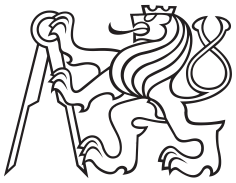
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Autonomní řízení dronu Ryze Tello

David Pařil

Vedoucí: RNDr. Petr Štěpán, Ph.D.
Květen 2021

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce RNDr. Petru Štěpánovi, Ph.D. za náměty, konzultace a všechen čas, který mi věnoval. Dále bych chtěl poděkovat mé rodině za podporu při studiu na ČVUT i v osobním životě.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2021

Abstrakt

Drony jsou v dnešní době stále používanějším prvkem v aplikacích jako přeprava zboží, záchranné mise, mapování prostor, zemědělství. Přetrvávajícím problémem dronů je jejich nízký výpočetní výkon často omezený rozměry, váhou či zdrojem napájení dronu. Tato práce se zabývá autonomním řízením dronu Ryze Tello systémem ROS přes rozhraní Wi-fi. Popisujeme vliv a kompenzaci zpoždění komunikace řídícího počítače s dronem, detekci ArUco značek z obrazu a jejich využití k lokalizaci a řízení dronu.

Klíčová slova: dron, řízení, lokalizace, ROS, Tello, ArUco

Vedoucí: RNDr. Petr Štěpán, Ph.D.
ČVUT v Praze,
Fakulta elektrotechnická,
Katedra kybernetiky,
Karlovo náměstí 13,
121 35 Praha 2

Abstract

Nowadays drones are increasingly used in applications such as parcel delivery, rescue missions, space mapping, agriculture. A persistent problem of drones is their low computing power, often limited by their size, weight or deficient power supply. This paper discusses autonomous drone control of the Ryze Tello drone using the ROS system via Wi-fi transmission. We describe the effects and compensation of the communication delay between the controlling computer and the drone, image-based detection of ArUco markers and their use for localization and control of the drone.

Keywords: drone, control, localization, ROS, Tello, ArUco

Title translation: Autonomous control of drone Ryze Tello

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Autonomní drony jsou ve světě často diskutovaným tématem pro jejich rostoucí využití ve vojenství, komerčním prostoru i domácnostech. Popularita dronů tkví v jejich široké škále využití, přístupu do oblastí člověku nedosažitelných či nebezpečných a jejich stále se zvyšující dostupnosti.

Mezi nejčastější obory použití dronů patří letecké snímkování, doručování zásilek, průzkumné a záchranné mise, zemědělství, bezpečnostní inspekce a mnohé jiné. Byť se jedná o relativně novou technologii, autonomní drony jsou pro svou všestrannost cílem nemála výzkumných týmů hledajících nové způsoby uplatnění dronů, jako například autonomní průzkumné mise či hašení požárů [1]. Nezávisle na aplikaci, spolehlivé řízení a lokalizace autonomního dronu jsou zásadní vlastností.

Mezi nejčastěji používané lokalizační systémy patří GPS (Global Positioning System), který poskytuje dostatečně přesné informace o poloze pro většinu úloh, ovšem je velice citlivý na kvalitu satelitního signálu, a proto není vhodný pro aplikace vyžadující vysokou spolehlivost systému.

Mnohé jiné lokalizační systémy s sebou často přináší nevýhody jako je vysoká pořizovací cena, hmotnost či rozměry systému. Pro své nepopíratelné výhody oproti ostatním lokalizačním systémům se lokalizace s využitím kamery stala velice atraktivní volbou mnoha výzkumných projektů.

1.1 Představení problematiky

V této práci se zabýváme řízením bezpilotního letadla (UAV) Ryze Tello, který disponuje nadstandardní senzorovou výbavou, nízkou pořizovací cenou a širokou škálou možností řízení vzdáleným počítačem. K jeho lokalizaci využíváme detekci příznaků v obraze z palubní kamery.

Bezpilotní letadlo Ryze Tello je drobná kvadrokoptéra (tj. čtyřrotorový vrtulník). V dalších částech textu budeme bezpilotní letadlo Ryze Tello označovat jako "dron", "Tello"či pouze "zařízení".

Řízení dronu Tello je možné pouze přes externí počítač s pomocí bezdrátové komunikace. Ta s sebou ovšem přináší nezanedbatelná zpoždění a nestabilitu komunikace. Cílem této práce je ověřit, zda-li je možné takto řídit dron Tello z externího počítače s využitím aplikačního rámce ROS.

Dron Tello nedisponuje žádným globálním lokalizačním systémem. Z tohoto důvodu v práci používáme lokalizaci pomocí značek pevně rozmístěných ve známém prostředí.

1.2 Nástin práce

Tato práce představuje námi navržený systém pro řízení dronu Ryze Tello na základě detekce polohy z obrazu palubní kamery dronu. Nejprve v kapitola 2 představí dron Ryze Tello. Následně kapitola 3 seznámí s aplikačním rámcem ROS, v němž je implementován řídící program. Kapitola 4 představí ROS balíček tello_driver využívaný ke komunikaci dronu s řídícím počítačem.

Kapitola 5 diskutuje výpočet polohy dronu z obrazu palubní kamery a jeho implementaci s pomocí ROS balíčku aruco_detect. V kapitole 6 implementujeme vybranou metodu řízení dronu. V kapitole 7 se zabýváme kompenzací vlivu zpoždění přenosu obrazu z dronu. Dosažené výsledky námi implementovaného systému jsou pak uvedeny v kapitole 8.

Kapitola 2

Dron Ryze Tello

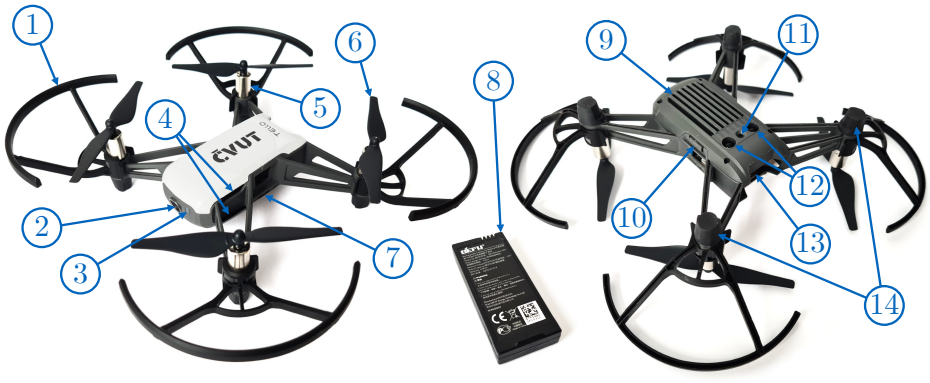
2.1 Základní informace o dronu

Dron Ryze Tello (dále jen Tello) je miniaturní kvadroptéra vyvinutá čínskou společností Ryze ve spolupráci s Intel a DJI. Mezi její klíčové vlastnosti patří kompaktní rozměry, nízká pořizovací cena, HD kamera, python SDK a v neposlední řadě její pokročilý řídící systém. Dron je napájen snadno vyměnitelnými lithium-iontovými (zkráceně Li-Ion) akumulátory o kapacitě 4.18Wh, které umožňují přibližně 10 minut letu. Dron je zobrazen na obrázku 2.1.

Obrázek 2.1: Fotografie dronu Ryze Tello.

Společnost Ryze vydala celkem tři mírně se lišící verze tohoto dronu. Mimo základní Tello vydala i Tello Iron Man a Tello EDU. Mezitím co u Tello Iron Man se jedná převážně o kosmetické změny, Tello EDU přináší řadu nových funkcí jako je detekce příznaků v obraze v podobě kódů nazývaných "Mission Pad"(volně přeloženo "úkolová podložka"), podporu létání v roji a vylepšené SDK 2.0. V této práci se zabýváme řízením základní verze.

2.2 Konstrukce a komponenty



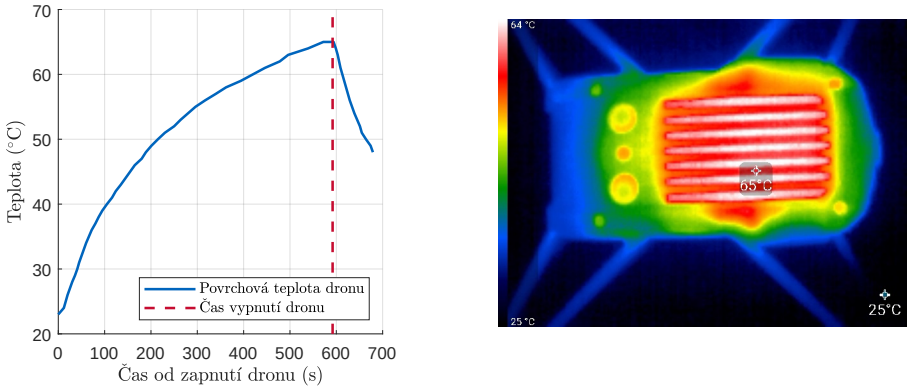
Obrázek 2.2: Schéma komponent dronu Ryze Tello.

Pohyb kvadroptéry zajišťují čtyři kartáčové DC motory (5) v kombinaci s odnímatelnými třípalcovými vrtulemi (6). Vrtule i motory dělíme na CW (clockwise neboli ve směru hodinových ručiček) a CCW (counterclockwise neboli proti směru hodinových ručiček). Jejich umístění a funkce je diskutována v sekci 2.3. Součástí balení jsou také čtyři nasazovací ochranné rámy (1), které chrání vrtule v případě boční srážky dronu s překážkou. Jejich použití při testování letu kvadrokoptéry ve vnitřních prostorech se dá označit za téměř nezbytné.

Na přední části zařízení se nachází hlavní kamera (2) disponující rozlišením 2592x1936 s možností přenášení videa 720p při třiceti snímcích za vteřinu. V blízkosti kamery se nachází stavový indikátor (3) ve formě RGB LED. Stavový indikátor zobrazuje pomocí odlišných barev a efektů blikání různé stavy, ve kterých se Tello může nacházet. Jejich kompletní seznam je dostupný v oficiálním manuálu dostupném na webových stránkách výrobce[1].

Pod předními vrtulemi se na obou stranách dronu nachází dva páry větracích otvorů (4). Ty během letu odvádí část vzduchu od vrtulí a poskytují tak aktivní chlazení řídící jednotce. Ta je chlazena také pasivním chladičem s větrací mřížkou na spodní části dronu (9), ovšem tento chladič není dostatečně účinný na chlazení zařízení ve vnitřních prostorách s pokojovou teplotou. Při dlouhodobé práci se zařízením v neletovém režimu je nutné poskytnout externí aktivní chlazení, jinak dojde k přehřátí zařízení a následnému vypnutí. Vývoj povrchové teploty dronu bez aktivního chlazení zaznamenaném termokamerou je zobrazen na obrázku 2.3.

[1] https://dl-cdn.ryzerobotics.com/downloads/Tello/20180404/Tello_User_Manual_V1.2_EN.pdf



Obrázek 2.3: Vlevo: Průběh maximální povrchové teploty od zapnutí dronu při přenosu videa. Vpravo: Měření termokamery v době vypnutí dronu (9 minut a 52 vteřin od zapnutí při okolní teplotě 25°C)

Na levé straně zařízení se nachází micro USB port (7) určený k nabíjení akumulátoru (8) vloženého do přihrádky v zadní části dronu (13). Naproti nabíjecímu portu je tlačítko na zapnutí či vypnutí zařízení (10).

Spodní strana dronu ukrývá mimo předem zmiňované větrací mřížky i sadu senzorů vizuálně polohovacího systému (dále VPS). Ten se skládá z malé kamery (11) a 3D infračerveného modulu (12). VPS se zapíná automaticky se zapnutím dronu a umožňuje Tellu za letu lépe držet požadovanou pozici. Tento systém je ovšem velice citlivý na podmínky v okolním prostředí. V naší práci využíváme výstup těchto senzorů při řízení letu, a proto se jim budeme blíže věnovat v kapitole 7.

Poslední diskutovanou částí dronu jsou jeho antény (14) skryté uvnitř úchytů dvou zadních motorů. Tello si takto vytváří vlastní 2.4 GHz 802.11n Wi-Fi, pomocí které komunikuje s připojeným zařízením - v našem případě řídícím počítačem.

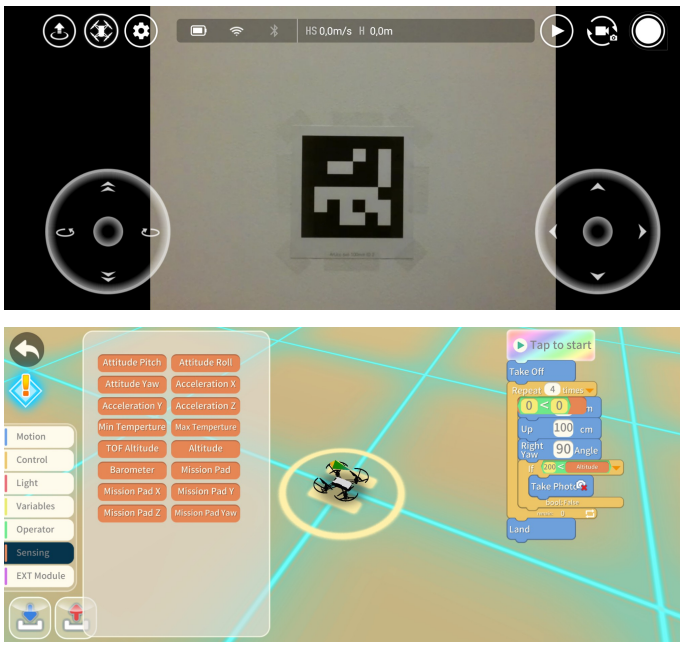
2.3 Ovládání a letové vlastnosti

Společnost Ryze uvedla kvadrokoptéru Tello na trh s představou malého dronu, který uživateli poskytne intuitivní manuální ovládání, stabilní let, kvalitní přenos a záznam z palubní kamery, ale i sadu nástrojů určených k programování dronu, nadstandardní senzorovou výbavu a řadu výukových

materiálů pro začátečníky i pokročilé. V této kapitole budeme hovořit o možnostech ovládání dronu Ryze Tello a jeho letových vlastnostech.

Právě díky předem zmíněné variabilitě použití dronu Tello existuje několik mobilních aplikací na jeho ovládání či programování. Všechny aplikace zmíněné v této kapitole jsou dostupné z oficiálních webových stránek výrobce[2]. První a pravděpodobně nejdůležitější z nich je aplikace "TELLO APP", která je především určena pro manuální ovládání dronu, ale zároveň poskytuje možnost aktualizace firmwaru Tella a mnohé další užitečné funkce.

Pro základní verzi dronu Tello jsou výrobcem doporučeny také dvě aplikace umožňující jednoduché programování dronu. Obě aplikace ("DroneBlocks"a "TELLO EDU APP") využívají vizuálního programovacího jazyku, který připomíná taktéž Tellem podporovaný programovací jazyk Scratch[3]. Poslední výrobcem doporučenou možností pro ovládání dronu je takzvané "Tello SDK"[2], které uživateli poskytuje celou řadou textových příkazů ke komunikaci s dronem pomocí protokolu UDP (User Datagram Protocol).

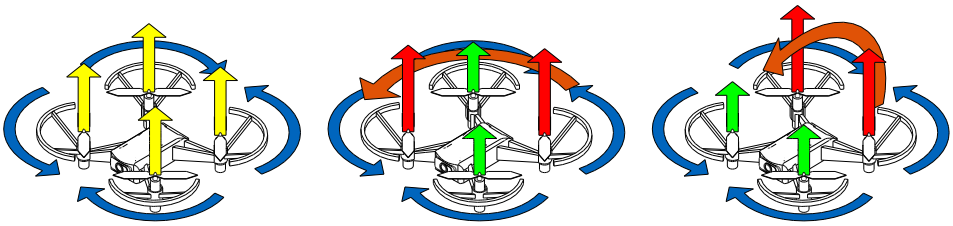


Obrázek 2.4: Snímky obrazovky ze zmíněných aplikací - nahoře TELLO APP, níže TELLO EDU APP. Pořízeno v systému Android 11.

