. 2021-5-16]. Dostupné z: http://wiki.ros.org/ROS/Introduction
[37] LIGOCKI, Adam. ROS. *Robotika* [online]. FEKT VUT [cit. 2021-5-16]. Dostupné z: https://sites.google.com/site/vutrobotika/navody/ros
[38] ROS/Tutorials - ROS Wiki. *Documentation - ROS Wiki* [online]. Open Robotics, 2020, 2021-04-09 [cit. 2021-5-16]. Dostupné z: http://wiki.ros.org/ROS/Tutorials
[39] Hardware in the Loop Simulation (HITL) | PX4 User Guide. *PX4* [online]. Dronecode, 2021 [cit. 2021-5-17]. Dostupné z: https://docs.px4.io/master/en/simulation/hitl.html
[40] Simulation | PX4 User Guide. *PX4* [online]. Dronecode, 2021 [cit. 2021-5-17]. Dostupné z: https://docs.px4.io/master/en/simulation/
[41] Using SITL. *ArduPilot* [online]. c2020 [cit. 2021-5-17]. Dostupné z: https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html
[42] MAVROS Offboard control example | PX4 User Guide. *PX4* [online]. Dronecode, 2021 [cit. 2021-5-17]. Dostupné z: https://docs.px4.io/master/en/ros/mavros_offboard.html
[43] Offb_node.py · GitHub. *GitHub Gist* [online]. c2021, 9 Aug 2016 [cit. 2021-5-17]. Dostupné z: https://gist.github.com/annesteenbeek/5370f62cf85bb9d6825327bff1b85293

SEZNAM PŘÍLOH

Přílohy jsou uloženy na přiloženém CD/DVD

Příloha A – setup_rpi.sh	(Bash skript)
Příloha B – setup_pc.sh	(Bash skript)
Příloha C – ros_ws.zip	(Pracovní adresář ROSu)