VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Katedra technických studií

Návrh programu pro ovládání dronu Parrot AR.Drone 2.0

bakalářská práce

Autor práce: Jaroslav Strnad

Vedoucí práce: PaedDr. František Smrčka, Ph.D.

Jihlava 2020

Vysoká škola
polytechnická
Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Jaroslav Strnad**
Studijní program:	Elektrotechnika a informatika
Obor:	Aplikovaná informatika
Název práce:	**Návrh programu pro ovládání dronu Parrot AR.Drone 2.0**
Cíl práce:	Cílem práce bude vytvoření programů pro ovládání letu a autonomního řízení letu dronu Parrot AR.Drone 2.0. Budou vytvořeny knihovny základních typických programů (start, visení, přistání, let po předem určeném obrazci, přelet a přistání). Programovacím jazykem budou JavaScript a Pithon. Hotové programy bude možné testovat ve 3D simulátoru Gazebo se sadou balíčků ROS a na reálném dronu. Simulátor Gazebo bude nainstalovaný ve virtualizovaném operačním systému Ubuntu. Pro virtualizaci bude použit virtualizační program Virtualbox. Součásti práce bude vyexportovaný obraz operačního systému Ubuntu včetně celého simulačního prostředí. Bakalářská práce bude obsahovat manuál o programování dronu Parrot AR.Drone 2.0.

PaedDr. František Smrčka, Ph.D.
vedoucí bakalářské práce

doc. Ing. Zdeněk Horák, Ph.D.
vedoucí katedry
Katedra technických studií

Abstrakt

Cílem této bakalářské práce je vytvoření ovládacích programů pro Parrot AR.Drone 2.0 ve virtualizovaném operačním systému Ubuntu. V úvodu se seznámíme s problematikou. Dále následuje analýza řešeného problému a náčrt použitých technologií. Následně se zaměříme na přípravu operačního systému a simulačního prostředí. V neposlední řadě si popíšeme proces programování ovládacích skriptů. V závěru zhodnotíme výsledky. Aplikace byla vytvořena za pomocí jazyku Python. Použité simulační prostředí je Gazebo podpořené knihovnami ROS. Operační systém Ubuntu je virtualizován v programu VirtualBox.

Klíčová slova

Drone, Ubuntu, Python, VirtualBox, Gazebo, ROS

Abstract

The goal of this bachelor's thesis is to create drivers for Parrot AR.Drone 2.0 in the virtualized operating system Ubuntu. In the introduction we will get acquainted with the issue. Next follows an analysis of the problem and an outline of the technologies used. Subsequently, we will focus on the preparation of the operating system and the simulation environment. Last but not least, we will describe the process of programming control scripts. In the end part of the thesis we evaluate results. Application was created with use of language Python. The simulation environment used is Gazebo, supported by ROS libraries. The Ubuntu operating system is virtualized in the VirtualBox application.

Key words

Drone, Ubuntu, Python, VirtualBox, Gazebo, ROS

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Souhlasím s umístěním bakalářské práce v knihovně VŠPJ a s jejím užitím k výuce nebo k vlastní vnitřní potřebě VŠPJ.

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 29. července 2020

…….
Podpis studenta

Poděkování

Zde bych chtěl poděkovat všem lidem, co mi byli oporou ve tvorbě této práce. Zejména vedoucímu mé práce PaedDr. Františkovi Smrčkovi, Ph.D. za cenné rady, konzultace a zapůjčení drona. Speciální poděkování patří MUDr. Haně Strnadové, mé matce, za to, že to se mnou vydržela.

Obsah

Seznam obrázků

Seznam použitých zkratek

UAV – Unmanned aerial vehicle, bezpilotní letadlo

GUI – Generated user interface, grafické uživatelské rozhraní

VDI – Virtual disk image, možný formát pro virtualizované systémy

VMDK – Virtual machine disk, možný formát pro virtualizované systémy

VHD – Virtual hard disk, možný formát pro virtualizované systémy

BSD – Berkeley Software Distribution, jeden z prvních unixových systémů

SDK – Software development kit, sada vývojových nástrojů

LTS – Long-term support, verze operačního systému s dlouhodobou podporou

BIOS – Basic Input-Output System, firmware osobních počítačů

APT – Advanced Package Tool, viz. 4.3. Příprava nainstalovaného Ubuntu

Sudo – substitute user do, příkaz k vykonání operace s administrátorskými právy

XML – Extensible Markup Language, značkovací jazyk

URDF – Unified Robot Description Format, XML formát reprezentující robotický model

Hz – Hertz, jednotka frekvence

NAT – Network Address Translation, způsob úpravy síťového provozu přepisem zdrojové, nebo cílové IP adresy

WANET – Wireless ad hoc network, decentralizovaný typ bezdrátové sítě, nezávislý na předem existující infrastruktuře

GPS – Global Positioning System, globální družicový polohový systém

1 Úvod

Dron (od anglického UAV – Unmanned aerial vehicle) je technologický fenomén 21. století. V dnešní době slouží především k domácímu užití, ve formě vzletné fotografické platformy, či jeden ze způsobů rekreačních závodů, a nebo také k průmyslovému užití v rámci přepravy balíčků, ať již v rámci regulérního transportu zboží (Amazon Prime Air), či ilegálních aktivit, například Kolumbijských a Mexických drogových kartelů.

Drony mají ovšem dlouholetou historii, datující se již od počátku 20. století s prvními patenty na vzletné zařízení s dálkovým ovládáním. Hlavním prvotním účelem dronů bylo nasazení pro armádní účely; první prototypy byly používány jako pokročilý způsob tréninku míření, dokud nepřišel nápad používat tato zařízení jako pozorovací/infiltrační zařízení v reálných válečných konfiktech, především ve Vietnamské válce a ve válce v Kosovu. Válečný pokrok těchto zařízení se ani poté nezastavil, a po tragédii 11. září byly armádní drony vybaveny raketami, pro možnost vyhlazení teroristických jednotek bez potřeby potenciální oběti pilota. Tyto armádně vybavené drony, již extrémně pokročilé a velice se lišící od civilních dronů, se používají ve válečných konfliktech dosud. A s nimi přibývají debaty o etice používání těchto zařízení.

Regulérního člověka ovšem zajímají spíše ony klasické, lehce dostupné drony. Pro klasického uživatele se hlavně jedná o hračku, či skvělý způsob pořizování fotografií. Tyto drony ovšem nejsou zrovna nejlevnější záležitost, s cenovým rozsahem od pár tisíc pro malé, indoor drony, do několika desítek tisíc pro high-end zařízení se 4K (rozlišení) kamerou. Z tohoto důvodu není nejlepší nápad tak drahé zařízení používat rovnou „na ostro", zejména pokud je uživatel začátečník, nemá zkušenosti s ovládáním a nezná možné vnější jevy, které by mohly s dronem reagovat/poškodit ho. Naštěstí v dnešní době máme způsoby, jak si takové zařízení vyzkoušet nanečisto, a to simulační prostředí pro roboty, mezi které tato zařízení patří.

Jedním takovým prostředím je Gazebo, otevřený software (open-source) pro robotickou simulaci. Aby práce s tímto prostředím nebyla tak náročná, hodí se použít open-source knihovny a nástroje ROS, které pomohou především s ovladači dronu a vizualizací. S těmito nástroji a za pomoci programovacího jazyku Python vytvoříme ovládací aplikaci pro specifický dron Parrot AR.Drone 2.0. Aplikace bude tvořena v operačním systému Ubuntu, distribuci GNU/Linux, která je nejvíce podporovaným operačním systémem pro

Gazebo. Ubuntu také bude virtualizováno v open-source virtualizačním prostředí VirtualBox, od společnosti Oracle.

2 Analýza řešeného problému

2.1 Seznámení s problematikou

Vzhledem k tomu, že neexistuje volně dostupný ovladač na drony typu Parrot AR.Drone 2.0, který by byl open-source a byl schopen ovládat daný dron jak v realistických simulačních prostředích, tak v reálném světě, je potřeba onen ovladač naprogramovat.

2.2 Vymezení cílů

Cílem této práce je naprogramování ovládacích aplikací pro dron typu Parrot AR.Drone 2.0 v simulačním prostředí Gazebo. Pro tento účel je potřeba zprovoznit operační systém Ubuntu na virtualizačním rozhraní VirtualBox. Takto zprovozněný operační systém se poté dá exportovat a implementovat na VirtualBoxu na jiném zařízení. Ve virtualizovaném operačním systému Ubuntu je potřeba mít nainstalované simulační prostředí Gazebo společně s knihovnami/nástroji ROS.

Hlavním úkolem je poté za pomoci programovacího jazyka Python naprogramovat ovládací aplikace. Ovládací aplikace budou podporovat letové funkčnosti drona. Ovládací aplikace bude možné využít jak v simulačním prostředí Gazebo na simulovaném dronu, tak na reálném Parrot AR.Drone 2.0

3 Použité technologie

Pro sestavení plnohodnotného simulačního prostředí ve virtualizovaném operačním systému budeme potřebovat velké množství technologií. Zde si uvedeme výčet těchto technologií a jejich popis.