Dron Tello je malá kvadrokoptéra, tedy helikoptéra se čtyřmi rotory. Princip letu kvadroptéry spočívá v přesunu centra tahu a změně točivého momentu vytvořeného čtyřmi vrtulemi. Využitím dvou vzájemně protichůdných, úhlopříčně umístěných párů vrtulí je kvadrokoptéra schopna dosáhnout nulového

[2]https://www.ryzerobotics.com/tello/downloads
[3]https://scratch.mit.edu/

točivého momentu. Mimoto lze ovládat otáčení dronu nesymetrickým vyvážením tahu mezi předem zmíněnými páry při zachování celkového tahu generovaného oběma páry. Podobným způsobem, pouze s páry vzájemně sousedících vrtulí, lze ovládat náklon dronu ve zbylých dvou osách. Pohyb nahoru a dolu kvadrokoptéra vykonává změnou celkového tahu všech čtyř vrtulí. Pohyb dopředu, dozadu a do stran pak zajistí již předem zmiňovaným náklonem, který část generovaného tahu odkloní a uvede tak kvadrokoptéru do pohybu.



Obrázek 2.5: Vizualizace předem popisovaných konfigurací, tedy kvadrokoptéra bez pohybu, otáčení a náklon. Na obrázku je taktéž modrými šipkami naznačen směr rotace dané vrtule.

Tello, jakožto bezpilotní letadlo, má na palubě letovou řídící jednotku, která zajišťuje stabilitu dronu ve dvou osách náklonu. K tomu dron využívá vestavěnou IMU (inertial measurement unit, česky inerciální měřící jednotka), což je zařízení, které využívá akcelerometru k měření lineárního zrychlení, gyroskopu k měření úhlové rychlosti a magnetometr. Spojením těchto dat pak získává orientaci v prostoru. Je potřeba dbát na to, aby IMU bylo vždy správně zkalibrované, jinak dron nedokáže správně vyhodnotit svůj stav, což vede k samovolnému pohybu. Jedním ze způsobů, jak provést kalibraci, je využití aplikace TELLO APP, která v nastavení tuto možnost nabízí. Takto lze samovolný pohyb výrazně omezit, avšak špatná kalibrace není jediným jeho zdrojem. Velký vliv na stabilitu kvadrokoptéry má například okolní pohyb vzduchu. Tello je určený i pro vnitřní použití, kde by samovolný pohyb mohl vést k častým srážkám dronu s okolím, proto je vybaven senzory VPS, které tyto vlivy částečně kompenzují. Aby tento systém správně fungoval, je zapotřebí, aby byly splněny následující podmínky:

- Dron se nachází od 0.3 metru až 6 metrů nad zemí
- Povrch pod dronem je neprůhledný, nereflektivní povrch s dobře viditelným, neopakujícím se, kontrastním vzorem
- Okolní osvětlení je dostatečné a konstantně intenzivní

V našem testovacím prostředí nebylo možné zajistit absolutní spolehlivost tohoto systému. Bližší informace o VPS, jeho využití a zpracování dat z něho získaných jsou uvedeny v kapitole 7.

Kapitola 3

ROS - Robot Operating System

3.1 Představení aplikačního rámce ROS

Řízení pokročilých robotických systémů, které zpravidla obsahují řadu akčních členů, senzorů, případně vícero řídících jednotek, vyžaduje kvalitní aplikační rámec (anglicky "framework"), jenž slučuje jednotlivé komponenty systému. Pro mnoho výzkumných týmů se jím stal právě ROS (volně přeloženo "Robotický Operační Systém).

ROS vznikl v roce 2007, kdy Willow Garage, jeho tvůrce, věnoval výrazné úsilí implementaci rozšířených konceptů předchozích badatelů v tomto oboru. S rostoucí komunitou se vývoj tohoto open-source (česky "Otevřený software") projektu výrazně urychluje. V době psaní této práce je ROS používaný desítkami tisíc uživatelů po celém světě. [2]

Zdůrazněme, že se nejedná o operační systém (jak by napovídal jeho název), ale o flexibilní aplikační rámec poskytující sadu konvencí, nástrojů a knihoven. Modularita je jednou z klíčových vlastností ROSu - mimo jiné přináší zpravidla vyšší stabilitu systému, snadnější implementaci multirobotických systémů a umožňuje jednoduché propojení částí psaných v různých programovacích jazycích (nativní podpora pro Python, C++, Lisp, případně experimentální implementace pro jazyky Java a Lua).

ROS aktuálně oficiálně podporuje pouze platformy založené na Unixu,

z nichž je doporučován open-source operační systém Ubuntu a operační systém Mac OS X. Jádro ROSu spolu s příslušnými nástroji a knihovnami je pravidelně vydáváno v distribucích. Tyto distribuce jsou zpravidla označovány jejich názvem. Mezi podporované verze v době psaní této práce patří:

- ROS Melodic Morenia; (23. května 2018 - květen 2023)
- ROS Noetic Ninjemys (23. května 2020 - květen 2025)
- ROS 2 Foxy Fitzroy (5. června 2020 - květen 2023)

Foxy Fitzroy je nejnovější distribucí ROSu 2 - ten přináší podporu pro operační systémy Microsoft Windows a řadu nových funkcí a vylepšení oproti předchozí verzi. ROS 1 a ROS 2 nejsou vzájemně kompatibilní a proto jsou prozatímně udržovány obě verze. Více informací na toto téma lze nalézt na stránkách ROS 2 design[1]. Jelikož v této práci využíváme knihovny vyvinuté pro původní ROS, využíváme distribuci Noetic Ninjemys.

Tato kapitola si dává za úkol představit základní funkční principy aplikačního rámce ROS Noetic Ninjemys (dále jen ROS), představit řadu využívaných nástrojů, které ROS poskytuje, a informovat čtenáře o způsobu spuštění námi vytvořeného programu.

3.2 Představení datového systému

Předem zmiňovaná modularita ROSu vede k organizaci zdrojů jednotlivých programů do skupin, které se nazývají balíčky (anglicky packages). Tyto balíčky mohou obsahovat skripty, knihovny, soubory typu launch a ostatní potřebné soubory pro běh programu. Povinnou součástí balíčku je manifest - ten v sobě uchovává informace jako kontaktní informace autora, verze, licence a závislosti balíčku.

ROS využívá proměnné prostředí, což umožňuje rychleji střídat jak mezi odlišnými verzemi ROSu, tak i mezi různými skupinami balíčků. Je proto důležité nezapomenout nahrát příslušné soubory s nastavením prostředí. Ve většině případů se jedná o dva příkazy načítající balíčky ROS a balíčky uvnitř uživatelsky vytvořeného pracovního prostředí:

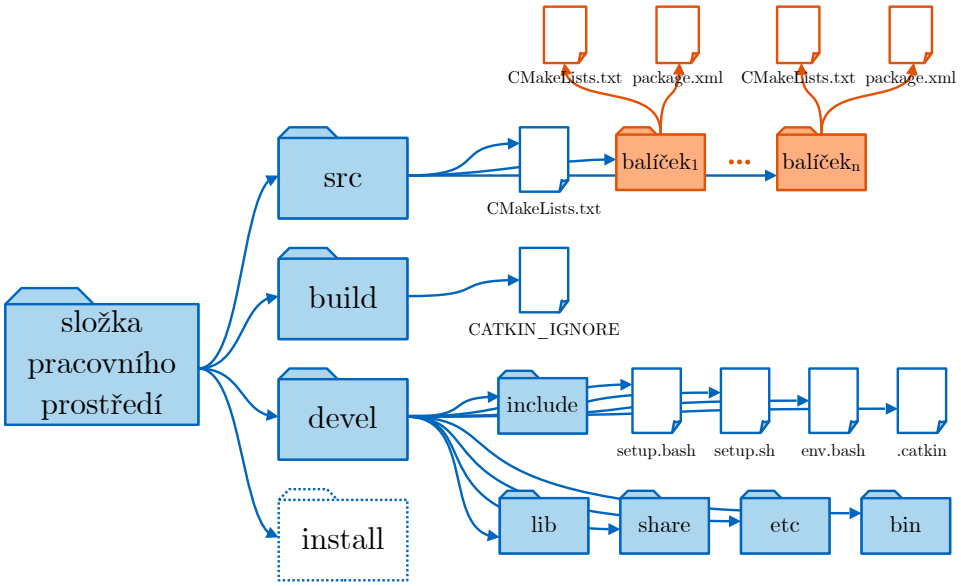
[1] http://design.ros2.org/

```
source /opt/ros/<nazev_distribuce>/setup.bash
source <cesta_k_pracovnimu_prostredi>/devel/setup.bash
```

Prvním popisovaným nástrojem použitým v této práci je nástroj catkin. Tento nástroj umožňuje uživateli snadno vytvářet a spravovat pracovní prostředí a sestavovat (anglicky build) balíčky v nich obsažené. Přejeme-li si sestavit balíčky a nechat si automaticky vytvořit příslušné prostředí, uděláme tak zavoláním příkazu

```
catkin_make
```

v adresáři obsahujícím složku s názvem "src", ve které se nachází používané balíčky, tedy v kořenové složce daného pracovního prostředí. Nově se v kořenové složce našeho pracovního prostředí vytvoří soubor "CMakeLists.txt"a složky "build"a "devel". Ve složce "devel"je pak vytvořen výše zmiňovaný soubor "setup.bash".



Obrázek 3.1: Kořenový systém pracovního prostředí catkin

Součástí ROSu je také řada nástrojů pro práci s balíčky. Jejich znalost není nutná, avšak výrazně usnadňuje orientaci a práci v systému ROS. V tabulce 3.1 uvádíme pouze výběr z dostupných nástrojů. Více informací o těchto nástrojích je k dispozici na webových stránkách ROS Wiki[2].

[2]http://wiki.ros.org/Tools

Nástroj	Primární funkce
rospack	Poskytuje informace o balíčku
roscd	Skok do adresáře obsahující balíček
rosls	Vypisuje položky v adresáři obsahující balíček
roscp	Umožňuje jednoduše kopírovat soubory z balíčku
rosrun	Vyhledá a spustí spustitelný soubor z balíčku
rosdep	Vyhledá a nainstaluje závislosti balíčku

Tabulka 3.1: Stručný přehled základních nástrojů pro práci s balíčky

3.3 Uzly a jejich vzájemná komunikace

Modularita ROSu nespočívá jen v propracovaném datovém systému a balíčkách, kterými jsme se zabývali v předchozí sekci. Součástí každého balíčku může být hned několik uzlů (anglicky node), což jsou spustitelné soubory, které mohou komunikovat s ostatními uzly v rámci systému ROS. Tato komunikace je klíčovou vlastností ROSu, a právě proto se jí budeme v této části textu zabývat.

V následující části uvádíme několik nových pojmů (tabulka 3.2).

Termín	Význam
roscore	spouští master, server parametrů a rosout
master (mistr)	umožňuje vzájemné vyhledání uzlů
server parametrů	uzly zde ukládají a načítají parametry
rosout	správa konzolového výstupu a jeho záznam
node (uzel)	spustitelný soubor komunikující s ostatními uzly
topic (téma)	spojující prvek při komunikaci mezi dvěma uzly
message (zpráva)	datové struktury, jimiž uzly komunikují
publisher (vydavatel)	uzel odesílající zprávy daného tématu (topicu)
subscriber (příjemce)	uzel přijímající zprávy daného tématu (topicu)
service (služba)	umožňuje komunikaci uzlů typu žádost-odpověď
server (server)	uzel odpovídající na žádosti dané služby
client (klient)	uzel zasílající žádosti dané službě

Tabulka 3.2: Přehled diskutovaných termínů s případnými překlady do jazyka českého uvedenými v závorce

Před používáním ROS programů je nutné zavolat příkaz

```
roscore
```

, kterým spustíme roscore. Jedná se o kolekci uzlů a programů, jejímž spuštěním se zapne master, server parametrů a protokolový uzel rosout. Server parametrů umožňuje uzlům ukládat a vyvolávat parametry. Protokolový uzel

rosout spravuje konzolový výstup ROSu a jeho záznam. Master umožňuje úzlům vzájemně se vyhledat a zároveň udržuje přehled všech aktivních vydavatelů, příjemců a služeb. Komunikace mezi uzly probíhá P2P (peer-to-peer, neboli klient-klient).

Zmíněné termíny vydavatel a příjemce popisují vztah uzlu k danému tématu (topic). Společným tématem se uzly vzájemně propojí a mohou si vyměňovat zprávy. Příjemce je uzel, který zprávy konkrétního tématu přijímá, vydavatel je pak uzel, který tyto zprávy naopak odesílá. Každé téma může být využíváno hned několika vydavateli a příjemci, zároveň jeden uzel může být vícenásobným příjemcem a vydavatelem. Hlavním znakem každého tématu je typ zprávy, kterou přenáší.

Zpráva je datová struktura zahrnující datový typ přenášených dat. K jejich definici se využívá zpravidla krátkých textových souborů s příponou "msg". Nejjednodušším příkladem zprávy je prázdná zpráva typu std_msgs/Empty.msg, která nepřenáší žádná data. Prázdné zprávy využívá například povel nouzového zastavení motorů kvadrokoptéry. Dalším jednoduchým příkladem zprávy je například zpráva definující bod v tří-dimenzionálním prostoru.

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

Zprávy mohou být definovány i s využitím již předem definovaných zpráv. Příkladem je zpráva, která přidá k předem definovanému bodu časovou informaci s využitím std_msgs/Header.msg.

geometry_msgs/PointStamped.msg

```
std_msgs/Header header
geometry_msgs/Point point
```

Součástí ROSu je značné množství již implementovaných základních typů zpráv, mezi něž patří i výše zmíněné příklady. Tento fakt opět přispívá k modularitě a přehlednosti programů vytvořených v systému ROS.

Témata jsou určena převážně pro jednosměrnou komunikaci mezi uzly, pro více směrnou komunikaci typu žádost-odpověď nabízí ROS takzvané služby. Služba je definována zprávou pro žádost a zprávou pro odpověď. Lze pozorovat určitou podobnost s tématem, které je ovšem definované pouze jednou zprávou. Uzel nabízející službu (taktéž nazývaný server) čeká na příjem zprávy s žádostí od klientského uzlu, na kterou následně, pokud možno neprodleně, odpovídá. Služba je definována textovým souborem s příponou "srv", který v sobě

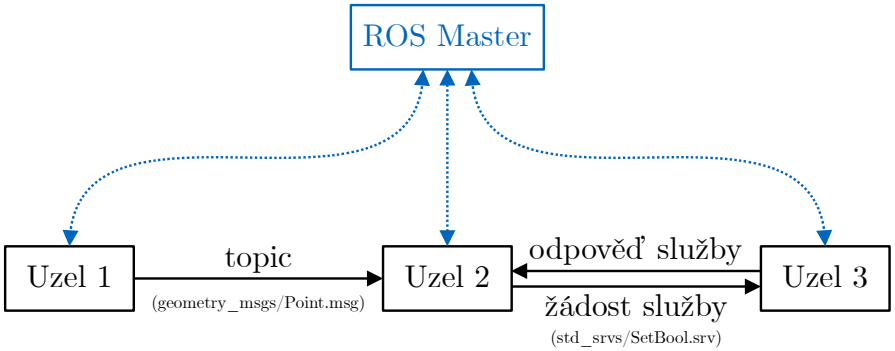
obsahuje typy žádosti a odpovědi (v tomto pořadí) vzájemně oddělené pomocí řetězce '—'. Jednoduchým příkladem může být služba, která žádá o nastavení hodnoty typu boolean a očekává odpověď obsahující potvrzení o úspěšném vykonání příkazu (opět typu boolean) a textový řetězec popisující případnou chybu.

std_srvs/SetBool.srv

```
bool data
---
bool success
string message
```

Podobně jako téma, může mít i server dané služby více klientů, ovšem žádná služba by neměla mít více jak jeden server.