3.1 Ubuntu

Operační systém Ubuntu od firmy Canonical Ltd je linuxová distribuce, odnož široce užívané distribuce Debian GNU/Linux. Ubuntu je na rozdíl od Debianu vytvořen tak, aby byl co nejvíce user friendly a aby fungoval „out of the box“, tudíž aby vše fungovalo hned po instalaci (podobně jako Windows), což u Linuxových distribucí většinou není. Uživatelským prostředím (GUI) je GNOME 3, které nedávno nahradilo uživatelské prostředí Unity, což bylo proprietární uživatelské prostředí vyráběné Canonical Ltd. Dají se ovšem také nainstalovat odnože Ubuntu jako například Kubuntu, či Xubuntu, které přicházejí s alternativními uživatelskými prostředími jako KDE a XFCE. [1]

Obrázek 1 Ubuntu od Canonical Ltd

3.2 VirtualBox

Oracle VM VirtualBox je open-source virtualizační hypervisor (program, umožňující spuštění virtuálních systémů a zařizující přidělení jejich prostředků) od společnosti Oracle Corporation. Podporuje virtualizaci mnoha různých operačních systémů, od

Windows, přes GNU/Linux do Solaris. V základu podporuje pouze 32 bitové operační systémy, ovšem procesory s podporou virtualizace (novodobé Intel a AMD procesory) jsou schopné virtualizovat 64 bitové operační systémy. Virtuální pevné disky VirtualBoxu jsou ve formátu VirtualBox Disk Image (.vdi). Jsou podporované i formáty VMDK (kompatibilní s Vmware Workstation) a VHD (kompatibilní s Windows Hyper-V). [2]

Obrázek 2 VirtualBox od Oracle

3.3 Python

Python je programovací jazyk vytvořený Python Software Foundation. Jedná se o vyšší, objektově orientovaný programovací jazyk, s jednou z nejjednodušších syntaxí. Je kompatibilní se všemi běžnými operačními systémy a ve většině Linuxových distribucí je v rámci základní instalace. V současné době jsou používány dvě verze tohoto jazyka, 2.7 a 3.X, přičemž podpora verze 2.7 bude již brzy končit. [3]

Obrázek 3 Python od Python Software Foundation

3.4 Gazebo

Gazebo je robotické simulační prostředí, vytvořené v roce 2002 na Univerzitě Jižní Kalifornie. Jedná se o open-source prostředí, podporované mnoha open-source knihovnami, zejména ROS, pro jejíž komunitu je to v dnešní době nejvíce používané prostředí. V dnešní době je Gazebo vlastněno společností Open Source Robotics Foundation. [4]

Obrázek 4 Gazebo od Open Source Robotics Foundation

3.5 ROS

Robot Operating System (ROS) je kolekce softwarových frameworků pro simulaci robotiky. Jedná se o open source software pod BSD licencí, napsaný v C++ a Pythonu. Podporuje především GNU/Linux a Windows, s experimentální MacOS verzí. [5]

Obrázek 5 ROS od Stanford Artificial Intelligence Laboratory

3.6 Ardrone_Autonomy

Ardrone_Autonomy je ROS ovladač pro Parrot AR.Drone 1.0 a 2.0 založený na oficiálním AR-Drone SDK (Software Developement Kit). Byl vytvořen v laboratořích autonomie na Univerzitě Simona Frasera v Kanadě, hlavním tvůrcem byl Mani Monajjemi. [6]

3.7 Tum_Simulator

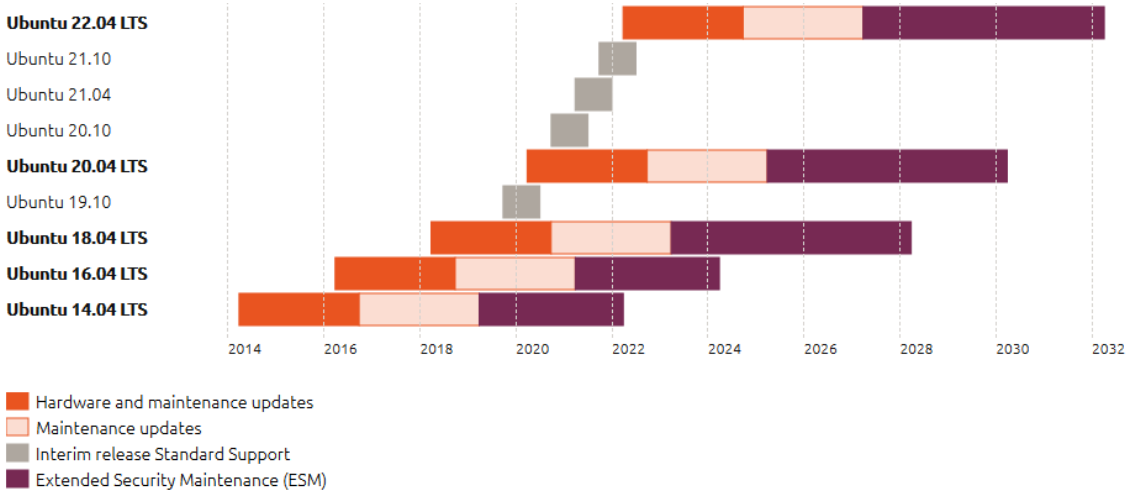
Tum_Simulator je implementace Parrot AR.Drone 2.0 do simulačního prostředí Gazebo. Byla vytvořena Juergenem Sturmem a Hongrongem Huangem z Technické Univerzity Mnichov. Implementuje plně funkční model dronu s plnou kompatibilitou Ardrone_Autonomy. [7]

4 Příprava prostředí k programování ovladačů

Před tím, než se můžeme pustit do samotného programování, potřebujeme plně funkční prostředí, ve kterém jej budeme vykonávat. V této sekci se budeme zabývat volbou verze operačního systému, samotnou instalací a přípravou nainstalovaného operačního systému do plně funkčního simulačního prostředí.

4.1 Volba verze Ubuntu

Prvním problémem, na který narazíme, je volba verze operačního systému Ubuntu. Verze operačního systému Ubuntu se rozlišují podle délky podpory od společnosti Canonical na verze LTS (Long Term Support, dlouhodobá podpora) a verze Interim (prozatimní, s podporou na devět měsíců). K dnešnímu datu nejnovější verze operačního systému Ubuntu je verze 20.04 LTS (kódové označení Focal Fossa), značící, že vyšla v dubnu roku 2020.



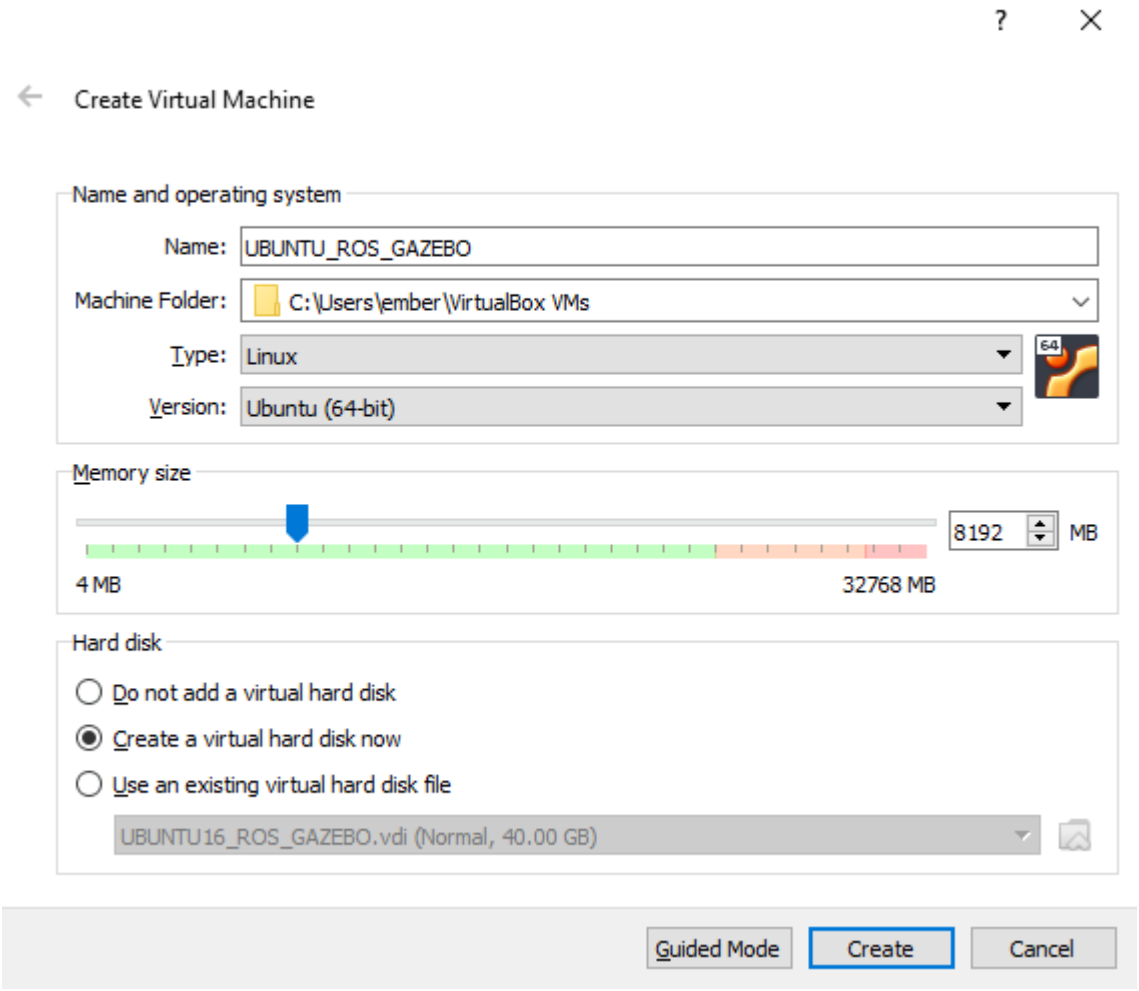
Obrázek 6 Současně podporované verze Ubuntu [8]

Nejnovější verzi ovšem použít nemůžeme, kvůli nekompatibilitě se simulačním prostředím Gazebo a balíčky ROS. Nejnovější verze Ubuntu, kompatibilní s těmito dvěma stěžejnímy prvky této bakalářské práce, je verze 16.04.6 LTS (kódové označení Xenial Xerus), která je ovšem na okraji kompatibility, a v mé práci jsem se v ní setkal s několika problémy. Ty byly většinově redukovány při použití mnou finálně zvolené verze, 14.04.6 LTS (kódové označení Trusty Tahr).

Pro účely práce jsem zvolil 64 bit verzi Ubuntu. VirtualBox oficiálně doporučuje při tvorbě virtuálního pevného disku vymezit minimálně 512 MB (megabyte) RAM paměti, práce byla ovšem tvořena na stolním počítači s 32 GB (gigabyte) paměti, a bude prezentována na laptopu s pamětí 8 GB. Mezi uživateli VirtualBox se praví, že nejjistější pro stabilitu jak nativního, tak virtualizovaného systému je doporučeno vymezit jednu čtvrtinu celkové RAM paměti počítače, naštěstí VirtualBox poskytuje možnost jednoduše měnit hodnotu vymezené paměti.

4.2 Instalace virtualizovaného Ubuntu

Vytvoření nového virtualizovaného systému začneme v programu VirtualBox kliknutím na tlačítko New, které nás vezme do instalace virtuálního disku. Virtualizované Ubuntu bude vytvořeno ve standartizovaném formátu VDI (Virtual Disk Image). Vytvořené Ubuntu bude mít přidělených 8 GB RAM paměti, z tohoto důvodu budeme vytvářet 64bit verzi operačního systému. V normálních podmínkách možnost přepnutí z 32-bit verze na 64-bit verzi není možná, pokud uživatel neumožnil procesorovou virtualizaci v BIOS (Basic Input-Output System, firmware osobních počítačů), jedná se o VT-x (Intel Virtualization Technology) pro procesory firmy Intel, nebo AMD SVM (Secure Virtual Machine) pro procesory firmy AMD.



Obrázek 7 Vytvoření Ubuntu VDI

V další části se nás VirtualBox zeptá, jak velký chceme virtualní pevný disk. Experimentálně jsem došel k závěru, že disk o velikosti 12 GB je dostačující ke všem našim potřebným úkonům. V případě, že by z nějakého důvodu tato velikost nestačila, můžeme také zvolit možnost vytvořit disk s velikostí dynamicky alokovanou, v tomto případě by disk rostl na kapacitě, bylo by to potřeba.

Po vytvoření virtuálního pevného disku, na který budeme systém Ubuntu instalovat, je potřeba zajít do nastavení virtuálního disku, a pod položkou Display vypnout 3D akceleraci virtuálního systému. Z tohoto důvodu celková grafická stránka systému bude pomalejší, protože nebude využívat akceleraci grafické karty hostitelského počítače, a místo toho bude využívat pouze software akceleraci VirtualBox, ovšem je nutné toto udělat, aby simulační prostředí Gazebo na virtualizovaném systému vůbec fungovalo. Gazebo má velice specifické požadavky na grafické karty, plně podporuje grafické karty značky NVIDIA a jisté integrované grafické karty značky AMD. [9] 3D akcelerace

VirtualBox je stále experimentální funkce, která se snaží podpořit grafickou výpočetní sílu virtualizovaného systému využitím grafické karty systému hostitelského, a ve většině případů funguje bez problémů. S akcelerovaným Gazebo ovšem nastávají problémy a v případě, že nespouštíme prázdnou simulaci bez jakýchkoliv modelů, odmítá se spustit. S vypnutou 3D akcelerací naše simulační prostředí funguje bez problémů, ovšem velice pomalu, co se týká počtu snímků za vteřinu.

Ostatní nastavení virtuálního disku se měnit nemusí, tudíž po spuštění virtualizovaného systému budeme žádáni o instalační soubor operačního systému. Samotná instalace Ubuntu je standartní plná instalace s přemazáním celého (dosud prázdného) disku bez nutnosti instalace dalších funkcí, jako jsou například proprietární ovladače pro tiskárny, či bluetooth.

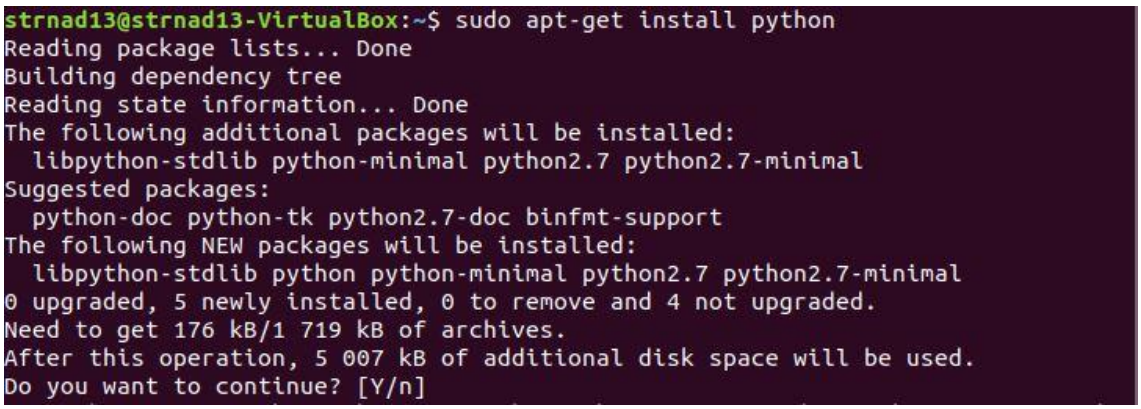
4.3 Příprava nainstalovaného Ubuntu

Po instalaci restartujeme virtualizované Ubuntu, aby se správně nastavily ovladače a bootloader (program uchovávající informaci o umístění operačního systému v úložném zařízení), čímž se celá instalace finalizuje. Po nahrání (boot) do hotového systému, první věc, kterou uděláme je, že zaktualizujeme Ubuntu, k čemuž budeme automaticky notifikováni aplikací Ubuntu System Updater, předinstalovanou s Ubuntu. Ta zlehčuje aktualizaci systému, a tudíž nemusíme aktualizovat skrze terminál, jak tomu je u jiných distribucí.

První věc, kterou potřebujeme nainstalovat, jsou knihovny jazyka Python. V nejnovějších verzích Ubuntu je nejnovější verze Python 3 již nainstalovaná jako součást operačního systému. Z důvodu případné kompatibility nainstalujeme také verzi Python 2.7, přezdívanou pouze Python.