Pro znázornění problematiky použijeme příklad líného farmáře. Mějme líného farmáře vlastnícího právě jednu ovci. Ovce se přes den pase na louce, v noci musí být zamknuta v ovčíně, aby nebyla usmrcena okolní dravou zvěří. Jelikož farmář považuje zavírání brány za ovcí za nesmírně namáhavou činnost, rozhodne se ovci vybavit polohovacím systémem a implementovat jednoduchý algoritmus s využitím aplikačního rámce ROS. Farmářem vytvořený program využívá tři uzly, z nichž první zpracovává informace z polohovacího systému a posílá informaci o poloze ovce druhému uzlu, který tato data porovnává s hranicí ovčína. Pokud druhý uzel vyhodnotí, že se ovce dostavila do ovčína, odesílá žádost službě, jejímž serverem je poslední z uzlů. Ten spravuje aktuátor manipulující s bránou ovčína. Jako odpověď klientovi (tedy druhému uzlu) zasílá informaci o úspěšném zavření brány či jejím případném zaseknutí. Architektura tohoto systému je zjednodušeně zobrazena na obrázku 3.2.



Obrázek 3.2: Zjednodušené zobrazení architektury popisovaného programu

Posledním tématem této sekce jsou nástroje související s uzly a komunikací mezi nimi. Stejně jako v předchozím případě uvádíme pouze výběr z dostupných nástrojů založený na osobní preferenci autora. Znalost těchto nástrojů

je velice užitečná, nikoliv však nezbytná pro pochopení této práce. Proto v tabulce 3.3 vypisuji pouze základní informace, podrobněji na webových stránkách[3].

Nástroj	Primární funkce
rosparam	přístup a úprava údajů na serveru parametrů
rosnode	informace o uzlech, jejich ukončení, test spojení
rostopic	informace o zprávách uvnitř topicu, jejich obsah
rosmsg	práce se soubory typu msg a výpis jejich struktury
rosservice	informace o službách, zasílání žádostí službě
rossrv	práce se soubory typu srv a výpis jejich struktury

Tabulka 3.3: Stručný přehled základních nástrojů pro práci s programy

3.4 Nástroje rosbag, rviz a rqt_graph

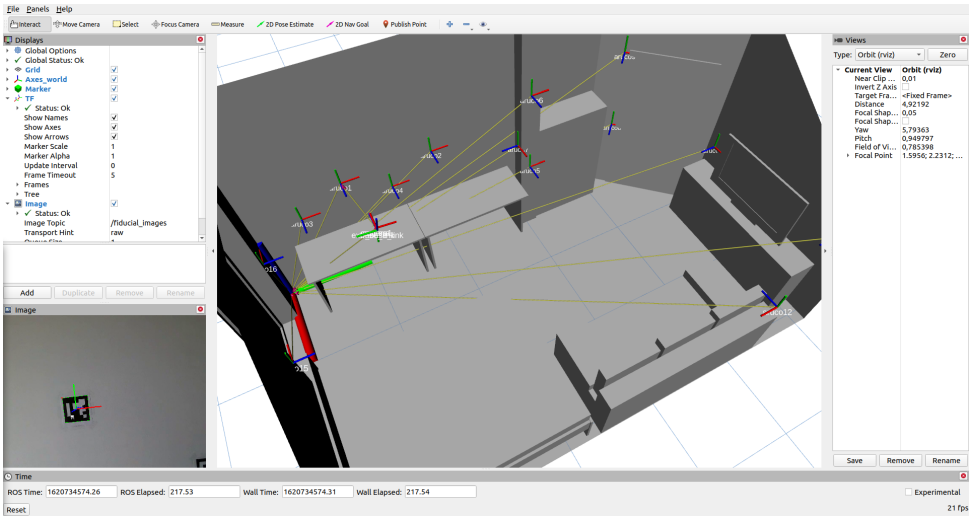
Následující část textu se věnuje třem nástrojům aplikačního rámce ROS, které si pro svoji užitečnost zasluhují větší pozornost. Jelikož testování letu bezpilotního letounu je komplikované, časově náročné a nebezpečné, je vhodné omezit jeho četnost. K tomu můžeme využít první z diskutovaných nástrojů.

Rosbag je nástroj umožňující záznam všech nebo vybraných zpráv zadaných témat a jeho následné přehrání. Takto můžeme při vývoji programu opakovaně analyzovat jeho chování s využitím záznamu naměřených dat z jednoho testovacího letu. Záznam se ukládá do souboru typu "bag". Existuje mnoho dalších nástrojů, které umí pracovat s těmito soubory. Velká část vizualizací dat v této práci byla vytvořena právě ze záznamů měření uložených v souborech typu "bag"s využitím nástroje MATLAB od společnosti MathWorks[4] a programovacího jazyka python.

K vizualizaci měřených dat v reálném čase slouží nástroj rviz. Ten umožňuje uživateli jednoduše vytvořit vizualizační prostředí a číst data přímo z daných témat. Na obrázku 3.3 je znázorněna námi vytvořená konfigurace nástroje rviz, ve které si lze povšimnou počátku světové souřadnicové soustavy umístěné v rohu testovací místnosti, souřadnicových systémů ArUco značek a souřadnicových systémů dronu Tello.

Posledním ze zmiňovaných nástrojů je rqt_graph, který umožňuje jednoduše vytvořit vizualizaci architektury programu. Lze tak snadno a rychle kontrolovat funkčnost spuštěných uzlů a komunikaci mezi nimi.

[3] http://wiki.ros.org/Tools
[4] https://www.mathworks.com/products/matlab.html



Obrázek 3.3: Vytvořená konfigurace ve vizualizačním nástroji rviz

3.5 Spuštění programu

Aplikační rámec ROS poskytuje nástroj roslaunch[5] využívaný ke spuštění souborů typu "launch". Tyto soubory slouží ke spuštění několika uzlů naráz. V souboru typu launch můžeme zároveň definovat parametry, s kterými se mají dané uzly spustit.

Součástí námi vytvořeného balíčku jsou i soubory typu launch spouštějící vytvořený programu. Hlavní funkce programu se zapne příkazem

```
roslaunch tello_aruco_demo main.launch
```

, který spouští všechny uzly zajišťující připojení k dronu a detekci polohy dronu v prostoru (viz kapitoly 4, 5 a 7) a vizualizační prostředí rviz s námi vytvořenou konfigurací (viz 3.4).

Po úspěšném navázání spojení s dronem lze spustit let pomocí příkazu

```
roslaunch tello_aruco_demo fly.launch
```

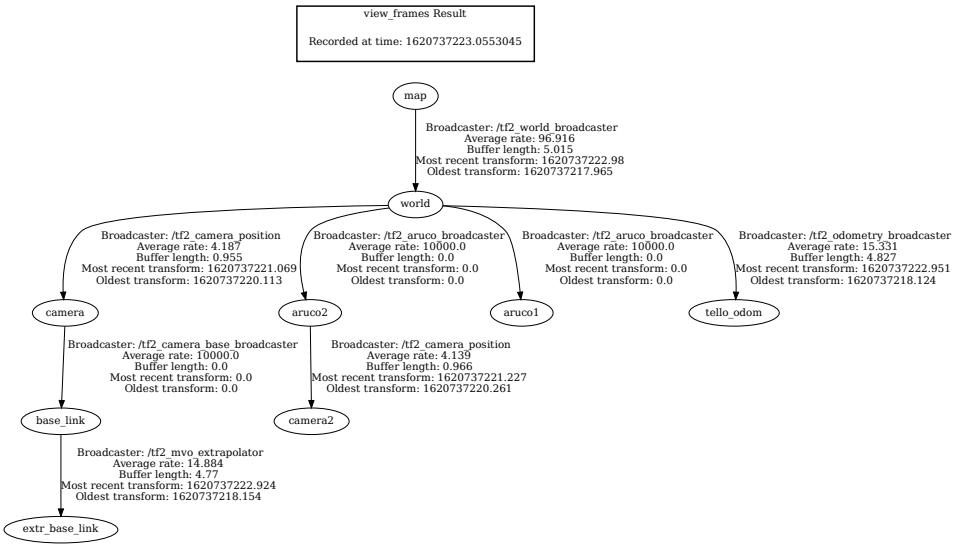
, který spouští uzel řídící let dronu (viz kapitolu 6) a uzel vysílající referenci polohy na základě definované trajektorie (viz kapitola 8). Upozorňujeme, že zavoláním tohoto příkazu dron vzlétne.

[5] http://wiki.ros.org/roslaunch

3.6 Balíček tf2

ROS balíček tf2[6] umožňuje jednoduše spravovat souřadnicové systémy a vztahy mezi nimi. Dále poskytuje celou řadu užitečných nástrojů pro práci s transformacemi v prostoru. Balíček tf2 je hojně využíván v průběhu této práce, a proto se mu budeme podrobněji věnovat.

Balíček umožňuje vytvořit uzly typu "tf2 hlasatel"(anglicky "tf2 broadcaster") a "tf2 posluchač"(anglicky "tf2 listener"). Uzel typu tf2 hlasatel odesílá transformace mezi různými souřadnicovými systémy tf2. Systém tf2 udržuje všechny souřadnicové systémy a vztahy mezi nimi ve stromové datové struktuře. Systém tf2 zároveň uchovává všechny získané transformace mezi systémy v různým časech. Díky tomu může uzel typu tf2 posluchač jednoduše nalézt transformaci mezi dvěma souřadnicovými systémy a to v libovolných dvou časech (existuje-li pro oba souřadnicové systémy v těchto časech záznam o transformacích propojujících daný souřadnicový systém s kořenem stromové datové struktury). Součástí balíčku tf2 je nástroj view_frames, který umožňuje jednoduše vykreslit výše popisovanou stromovou datovou strukturu systému. Takto vygenerovaná vizualizace datové struktury tf2 programu popisovaného v této práci je zobrazena na obrázku 3.4.



Obrázek 3.4: Vizualizace stromové datové struktury systému tf2 vygenerovaná nástrojem view_frame

Pro lepší přehlednost vizualizace na obrázku 3.4 byl omezen počet značek ArUco v prostředí na dvě, z nichž pouze značka aruco2 je viděna kamerou dronu.

[6] http://wiki.ros.org/tf2

Kapitola 4

Balíček tello_driver

Ve dvou předchozích kapitolách jsme představili používaný dron Ryze Tello a aplikační rámec ROS. V této kapitole se budeme věnovat volně dostupnému ROS balíčku tello_driver[1], který využíváme ke komunikaci s dronem Ryze Tello.

Balíček tello_driver nevyužívá žádný z oficiálních způsobů komunikace s dronem (viz sekci 2.3), nýbrž jedná se o nadstavbu neoficiální knihovny TelloPy[2] vytvořené zpětným inženýrstvím oficiálních aplikací ovládajících dron Tello. TelloPy oproti oficiálnímu Tello SDK poskytuje větší množství funkcí, bohužel jen část z nich je dostupná přes tello_driver.

4.1 Zprávy a komunikace

Po zapnutí Tella a připojení řídícího počítače k jeho Wi-Fi může být spuštěn uzel tello_driver_node. Pokud dojde k úspěšnému připojení dronu, tello_driver_node vytvoří několik témat (topic), jejichž stručný přehled je dostupný v tabulkách 4.1 a 4.2.

Kromě témat /tello/cmd_vel a /tello/flip, přenáší všechna témata uvedená v tabulce 4.1 prázdnou zprávu (std_msgs/Empty.msg). Jedním z nejdůležitěj-

[1] http://wiki.ros.org/tello_driver
[2] https://github.com/hanyazou/TelloPy

Téma (topic)	Popis funkce
/tello/takeoff	autonomní vzlétnutí dronu z rovného, statického povrchu
/tello/throw_takeoff	čeká na vyhození dronu do vzduchu, poté spouští řízení letu
/tello/land	autonomní přistání dronu
/tello/palm_land	autonomní přistání dronu na ruce uživatele
/tello/emergency	neprodlené vypnutí všech motorů
/tello/cmd_vel	požadovaný pohyb dronu (translace a rotace)
/tello/fast_mode	přepnutí dronu do rychlého letového režimu
/tello/flip	dron autonomně vykoná vzdušný manévr
/tello/flattrim	kalibrace IMU jednotky

Tabulka 4.1: Přehled témat, pro něž je uzel tello_driver_node příjemcem

ších témat v této práci je /cmd_vel, kterým řídíme požadovaný pohyb dronu. Přestože zpráva přenášená tímto tématem (geometry_msgs/Twist.msg) obsahuje proměnné udávající intenzitu pohybu v šesti stupních volnosti, Tello pracuje pouze se čtyřmi z nich (tři osy translace a otočení dronu).

Téma (topic)	Popis funkce
/tello/imu	informace z palubní IMU jednotky
/tello/odom	vestavěná odometrie dronu závisející na VPS
/tello/status	základní informace o stavu dronu
/tello/image_raw/h264	zakódovaný video stream
/tello/camera/image_raw	dekódovaný video stream
/tello/camera/camera_info	základní informace o kameře, její distorzní model atd.

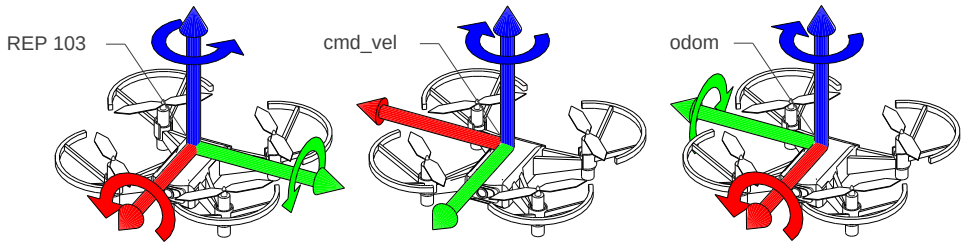
Tabulka 4.2: Přehled témat, pro něž je uzel tello_driver_node vydavatelem

Témata uvedená v tabulce 4.2 přenáší převážně informace ze senzorů Tella. Data zasílaná pomocí tématu /tello/camera/camera_info jsou převážně statická (tedy kromě záhlaví se jejich obsah v čase nemění). Téma /tello/camera/camera_info je zároveň jediným tématem z tabulky 4.2, u něhož data v něm procházející nepochází z dronu Tello. V tomto tématu se přenášejí základní informace o kameře, převážně pak její kalibrační údaje. V této práci používáme kalibrační parametry kamery obsažené v balíčku tello_driver. Téma, v němž jsou přenášena samotná data video streamu, se liší v závislosti na tom, zdali tello_driver poskytuje již dekódovaný nebo ještě zakódovaný video stream.

4.2 Souřadnicový systém, transformace

Aplikační rámec ROS poskytuje celou řadu konvencí, zpravidla popsaných v dokumentech nazývaných REP - ROS Enhancement Proposal (volně přeloženo: návrh na vylepšení ROSu). V této části textu se budeme zabývat obsahem dokumentu REP 103 [3], který porovnáme s implementací dronu Tello.

Na obrázku 4.1 je zobrazeno porovnání souřadnicových systémů používaných pro dron Tello v této práci. Naší snahou bylo vždy převádět souřadnicové systémy do tvaru popisovaném v REP 103. Povšimněme si, že téma cmd_vel bere v potaz pouze jednu rotační souřadnici, jak bylo popsáno v předchozí části textu 4.1.



Obrázek 4.1: Používané souřadnicové systémy dronu Tello v systému ROS - červenou barvou je značena osa X, zelenou osa Y a modrou osa Z.

Přestože je dle REP 103 doporučené vyjadřovat rotace ve formě kvaternionů, setkáváme se často i s jinými reprezentacemi rotací. Tento problém řešíme s využitím nástrojů obsažených v balíčku tf2 (viz 3.6), které snadno umožňují přechod mezi různými reprezentacemi rotací.