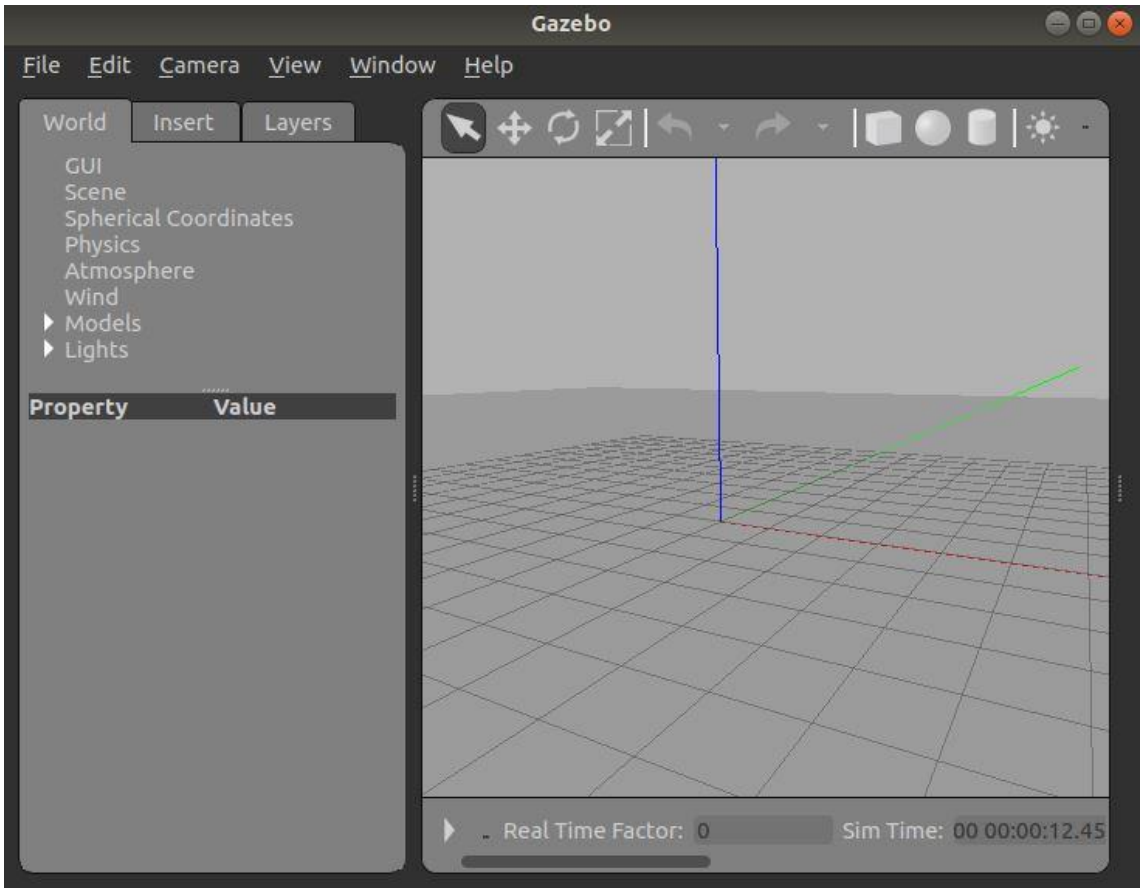
V Ubuntu se dají balíčky jednoduše instalovat skrze Ubuntu Software aplikaci. Já ovšem preferuji instalaci skrze samotný terminál. Ubuntu, protože je založené na distribuci Linux Debian, používá k přístupu k repozitářím systém APT (Advanced Packaging Tool). Tento systém se dá využít z mnoha různých terminálových aplikací, z čehož uživatelsky nejpřívětivější je apt-get. APT je mnohoúčelový systém, schopen nejen instalace balíčků, ale také jejich odebírání, či aktualizace (ať již balíčků, či také celého systému).

Terminál buď můžeme přepnout do administrátorského (sudo) módu, kdy všechny příkazy budou automaticky s administrátorskými právy, nebo můžeme před samotný příkaz přidat “sudo”, načež se nás terminál zeptá na administrátorské heslo. Po zadání příkazu “sudo apt-get install Python” se vyčtou všechny potencionálně nainstalované balíčky, aby se uživatel ujistil, že neinstaluje nic, co by nechtěl. Po akceptování tohoto výběru se balíčky stáhnou, extrahují a nainstalují již automaticky.



Obrázek 8 Instalace balíčků v terminálu

Pro instalaci simulačního prostředí Gazebo si můžeme vybrat jeden ze dvou způsobů. Buď instalaci pomocí podpůrného stahovacího programu Curl, a nebo několika krokovou instalaci za pomoci přidávání oficiálních Gazebo zdrojů do systému APT. Kvůli praktičnosti jsem vybral jednodušší možnost za pomocí programu Curl. Curl je software pro stahování souborů přes počítačovou síť za pomocí velké škály protokolů. Pomocí něj, instalace Gazebo zabere pouhý jednořádkový příkaz v terminálu. [10]



Obrázek 9 Nainstalované Gazebo

Dále potřebujeme nainstalovat podpurné knihovny ROS. Verze ROS musí být ROS Indigo Igloo, verze podporující vybranou verzi operačního systému Ubuntu 14.04.06. Instalace těchto knihoven je více rozsáhlá a obtížnější, protože se knihovny nevyskytují v oficiálních Ubuntu repozitářích. Operační systém Ubuntu obsahuje seznam repozitářů, do kterého lze přidávat ty, které nejsou oficiálně podporované. Seznam repozitářů se nachází ve složce "/etc/apt/sources.list.d/", kam lze přidávat nové listy repozitářů. APT také potřebuje tzv. APT key, klíč díky kterému Ubuntu bude vědět, že souborům z přidaného repozitáře se dá věřit, a tudíž nebude mít problémy s jejich stažením a instalací. Jedná se o proces, který se lze udělat jak přes terminál, tak přes zabudovanou aplikaci Ubuntu Software Centre.

Po instalaci je potřeba inicializovat příkaz "rosdep", který byl instalovaný společně s ROS a zjišťuje a instaluje potřebné závislosti k běhu ROS a v další části připraví vybrané prostředí na sestavení do funkční simulace. [11]

Po instalaci dvou nejkritičtějších prvků, které nám umožní vytvářet ovládání drona, si potřebujeme vymezit pracovní místo na disku pro tyto úkony. Vytvoříme složku

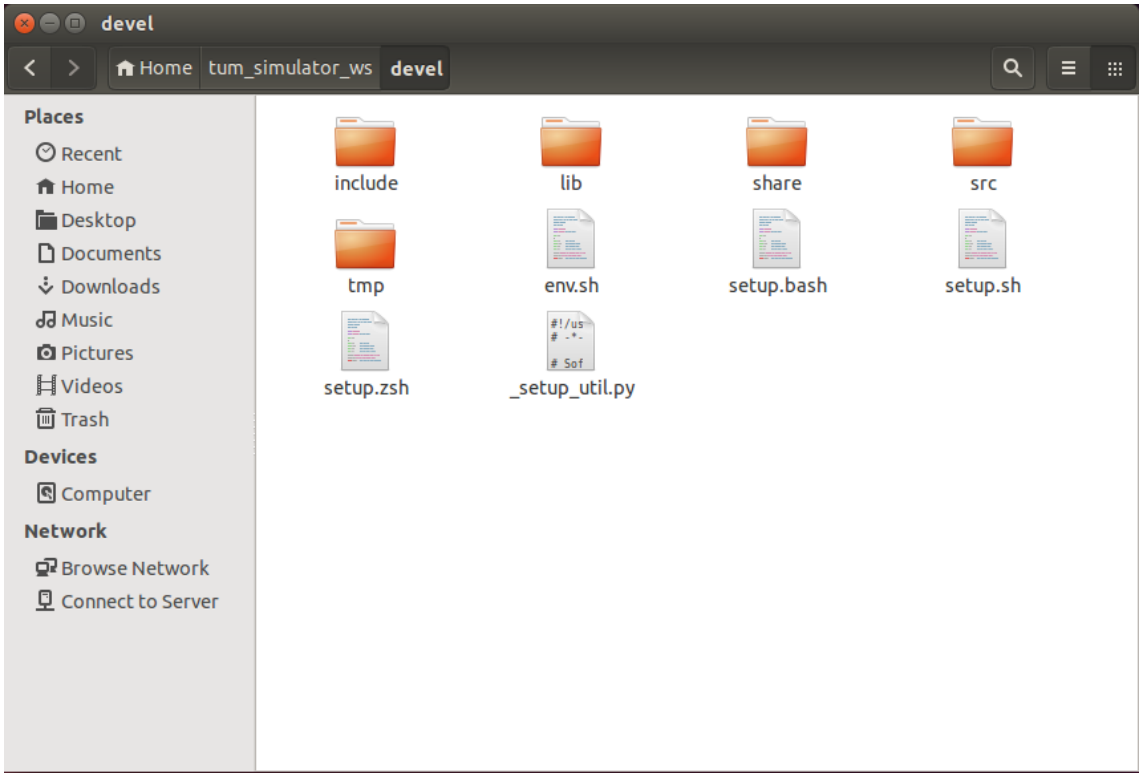
libovolného jména (v našem případě “tum_simulator_ws”, poté v ní vytvoříme složku “src”, která bude obsahovat další potřebné složky a námi vytvořené ovládací skripty. Pro naše účely programování ovládání pro Parrot AR.Drone 2.0 budeme potřebovat dva dodatečné prvky: Tum_Simulator a Ardrone_Autonomy. Tum_Simulátor je také závislý na Ardrone_Autonomy, tudíž i kdybychom jej nechtěli, tak je potřebujeme nainstalovat společně. Tyto balíčky stáhneme do námi vytvořené složky za pomoci aplikace git, vrátíme se do předchozí hlavní složky, a poté kombinací “rosdep” a “catkin_make” sestavíme finální použitelný simulátor.

5 Nastavení prostředí a tvorba ovladačů

Teď, když máme všechny potřebné komponenty nainstalované, musíme nastavit simulační prostředí Gazebo, aby kooperovalo s našimi následujícími ovladači. Vysvětlíme si, jak překrýt shell pracovním prostředím, jak pracovat s ROS soubory a nakonec jak programovat ovládání samotného drona.

5.1 Příprava terminálu pro práci s ROS

Pokaždé, když otevřeme nový terminál pro interakci s naší simulací, potřebujeme zajistit, aby náš pracovní prostor (workspace) překryl prostředí proměnných Unix shell (textové uživatelské rozhranní, v našem případě defaultní Bash) za pomoci instalačních skriptů. Tyto skripty byly vytvořeny v předchozím kroku při sestavení finalizované simulátoru příkazem “catkin_make”. Nachází se v mateřské složce simulátoru, v podsložce “devel”.



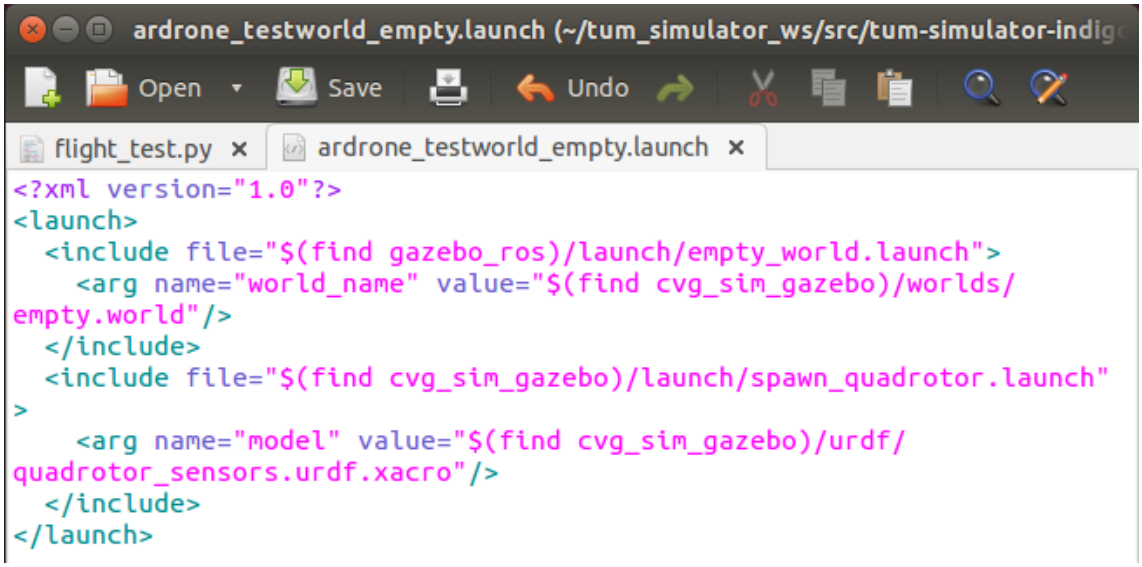
Obrázek 10 Složka /devel/ obsahující instalační skripty

V této složce najdeme tři různé instalační skripty “setup”, každý s příponou, značící pro který Unix shell jsou určené (.bash pro defaultní Bash shell, .sh pro Bourne shell a .zsh pro Z shell). Abychom mohli překrýt shell naším pracovním prostředím simulátoru, a tudíž s ním dále interagovat, využijeme vestavěný příkaz “source”, který zajistí načtení

onoho instalační skriptu a jeho provedení v aktivním shell. Abychom si ulehčili práci a nemuseli plný příkaz “source ~/tum_simulator_ws/devel/setup.bash” psát při otevření každého nového terminálu, můžeme příkaz vepsat do souboru “.bashrc”, který obsahuje veškeré příkazy, které se provádí při spuštění nového terminálu.

5.2 Práce s ROS

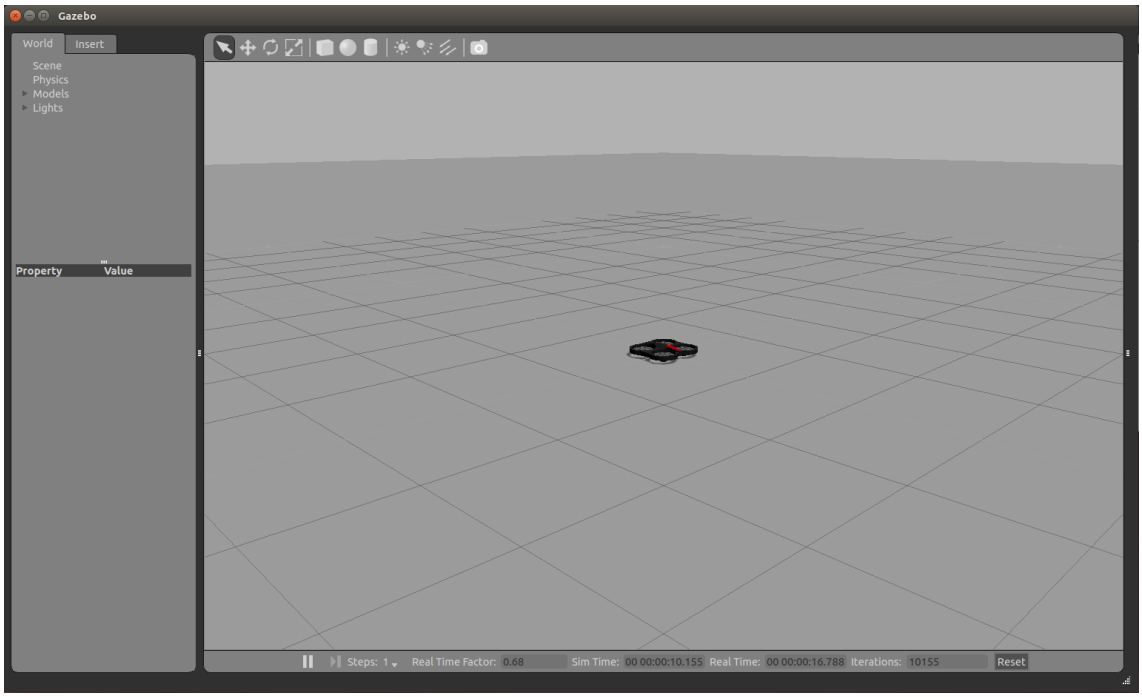
Nyní máme vše potřebné k programování a simulaci Parrot AR.Drone 2.0 hotové a připravené. Fungující simulátor, obsahující Parrot AR.Drone 2.0 můžeme spustit za pomoci příkazu “roslaunch”, který se stará o spuštění několika ROS Node (spustitelný proces vytvářející výpočet v rámci ROS). [12] Tyto ROS node jsou sepsané v rámci XML (Extensible Markup Language, datový soubor) souborů s příponou “.launch”. Ty obsahují informace, které ROS node spustit, jejich parametry, a zda chceme společně s nimi spustit další launch soubory. Veškeré launch soubory budou uloženy v adresáři “~/tum_simulator_ws/src/tum-simulator-indigo/cvg_sim_gazebo/launch”. Díky překrytí naším pracovním prostorem nemusíme vypisovat celou tuto adresu, stačí například “roslaunch cvg_sim_gazebo ardrone_testworld.launch”.



Obrázek 11 Ukázka základního launch souboru

Tento launch soubor, se kterým budeme budeme pracovat pro účely naší simulace, spustí dva další launch soubory, každý s jedním argumentem. První spuštěný launch soubor obsahuje ROS node informace a typ světa, který se vytvoří v prostředí Gazebo, druhý ROS node zajišťuje incializaci samotného prostředí Gazebo. V druhém launch souboru

máme ROS node informace o typu modelu dronu, který bude vybrán (v našem případě Parrot AR.Drone 2.0) ve formátu URDF (Unified Robot Description Format, XML soubor reprezentující robotický model), informace v jakém stavu bude dron vytvořen ve světě a nakonec samotnou akci vytvoření funkčního modelu dronu. Po spuštění tohoto launch souboru za pomoci již zmíněného příkazu "roslaunch" získáme otevřené prostředí Gazebo s nakonfigurovanou krajinou a funkčním modelem dronu. Tuto simulaci můžeme nechat běžet na pozadí, zatímco budeme programovat ovládací skripty, pokud nám nebude vadit, že celkový výkon systému bude pomalejší, kvůli již zmíněné vyplé 3D akceleraci.



Obrázek 12 Funkční Gazebo prostředí s Parrot AR.Drone 2.0

5.3 Programování ovladačů pro Parrot AR.Drone 2.0

Teď se můžeme pustit do tvorby samotných ovládacích programů/skriptů. Ve složce "~/tum_simulator_ws/src/" jsem si vytvořil složku s názvem controls, která bude obsahovat všechny vytvořené ovladače. Složky v tomto smyslu slova jsou takzvané catkin package, části našeho pracovního prostoru, který jsme vytvořili a který je spravován pomoci přikazu "catkin", který jsme již používali při sestavě celé struktury simulátoru. Nemůžeme tudíž vytvořit pouhou složku, ale za pomocí příkazu "catkin_create_pkg controls std_msgs rospy roscpp" vytvoříme nový catkin package,

který bude obsahovat potřebné soubory pro další chod, v našem případě “package.xml”, manifest, který je nutný v každém catkin package, obsahující informace o novém catkin package včetně výpisu podpurných knihoven, které jsme indikovali v příkazu po názvu catkin package, a CmakeLists.txt, obsahující informace o sestavení nového catkin package. Po vytvoření nového catkin package jej musíme sestavit, opět za pomoci “catkin_make” v mateřské složce. [13]

Nyní máme místo k ukládání našich ovladačů, tudíž se můžeme přesunout k jejich tvorbě. Ovladače budou tvořeny v jazyce Python, z tohoto důvodu si vytvoříme prázdný dokument s příponou “.py”. První ovladač, který budeme programovat, zajistí základní autonomní operaci vzestupu, letu dopředu, sestupu a vypnutí motorů. Pro tyto účely jsem vytvořil soubor “flight_test.py”.