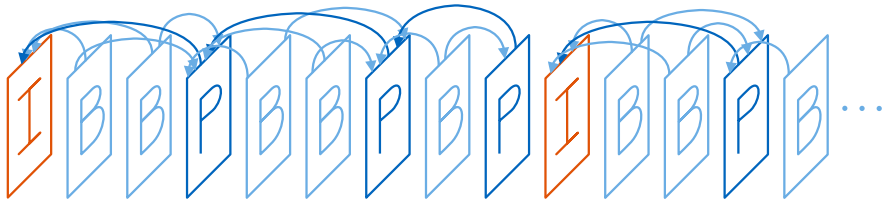
4.3 Přenos H.264 videa, dekódování a časový podpis snímku

K řízení dronu využíváme zpětnou vazbu získanou z obrazu palubní kamery dronu Tello. Z tohoto důvodu je potřeba zajistit kvalitní a stabilní přenos obrazu mezi dronem a řídícím počítačem. Přenos obrazu z kamery probíhá přes Wi-Fi vytvářenou dronem pomocí UDP protokolu. Kvůli redukci množství přenášených dat kóduje Tello video stream z palubní kamery ve formátu H.264 (taktéž nazývaný MPEG-4 AVC) [4]. Poznamenejme, že VPS využívá kameru umístěnou na spodní straně dronu - obraz z ní ovšem není dostupný.

Dron Tello zasílá řídícímu počítači několikrát za sekundu paket obsahující několik snímků videa z kamery. Aby bylo možné s obrazem dále pracovat, je potřeba ho nejprve v řídícím počítači dekódovat. Jak bylo zmíněno v části 4.1, balíček tello_driver umožňuje publikovat obraz z kamery buď kódovaný ve formátu H.264 nebo již dekódovaný.

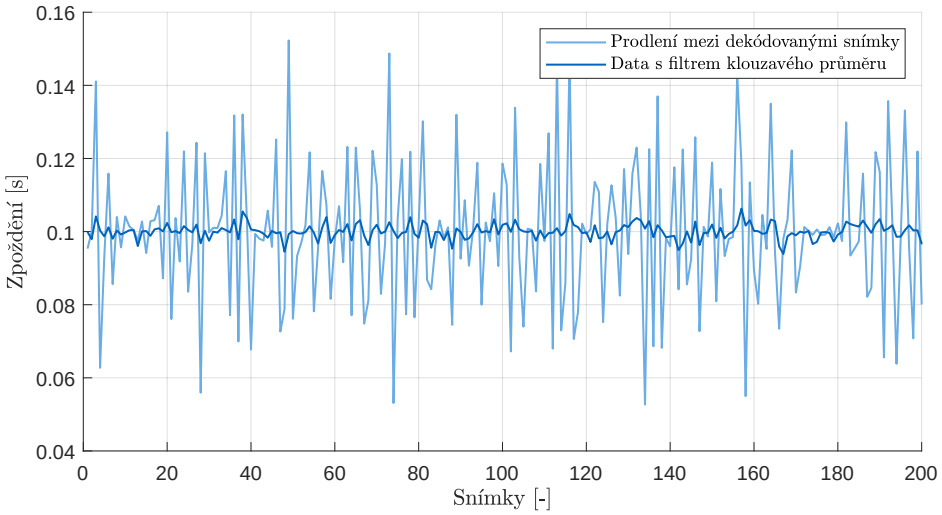
Bylo potřeba provést několik úprav, abychom mohli k dekódování obrazu využít balíček tello_driver. V první řadě bylo zavedeno omezení velikosti fronty dekodéru snímků na maximálně 20 snímků. Tento krok byl proveden na základě dvou problémů. Prvním z nich bylo vytvoření dlouhé fronty snímků v čase mezi zahájením přenosu obrazu mezi řídícím počítačem a Tellem a spuštěním dekodéru. Druhým důvodem bylo vytvoření dlouhé fronty snímků v případě zpoždění dekodéru způsobeném například nadměrným zatížením řídícího počítače. Vznikne-li na vstupu dekodéru dlouhá fronta snímků, pak na jeho výstupu získáváme zastaralý obraz z kamery dronu. Formát H.264 využívá tři typy snímků (viz 4.2). Snímek typu I (též nazývané key frame, tedy klíčový snímek) je nezávislý na ostatních snímcích. Snímek typu P vychází z předchozího snímku I nebo P. Snímek B vychází z předchozích a následujících snímků I, P nebo B. Omezení fronty, tedy mazání starých snímků, vede vzhledem ke struktuře snímků formátu H.264 k narušení video přenosu. Je tedy potřebné zajistit dostatečný výpočetní výkon řídicího počítače.

Snímky z Tella nejsou časově podepsané, je tedy téměř nemožné určit přesné stáří snímku. Vzhledem k implementaci dekodéru je navíc velice obtížné provést časový podpis snímku před jeho dekódováním. Na základě měření zpoždění mezi pořízením snímku dronem Tello a jeho vykreslením na řídícím počítači (viz obrázek 4.4) bylo stanoveno průměrné zpoždění snímku před jeho časovým podepsáním na 175.68 milisekund se standardní směrodatnou odchylkou 17.85 milisekund. Výsledky měření jsou založeny na padesáti vzorcích. Měření zanedbává zpoždění zobrazovacího zařízení.

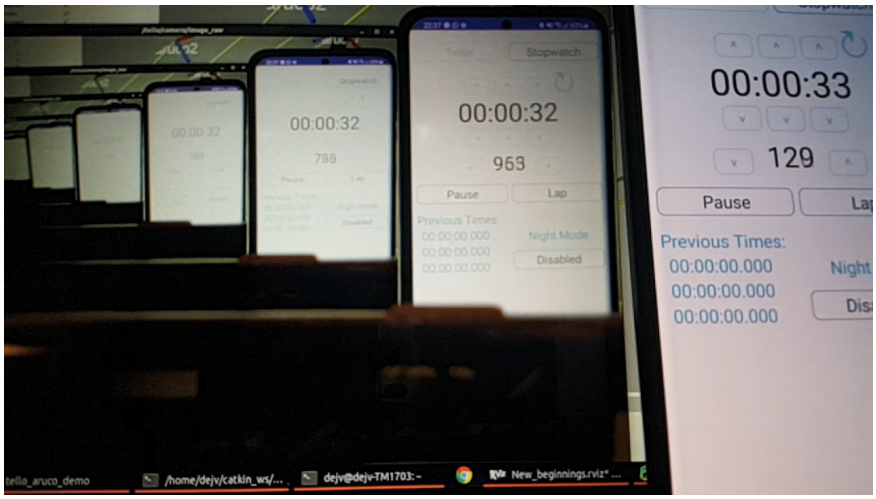


Obrázek 4.2: Ilustrace snímků kódovaného videa H.264

Na obrázku 4.3 je zobrazeno zpoždění mezi jednotlivými snímky na výstupu dekodéru, kdy uvažujeme pouze každý třetí snímek (viz 5.2). Při implementaci časového podpisu dekódovaných snímků do balíčku tello_driver byl na základě odhaleného šumu v tomto měření použit filtr klouzavého průměru.



Obrázek 4.3: Zpoždění mezi jednotlivými snímky na výstupu dekodéru balíčku tello_driver a jejich klouzavý průměr (při deseti vzorcích)



Obrázek 4.4: Snímek z vysokorychlostního video záznamu (120 fps) využitém k zjištění stáří snímku z video přenosu dronu Tello (30 fps) po dekódování na řídícím počítači (obrazovka s obnovovací frekvencí 60 Hz) s časomírou s obnovovací frekvencí obrazovky 120 Hz

Kapitola 5

Detekce polohy z obrazu

K řízení polohy dronu ve známém prostředí je zapotřebí poskytnout řídícímu systému zpětnou vazbu ve formě aktuální polohy dronu vůči danému prostředí. Jedním z nejčastěji používaných řešení při řízení kvadrokoptéry je využití globálního polohového systému (GPS). Ten ovšem spoléhá na viditelnost družic a jeho přesnost není dostatečná pro orientaci ve zúžených prostorech. GPS tedy není vhodný pro řízení dronu například ve vnitřních prostorách budov. Často využívaným řešením pro lokalizaci a mapování ve vnitřních prostorách bez přístupu k GPS je využití LIDARu [5]. Nevýhodou této metody je především vysoká pořizovací cena, výpočetní náročnost zpracování dat, rozměry a hmotnost zařízení. Výhodou je pak robustní orientace ve vnitřním prostředí, jak dokazuje systém Dronument[6], i venkovním prostředí [7].

Nemalé úsilí mnoha výzkumných týmů se věnuje využití dat z palubní kamery dronu k jeho lokalizaci v okolním prostoru. Kamera poskytuje velké množství informací, přesto jsou její rozměry a váha vhodné pro použití na palubě dronu. Pořizovací cena kamer se díky vysoké poptávce navíc v průběhu let výrazně snížila.

Existuje mnoho přístupů k zpracování dat z kamery využívaných pro řízení dronu - například využití vizuální simultánní lokalizace a mapování (VSLAM) nebo využití neuronových sítí k vytvoření hloubkových map z obrazu kamery. Jedním z nejčastěji používaných přístupů pro lokalizaci ve známém prostředí je využití detekce značek rozmístěných v prostoru z obrazu kamery. Tento přístup přináší mnohé výhody jako například relativně kvalitní lokalizaci a nadstandardní rychlost zpracování dat.

K detekci polohy kamery se využívá mnoho různých typů značek [8][9]. Zpravidla se jedná o kontrastní značky s pevně danou orientací a identifikátorem. V této práci využíváme značky ArUco pro jejich všestranně výhodné vlastnosti při detekci pozice kamery.

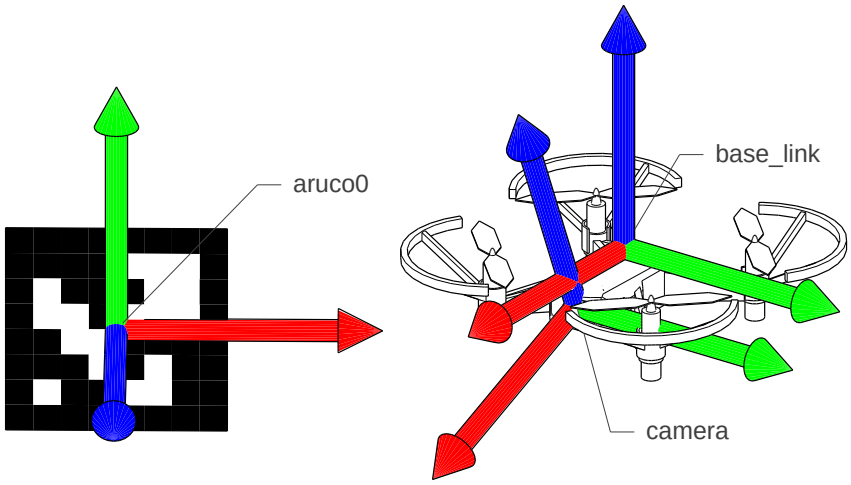
5.1 Značky ArUco

Detekce ArUco značek v obrazu je robustní metodou na zjištění pozice kamery. Tyto čtvercové značky se skládají z černého ohraničení a vnitřního binárního kódu. Černé ohraničení umožňuje snadnou detekci značky v obraze a aproximaci její pozice. Vnitřní binární kód obsahuje informaci o identifikačním čísle značky, její orientaci a data umožňující detekci a korekci chybného sejmutí kódu.

Vzhled každé ArUco značky je dán slovníkem, z kterého pochází, a jejím identifikačním číslem. Slovník je soubor všech kódů značek v něm obsažených. Velikost každého slovníku (celkový počet různých kódů) a velikosti jeho značek (počet bitů vnitřního kódu) jsou pevně stanoveny. V této práci využíváme předdefinovaný slovník DICT_6X6_50, tedy slovník s velikostí binárního kódu 6×6 a maximálním počtem padesáti značek.

Slovník dané ArUco značky poskytuje dostatek informací pro její detekci v obraze. Pro výpočet pozice kamery z ArUco značky je však nutné znát navíc i její fyzické rozměry. Ty nejsou dány slovníkem značky a mohou se tak vzájemně lišit mezi jednotlivými značkami konkrétního slovníku. V této práci se používají značky o rozměru 100×100 milimetrů.

Souřadnicový systém ArUco značek je stanovený následujícím způsobem. Její počátek je umístěný ve středu značky a všechny osy jsou si vzájemně ortogonální, osa Z je ortogonální k ploše značky a je kladná ve směru vycházejícím ze značky. Osa Y prochází středem horní strany značky, osa X prochází středem pravé strany značky. Vizualizace ArUco značky a použitých souřadnicových soustav při detekci aruco kódů je zobrazena na obrázku 5.1. Další informace na toto téma jsou dostupné na webových stránkách [10].



Obrázek 5.1: ArUco značka slovníku DICT_6X6_50 s identifikačním číslem 0, souřadnicové soustavy ArUco značek a Tella

5.2 Představení balíčku aruco_detect

K detekci příznaků v obraze v podobě ArUco značek využíváme ROS balíček aruco_detect[1]. Balíček je založen na implementaci obsažené v OpenCV knihovně [11][12]. Balíček umožňuje jednoduše detekovat ArUco značky v obraze a vypočítat jejich transformace od kamery ke značce. Úplný seznam témat využívaných balíčkem aruco_detect je zaznamenán v tabulce 5.1.

Vztah	Název	Popis funkce
Příjemce	/camera/compressed	dekódovaný video stream
Příjemce	/camera_info	základní informace o kameře
Vydavatel	/fiducial_vertices	pozice čtyř rohů značek ve snímku
Vydavatel	/fiducial_transforms	transformace od kamery ke značkám
Vydavatel	/fiducial_images	vizualizace detekovaných značek

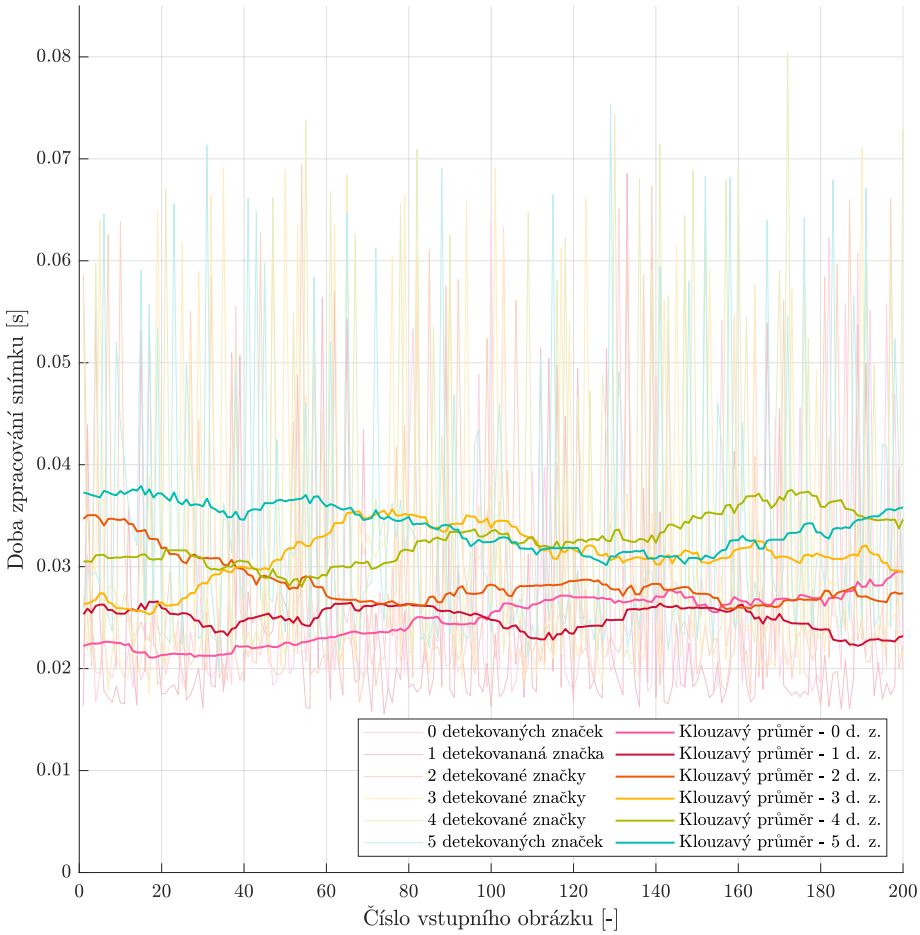
Tabulka 5.1: Seznam všech témat využívaných balíčkem aruco_detect

Při spuštění balíčku je nutné definovat několik parametrů, konkrétně používaný slovník ArUco značek a fyzickou velikost značek. Volitelné parametry pak umožňují například definovat ignorované značky a značky s jinými fyzickými rozměry. Dále umožňují nastavení parametrů samotné detekce značek knihovnou OpenCV. V průběhu této práce využíváme výchozí hodnoty těchto parametrů.