5.3.1 Ovladač autonomního letu

Soubor flight_test.py se dělí na tři hlavní části: import potřebných knihoven a hodnot z “.msg” souborů, což jsou specifické ROS soubory popisující datové hodnoty, dále tvorbu třídy obsahující všechny potřebné funkce pro ovládání dronu, a v poslední řadě samotné tělo ovladače, využívající funkce z již vytvořené třídy k provedení různých letových úkonů.

```
#!/usr/bin/env python

import rospy
import time
import sys

from geometry_msgs.msg import Twist
from std_msgs.msg import String
from std_msgs.msg import Empty
from ardrone_autonomy.msg import Navdata
```

Obrázek 13 Import knihoven

Do ovladače importujeme:

- Rospy – umožňující jednoduchou komunikaci jazyku Python se zbytkem ROS
- Time – poskytuje nám možnost nastavovat akce podle času

- Sys – umožňuje manipulaci se samotným skriptem, například automatické ukončení

Následně importujeme určité funkce message souborů:

- Twist z geometry_msgs.msg – přímo interaguje s funkcemi Ardrone_Autonomy pro pohybové účely dronu
- String z std_msgs.msg – datový typ String
- Empty z std_msgs.msg – speciální datový typ použitelný pro posílání prázdného signálu
- Navdata z ardrone_autonomy.msg – obsahuje přídavné funkce specializované pro Parrot AR.Drone 2.0, jako jsou například hlášení stavu baterie, či v jakém stavu se dron právě nechází

```
class AutonomousFlight():

    def __init__(self):
        self.status = ""
        rospy.init_node('flighttest', anonymous=False)
        self.rate = rospy.Rate(10)
        self.pubTakeoff = rospy.Publisher("ardrone/takeoff",Empty, queue_size=10)
        self.pubLand = rospy.Publisher("ardrone/land",Empty, queue_size=10)
        self.pubCommand = rospy.Publisher('cmd_vel',Twist, queue_size=10)
        self.command = Twist()
        self.state_change_time = rospy.Time.now()
        rospy.on_shutdown(self.SendLand)

    def SendTakeOff(self):
        self.pubTakeoff.publish(Empty())
        self.rate.sleep()

    def SendLand(self):
        self.pubLand.publish(Empty())

    def SetCommand(self, linear_x, linear_y, linear_z, angular_x, angular_y, angular_z):
        self.command.linear.x = linear_x
        self.command.linear.y = linear_y
        self.command.linear.z = linear_z
        self.command.angular.x = angular_x
        self.command.angular.y = angular_y
        self.command.angular.z = angular_z
        self.pubCommand.publish(self.command)
        self.rate.sleep()
```

Obrázek 14 Třída obsahující potřebné funkce

Hlavní třída AutonomousFlight() obsahuje všechny potřebné funkce ke tvorbě letové funkcionality. Skládá se z konstuktoru a tří vlastních funkcí.

Konstruktor __init__(self):

- rospy.init_node – vnitřně pojmenuje spuštěný ROS node (v našem případě na flighttest), v případě, že chce spouštět více těchto ovládacích skriptu (což v následných případech budeme), je potřeba aby každý ROS node byl pojmenován unikátně. Případně, pokud nám příliš nejde o unikátní jména, můžeme atribut anonymous přepnout na True, čímž za každé jméno ROS node získáme náhodné číslo, aby automaticky byli unikátní.
- rospy.Rate(10) – jedná se o součást importované knihovny time, zajišťuje průchod cyklem hodnotou, udanou v Hz (Hertz)
- rospy.Publisher – způsob propojení funkce s určitým zadaným ROS topic (jmenná cesta pro komunikaci mezi jednotlivými ROS node), ardrone/takeoff a ardrone/land jsou ROS topic knihoven Ardrone_Autonomy starající se o vzestup a sestup, zatímco cmd_vel je ROS topic knihovny geometry_msgs starající se o pohybové vektory
- rospy.Time.now – získání současného času

Funkce SendTakeOff(self):

- pubTakeoff.publish(Empty()) – využívá komunikaci ROS topic pro nastartování motorů dronu
- rate.sleep – podle již určené hodnoty v Hz uspí funkce ROS, než započne další krok

Funkce SendLand(self):

- pubLand.publish(Empty()) – opět pomocí ROS topic dá dronu úkon přistát, tentokrát bez funkce uspání, protože chceme, aby přistání proběhlo celé

Funkce SetCommand(self, linear_x, linear_y, linear_z, angular_x, angular_y, angular_z):

- command.linear/angular – využívá Twist message pro přidělení směru vektoru a jejich hodnoty, která může být od -1 do 1
- pubCommand.publish(self.command) – pomocí ROS topic přepošle zvolené hodnoty dronu

```
if __name__ == '__main__':
        try:

                i = 0
                uav = AutonomousFlight()

                while not rospy.is_shutdown():

                        uav.SendTakeOff()
                        if i <= 30 :
                                uav.SetCommand(0,0,1,0,0,0)
                                i+=1

                        elif i<=60 :
                                uav.SetCommand(1,0,0,0,0,0)
                                i+=1

                        elif i<=90 :
                                uav.SetCommand(0,0,0,0,0,0)
                                uav.SendLand()
                                i+=1
                        elif i<=120 :
                                sys.exit()

        except rospy.ROSInterruptException:
                pass
```

Obrázek 15 Funkční tělo ovladače

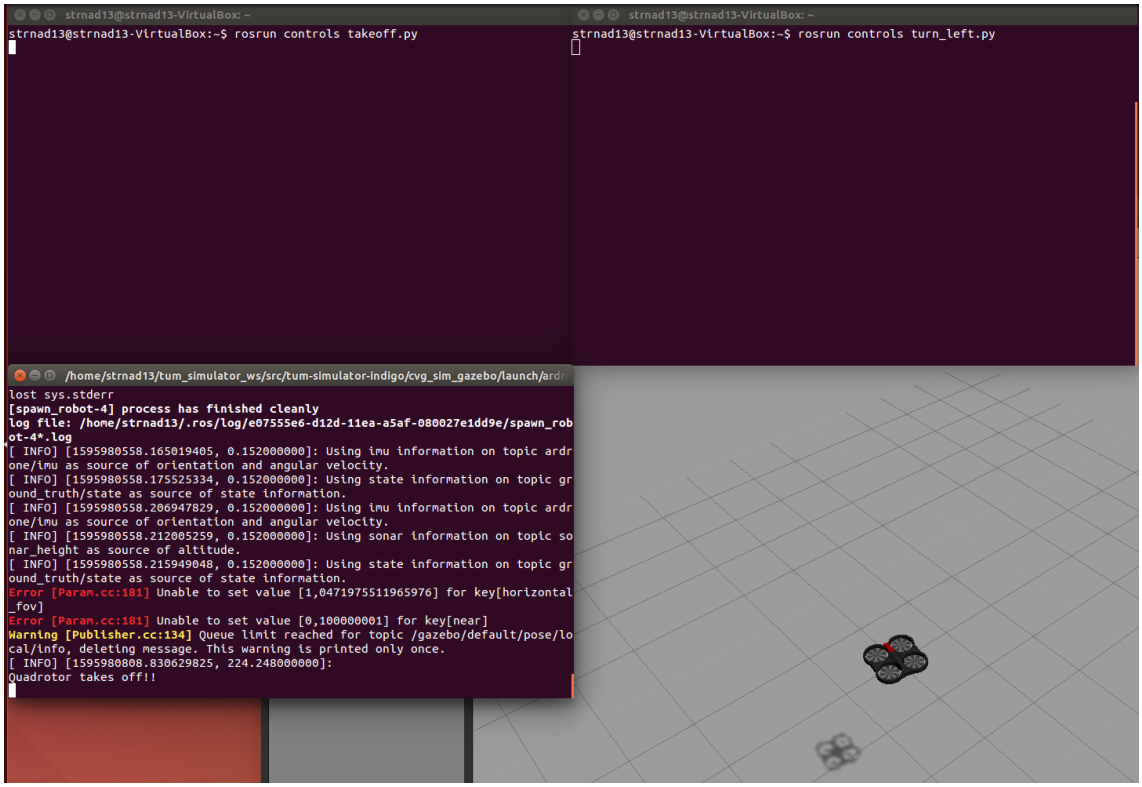
Ve funkčním těle ovladače využíváme námi definované funkce se zvolenými hodnotami, abychom vytvořili autonomní let drona podle našich představ. Vytvoříme blok Try a blok Except. V bloku Try, který bude probíhat, pokud bude program funkční, si definujeme proměnnou i, s hodnotou nula a vyvoláme třídu AutonomousFlight() na proměnnou uav (pojmenováno pod Unnamed Aerial Vehicle, bezpilotní letadlo). Následuje while not cyklus, kde kauzalita not bude v případě, že program bude v průběhu vypnut. V samotném cyklu vyvoláme funkci pro start motorů a vzlet. Proměnná i začne postupně růst na hodnotě v průběhu opakování cyklu, protože jsme zvolili rospy.Rate(10) v konstruktoru, každý if blok bude trvat 3 reálné vteřiny. První tři vteřiny dostane dron jedničkový příkaz letět po lineárním z vektoru, tudíž začne vzlétat po ose z přímo nahoru. Po třech vteřinách dostane jedničkový příkaz letět po lineární ose x, tudíž poletí dopředu. Po dalších třech vteřinách získá vynulovaný vektorový příkaz, tudíž se zastaví na místě, a pomocí příkazu SendLand() začne klesat dokud nepřistane, načež v poslední části využijeme interakci se samotným skriptem, který se terminuje, a tudíž se terminuje

veškerá funkce ROS a s ní i motory drona. Blok except proběhne v případě, že není možnost projít do bloku try, tudíž v ovladači existuje chyba.

Tyto ovládací skripty se spouští za pomoci ROS příkazu “rosrun” a potřebují Unix práva pro spuštění, která jim dáme za pomoci Unix příkazu “chmod”.

5.3.2 Ovladač manuálního letu

Tímto způsobem je možné dron naprogramovat na mnoho různých autonomních letů, kdy jen stačí ovládací skript spustit, a sledovat chování dronu. Můžeme ovšem také dron ovládat manuálně a dávat mu příkazy za běhu. Z tohoto důvodu bylo potřeba, aby zmíněné názvy ROS node byly unikátní, protože jsme schopni vytvořit mnoho různých ovládacích skriptů, jako byl náš autonomní, a řetězit je ve více terminálech. V těchto případech ovladače budou obdobné, co se týká obsahu, ovšem s limitovanými, specializovanými funkcemi. Ve složce controls jsem vytvořil mnoho takových ovladačů, od startu a visení, otáčení se, pohybu podle osy y bez otáček, až po regulerní sestup a přistání.



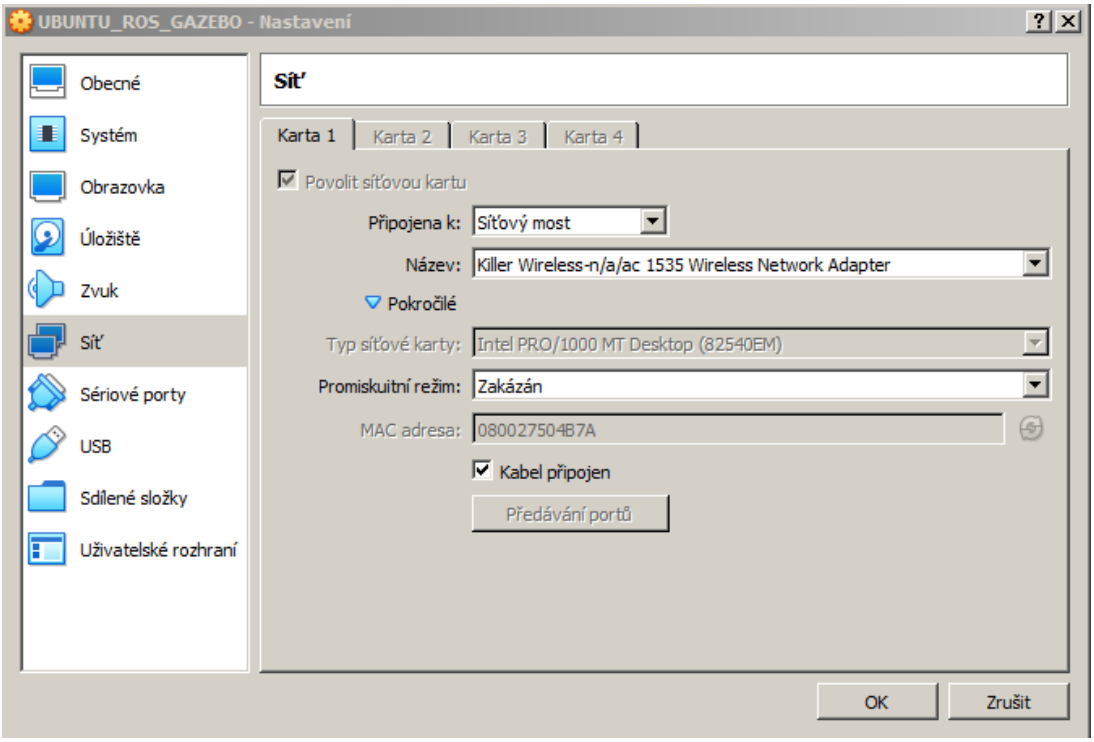
Obrázek 16 Řetězení ovládacích skriptů pro manuální ovládání dronu

Toto ovládání nemá zabudované automatické ukončení, tudíž pokud chceme skript a jeho operaci terminovat, terminujeme jej klávesovou zkratkou CTRL+C

5.3.3 Využití ovladačů na reálném dronu

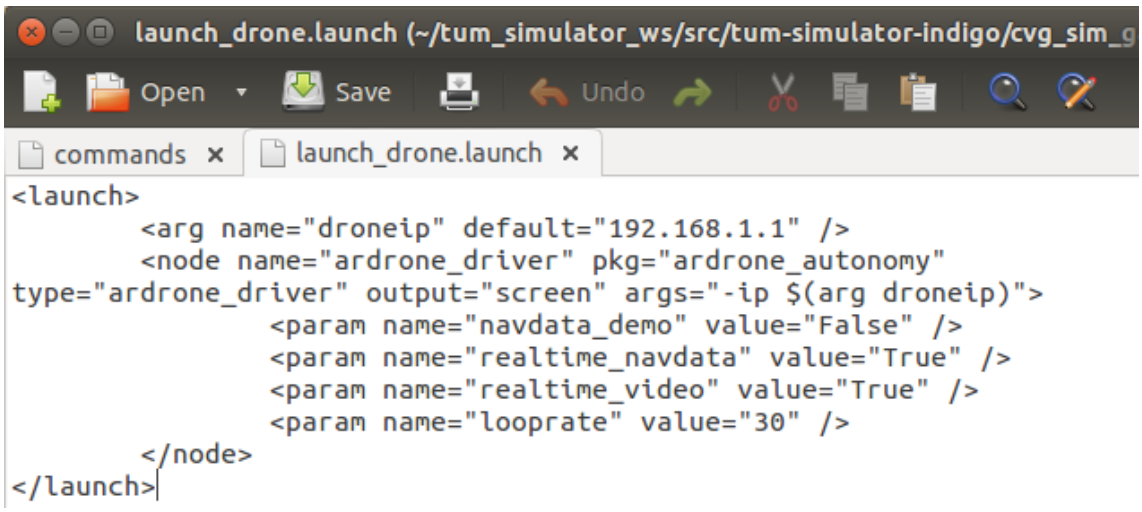
Veškeré tyto ovládací skripty jsou plně funkční s reálným Parrot AR.Drone 2.0. Pro jejich používání s fyzickým dronem je potřeba nastavit VirtualBox pro wifi konektivitu a vytvořit přídavný launch soubor pro konektivitu s dronem.

Virtualizované systémy ve VirtualBox využívají virtualizovaný síťový adaptér, který lze napojit na různé způsoby síťového provozu, jako například NAT (Network Address Translation, překlad síťových adres), nebo Cloud síť. Parrot AR.Drone 2.0 se k zařízením připojuje přes svůj vlastní WANET (Wireless ad hoc network) a tudíž v našem případě potřebujeme síťovou kartu virtualizovaného Ubuntu nastavit do módu síťového mostu, kdy virtualizované Ubuntu bude využívat přímo Wi-Fi adaptér hostitelského počítače. Poté jsme schopni se hostitelským počítačem připojit na Wi-Fi síť tvořenou samotným dronem, a virtualizované Ubuntu bude také schopné se k němu připojit a interagovat s ním.