Detekce ArUco značek v obraze je výkonově náročná. Tello přenáší video s

[1] http://wiki.ros.org/aruco_detect

frekvencí 30 snímků za sekundu. Vzhledem k časové náročnosti zpracování jednoho snímku balíčkem aruco_detect zobrazené na obrázku 5.2 bylo nutné upravit balíček a omezit frekvenci detekce příznaků na 10 snímků za sekundu. Pokud dojde k přetížení řídícího počítače, pak se opozdí i dekódování video přenosu a následné dojde k poškození jeho obrazu.



Obrázek 5.2: Časová náročnost zpracování jednoho snímku z kamery dronu Tello knihovnou aruco_detect v závislosti na počtu detekovaných značek ArUco. Pro přehlednost jsou data uvedena i s klouzavým průměrem o padesáti vzorcích

Všechna měření byla provedena na laptopu Xiaomi Mi Notebook Air 13.3 2018 s operačním systémem Ubuntu 20.04.2.0 LTS, procesorem Intel® Core™ i5-8250U, 8 GB DDR4 operační paměti a grafickou kartou NVIDIA® GeForce® MX150.

5.3 Detekce pozice dronu ve známém prostředí s využitím ArUco značek

K detekci pozice dronu ve známém prostředí využíváme několik uzlů vzájemně komunikujících převážně přes rozhraní balíčku tf2 (viz 3.6). Seznam uzlů používaných při detekci pozice dronu je uveden v tabulce 5.2.

Uzel	Posluchač transformací	Hlasatel transformací
aruco_detect		camera_id → aruco_id
tf2_aruco_broadcaster		world → aruco_id
tf2_camera_base_broadcaster		camera → base_link
tf2_camera_position	world → camera_id	aruco_id → camera_id
		world → camera

Tabulka 5.2: Seznam uzlů používaných při detekce pozice dronu

Souřadnicový systém world slouží jako počátek světové souřadnicové soustavy a je definovaný jako jeden z rohů testovací místnosti. Tento statický souřadnicový systém je publikován uzlem static_transform_publisher zahrnutým v aplikačním rámci ROS.

Uzel tf2_aruco_broadcaster je hlasatel statických transformací mezi počátkem světové souřadnicové soustavy a jednotlivými kódy ArUco. Název souřadnicového systému daného ArUco vždy končí jeho identifikačním číslem. Seznam všech pozic a identifikačních čísel značek ArUco rozmístěních v prostředí je definován v souboru \environment\ArUco_list.json v námi vytvořeném balíčku.

Hlasatelem statické transformace mezi souřadnicovým systémem kamery dronu (camera) a souřadnicovým systémem base_link (viz obrázek 5.1) je uzel tf2_camera_base_broadcaster. Translační složka této transformace byla změřena pomocí posuvného měřítka. Rotační složka transformace byla změřena s využitím balíčku aruco_detect z transformace camera_id → aruco_id, kdy dron byl umístěn na rovný povrch a ArUco značka na kolmou stěnu.

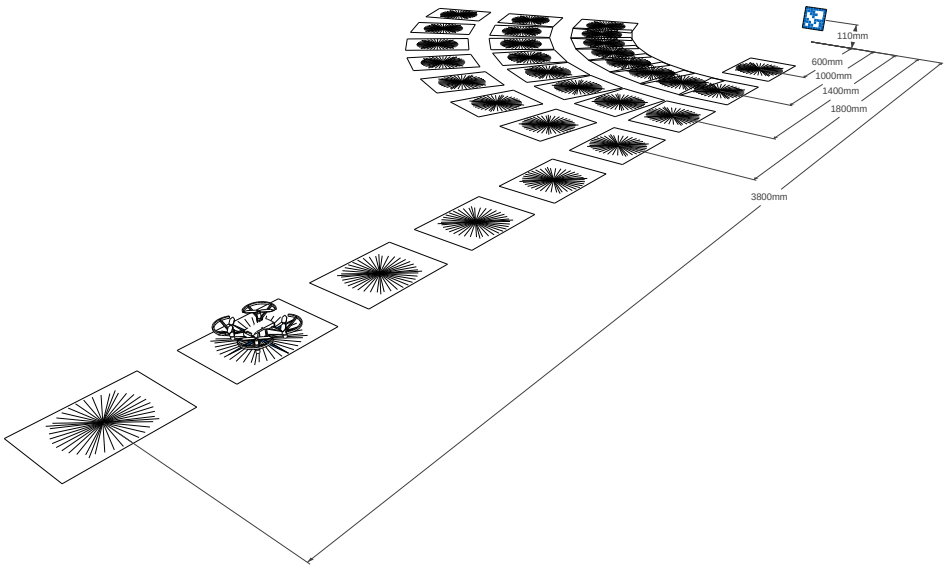
Uzel tf2_camera_position je příjemcem tématu /fiducial_transforms, v němž uzel aruco_detect (viz 5.2) vysílá zprávy obsahující transformace camera_id → aruco_id všech detekovaných ArUco značek v daném snímku z kamery. Zpráva v sobě nese časový podpis tohoto snímku po jeho dekódování (viz 4.2). Pro každou z přijatých transformací vypočte uzel inverzní transformaci (tedy aruco_id → camera_id) a tu publikuje do systému tf2. Statické transformace publikované uzlem tf2_aruco_broadcaster umožňují uzlu

tf2_camera_position neprodleně získat transformace world → camera_id ze systému tf2. Nakonec uzel tf2_camera_position provede vážené průměrování těchto transformací podrobně popsané v části 5.5 a výsledek publikuje jako transformaci world → camera. Princip funkce popisovaného systému lze pozorovat i na obrázcích 3.3 a 3.4.

5.4 Měření přesnosti detekce ArUco značek

V této části textu se zabýváme měřením přesnosti detekce jedné ArUco značky vzhledem k různým polohám kamery - převážně pak přesností transformací camera_id → aruco_id získaných z tématu /fiducial_transforms balíčku aruco_detect.

Všechny výsledky uvedené v této kapitole byly získány na měřící soustavě zobrazené na obrázku 5.3. Popisovaná měřící soustava se skládá z jedné ArUco značky 6×6 s identifikačním číslem 0 a velikosti 100 milimetrů, dronu Tello, plastového měřícího nástavce na dron Tello a řady pevně umístěných úhloměrů vytištěných na papírech formátu A4. Měřící nástavec na dron je 3D vytištěný kříž s výstupky, které lze zachytit do větrací mřížky na spodní straně dronu (viz obrázek 2.2). Střed tohoto kříže se nachází přímo pod souřadnicovým systémem base_link. V kombinaci s papírovými úhloměry umožňuje měřící nástavec rychle a přesně polohovat dron vůči značce ArUco.

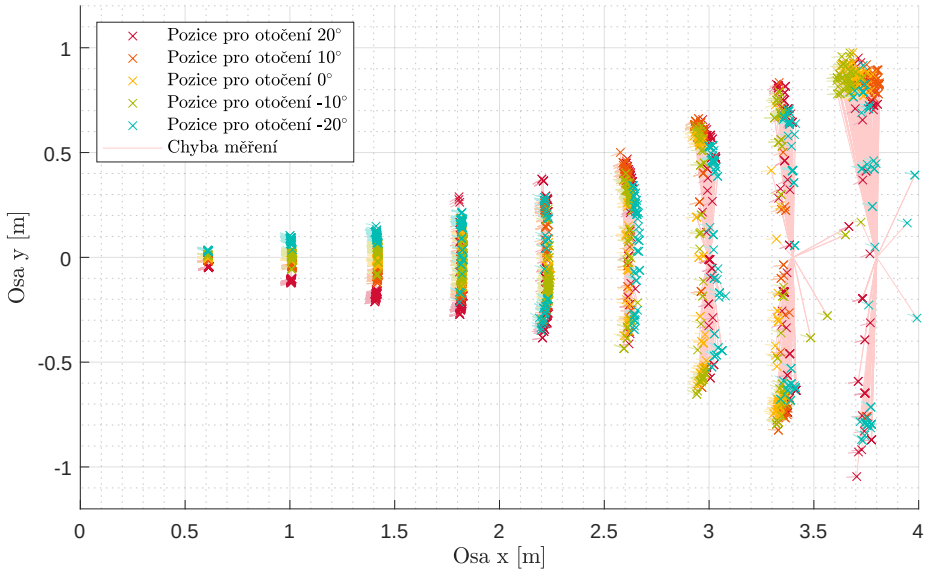


Obrázek 5.3: Ilustrace měřící soustavy přesnosti detekce ArUco značek

Na měřící soustavě byly provedeny dvě sady měření. V obou měřeních byl

dron umísťován na různé pozice na měřící soustavě a na každé z nich byl po dobu třiceti sekund zaznamenáván průběh získaných transformací.

V první sadě měření se dron postupně vzdaloval kolmo od značky ArUco a to ve vzdálenostech 60 cm až 380 cm od značky s rozestupy 40 cm. V každé měřené vzdálenosti byl dron dále orientován v pěti různých otočeních, konkrétně -20°, -10°, 0°, 10° a 20° vzhledem k otočení dronu ve směru značky. Výsledky tohoto měření jsou zobrazeny na obrázku 5.4.

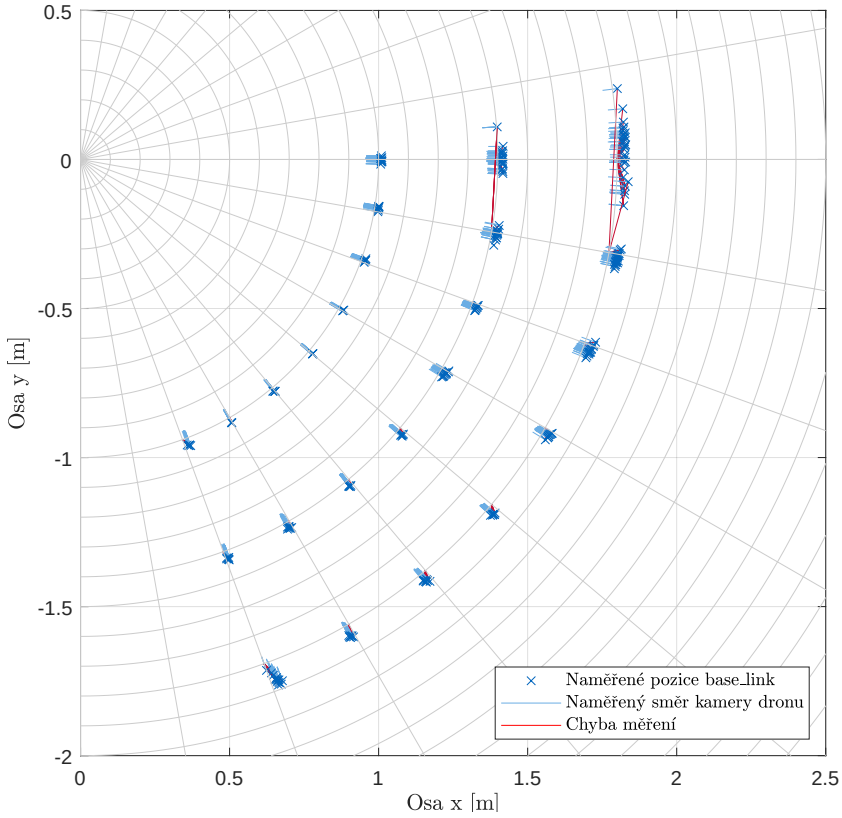


Obrázek 5.4: Výsledky prvního měření přesnosti detekce ArUco značek

V druhé sadě měření byl dron vždy orientován tak, aby kamerou mířil přímo na značku ArUco. Měření probíhala ve vzdálenostech 100 cm, 140 cm a 180 cm od značky ArUco. Dron v průběhu měření měnil svojí orientaci otočením o 0° až 70° s krokem 10° okolo bodu umístěném ve středu značky ArUco ve směru hodinových ručiček při pohledu shora na měřící soustavu (dále úhel φ). Výsledky druhého měření jsou zobrazeny na obrázku 5.5

5.5 Filtrace dat (vážený průměr)

Z měření zobrazených na obrázcích 5.4 a 5.5 lze pozorovat závislost mezi umístěním dronu Tello na měřící soustavě a přesností měřené polohy získané ze značky ArUco. Fakt, že se v našem testovacím prostředí (viz obrázek 3.3) vyskytuje hned několik značek ArUco, nám dovoluje zpracovávat získaná data tak, abychom dosáhli vyšší přesnosti měření pozice.



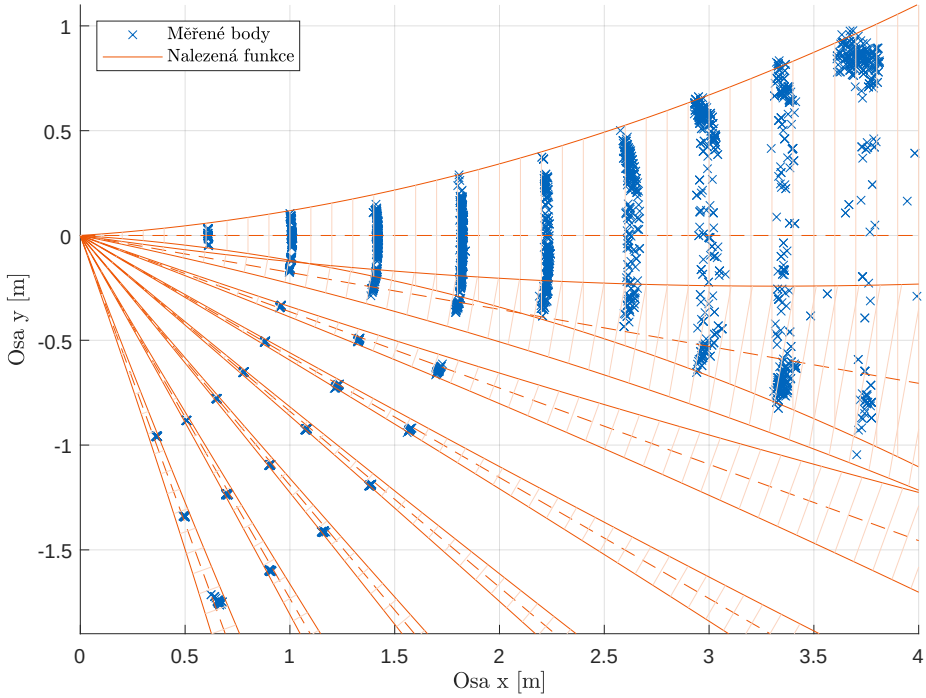
Obrázek 5.5: Výsledky druhého měření přesnosti detekce ArUco značek

K určení přesnosti detekce polohy byla na základě změřených dat nalezena funkce 5.1 přibližně popisující maximální odchylku od reálné polohy ve směru kolmém kameře dronu v závislosti na vzdálenosti d [m] dronu od značky ArUco a úhlu φ [rad] využívaném v druhém měření. Výstup funkce je ilustrován na obrázku 5.6.

$$f(d, \varphi) = \left(\frac{d^2}{20} + \frac{d}{16}\right) \cdot \left(0.1 + \left||\varphi| - \frac{\pi}{4}\right|^4 \cdot 2.5\right) \tag{5.1}$$

Dále využíváme funkci $g(d) = \frac{d}{60}$, která přibližně vyjadřuje maximální odchylku od reálné polohy ve směru kamery dronu. V tomto směru je měření zpravidla výrazně přesnější.

Funkce 5.1 byla implementována v uzlu tf2_camera_position (viz 5.3), kde se na základě výstupů této funkce provádí vážené průměrování transformací získaných z ArUco značek ve třech stupních volnosti (translační složky transformace).



Obrázek 5.6: Ilustrace výstupu nalezené funkce 5.1 pro úhel $\varphi = 0°, 10°, ...,70°$ a $d >= 0$ spolu s daty z obou měření

Jelikož měřená natočení dronu byla dostatečně přesná pro naše účely, nebylo potřeba aplikovat algoritmus váženého průměru i na zbylé tři stupně volnosti (rotační složky transformace). Přesto je uzel tf2_camera_position implementován tak, aby jednoduše umožňoval přidání váženého průměrování rotačních složek.

Poté, co uzel tf2_camera_position získá všechny souřadnicové systémy camera_id z aktuálního snímku, ukládá si jejich transformace vůči počátku světové souřadnicové soustavy do matic

$$T = \begin{bmatrix} x_{ArUco_1} & y_{ArUco_1} & z_{ArUco_1} \\ \vdots & \vdots & \vdots \\ x_{ArUco_n} & y_{ArUco_n} & z_{ArUco_n} \end{bmatrix},$$
$$Q = \begin{bmatrix} x_{ArUco_1} & y_{ArUco_1} & z_{ArUco_1} & w_{ArUco_1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{ArUco_n} & y_{ArUco_n} & z_{ArUco_n} & w_{ArUco_n} \end{bmatrix}. \tag{5.2}$$

Program na základě transformací world → camera_id a world → aruco_id vypočítá váhy jednotlivých ArUco značek ve třech osách translace světové souřadnicové soustavy. Část programu zodpovědná za výpočet těchto vah je uvedena jako algoritmus 5.1. Algoritmus předpokládá umístění značek ArUco osou Y jejich souřadnicového systému směrem vzhůru a polohu dronu osou Z

jeho souřadnicového systému směrem vzhůru.

```
def getWeights(self, aruco, tello):
    x = tello.transform.translation.x − aruco.transform.translation.x
    y = tello.transform.translation.y − aruco.transform.translation.y
    z = tello.transform.translation.z − aruco.transform.translation.z
    alpha_aruco = trans.euler_from_quaternion([aruco.transform.rotation.x,
                                               aruco.transform.rotation.y,
                                               aruco.transform.rotation.z,
                                               aruco.transform.rotation.w], 'szyx')[0]
    d = math.sqrt(x*x+y*y+z*z)  # Vzdalenost dronu od znacky
    alpha_tello = math.atan2(y, x) + math.pi / 2  # Uhel dronu vuci znacce
    alpha = alpha_tello − alpha_aruco  # Uhel alpha mezi dronem a znackou
    beta = math.atan2(z,math.sqrt(x*x+y*y))  # Predpoklada osu Y ArUca vzhuru

    #  Vypocet vah na zaklade funkci f a g (Vaha = 1/maximalni odchylka)
    w_lr = 1/(((d*d)/20+d/16)*(0.1 + math.pow(abs(abs(alpha)−math.pi/4),4)*2.5))
    w_fb = 1/(d/60)
    w_ud = 1/(((d*d)/20+d/16)*(0.1 + math.pow(abs(abs(beta)−math.pi/4),4)*2.5))

    #  Orientace vah vzhledem k osam svetove souradnicove soustavy
    w_x = abs(w_lr * math.cos(alpha_tello) − w_fb * math.sin(alpha_tello))
    w_y = abs(w_lr * math.sin(alpha_tello) + w_fb * math.cos(alpha_tello))
    w_z = w_ud

    # Nastaveni vah − pro rotace ponechavame jednotkovou vahu
    weights = [w_x, w_y, w_z, 1]

    return weights
```

Algoritmus 5.1: Funkce počítající váhy ArUco značky na základě vstupních transformací world → camera_id a world → aruco_id

Nalezením vah všech detekovaných značek ArUco pomocí algoritmu 5.1 získáváme matici

$$W = \begin{bmatrix} w_{Tx_1} & w_{Ty_1} & w_{Tz_1} & w_{Q_1} \\ \vdots & \vdots & \vdots & \vdots \\ w_{Tx_n} & w_{Ty_n} & w_{Tz_n} & w_{Q_n} \end{bmatrix}. \tag{5.3}$$

Program poté provádí vážené průměrování translačních složek dle vzorců

$$\begin{aligned} T_{\text{vážené}} &= T \odot W_T, \\ T_{norm} &= T_{\text{vážené}} \oslash \begin{bmatrix} \sum_{i=1}^{n} w_{Tx_i} & \sum_{i=1}^{n} w_{Ty_i} & \sum_{i=1}^{n} w_{Tz_i} \end{bmatrix}, \\ T_{\text{průměrné}} &= \begin{bmatrix} \sum_{i=1}^{n} T_{norm_{i1}} & \sum_{i=1}^{n} T_{norm_{i2}} & \sum_{i=1}^{n} T_{norm_{i3}} \end{bmatrix}, \end{aligned} \tag{5.4}$$

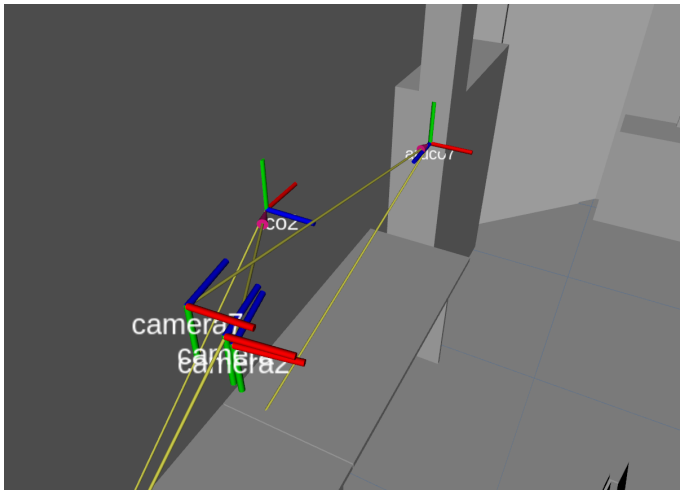
kde W_T představuje první tři sloupce matice W, symbol $\odot$ představuje Hadamardův součin (násobení matic po složkách) a symbol $\oslash$ představuje dělení matic po složkách.

Dále program provádí vážené průměrování rotačních složek ve formě kvaternionů pomocí vzorců

$$\begin{aligned} Q_{vážené} &= Q \odot W_Q, \\ Q_{norm} &= Q_{vážené} \oslash \sum_{i=1}^{n} w_{Q_i}, \\ [D, V] &= eigen(Q^T \cdot Q), \end{aligned} \tag{5.5}$$

kde W_Q představuje poslední sloupec matice W, $eigen$ představuje rozklad na vlastní čísla D a vlastní vektory V. Vlastní vektor V odpovídající největšímu vlastnímu číslu D pak představuje námi hledaný vážený průměr rotační složky.

Sloučením získané translační a rotační složky získává uzel tf2_camera_position transformaci world → camera. Takto se nám podařilo výrazně zvýšit přesnost detekce polohy v prostoru z příznaků v obrazu z palubní kamery dronu Tello. Výsledky této metody lze pozorovat na obrázku 5.7, který byl pořízen ve vizualizačním prostředí rviz.



Obrázek 5.7: Vizualizace funkčnosti popisovaného systému váženého průměrování transformací získaných ze značek ArUco - poloha camera2 je vypočtena z detekce značky aruco2, poloha camera7 je vypočtena z detekce značky aruco7 a poloha camera je váženým průměrem poloh camera2 a camera7

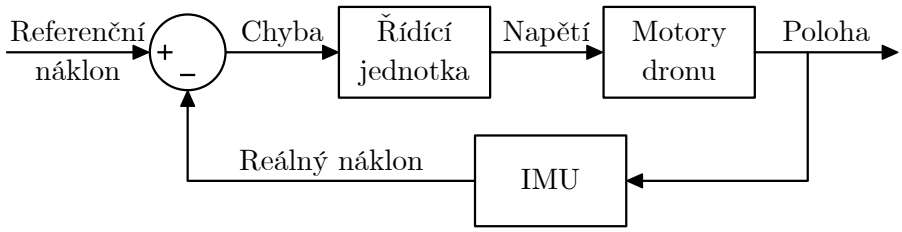
Kapitola 6

Řízení dronu

Cílem této práce je řízení letu cenově dostupného dronu Ryze Tello z externího počítače. V sekci 2.3 jsme představili princip letu dronu řízený vestavěnou řídící jednotkou. Jejím prioritním cílem je stabilizace dronu řízením rychlostí motorů. Díky vestavěnému VPS je navíc Tello schopné kompenzovat špatnou kalibraci IMU a vlivy okolí (například vítr) a omezit tak samovolný pohyb dronu. K autonomnímu řízení dronu je ovšem zapotřebí znát také jeho pozici vůči okolnímu prostředí, čemuž jsme se věnovali v předchozí kapitole. Tato kapitola naváže na detekci polohy (kapitola 5) a představí způsob řízení dronu Tello podle pozice ve známém prostředí.

Pří řízení libovolného systému využíváme otevřené či uzavřené řídící smyčky. Otevřená smyčka řídí systém nezávisle na jeho výstupu, proto se využívá převážně v systémech, které mají tendenci se samovolně vrátit do určitého ekvilibria. Při řízení kvadrokoptér se proto využívá zpravidla uzavřených řídících smyček, které svůj výstup upravují na základě zpětné vazby systému. Složitější systémy, jako je ten náš, využívají celou řadu řídících smyček. Příkladem řídící smyčky může být předem zmiňovaná stabilizace dronu na základě dat z IMU (viz obrázek 6.1).

Řídící jednotka na obrázku 6.1 plní funkci regulátoru systému. Regulátor je zařízení, jehož cílem je pomocí úpravy signálů na jeho výstupu minimalizovat vstupní odchylky od požadované hodnoty, tedy chybu.



Obrázek 6.1: Schéma uzavřené řídící smyčky náklonu kvadrokoptéry

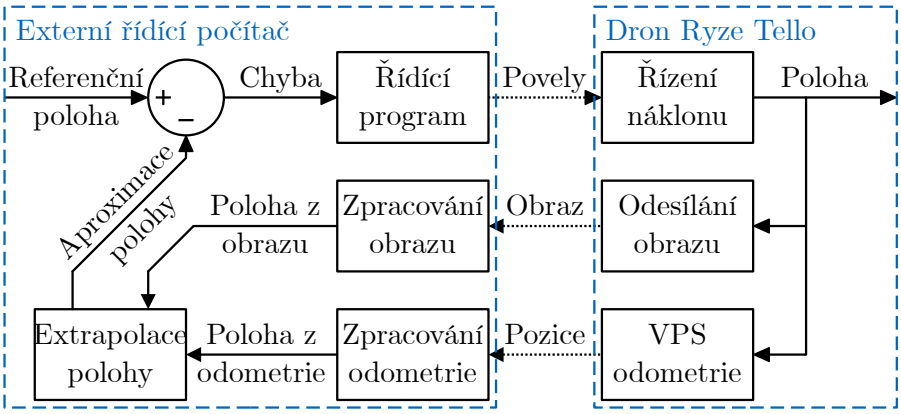
6.1 Výběr regulátoru

Regulátory jsou v dnešní době běžně využívaným prvkem pro řízení široké škály systémů. Setkat se s nimi lze například v letadlech, výtazích, myčkách na nádobí a podobně. Aby byla optimálně uspokojena jejich široká škála využití, existuje mnoho různých typů regulátorů. Výběr správného typu regulátoru je fundamentální částí návrhu řízení každého systému.

Většina regulátorů předpokládá téměř okamžitou odezvu výstupu systému zpět na vstup, popřípadě opožděný vliv akčního členu systému (příkladem může být regulace teploty domu). V našem případě, kdy regulujeme polohu dronu vůči známému prostředí pomocí vzdáleného počítače, se setkáváme s nezanedbatelnými zpožděními a to převážně při přenosu a zpracování obrazu, ze kterého získáváme zpětnou vazbu polohy dronu v prostředí.

Pokud bychom tato velká zpoždění zanedbali, vedlo by to k silným oscilacím, případně úplnému selhání regulátoru. Nabízí se využít modelové prediktivní řízení, které se snaží předvídat reakce systému vyvolané výstupem regulátoru, a s jejich pomocí systém řídit. Jak z názvu této metody vyplývá, je zapotřebí mít k dispozici co nejkvalitnější model systému. Existuje mnoho způsobů, jak tento model získat. Jedním z nejběžnějších způsobů je vytvoření matematického modelu dronu [13], to ovšem může být velice komplikované, obzvlášť pokud, jako v našem případě, nejsou všechny fyzikální vlastnosti dronu známé. Dalším z možných způsobů je trénování modelu na základě velkého datasetu zaznamenaných letů, jak je diskutováno v [14]. Výsledky této metody ovšem silně závisí na velikosti a kvalitě datasetu. Pro náš případ by bylo taktéž možné využít metodu zmiňovanou v [15], kde je využito parametrické identifikace k nalezení aproximace dynamických vlastností dronu. Máme-li k dispozici model diskutovaného systému, můžeme pak k jeho řízení využít například Smithův prediktor [16].

Tello ovšem disponuje i nadstandardním vizuálním polohovacím systémem, jehož data lze využít k extrapolaci pozice dronu (kapitola 7) s vysokým zpožděním získané z příznaků v obraze (kapitola 5). Takto byl výrazně snížen vliv zpoždění obrazu, což umožnilo využití neprediktivního regulátoru. Zjednodušené schéma uzavřené řídící smyčky finálního systému řízení polohy v prostoru je zobrazené na obrázku 6.2.



Obrázek 6.2: Schéma uzavřené řídící smyčky pozice dronu v prostoru

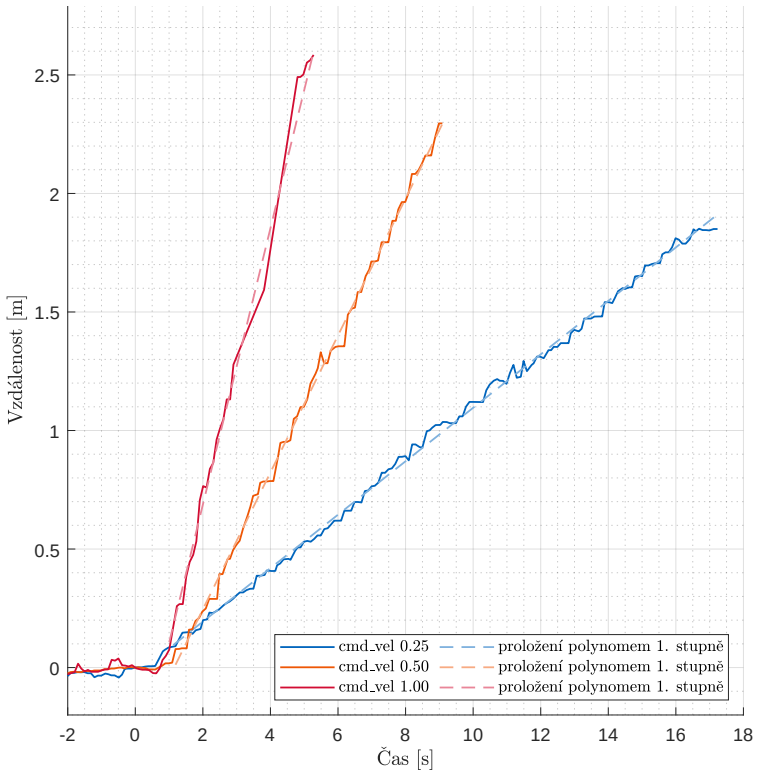
6.2 Výstup regulátoru

Než se začneme zabývat návrhem zvoleného regulátoru, je dobré věnovat pozornost jeho výstupu. V této části textu se zaměříme na to, jak Tello reaguje na výstup regulátoru, tedy na povely zasílané dronu pomocí tématu /tello/cmd_vel (viz 4.1).