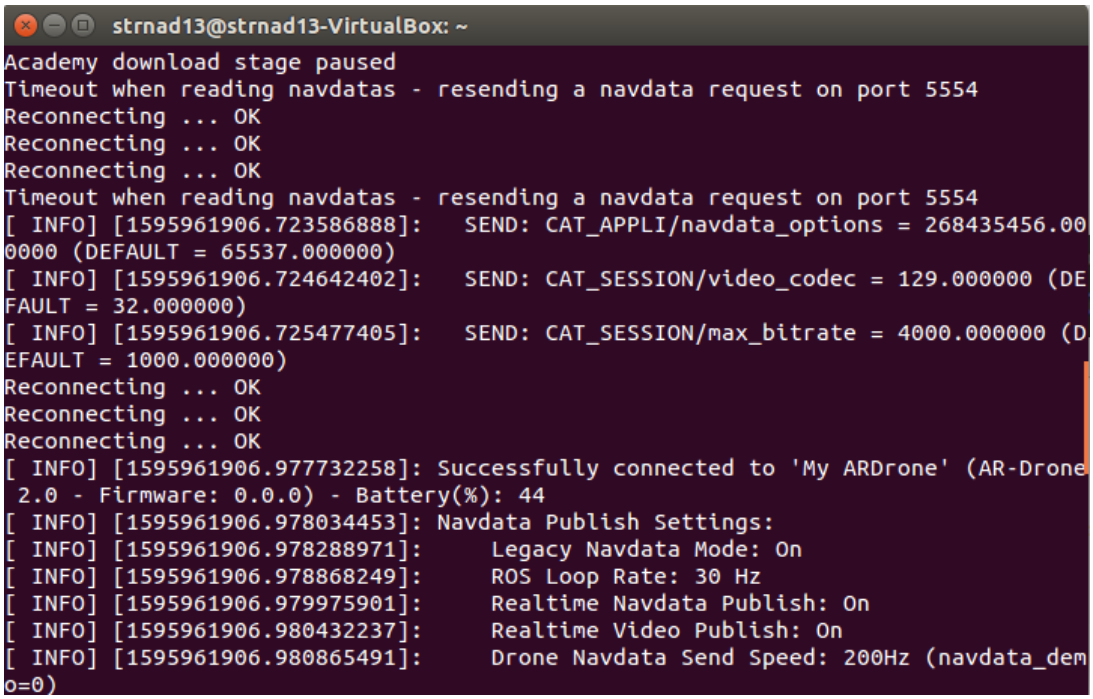
Obrázek 17 Nastavení síťové karty virtualizovaného Ubuntu

V druhé části musíme vytvořit dodatečný launch soubor, díky kterému se napojíme k dronu, a budeme mu schopni posílat příkazy ovladači, stejně jako v simulačním prostředí Gazebo. Pro jednoduchost při spouštění vytvoříme nový launch soubor ve stejné složce kde se vyskytují všechny ostatní (viz. 5.2 Práce s ROS).



Obrázek 18 Připojující launch soubor pro reálný dron

Standartní IP adresa Parrot AR.Drone 2.0 je 192.168.1.1, tudíž jí zapíšeme jako argument. Následuje pojmenování ROS node a deklarace, že budeme využívat ovladače Ardrone_Autonomy. Pomocí parametru output by se teoreticky dal získat i video záznam z drona. Parametr realtime_navdata nám bude vypisovat informace o dronu v příkazové řádce, opět díky funkcionalitě Ardrone_Autonomy. Poslední parametr looprate udává frekvenci v Hz, udávající jak často se budou aktualizovat příkazy z tohoto připojovacího ovladače do dronu. Je dobré ji nastavit větší, či stejnou, jako je v ovládacích skriptech, jinak hrozí nebezpečí ztráty dat.



Obrázek 19 Připojený dron s výpisem dat

6 Závěr

Cílem projektu bylo vytvoření funkčního virtualizovaného prostředí pro simulaci a vytvoření programů pro ovládání a autonomní let specifického drona Parrot AR.Drone 2.0. Vzhledem k tomu, že ovladače jsou lehce modifikovatelné a funkční jak v simulaci, tak na reálném dronu, tak z mého pohledu se mi podařilo splnit cíl práce, a jsem spokojen s výsledkem. Simulace robotiky je velice zajímavé téma, a tudíž jsem rád, že jsem se s tímto oborem informatiky mohl touto cestou seznámit.

Nejvíce problematickou částí celé práce bylo samotné vytvoření funkčního simulačního prostředí ve virtualizovaném Ubuntu, úkon, který jsem nakonec dělal natřikrát. Hlavními problémy byla volba správné verze operačního systému Ubuntu, která by byla schopná akomodovat veškeré potřebné moduly pro přeměnu regulerního Ubuntu na plnohodnotné simulační prostředí, zejména z důvodu, že Parrot AR.Drone 2.0 již není v prodeji, a tudíž i veškerá podpora ohledně simulace tohoto specifického drona odpadá. Z tohoto důvodu jsem musel experimentovat se staršími verzemi operačního systému a doufat v neoficiální repozitáře, které by mohli být aktualizovány na verzi, kterou jsem zrovna zkoušel, což se často ve výsledku změnilo v sisyfovskou práci.

Druhým největším problémem byla již zmíněná nemožnost 3D akcelerace, a tudíž samotná práce na programování ovladačů byla náročná kvůli celkovému výkonu systému, nemluvě o výsledném výkonu v simulačním prostředí. Jedná se bohužel o problém s virtualizací, a tudíž bych do budoucna plně doporučil pro tyto účely používat systém Ubuntu nativně.

Jedinným nedostatek práce je neschopnost dronu otočit se o určitý počet stupňů. Jedná se o nedostatek způsobu pohybu založeném na frekvenci, a i když přídavné sensory z knihoven Ardrone_Autonomy by byly schopné říct o kolik stupňů je dron právě otočen, nedá se tato skutečnost použít při samotném ovládání.

V rámci využívání těchto ovladačů na reálném dronu, bych rád zmínil, že by bylo jednodušší používat metodu manuálního ovládání (viz. 5.3.2 Ovladač manuálního letu), protože specifikace simulovaného Parrot AR.Drone 2.0 se zdáli lehce odlišné od reálného dronu, s největší pravděpodobností větší hmotností reálného dronu a vlivem gravitace. Autonomní ovladače stále plně fungují, bylo by ovšem potřeba zkoumat časové rozpětí mezi akcemi v různých podmínkách.

Druhý programovací jazyk JavaScript jsem po předchozí konzultaci s Dr. Smrčkou v práci nepoužil, na veškeré úkony byl jazyk Python více než dostačující, a také mi přišlo, že především knihovny ROS silně doporučovaly práci s jazykem Python, především kvůli knihovně rospy.

6.1 Možné rozšíření práce

Práce by se dala rozšířit o mnoho technologií, od jednodušších jako například GUI (generated user interface) a klávesnicové ovládání, po složité, jako například plná letová autonomie s využitím sensorů drona. Speciální verze Parrot AR.Drone 2.0 také disponovala GPS, další funkcí, která by mohla napomoci rozšíření, ovšem stejně jako regulerní Parrot AR.Drone 2.0 se již nevyrábí. Možnou adicí by mohla být kompatibilita s více druhy dronů.

Seznam použité literatury

[1] Hill, B. M., 2009. *The official Ubuntu book* [online]. 4. vyd. Boston: Pearson Education, 2009 [cit. 27.7.2020]. Dostupné z: https://books.google.cz/books?id=uzp2AyXipNwC&hl=cs&source=gbs_similarbooks

[2] Oracle Corporation. *VirtualBox Manual* [online]. [Cit. 27.7.2020]. Dostupné z: https://www.virtualbox.org/manual/ch01.html

[3] Python Software Foundation, 2020. *General Python FAQ* [online]. [Cit. 27.7.2020]. Dostupné z: https://docs.python.org/3/faq/general.html

[4] Open Source Robotics Foundation, 2014. *Gazebo History* [online]. [Cit. 27.7.2020]. Dostupné z: http://gazebosim.org/

[5] Amanda Dattalo, 2018. *ROS/Introduction* [online]. [Cit. 27.7.2020]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[6] Mani Monajjemi, 2012. *ardrone_autonomy* [online]. 2014 [Cit. 27.7.2020]. Dostupné z: https://ardrone-autonomy.readthedocs.io/en/latest/

[7] Juergen Sturm. *tum_simulator* [online]. Hongrong Huang. Technical University of Munich. [Cit. 27.7.2020]. Dostupné z: http://wiki.ros.org/tum_simulator

[8] Canonical Ltd., 2020. *The Ubuntu lifecycle and release cadence* [online]. [Cit. 27.7.2020]. Dostupné z: https://ubuntu.com/about/release-cycle

[9] Dave Coleman, 2013. *simulator_gazebo/SystemRequirements* [online]. [Cit 27.7.2020]. Dostupné z: http://wiki.ros.org/simulator_gazebo/SystemRequirements

[10] Open Source Robotics Foundation, 2014. *Install Gazebo using Ubuntu packages* [online]. [Cit 27.7.2020]. Dostupné z: http://gazebosim.org/tutorials?tut=install_ubuntu

[11] Shane Loretz, 2020. *rosdep* [online]. [Cit. 27.7.2020]. Dostupné z: http://wiki.ros.org/rosdep

[12] The Robotics Back-End, 2020. *What is a ROS Node?* [online]. [Cit 27.7.2020]. Dostupné z: https://roboticsbackend.com/what-is-a-ros-node/

[13] Dave Coleman, 2013. *Creating a ROS Package* [online]. [Cit. 27.7.2020]. Dostupné z: http://wiki.ros.org/catkin/Tutorials/CreatingPackage

Přílohy

Společně s bakalářskou prací odevzdávám vyexportovaný virtuální disk ve formátu VDI obsahující operační systém Ubuntu, připravený na simulaci a práci s dronem, podle popisu v práci.