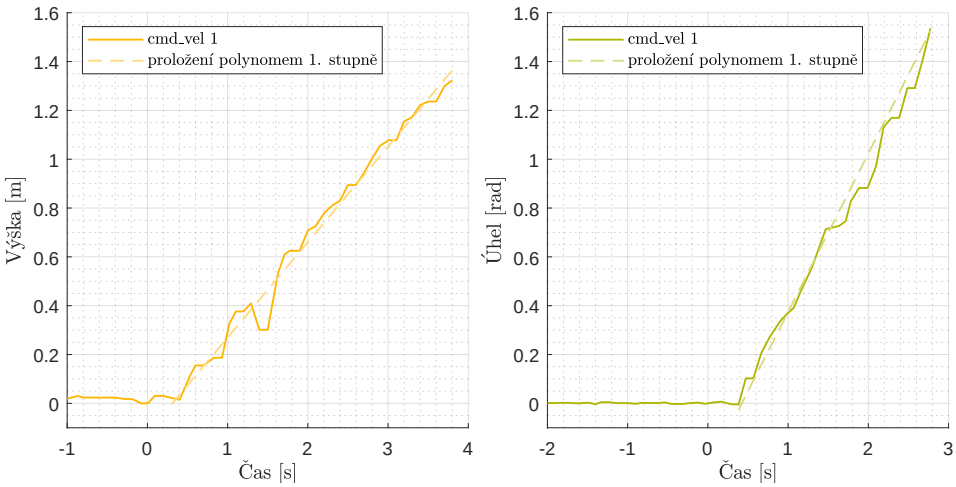
Bylo provedeno několik měření, jejichž výsledky jsou zaznamenány na obrázku 6.3. V průběhu všech měření byl dron v čase t=0 uveden do pohybu zasláním zprávy tématu /tello/cmd_vel s právě jedním ze čtyř stupňů volnosti nastaveným na nenulovou úroveň. Poznamenejme, že poloha dronu je měřena pomocí příznaků v obrazu z palubní kamery (v datech se projevuje zpoždění).

Z provedených měření pozorujeme přibližně lineární závislost mezi povelem tématu /tello/cmd_vel a rychlosti pohybu v dané ose. Tento jev se vztahuje na všechny osy pohybu ovládané pomocí tématu /tello/cmd_vel.

Proložením měřených dat polynomem 1. stupně můžeme získat přibližnou rychlost dronu (viz tabulka 6.1). Povšimněme si, že dron Tello reaguje na příkazy cmd_vel stejné hodnoty v různých osách rozdílnou rychlostí pohybu.



(a) : Výsledky měření vlivu výstupu regulátoru na pohyb dronu v ose Y dle REP 103



(b) : Výsledky měření vlivu výstupu regulátoru na pohyb dronu v ose Z (vlevo) a rotaci okolo osy Z (vpravo) dle REP 103

Obrázek 6.3: Výsledky měření vlivu výstupu regulátoru na pohyb dronu

REP 103 osa	cmd_vel hodnota	rychlost dronu
osa Y	0.25	0.1127 [m/s]
osa Y	0.50	0.2889 [m/s]
osa Y	1.00	0.5829 [m/s]
osa Z	1.00	0.3911 [m/s]
rotace okolo Z	1.00	0.6488 [rad/s]

Tabulka 6.1: Přehled měřených příkazů cmd_vel a rychlostí dronu těmito příkazy vyvolanými

6.3 PID regulace

Ná základě závěrů z části 6.1 byl pro řízení polohy dronu vůči známému prostředí vybrán PID regulátor. Jedná se o jeden z nejpoužívanějších regulátorů, jelikož je relativně jednoduchý, efektivní a dostatečně účinný pro celou řadu aplikací. PID regulátor se skládá z proporcionální, integrační a derivační složky. Váhu jednotlivým složkám regulátoru dávají tři konstanty zesílení K_P,K_I a K_D. Můžeme se ovšem setkat i s případy, kdy jsou některé z konstant zesílení nulové (tedy daná složka nemá žádný vliv na výstup), pak hovoříme například o P, PI nebo PD regulátoru.

Proporcionální složka (P) je přímo úměrná chybě na vstupu. Integrační složka (I) je přímo úměrná integrálu chyby na vstupu. Derivační složka (D) je pak přímo úměrná derivaci chyby na vstupu. Součtem všech tří složek pak získáváme výstup PID regulátoru. Matematicky lze tyto rovnosti zapsat ve tvaru 6.1 pro spojitý PID regulátor, popřípadě ve tvaru 6.2 pro diskrétní formu PID regulátoru.

$$PID(t) = P(t) + I(t) + D(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t) + K_D \cdot \frac{de(t)}{dt} \quad (6.1)$$

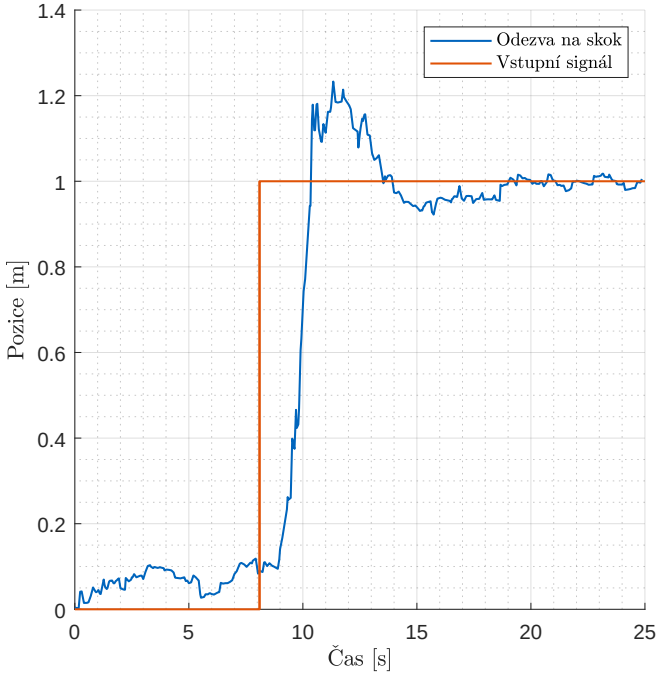
$$\begin{aligned} PID(k) &= P(k) + I(k) + D(k) \\ &= K_P \cdot e(k) + K_I \cdot \sum_{i=0}^{k} (e(i) \cdot \Delta t(i)) + K_D \cdot \frac{e(k) - e(k-1)}{\Delta t(k)} \end{aligned} \quad (6.2)$$

Námi navržený řídící systém využívá čtyři nezávislé PID regulátory v diskrétní formě - tři z nich regulují translaci ve světových souřadnicích, čtvrtý pak natočení dronu vůči počátku světových souřadnic. Každý z regulátorů má vlastní sadu tří konstant zesílení, jejichž hodnotu je potřeba co nejlépe nastavit. Výstupy regulátorů natočení a translace v ose Z není třeba upravovat, mezitím co výstupy regulátorů os X a Y světových souřadnic musejí prvně být otočeny dle natočení dronu vůči počátku. Je důležité připomenout, že dron Ryze Tello využívá jinou definici os souřadnicové soustavy (viz 4.1), před odesláním pokynů je třeba souřadnice náležitě převést. Celý algoritmus byl implementován v uzlu PID_controller_node.

Iterativní úpravou všech konstant zesílení jsme dospěli k hodnotám uvedených v tabulce 6.2. Odezvu regulátoru na skok na vstupu v ose y lze pozorovat na obrázku 6.4. Poloha dronu na obrázku 6.4 je získaná pomocí detekce značek z obrazu dronu (projevuje se zpoždění). Vidíme, že regulátor reaguje velice rychle s překmitem přibližně 20 centimetrů.

Regulátor	K_P	K_I	K_D
PID_X	4/3	0.05	0.20
PID_Y	4/3	0.05	0.20
PID_Z	1.00	0.10	0.10
PID_φ	0.50	0.05	0.10

Tabulka 6.2: Hodnoty PID regulátorů řídícího systému



Obrázek 6.4: Odezva systému na skok na vstupu v ose Y

Kapitola 7

Odometrie a extrapolace polohy dronu

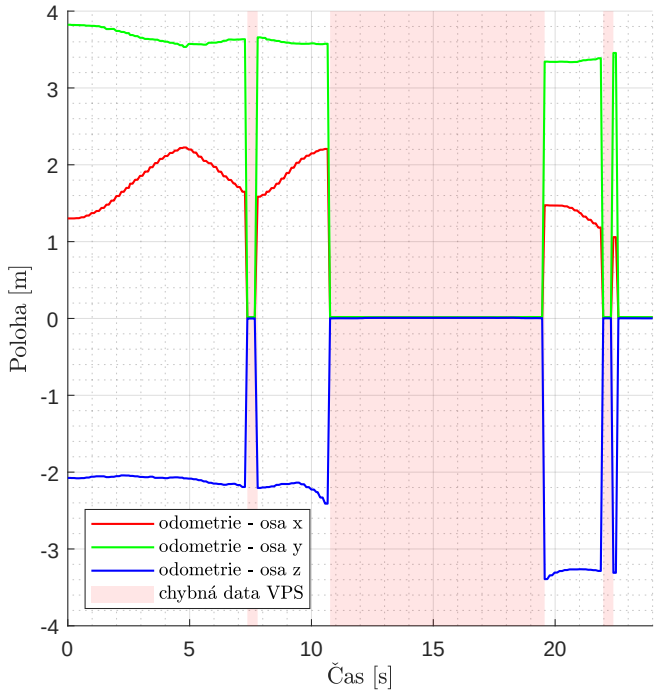
V předchozí kapitole jsme představili způsob řízení letu dronu dle pozice ve známém prostředí pomocí PID regulátoru. PID regulátor ovšem předpokládá okamžitou zpětnou vazbu. V této kapitole se zabýváme kompenzací zpoždění pozice dronu z obrazu pomocí extrapolace polohy z dat vizuálně pozičního systému (VPS).

Data VPS posílá dron Tello řídícímu počítači s frekvencí přibližně 20 Hz. Přestože detekce polohy z obrazu kamery (kapitola 5) může fungovat až na frekvenci 30 Hz, oproti VPS nemá zanedbatelné zpoždění přenosu dat do řídícího počítače. Data VPS přenáší tello_driver v tématu /tello/odom zprávami typu nav_msgs/Odometry.msg .

7.1 Měřená data a jejich zpracování

Uvnitř zprávy tématu tello_driver nalezneme informaci o poloze dronu a jeho rychlostech ve všech šesti stupních volnosti. V naší práci využíváme pouze polohu získanou z odometrie dronu.

Připomeňme část textu 4.1, kde diskutujeme rozdílnost souřadnicové soustavy odometrie dronu oproti definici v REP 103. Dále připomeňme část textu 2.3, kde uvádíme podmínky správného fungování VPS. Nejsou-li splněny tyto podmínky, zprávy odometrie obsahují chybná data (viz obrázek 7.1).



Obrázek 7.1: Poloha dronu z odometrie tématu tello_driver při splnění i nesplnění podmínek správné funkčnosti VPS

Poloha odometrie nevyjadřuje polohu dronu v místnosti, navíc často dochází k driftu a může tak dosahovat velice vysokých kladných i záporných hodnot (až stovky metrů od počátku). Pokud dojde k narušení podmínek funkce VPS, poloha odometrie se přesune na hodnoty do tří centimetrů od počátku (nejedná se o nulové hodnoty, jak by se mohlo zdát z obrázku 7.1). Po obnovení správné funkčností VPS může být dron opět přemístěn až stovky metrů od polohy před narušením funkčnosti VPS. Poznamenejme, že rotační složka polohy odometrie není funkčností VPS nijak ovlivněna, jelikož využívá senzory IMU.

Uzel tf2_odometry_broadcaster, jakožto příjemce tématu /tello/odom, řeší výše zmíněné problémy a odesílá opravené transformace systému tf2 (je hlasatelem transformace world → tello_odom). Uzel přechází na základě funkčnosti VPS mezi dvěma stavy. Data jsou prohlášena za korektní, právě pokud libovolná translační složka je od počátku vzdálena více jak 3 centimetry.

První ze stavů se vykonává pokud jsou data VPS chybná. V tomto stavu uzel odesílá poslední validní translační složku spolu s aktuální rotační složkou.

Při přechodu z prvního do druhého stavu si dron uloží kalibrační translaci,

která umožňuje přesunout nově získanou validní translační složku do předchozí validní translační složky. Díky tomu souřadnicový systém tello_odom spolu s obnovením funkčnosti VPS plynule naváže na předchozí pohyb (v případě chybných dat je díky prvnímu stavu translační složka transformace statická).

Program se nachází v druhém stavu právě když systém VPS funguje korektně. V tomto stavu program odesílá rozdíl mezi translační složkou přijaté polohy odometrie a uloženou kalibrační translační složkou spolu s aktuální rotační složkou.

Uzel tf2_odometry_broadcaster navíc transformuje osy odometrie do tvaru doporučeném REP 103 (viz obrázek 4.1). Uzel nám tak umožňuje spolehlivě využívat souřadnicový systém tello_odom k extarpolaci polohy dronu.

7.2 Extrapolace polohy

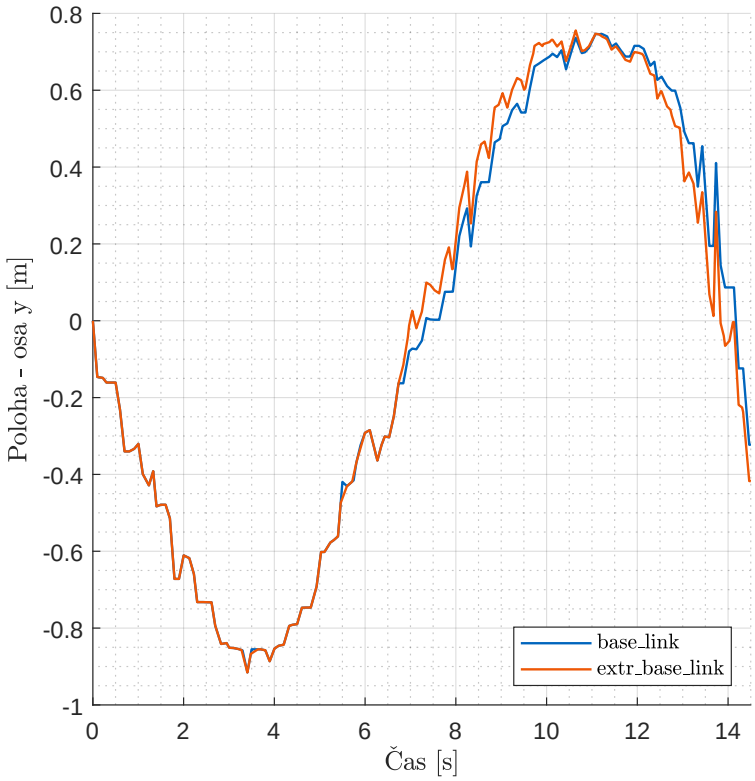
V této části textu se budeme zabývat uzlem tf2_mvo_extrapolator, který na základě zpoždění polohy z obrazu extrapoluje tuto polohu o data odometrie, načež výsledná transformace je odeslána systému tf2 (uzel je hlasatelem transformace world → extr_base_link).

Uzel tf2_mvo_extrapolator je posluchačem transformace world → base_link, ze které získává časový podpis snímku, z něhož byla transformace spočtena. Odečtením aktuálního času systému a tohoto časového podpisu získáváme stáří snímku od jeho dekódování. Součtem této hodnoty s nalezeným koeficientem zpoždění snímku před dekódováním (viz 4.3) získáváme celkové stáří snímku.

Znalost stáří původního snímku, z kterého byla získána poloha souřadnicového systému base_link, umožňuje uzlu tf2_mvo_extrapolator využít systém tf2 k nalezení transformace tello_odom(t_1) → tello_odom(t_2), kde t_1 představuje čas pořízení původního snímku a t_2 aktuální čas.

Získanou transformaci s aktuálním časovým podpisem pak uzel odesílá jako transformaci base_link → extr_base_link. Uzel tf2_mvo_extrapolator nám takto umožňuje extrapolovat polohu získanou z obrazu kamery dronu Tello. Funkčnost tohoto systému lze pozorovat na obrázku 7.2. Na obrázku si lze povšimnout, že systém nejprve přijímá chybná data VPS, posléze dojde k obnovení validních dat a zahájení extrapolace polohy. Extrapolace polohy

úspěšně předpovídá pohyby souřadnicového systému base_link získaného z obrazu kamery dronu.



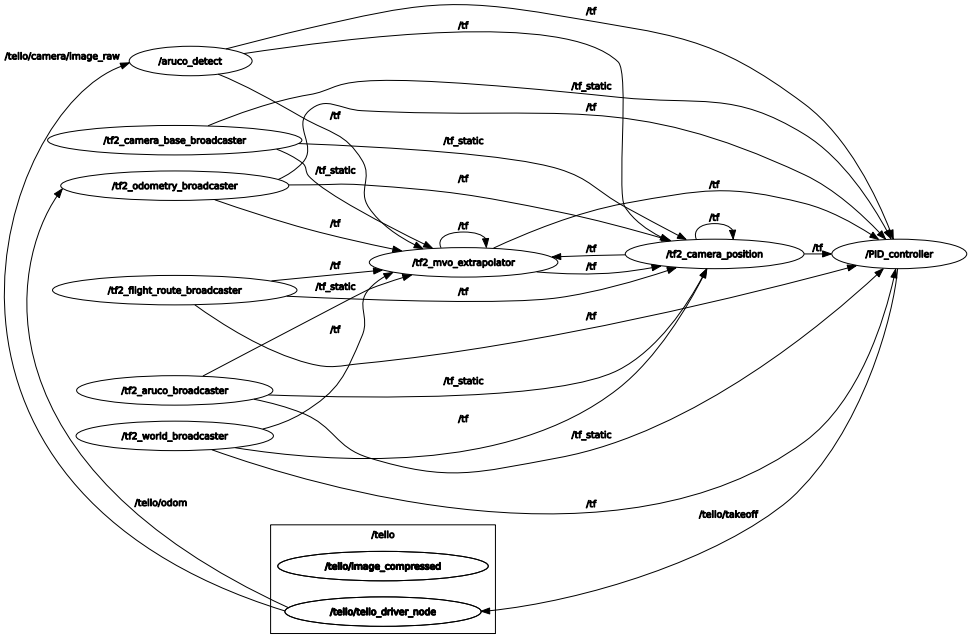
Obrázek 7.2: Průběhy os Y translačních složek souřadnicových systémů base_link a extr_base_link v čase. V čase od 0 sekund do 7 sekund se obě křivky překrývají.

Mimo výše zmíněné, systém extrapolace polohy navíc umožňuje v případě výpadku detekce ArUco kódů z kamery dronu krátkodobě fungovat jako hlavní zdroj polohy dronu (za předpokladu, že jsou splněny podmínky funkčnosti VPS).

Kapitola 8

Implementace a výsledky

V předchozích kapitolách této práce jsme si představili rozsáhlý systém pro řízení dronu Ryze Tello vytvořený v aplikačním rámci ROS. Vykreslíme-li strukturu uzlů tohoto systému pomocí nástroje rqt_graph, získáme graf zobrazený na obrázku 8.1.



Obrázek 8.1: Vizualizace nástroje rqt_graph námi vytvořeného systému na řízení dronu Ryze Tello

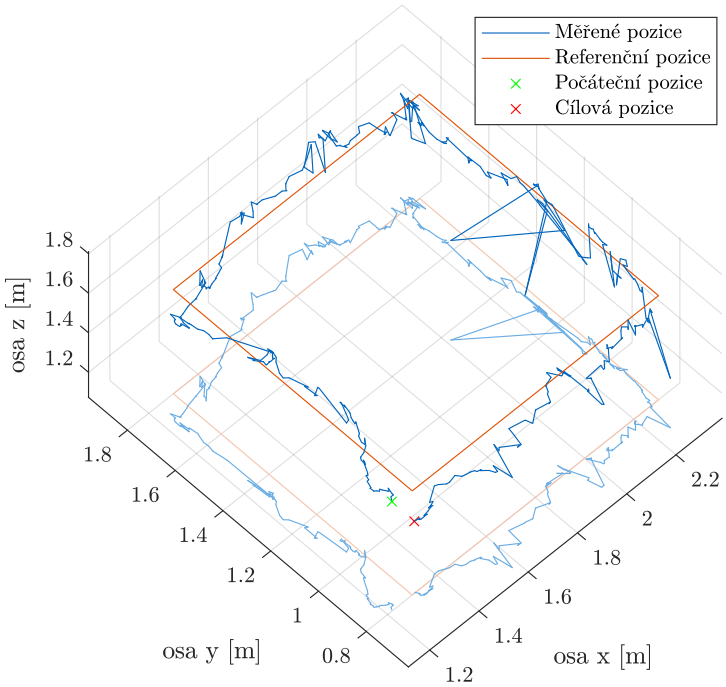
Abychom ověřili funkčnost představeného systému, byla provedena dvě měření autonomního letu dronu. Cílem těchto měření bylo zaznamenat stabilitu

autonomního letu dronu po uživatelem nastavené trajektorii. K vysílání požadované polohy dronu v prostoru využíváme uzel tf2_flight_route_broadcaster, který je hlasatelem transformace world → reference. Posluchačem této transformace je uzel PID_controller_node, který reguluje let dronu Tello tak, aby co nejpřesněji následoval pohyb počátku souřadnicového systému reference.

Uzel tf2_flight_route_broadcaster získává podobu vysílané trasy ze souboru \environment\Flight_route_list.json v námi vytvořeném balíčku. Tento soubor obsahuje seznam všech bodů, kterými souřadnicový systém reference prochází, natočení dronu v těchto bodech a rychlost [m/s], kterou se reference mezi jednotlivými body pohybuje. Uzel tf2_flight_route_broadcaster vysílá aktualizovanou polohu reference padesátkrát za sekundu.

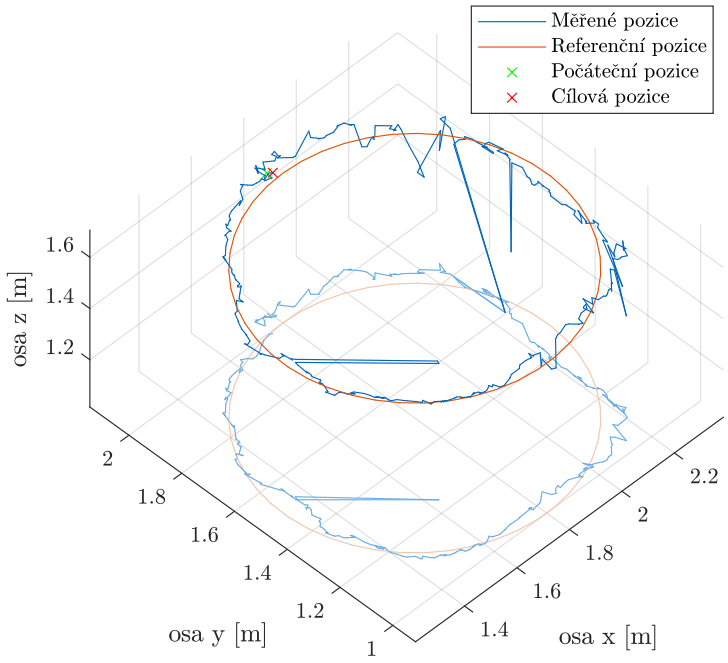
Na obrázcích 8.2 a 8.3 je zobrazen průběh autonomního letu dronu Tello řízený představeným systémem. Záznam letu dronu je získaný z poloh souřadnicového systému base_link, jehož poloha vychází z detekce značek ArUco v obrazu z palubní kamery dronu. Z tohoto důvodu obsahují představené záznamy letu velké množství šumu.

Na prvním obrázku (8.2) je zobrazen záznam letu dronu, při kterém se reference pohybovala po čtvercové trajektorii s délkou strany 1 m a rychlostí 0.1 m/s.



Obrázek 8.2: Záznam letu dronu po čtvercové trajektorii

Na druhém obrázku(8.3) je zobrazen záznam letu dronu, při kterém se reference pohybovala po kruhové trajektorii s poloměrem 1 m a rychlostí 0.1 m/s.



Obrázek 8.3: Záznam letu dronu po kruhové trajektorii

Dron Tello v průběhu obou letů úspěšně sledoval referenci a to s malou odchylkou od požadované trajektorie. Povšimněme si, že systém dokáže stabilně řídit let dronu i přes občasné silné výkyvy polohy souřadnicového systému base_link.

Kapitola 9

Závěr

V průběhu této práce jsme se podrobně seznámili s aplikačním rámcem ROS a dronem Ryze Tello. Představili jsme využívaný způsob komunikace dronu s řídícím počítačem, diskutovali jsme problematiku přenosu videa, jeho dekódování a výpočet zpoždění přijatých snímků.

V kapitole 5 jsme představili způsob detekce polohy dronu z obrazu jeho palubní kamery pomocí značek ArUco. Provedli jsme měření přesnosti polohy získané pomocí značek ArUco a úspěšně navrhli algoritmus zvyšující přesnost detekované polohy díky váženému průměrování ve třech stupních volnosti poloh získaných z jednotlivých značek v daném snímku.

Znalost polohy dronu byla využita v kapitole 6, kde jsme se nejdříve seznámili s možnostmi řízení dronu, poté jsme představili námi zvolený PID regulátor a jeho implementaci v programu. Ukázali jsme reakce dronu na výstup regulátoru a jeho odezvu na skok na vstupu.

Kapitola 7 představila způsob využití odometrie dronu ke kompenzaci zpoždění obrazu z palubní kamery. Byl uveden způsob zpracování dat VPS a IMU obsažených v odometrii dronu. Následně jsme představili program zajišťující extrapolaci polohy ze značek ArUco pomocí dat odometrie.

Nakonec jsme v kapitole 8 ukázali výsledky celého systému, včetně komunikace mezi všemi představenými uzly a provedli měření přesnosti a stability autonomního letu dronu Tello ovládaného námi navrženým řídícím systémem. Z naměřených dat bylo vidět, že řídící systém funguje korektně a dron je

schopný sledovat uživatelem nastavenou trajektorii ve známém prostředí se značkami ArUco. Všechny body zadání této bakalářské práce tak byly splněny.

V práci by bylo možno pokračovat implementací Kalmanova filtru. Ten by umožnil využít více dostupných dat (například data IMU, rychlosti z odometrie či samotné příkazy zasílané dronu tématem /tello/cmd_vel) pro výpočet polohy dronu v prostoru a pravděpodobně by se tak výrazně zvýšila její přesnost. Dále se nabízí možnost nalézt lepší konstanty zesílení využívaného PID regulátoru. Přesnost detekce polohy z ArUco značek by mohla být zlepšena využitím přesnější kalibrace kamery dronu.

Příloha A

Literatura

[1] V. Pritzl, P. Stepan, and M. Saska, "Autonomous flying into buildings in a firefighting scenario," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[2] "Ros.org history." [online]`https://www.ros.org/history/`. cit. 20.4.2021.

[3] T. Foote and M. Purvis, "Standard units of measure and coordinate conventions." [online]`https://www.ros.org/reps/rep-0103.html`. cit. 20.4.2021.

[4] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[5] M. Petrlík, "Onboard localization of an unmanned aerial vehicle in an unknown environment," 05 2018. Diplomová práce, ČVUT.

[6] P. Petráček, V. Krátký, and M. Saska, "Dronument: System for reliable deployment of micro aerial vehicles in dark areas of large historical monuments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2078–2085, 2020.

[7] A. Ahmad, V. Walter, P. Petracek, M. Petrlik, T. Baca, D. Zaitlik, and M. Saska, "Autonomous aerial swarming in gnss-denied environments with high obstacle density," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[8] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Experimental comparison of fiducial markers for pose estimation," in

2020 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 781–789, 2020.

[9] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *Journal of Intelligent & Robotic Systems*, 2014.

[10] "Detection of aruco markers." [online]`https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html`. cit. 20.4.2021.

[11] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing*, vol. 76, 06 2018.

[12] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern Recognition*, vol. 51, 10 2015.

[13] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a quad-rotor robot," in *Proceedings of the 2006 Australasian Conference on Robotics and Automation*, pp. 1–10, The Australian Robotics and Automation Association Inc., 2006.

[14] A. Mahé, C. Pradalier, and M. Geist, "Trajectory-control using deep system identication and model predictive control for drone control under uncertain load.," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 753–758, 2018.

[15] H. Murcia, A. Hernandez, J. Cerquera, C. Copot, and R. KEYSER, "Model predictive path-following control of an ar.drone quadrotor," 10 2014.

[16] L. Langebro and B. Puche Moreno, "Design of control system for uav-based video recording and tracking," 2020. Student Paper.

Příloha B

Přiložené soubory

K této práci jsou přiloženy všechny zdrojové soubory potřebné ke spuštění popisovaného programu. Stromová struktura přiloženého souborového systému je ilustrována níže.

```
src
├── .idea
│   └── ...
├── camera_info_manager_py
│   └── ...
├── fiducials
│   ├── aruco_detect
│   │   └── ...
│   └── ...
├── tello_aruco_demo
│   ├── environment
│   │   ├── ArUco_list.json
│   │   ├── Flight_route_list.json
│   │   └── ...
│   ├── launch
│   │   ├── fly.launch
│   │   └── main.launch
│   ├── meshes
│   │   └── Environment.stl
│   ├── nodes
│   │   ├── PID_controller.py
│   │   ├── rviz_mesh_publisher.py
│   │   ├── tf2_aruco_broadcaster.py
│   │   ├── tf2_camera_base_broadcaster.py
│   │   ├── tf2_camera_position.py
│   │   ├── tf2_flight_route_broadcaster.py
│   │   ├── tf2_mvo_extrapolator.py
│   │   ├── tf2_odometry_broadcaster.py
│   │   └── ...
│   ├── rviz
│   │   └── rviz_configuration.rviz
│   └── ...
├── tello_driver
│   └── ...
└── CMakeLists.txt
```

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pařil** Jméno: **David** Osobní číslo: **483565**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Autonomní řízení dronu Ryze Tello

Název bakalářské práce anglicky:

Autonomous Control of Drone Ryze Tello

Pokyny pro vypracování:

1) Seznamte se s ovládáním dronu Ryze Tello přez rozhraní Wi-fi pomocí Tello SDK, seznamte se se systémem ROS a se systémem ArUco značek.
2) Navrhněte a implementujte systém, kdy se dron Tello bude řídit podle značek, které uvidí.
3) Otestujte možnosti řídicí smyčky s použitím obrazu z kamery přenášené rozhraním Wi-fi na externí počítač.

Seznam doporučené literatury:

[1] Ryze Tello SDK, online: https://www.ryzerobotics.com/tello/downloads
[2] OpenCV ArUco Tutorial, online: https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Petr Štěpán, Ph.D., Multirobotické systémy FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

RNDr. Petr Štěpán, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta