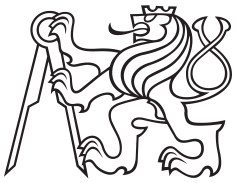
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Mapování prostoru robotickou helikoptérou

Ondřej Masopust

Vedoucí práce: Ing. Jan Chudoba
Studijní program: Kybernetika a robotika
Květen 2021

Poděkování

Chtěl bych tímto poděkovat svojí rodině za veškerou formu podpory během mého studia. Také bych chtěl vyjádřit díky celému sboru pedagogů, kteří dávají svůj drahocenný čas, aby vychovali nové inženýry. Speciálně pak Ing. Janu Chudobovi za poskytnutí možnosti uskutečnit tuto bakalářskou práci a za jeho vedení během vypracovávání. V neposlední řadě bych chtěl poděkovat uživateli „marty cohen“ ze serveru math.stackexchange.com za jeho příspěvek [1], který mi přispěl k nasměrování během vymýšlení důkazu věty 5.1.

[1] `https://math.stackexchange.com/a/54869`

Prohlášení

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 prací.

V Praze dne 20. května 2021

. . . . . .
podpis autora práce

Abstrakt

Tato práce se zabývá rozborem a řešením problému mapování okolního prostředí robotické helikoptéry s využitím dálkového senzoru na palubě. Je proveden rozbor typů senzorů potenciálně vhodných pro tuto aplikaci. Z těch byl po konzultaci s vedoucím práce vybrán laserový dálkoměr rozmítající paprsek v jedné rovině. Dále je pojednáno o možnostech zpracování a uložení naměřených dat. Jako hlavní datová struktura byl vybrán octree, který je implementován ve volně dostupné knihovně OctoMap. Následně je popsán implementovaný softwarový systém v jazyce C/C++. Ten zajišťuje sběr dat ze senzoru a jejich ukládání do octree. Dále poskytuje funkce pro prohledávání naměřených dat za účelem detekce kolize helikoptéry s překážkou na letové trase. Experimenty ukázaly, že softwarové řešení vymezené zadáním splňuje nároky pro nasazení do palubního počítače robotické helikoptéry. Pro zlepšení mapování je na základě experimentů doporučeno vyzkoušet použití 3D lidaru.

Klíčová slova: mapování, UAV, octree, detekce kolize, prevence kolize

Vedoucí práce: Ing. Jan Chudoba
Jugoslávských partyzánů 1580
Praha 6

Abstract

This thesis aims to analyse and work out the problem of mapping the robotic helicopter's environment with an on-board range sensor. Potentially suitable range sensors are theoretically analyzed. A lidar transmitting a laser ray in a 2D plane was chosen after a consultation with the thesis supervisor. Next, an analysis of effective way of processing and storing the sensor data is presented. Octree, which is implemented in n open-source library OctoMap, was chosen as the main data structure. A software system implemented in C/C++ is then described. The software collects range sensor data, stores them in the octree and provides an API for searching the measured data for the sake of collision detection. Experiments showed that the implemented software defined by the assignment is suitable for deployment to the onboard helicopter computer. Furthermore based on the experiments, a 3D lidar is recommended to try for the environment mapping.

Keywords: mapping, UAV, octree, collision detection, collision prevention

Title translation: Environment mapping using a robotic helicopter

Obsah

Obrázky

Grafy

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Masopust** Jméno: **Ondřej** Osobní číslo: **483591**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mapování prostoru robotickou helikoptérou

Název bakalářské práce anglicky:

Environment mapping using a robotic helicopter

Pokyny pro vypracování:

Cílem práce je návrh a implementace software pro mapování prosotru vhodné pro autonomní robotickou helikoptéru. Vytvořená mapa bude sloužit jako struktura pro plánování bezkolizního pohybu helikoptéry v daném prostoru. Lokalizace bude v rámci úlohy řešena motion capture systémem Vicon. Dále je přípustné po domluvě s vedoucím práce přijmout další zjednodušující podmínky úlohy.

- Seznamte se senzory používanými v robotice pro měření vzdáleností k překážkám a základními metodami pro zpracování senzorických dat a reprezentaci prostoru.
- Po konzultaci s vedoucím zvolte vhodnou konfiguraci senzorů a metodu pro mapování prostoru.
- Implementujte navrženou metodu mapování a ověřte její funkci na reálném modelu v laboratoři.
- Navrhněte a implementujte metodu pro zjištění, zda je libovolná trajektorie v prostoru zadaná sekvencí lineárních úseků volná pro průlet helikoptérou.
- Vyhodnoťte funkci metody a diskutujte její možná rozšíření s ohledem na případné omezující podmínky, které bylo nutné přijmout s ohledem na řešitelnost problému v rámci rozsahu bakalářské práce.

Seznam doporučené literatury:

[1] Hornung A., Wurm K.M., Bennewitz M., Stachniss C., and Burgard W., "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees" in Autonomous Robots, 2013; DOI: 10.1007/s10514-012-9321-0.
[2] Meagher, Donald. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, (1980).
[3] Elfes, Alberto. Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception, (2013).
[4] Fisher, Robert & Konolige, Kurt.. Handbook of Robotics Chapter 22-Range Sensors, (2008).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Chudoba, inteligentní a mobilní robotika CIIRC

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

Datum zadání bakalářské práce: **05.01.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Jan Chudoba	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

Kapitola 1
Úvod

Dříve volnočasová aktivita leteckého modelářství se v poslední době přesunula více do veřejného povědomí. Nejvíce pak ve formě dronů či jiných multirotorových helikoptér. Za zmínku stojí např. velmi známá firma DJI či závod „FAI World Drone Racing Championship“.

Helikoptéry se dostaly i do výzkumné oblasti a v poslední době je jejich vývoji věnováno značné úsilí. Drony nacházejí uplatnění např. ve filmovém průmyslu, v zemědělství nebo i jako prostředek pro záchranáře a policisty. S vývojem bezpilotních letounů (UAV), které mají za úkol samostatně se pohybovat v prostředí, je spojena i potřeba mapování jejich okolí pomocí vhodných senzorů. S měřením je pak spojena i úloha efektivního a smysluplného zpracování a uložení dat.

1.1 Související práce

Jednou z prací, kde můžeme vidět uplatnění dronů, je diplomová práce Ing. Petráčka zabývající se dokumentací historických památek pomocí UAV [15]. Petráček zde postupuje tak, že okolí památky je nejprve naskenováno statickým lidarem. Data z lidaru jsou poté použita pro snazší orientaci helikoptéry v prostoru. Úkolem helikoptéry je pak pořídit přesnější 3D sken zvolené památky (mozaiky, sochy...). 3D sken památky je pořízen pomocí dálkoměrů na helikoptéře.

Další prací využívající bezpilotních helikoptér je lokalizace zdroje ionizujícího radioaktivního záření [18]. Autoři uvádějí využití této úlohy např. při události veřejného ohrožení důsledkem radioaktivního záření.

Jednou z prací zabývající se podobným problémem, jaký je adresován v této bakalářské práci, je bakalářská práce Jana Cabiara [4]. Ten se zabývá použitím existujících algoritmů pro 3D rekonstrukci prostředí pomocí stereokamery upevněné na kvadkoptéře.

Tento text navazuje na diplomovou práci Martina Jirouška [10], který sestavil matematický model šestirotorové helikoptéry, který byl následně použit pro stabilizaci helikoptéry ve vzduchu a k usnadnění jejího řízení. Na této helikoptéře budou výsledky této práce testovány.

1.2 Cíl práce

Cílem této bakalářské práce je sestavit programový celek, který zajistí mapování okolí dronu pomocí vhodného senzoru. Implementovaný software nabídne metodu zjištění polohy nebo vzdálenosti nejbližší překážky v zadaném směru. Této funkce bude využito ke korekci letového příkazu, aby dron nenarazil do překážky. Dále bude k dispozici metoda, která ze zadané sekvence bodů zjistí, zda cesta definovaná úsečkami mezi sousedními dvěma body je volná pro průlet, či zda se na cestě objevuje překážka. Bude nutno zvolit vhodný senzor pro skenování okolí a efektivní přístup ke zpracovávání a ukládání naměřených dat.

Kapitola 2
Přehled senzorů

Jedním z prvních úkolů je vybrat správný senzor pro tuto aplikaci. Výběrovou množinu tvoří dálkové senzory měřící vzdálenost pomocí ultrazvuku, rádiových vln, laseru, nebo uzpůsobené kamery. V této kapitole jsou poskytnuty detaily k jednotlivým typům senzorů a pojednání o vhodnosti jejich využití pro skenování okolí hexakoptéry.

2.1 Ultrazvukové senzory

Senzory využívající zvukové vlny se také nazývají sonary. Výlučně se využívá frekvencí nad slyšitelné spektrum. Jde tedy o ultrazvuk. Z hlediska dosahu lze ultrazvukové senzory použít do vzdálenosti jednotek metrů. Dosažitelné rozlišení je v desetinách milimetru [21]. Nebyl nalezen žádný sonar, který by poskytoval skenování ve více jak jedné dimenzi. Pro skenování okolí by tedy musel být zhotoven vlastní systém otáčení senzoru. Z tohoto důvodu jsou sonary nevhodné pro tento typ aplikace. Ultrazvukové senzory by ovšem šlo použít například pro měření výšky letu od země, nebo pro měření v prostoru přímo před helikoptérou pro zvýšení spolehlivosti detekce překážek.

2.2 Radary

Radary měří vzdálenost pomocí rádiových vln. Používají se hlavně v letectví pro monitorování letového provozu. Pro jejich použití je zapotřebí anténa, které dokáže vyslat přesně směrovaný paprsek a poté ho zpětně detekovat. Dosah radarů a jejich odolnost vůči vnějším vlivům, jako je počasí, je velmi uspokojivá. Radary jsou především používány na větších strojích. Moduly radaru použitelné pro roboty také existují. Možno zmínit například modul SRD-D1 od firmy Ainstein. Jeho nevýhodou ovšem je rozlišení pouze 0.6 m a úhlový rozsah $\pm 30°$. Tento modul by byl dobrým kandidátem pro automatickou detekci kolize, ovšem bez ukládání dat o okolí. Vhodný radar pro kontinuální mapování okolí nebyl nalezen.

2.3 Lidary

Lidary jsou založené na měření vzdálenosti pomocí světelných paprsků. Jsou zřejmě nejrozšířenější v robotických aplikacích mapování okolí. Pro měření vzdálenosti lze využít buď triangulace, kdy se měří úhel, pod kterým dopadl odražený paprsek na detektor, nebo metody měření času letu (ToF). Metoda ToF je vhodnější pro měření delších vzdáleností, zatímco metodu triangulace lze použít i pro měření vzdáleností kratších, které metodou ToF měřit nelze z důvodu nedostatečného časového rozlišení použitelné technologie. Další možností je využití modulace světelného paprsku.

Dosah lidaru může sahat do desítek metrů. Rozlišení v měření vzdálenosti pak může být v jednotkách centimetrů (například LIDAR-Lite v3 od firmy Garmin).

Přesné směrování světelného paprsku není pro současné technologie problém. Další výhodou použití světla je i možnost celkem snadného rozmítání měřícího paprsku do více směrů. K lze použít například rotující zrcadlo. S tímto přístupem lze dosáhnou velmi dobrého rozlišení v úhlu. Například lidar TiM581 od firmy SICK údajně vykazuje úhlové rozlišení 0.33°.

Lidary se vyrábí v různých provedeních lišících se mírou záběru okolního prostředí. Jednak se vyrábí lidary měřící pouze v jednom směru (například již zmíněný LIDAR-Lite v3). Poté existují lidary, které rozmítají paprsek postupně v jedné rovině (například Hokuyo URG-04LX-UG01)[1]. Existují i senzory, které rozmítají paprsky ve více rovinách. Jejich záběr je pak obohacen o další dimenzi (viz například MRS1000 od firmy SICK). Stále se ale nejedná o měření v plném prostorovém úhlu. Lidary měřící v plném prostorovém úhlu také existují. Jako příklad je možno uvést RobotEye RE05 od firmy Ocular Robotics.

Nevýhodou lidarů může být jejich závislost na osvětlení okolního prostředí. Lidary mohou mít problémy s měřením, pokud jsou v prostředí přímého slunečního záření. Můžeme tedy říci, že pokud chceme robustní senzor, který se vyrovná s různými světelnými podmínkami, lidar nemusí být tou nejlepší možností.

2.4 Kamery

Pro měření hloubkové struktury okolní scény je možné využít i kamer. Jednou z možností je sestava nazvaná jako stereo kamera. Jedná se o dvojici kamer mírně od sebe vzdálených směřujících v podobném směru na scénu. Jde o imitaci procesu, který mozek provádí s obrazy ze dvou očí. Princip funkce je založen na triangulaci, kdy ze známé vzdálenosti kamer a známého úhlu, pod kterým obě kamery vidí daný objekt, je spočtena vzdálenost objektu od kamer. S tím je spojeno řešení problému nalezení odpovídajících si bodů v obrazech z obou kamer. Ke zjednodušení tohoto problému se používá

[1] Existují metody, jak využít 2D senzor pro mapování 3D prostoru. Více je diskutováno v sekci 2.5.

epipolární geometrie, díky které stačí prohledat v druhém obrazu menší množinu pixelů za účelem nalezení korespondujícího bodu. Tato metoda nemá zvláštní nároky na osvětlení a je v běžném prostředí nejuniverzálnější.

Další možnost, která zde bude uvedena, je použití strukturovaného osvětlení. Osvětlí-li se scéna světlem s různou intenzitou v různých oblastech, lze z obrazu jedné kamery usoudit na hloubkový profil okolní scény. Nevýhodou této metody je zmíněná potřeba zdroje strukturovaného osvětlení (SO). Tedy pro použití pro kontinuální mapování okolí helikoptéry by bylo nutné přijmout předpoklad, že okolní scéna je osvětlena okolními zdroji s dostatečně malou intenzitou, aby osvětlení z palubního zdroje SO bylo v obrazu kamery zřetelné. Další možností je přijmout předpoklad, že by zdroj SO byl umístěn mimo palubu helikoptéry. Tím by mohlo dojít ke zvýšení intenzity zdroje SO a tedy snížení nároku na nízkou intenzitu ostatních rušivých zdrojů. Vhodným prostředím pro použití této metody by mohly být např. podzemní prostory, ve kterých se odehrávají soutěže „DARPA Subterranean Challenge“.

Obecnou nevýhodou použití kamer je jejich omezený zorný úhel. Pokud je požadavek na mapování okolí helikoptéry ve všech směrech, bylo by nutné vytvořit systém otáčení kamer. Z důvodu zamezení rozmazanosti obrazů z kamer by maximální rychlost otáčení pravděpodobně nebyla dostatečná pro kontinuální mapování celého prostoru. V opačném případě by bylo nutné použít kamery s vysokou rychlostí pořizování snímků.

2.5 Způsob využití 2D senzoru pro mapování prostoru

Pro skenování prostoru se většinou využívají lidary rozmítající paprsky do tří dimenzí. Mapování prostoru lze ale provádět i s 2D lidarem. V této části budou představeny možné přístupy.

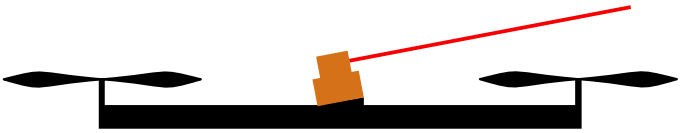
2.5.1 Natáčení senzoru

Prvním navrhovaným řešením je připevnit senzor na otočnou plošinu a pohybovat s ní takovým způsobem, aby se měnilo natočení roviny, ve které senzor měří. Při použití této metody by bylo nutné zajistit měření natočení senzoru tak, aby přijatá měření byla uložena na správné pozice. Nevýhodou tohoto přístupu je přidaná složitost spočívající v konstrukci takové plošiny a zároveň i její přidaná hmotnost. Pro účely použití na dronu je ve většině případů cílem dosáhnout co nejnižší letové hmotnosti. Dalším aspektem je zvýšení odběru z palubního zdroje napětí z důvodu přidaných akčních členů, které polohují plošinu.

2.5.2 Senzor se stálou polohou

Další možností je nechat senzor nepohyblivě připevněný k helikoptéře. Pak se nabízejí dvě možné konfigurace.

Obrázek 2.1: Konfigurace senzoru rovnoběžně s UAV



Obrázek 2.2: Konfigurace nakloněného senzoru

První konfigurace uvažuje senzor připevněný takovým způsobem, že rovina měření je rovnoběžná s rovinou vymezenou rotory helikoptéry (viz obrázek 2.1). Pro zmapování okolí by pak helikoptéra musela udělat vertikální přelet, který by zajistil sejmutí hloubkového profilu ve všech potřebných výškových hladinách. Nevýhodou tohoto přístupu je fakt, že nelze měřit prostor přímo pod a nad helikoptérou. Informace o poloze stropu nebo podlahy je ovšem pro provedení vertikálního přeletu zásadní. Bez ní nelze určit, jak vysoko může UAV vyletět. Řešením by mohlo být přidání dalších 1D senzorů směřujících do prostoru nad a pod helikoptéru. Tyto senzory by poskytovaly informaci o možnosti vertikálního pohybu. Pokud se helikoptéra dostane do oblasti, která není zmapovaná, musí zastavit horizontální pohyb, aby byl prostor zmapován. Tento fakt se také jeví jako nevýhoda, pokud je zásadní rychlost pohybu.

Další možností, která nevyžaduje vertikální manévr, je připevnit senzor nakloněný vůči rovině rotorů, jak je zobrazeno na obrázku 2.2 Pokud tedy bude helikoptéra ve vodorovné poloze, rovina paprsku rozmítaná senzorem bude vůči vodorovině nakloněná. Následným manévrem ve formě jedné otočky okolo vertikální osy se dosáhne zmapování části okolního tří-dimenzionálního prostoru do určité výšky nad a pod helikoptérou. Zajištění kontinuálního mapování prostoru před helikoptérou během letu vpřed nebo vzad je možné dosáhnou tím, že naklonění senzoru vůči helikoptéře bude provedeno např. na pravém boku helikoptéry. To znamená, že paprsky snímající prostor v největší výšce budou na pravém straně a paprsky snímající prostor v nejmenší výšce na straně levé. Paprsky směřující vpřed a vzad budou mít pak úhel náklonu shodný s úhlem náklonu helikoptéry. Nevýhodou tohoto přístupu je nutnost provedení výše popsaného manévru pro získání nového 3D skenu. Tento manévr je ovšem možné provést i za letu. Ani tato metoda neposkytuje mapování prostoru přímo nad a pod helikoptérou. Mapovací manévr ovšem tuto informaci nepotřebuje.

Kapitola 3

Zpracování dat

Měřením hloubkového profilu okolního prostředí je získáváno velké množství dat. Tato data je zapotřebí efektivně uložit tak, aby bylo docíleno snížení nutného úložného prostoru a rychlého vyhledávání. Tato kapitola pojednává o možnostech komprese naměřených dat, jejich filtrování a způsobech reprezentace obsazenosti reálného prostoru.

3.1 Komprese dat

Data ze senzoru jsou obdržena většinou ve formě seznamu bodů. V případě 2D lidarů se může jednat o seznam vzdáleností, které senzor naměřil, pro jednotlivé úhlové polohy. Uložení těchto dat většinou znamená zanesení záznamu do mřížky reprezentující diskretizovaný model okolního prostoru. Každé buňce mřížky se běžně říká voxel[1]. V této části bude pojednáno o možnostech komprese takové mřížky.

3.1.1 Poziční komprese

Jednou z možností komprese je pamatovat si pouze strukturu okolí v nějakém definovaném okruhu okolo UAV. Pokud tedy helikoptéra odletí daleko z pozice, ze které vzlétla, struktura prostoru okolo místa vzletu je zapomenuta. Pokud je implementováno měření prostoru v plném prostorovém úhlu, může tento přístup znamenat pamatování pouze několika posledních měření a stará měření zapomínat. Pokud pořízení 3D skenu okolí vyžaduje nějaký manévr robota (například z důvodu použití 2D lidaru), může být metoda zapomínání nevýhodná. Robot by musel často vykonávat skenovací manévr, což by mohlo být zdržením v jeho úkolu. Na druhou stranu v oblasti automatického řízení automobilů je tato metoda vhodná, jelikož automobil se většinou pohybuje směrem vpřed a nevrací se na dříve navštívené pozice.

3.1.2 Použití kompresní datové struktury

Další možností komprese je použití tzv. KD stromů. Jak je již patrné z názvu, jde o reprezentaci prostoru pomocí datové struktury známé jako strom. KD

[1] z anglického „volume pixel“

stromy poskytují reprezentaci prostoru libovolné dimenze. Jejich smyslem je rozdělení prostoru na vnořené k-dimenzionální krychle. Postup od kořene stromu k jeho listům odpovídá vnořování se do menších krychlí. Pokud nastane situace, že pro nějaký uzel mají všichni potomci (všechny vnořené krychle) stejný stav (např. plno/prázdno), může dojít k prořezání stromu. Tedy není potřeba ukládat v paměti potomky tohoto uzlu a daný uzel se stane listem.

Octree

Speciálním případem KD stromu je tzv. octree [12]. Jde o strom používaný k reprezentaci 3D prostoru. Každý uzel, který není listem, má právě osm potomků, kteří reprezentují osm vnořených krychlí o poloviční délce hrany, než je délka hrany jejich rodiče. K významné kompresi dochází, nachází-li se v datech souvislý objekt velkých rozměrů. Jemu vepsaná krychle je pak reprezentována jediným uzlem ve stromu.

Vyskytuje se ovšem jeden problém s použitím komprese senzorových dat pomocí octree. Dálkoměry měří pouze body na povrchu objektů. Vnitřní body v objemu skenovaných těles nebudou označeny jako obsazené, jelikož nejsou pro senzor vzdálenosti viditelné. Objemová komprese tedy v tomto případě není ideální. Tento problém je ovšem vyvážen skutečností, že dojde ke značné kompresi volného prostoru, ve kterém se helikoptéra pohybuje.

Komprese povrchových dat

S problémem skenování povrchu se vypořádávají výškové mapy[2]. Ty jsou reprezentovány 2D mřížkou, kde každá buňka mřížky obsahuje informaci o výšce povrchu v daném místě. Tento přístup je vhodný zejména pro pozemní roboty pohybující se ve specifickém prostředí, kde se nenacházejí oblasti typu podjezd pod mostem. Jakmile se totiž v prostředí objeví oblast, ve které by bylo potřeba zaznamenat více výškových profilů v jedné buňce[3], tento přístup selhává.

S řešením přišel Triebel a spol. [20], který navrhuje použití tzv. víceúrovňových povrchových map. Každá buňka 2D mřížky je zde seznamem objektů reprezentujících překážky v různých výškách. Podobným řešením jako víceúrovňové povrchové mapy je i tzv. SkiMap [5]. Hlavní datovou strukturou, kterou SkiMap využívá, je Skip list, který představil Pugh [16].

3.2 Metody vkládání dat

Jednou z možností vkládání nových dat do datové struktury je pouze aktualizovat hodnotu voxelu odpovídajícího koncovému bodu měřicího paprsku. Do paměti jsou tedy přidávána pouze nově zjištěná data o obsazených voxelech. Tento způsob je časově nenáročný. Jeho nevýhodou ovšem je skutečnost, že

[2] anglicky „elevation maps“

[3] např. prostor pod mostem a na mostě

pohybující se objekt (chodec, automobil...), zanechá v mapě trvalý záznam, který v čase přetrvá.

Řešením tohoto problému je vkládání nově naměřených informací takovým způsobem, že mimo aktualizace koncového bodu jsou aktualizovány všechny body na trase, po které měřící paprsek letěl. Pro trajektorii mezi senzorem a koncovým bodem lze totiž zřejmě udělat předpoklad, že není obsazená. Takto lze aktualizovat okolní scénu, která se mění dynamicky.

3.3 Filtrování naměřených dat

Senzorová data mohou obsahovat měření, která neodpovídají realitě. Taková měření lze poznat např. podle toho, že vykazují značnou odchylku vůči sousedním naměřeným datům. Pro odstranění těchto parazitních jevů je možné senzorová data filtrovat. Za tímto účelem je spočítán rozdíl naměřených vzdáleností s oběma sousedními body. Pokud jsou oba rozdíly v absolutní hodnotě větší, než je stanovená mez, je velmi pravděpodobné, že nastala chyba v měření. Takový bod pak není uložen.

3.4 Metody reprezentace obsazenosti voxelu

Zřejmou možností je ukládat ve voxelu pouze ternární informaci o stavu, zda je objem obsazený nějakým reálným objektem, zda je volný, nebo zda o jeho stavu není zatím nic známo. Tento přístup disponuje tou výhodou, že k uložení takto jednoduché informace stačí velmi málo paměti. S tímto přístupem má každé měření velký vliv na výsledný stav voxelu. To je nevýhodou z toho důvodu, že reálné senzory vykazují šum měření. Tento šum se pak výrazně projeví v uložených datech.

3.4.1 Pravděpodobnostní přístup

Tomuto lze předejít použitím pravděpodobnostního přístupu [6]. Ze znalosti přesnosti měření lze dopředu stanovit, jak velký vliv na změnu stavu voxelu bude mít každá aktualizace (neboli každé vložení nového měření). Velké množství aktualizací stejné povahy[4] bude mít za následek, že následná chyba měření bude mít pouze malý vliv na stav voxelu. Takto lze měření zpřesnit a eliminovat šum.

Pravděpodobnostní přístup neřeší pouze chyby měření způsobené senzorem vzdálenosti. Je známo, že gyroskopické senzory založené na měření úhlových rychlostí vykazují značné chyby v měření a vlivem integrace může docházet k chybným hodnotám úhlu náklonu. Pokud tedy dron využívá k získávání dat o naklonění některý z takových senzorů, pravděpodobnostní přístup může tyto chyby do jisté míry potlačit.

[4] např. přijetí informace o obsazenosti daného bodu

Kapitola 4
Použité prostředky

V této kapitole jsou popsány technické prostředky použité k vyřešení zadané úlohy.

4.1 Lokalizační systém Vicon

Lokalizace helikoptéry je v této práci řešena systémem[1] Vicon [2]. Systém se skládá z kamer snímajících polohu lokalizovaného objektu a softwaru, který vyhodnocuje obraz z kamer. Data jsou z pozičního systému získávána přes síťový protokol TCP nebo UDP. Systém má zjednodušit úlohu za účelem poskytnutí možnosti zaměřit se na mapování okolí. Úloha autonomní lokalizace je předmětem dalšího vývoje. Více je popsáno v části 7.2.1.

Systém Vicon používá k lokalizaci odrazivé kuličky přilepené na měřený objekt. Podoba tohoto systému v laboratoři výzkumné skupiny Intelligent and Mobile Robotics Group (IMR)[2] je zobrazena na obrázku 4.1.

Každá z kamer dodává informaci o přímce, na které se sledovaný objekt nachází. Jedním z možných řešení lokalizace pomocí více kamer je vyřešit optimalizační úlohu nalezení průsečíku daných přímek. Za účelem získání informace o natočení objektu je nutné mít takové měřené body alespoň tři na sledovaném objektu. Jakým způsobem přesně systém Vicon vyhodnocuje pozici měřených kuliček nebylo nalezeno.

4.2 Knihovna OctoMap

Se souhlasem vedoucího práce byla pro reprezentaci a kompresi naměřených dat zvolena datová struktura octree. Jedny z dostupných knihoven, které implementují ukládání dat pomocí octree, jsou PCL [19] a OctoMap [9]. Bylo zjištěno, že knihovna PCL nabízí strukturu octree pouze pro kompresi za účelem přesunu dat a nepodporuje dynamické přidávání nových měření. Z toho důvodu byla použita knihovna OctoMap.

Přidávání nových měření v knihovně OctoMap je možno provést jak formou aktualizace pouze koncového bodu, tak i aktualizací všech bodů na trase

[1] anglicky nazýváno „motion capture system“

[2] Jde o výzkumnou skupinu vedoucího této práce.

paprsku. Standardně je nastaven limit 16 vrstev stromu, do kterého se data ukládají. Při použitém rozlišení 5 cm lze tedy takto zmapovat prostor ve tvaru krychle o délce hrany přes 1.5 km. Limit 16 vrstev lze změnit, jelikož zdrojový kód je volně přístupný. Pro reprezentaci obsazenosti voxelu je využit pravděpodobnostní přístup. Konkrétně jde o použití logaritmických šancí[3], které přinášejí tu výhodu, že aktualizace obsazenosti voxelu využívá pouze operace sčítání a odčítání. Bližší informace jsou popsány u Hornunga [9].

Součástí projektu OctoMap je i vizualizační program Octovis. Ten umožňuje zobrazit naměřená data v jednotlivých vrstvách stromu.

4.3 Senzorické vybavení

Po konzultaci s vedoucím práce byla zvolena senzorová konfigurace v podobě jednoho dálkoměru Hokuyo URG-04LX-UG01 [17]. Senzor je zobrazen na obrázku 4.2. Hlavním důvodem výběru tohoto senzoru byla jeho okamžitá dostupnost v laboratoři skupiny IMR. Tento senzor disponuje měřením v jedné rovině s úhlem otevření 240°. Výrobce uvádí přesnost ±30 mm pro měření do 1 m. Na větší vzdálenosti garantuje přesnost ±3%. Maximální měřená vzdálenost je 5.6 m. Pořídit jeden sken trvá senzoru údajně 100 ms. Data a napájení jsou poskytnuta přes sběrnici USB.

Významným omezením při použití tohoto senzoru je, jak výrobce uvádí, použití pouze ve vnitřních prostorech. Při použití za přímého slunečního světla může docházet k chybám měření.

[3] anglicky „log odds“

Obrázek 4.1: Lokalizační systém Vicon v testovací laboratoři a detekční kuličky

Obrázek 4.2: Použitý senzor Hokuyo

Kapitola 5

Implementace řešení

V této kapitole je popsáno algoritmické řešení problému mapování okolí a detekce překážek s využitím knihovny OctoMap.

5.1 Návrh struktury programu

Celkové schéma implementovaného softwarového systému je zobrazeno na obrázku 5.1. Hlavním blokem, který spojuje jednotlivé funkce, je třída „Octree Manager“. Ta také poskytuje konečné API pro komunikaci s řídícím systémem helikoptéry. Získávání dat zajišťuje vlákno „Range sensor Data Manager“, které přijímá data ze senzoru přes USB. Iterátory napojené na blok „Obstacle Detection OcTree“ vykonávají prohledávání v uložených datech. Veřejné metody třídy „Obstacle Detection OcTree“ dávají možnost přistupovat k těmto iterátorům na vyšší úrovni abstrakce. V následujících částech je uveden podrobnější popis jednotlivých komponent naimplementovaného systému.

5.2 Třída ObstDetOcTree

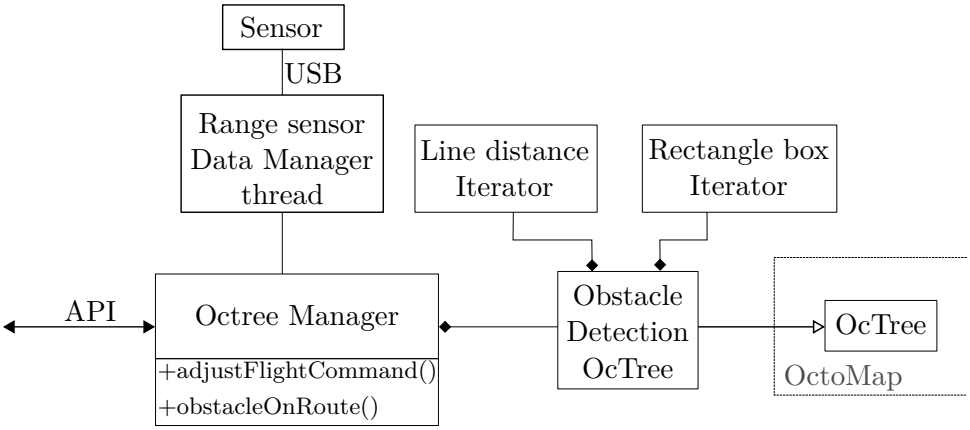
Jak již bylo naznačeno, zkratkový název je odvozen ze spojení „Obstacle Detection OcTree“, což lze volně přeložit jako “OcTree určený pro detekci překážek”. Tato třída rozšiřuje třídu OcTree o funkce vyhledávání v uložených datech. Jedná se o metody hledání nejbližšího obsazeného voxelu v určitém ohraničeném prostoru. Těmito prostory mohou být

- koule,
- válec, nebo
- kvádr.

Detaily jsou poskytnuty v následujících podsekcích.

5.2.1 Detekce překážky v okolí helikoptéry

Jak již bylo zmíněno, třída ObstDetOcTree poskytuje metodu pro prohledávání prostoru vymezeného povrchem koule. Tato metoda byla nazvána



Obrázek 5.1: Diagram softwarového řešení

`getNearestPoint` a vrací souřadnice obsazeného uzlu (voxelu), který je nejblíže středu prohledávané koule. Pokud neexistuje překážka v prohledávané kouli, metoda vrátí souřadnice (NAN, NAN, NAN).

Knihovna OctoMap poskytuje iterátor `leaf_bbx_iterator`, který iteruje přes krychli zarovnanou se souřadnicovými osami. Toho je využito při prohledávání koule. Iteruje se tedy přes krychli, ovšem jen uzly, které patří do koule, jsou brány v potaz. Z těch je vybrán obsazený uzel, který je nejblíže středu koule.

Tuto metodu lze použít k detekci překážky překážky v okolí helikoptéry ve všech směrech. Detekce překážky v konkrétním směru je popsána v sekci 5.2.2.

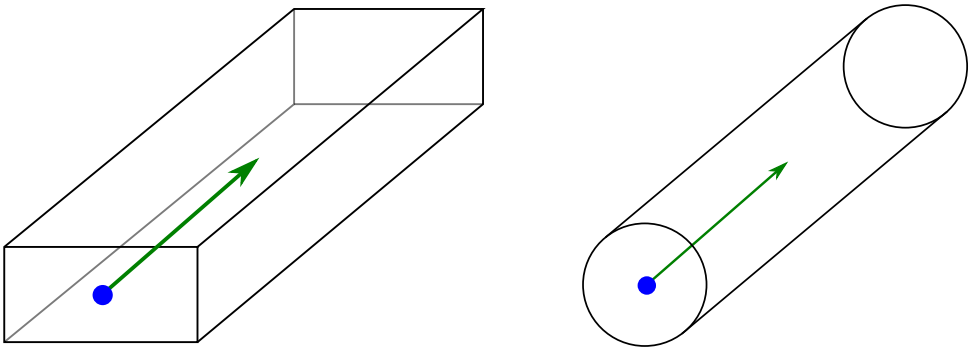
5.2.2 Detekce překážky v zadaném směru

Jak již bylo naznačeno, třída ObstDetOcTree disponuje také metodou nazvanou `getNearestObstacleDistance`. Ta provádí hledání překážky v zadaném směru. Toho lze využít zejména pro prevenci kolize během přeletu helikoptéry. Zároveň je na této metodě postavena funkce prohledávání zadané letové trasy.

K tomuto problému bylo přistoupeno dvěma různými přístupy. První přístup prohledává objem pomyslného kvádru. Druhý přístup prohledává objem válce. Nejdříve budou uvedeny jejich společné rysy.

Oba přístupy sdílí podobné inicializační parametry. Prvním z nich je pozice. Ta určuje umístění prohledávaného objektu v prostoru. Tento bod je poté interpretován jako střed jedné z podstav (resp. stěn) válce (resp. kvádru). Tento bod je také brán jako počátek směrového vektoru, který určuje směr prohledávání. Výše popsané je znázorněno na obrázcích 5.2a a 5.2b. Dalšími inicializačními parametry jsou rozměry objektů, tedy vzdálenost, ve které se prohledává a poté poloměr (resp. délky hran) podstav.

Pro oba přístupy prohledávání byly vytvořeny vlastní iterátory. Během implementace byla čerpána inspirace z iterátorů, které obsahuje knihovna OctoMap. Funkce obou dvou je založena na prohledávání stromu do hloubky. Jako pomocná datová struktura je tedy použit zásobník. Rozdíl těchto iterátorů je v podmínce, která vyhodnocuje, zda uložit potomka uzlu na zásobník,



(a) : Prohledávání kvádru

(b) : Prohledávání válce

Obrázek 5.2: Ilustrace k popisu hledání překážky v zadaném směru. Modrý bod reprezentuje zadanou pozici, zelená šipka pak směr prohledávání.

nebo ne. Tyto rozdíly jsou popsány v následujících podsekcích.

Prohledávání kvádru

V části 5.2.1 byl zmíněn `leaf_bbx_iterator`. Ten zde nelze použít, jelikož poskytuje pouze prohledávání krychle, jejíž hrany jsou zarovnané se souřadnicovými osami mapovaného prostředí. Proto byl implementován vlastní `RectangleBoxIterator`, který zprostředkovává prohledávání libovolně orientovaného kvádru.

Rozhodnutí, zda se aktuálně navštívený uzel přidá na zásobník pro další prohledávání, je zpracováno na základě podmínky, zda zasahuje svým objemem do prohledávaného kvádru. Za účelem zpracování této podmínky je uvedena následující definice a věta.

Definice 5.1. Konvexní mnohostěn je průnik konečně mnoha poloprostorů.

Poznámka 5.2. Kvádr je konvexní mnohostěn.

Věta 5.1. *Nechť $K_1, K_2 \subset \mathbb{R}^3$ jsou uzavřené množiny ve tvaru kvádru obsahující svůj vnitřní objem. Dále nechť h_i^1 (resp. h_i^2), $i \in \{1, 2, \ldots, 12\}$ jsou úsečky reprezentující hrany kvádru K_1 (resp. K_2). $K_1 \cap K_2 \neq \emptyset$, právě když existuje i takové, že $h_i^1 \cap K_2 \neq \emptyset$ nebo $h_i^2 \cap K_1 \neq \emptyset$.*

Důkaz. $\Rightarrow$:
Pokud platí $K_1 \cap K_2 \neq \emptyset$, pak dimenze $dim(K_1 \cap K_2)$ může nabývat hodnot z množiny $\{0, 1, 2, 3\}$.

- $dim(K_1 \cap K_2) = 0$ platí v případě, že průnikem je jediný bod. Tato situace nastane, pokud jeden z kvádrů přispívá do průniku pouze jedním svým vrcholem, nebo jde pouze o průnik hran v jediném bodě. V obou případech ale platí $K_1 \cap K_2 \in h_i^1$ nebo $K_1 \cap K_2 \in h_i^2$ pro nějaké i. Tedy závěr implikace je pravdivý.

- $dim(K_1 \cap K_2) = 1$ platí v případě, že průnikem je úsečka. To nastane v situaci, kdy jeden z kvádrů do průniku přispívá pouze svou hranou, nebo

její částí. I z toho ale okamžitě plyne, že $K_1 \cap K_2 \subseteq h_i^1$ nebo $K_1 \cap K_2 \subseteq h_i^2$ pro nějaké i. Tedy závěr implikace je pravdivý.

- $dim(K_1 \cap K_2) = 2$ nastane v případě, že kvádry spolu sdílejí část své stěny, nebo celou stěnu. Průnik dvou stěn obsahuje alespoň část jedné z hran jedné stěny. Tedy i v tomto případě platí $K_1 \cap K_2 \cap h_i^1 \neq \emptyset$ nebo $K_1 \cap K_2 \cap h_i^2 \neq \emptyset$ pro nějaké i.

- $dim(K_1 \cap K_2) = 3$ nastane ve všech ostatních případech. Průnikem je pak konvexní mnohostěn. Konvexní mnohostěn je dle definice 5.1 tvořen poloprostory. Tyto poloprostory jsou ohraničeny rovinami (v případě $\mathbb{R}^3$). Průniky těchto rovin tvoří hrany vzniklého mnohostěnu. Hrana nově vzniklého mnohostěnu může být tvořena původní hranou jednoho z kvádrů, nebo může být vytvořena nová hrana průnikem stěn obou kvádrů. V každém případě je ale alespoň jedna z hran vzniklého mnohostěnu tvořena původní hranou jednoho z kvádrů, nebo částí této hrany. Tím ale nabývá pravdivosti výrok $K_1 \cap K_2 \cap h_i^1 \neq \emptyset$ nebo $K_1 \cap K_2 \cap h_i^2 \neq \emptyset$ pro nějaké i.

$\Leftarrow$:
Platí $h_i^1 \subset K_1$, tedy i $K_1 \cap K_2 \neq \emptyset$ (identicky pro h_i^2) [1] □

Tedy pro zjištění, zda uzel stromu zasahuje svým objemem do prohledávaného kvádru, stačí otestovat, zda jedna z hran prohledávaného kvádru zasahuje do objemu uzlu, nebo zda jedna z hran uzlu zasahuje do objemu prohledávaného kvádru.

Pro implementaci tohoto testu do kódu bude brána parametrická rovnice úsečky

$$\begin{aligned} x &= p_1 + t(q_1 - p_1) \\ y &= p_2 + t(q_2 - p_2) \\ z &= p_3 + t(q_3 - p_3), \end{aligned} \tag{5.1}$$

kde (x, y, z) jsou souřadnice bodů úsečky, $\boldsymbol{p} = (p_1, p_2, p_3)$ je počáteční bod úsečky, $\boldsymbol{q} = (q_1, q_2, q_3)$ je koncový bod úsečky a $t \in \langle 0, 1 \rangle$ je proměnný parametr. Dále pak dvě rovnice poloprostorů v $\mathbb{R}^3$

$$\begin{aligned} n_1 x + n_2 y + n_3 z + d &\leq 0 \\ n_1 x + n_2 y + n_3 z + d &\geq 0, \end{aligned} \tag{5.2}$$

kde $\vec{n} = (n_1, n_2, n_3)$ je normálový vektor hraniční roviny poloprostoru a d je parametr určující umístění hraniční roviny v prostoru.

Prvním případem je testování hran uzlu, zda leží v prohledávaném kvádru. Jelikož hrany uzlu jsou rovnoběžné se souřadnicovými osami, zjednoduší se jejich parametrický zápis a dvě souřadnice budou konstantní. Např. pro

[1] Kontrola důkazu provedena Doc. RNDr. Jaroslavem Tišerem, CSc. (Katedra matematiky, FEL, ČVUT)

úsečku (hranu) rovnoběžnou s osou x by rovnice vypadaly

$$\begin{aligned} x &= p_1 + t(q_1 - p_1) \\ y &= p_2 \\ z &= p_3. \end{aligned} \tag{5.3}$$

Nyní bude odvozen vzorec, který rozliší, zda taková úsečka zasahuje do části prostoru vymezeného dvěma rovnoběžnými rovinami. Tuto část prostoru lze popsat nerovnicí

$$-d_l \leq n_1 x + n_2 y + n_3 z \leq -d_u. \tag{5.4}$$

Můžeme předpokládat, že $-d_l \leq -d_u$, protože objem prohledávaného kvádru je vždy nenulový. Do nerovnice (5.4) budou dosazeny rovnice úsečky (5.3) a výraz bude upraven.

$$\begin{aligned} -d_l &\leq n_1(p_1 + t(q_1 - p_1)) + n_2 p_2 + n_3 p_3 \leq -d_u \\ -d_l &\leq t n_1(q_1 - p_1) + n_1 p_1 + n_2 p_2 + n_3 p_3 \leq -d_u \\ -d_l &\leq t n_1(q_1 - p_1) + \vec{n} \cdot \boldsymbol{p} \leq -d_u \\ -\vec{n} \cdot \boldsymbol{p} - d_l &\leq t n_1(q_1 - p_1) \leq -\vec{n} \cdot \boldsymbol{p} - d_u \end{aligned} \tag{5.5}$$

$$\frac{-\vec{n} \cdot \boldsymbol{p} - d_l}{n_1(q_1 - p_1)} \leq t \leq \frac{-\vec{n} \cdot \boldsymbol{p} - d_u}{n_1(q_1 - p_1)}. \tag{5.6}$$

Nyní by měla proběhnout diskuze o tom, zda v posledním kroku odvození je potřeba změnit smysl nerovnosti v závislosti na znaménku členu $n_1(q_1 - p_1)$. Víme, že musí platit jedna z relací

$$\frac{-\vec{n} \cdot \boldsymbol{p} - d_l}{n_1(q_1 - p_1)} \leq \frac{-\vec{n} \cdot \boldsymbol{p} - d_u}{n_1(q_1 - p_1)} \tag{5.7}$$

$$\frac{-\vec{n} \cdot \boldsymbol{p} - d_l}{n_1(q_1 - p_1)} \geq \frac{-\vec{n} \cdot \boldsymbol{p} - d_u}{n_1(q_1 - p_1)}. \tag{5.8}$$

Jelikož prohledávaný kvádr má nenulový objem, pak nerovnice (5.4) musí mít řešení pro nějakou trojici (x, y, z). Tedy správný smysl nerovnosti (5.6) lze určit z toho, která z relací (5.7), (5.8) je pravdivá. Označme tedy

$$b_{max} \coloneqq max\left(\frac{-\vec{n} \cdot \boldsymbol{p} - d_l}{n_1(q_1 - p_1)}, \frac{-\vec{n} \cdot \boldsymbol{p} - d_u}{n_1(q_1 - p_1)}\right) \tag{5.9}$$

$$b_{min} \coloneqq min\left(\frac{-\vec{n} \cdot \boldsymbol{p} - d_l}{n_1(q_1 - p_1)}, \frac{-\vec{n} \cdot \boldsymbol{p} - d_u}{n_1(q_1 - p_1)}\right) \tag{5.10}$$

Jelikož $t \in \langle 0, 1 \rangle$, výsledná podmínka má tvar

$$(b_{max} \geq 0) \wedge (b_{min} \leq 1). \tag{5.11}$$

Pokud je splněna, úsečka zasahuje do části prostoru vymezeného nerovnicí (5.4). Jsou-li k dispozici rovnice rovin, které tvoří stěny kvádru, lze takto určit, zda úsečka zasahuje do objemu kvádru postupným ověřováním podmínky (5.11) pro všechny tři dvojice rovnoběžných stěn.

Odvození bylo provedeno pro úsečku rovnoběžnou s osou x. Pro úsečku rovnoběžnou s osou y (resp. z) platí odvození identicky s tím rozdílem, že ve jmenovateli výrazů (5.6) a dalších budou indexy s číslem 2 (resp. 3).

Byla odvozena podmínka, kterou lze v kódu implementovat, s jejíž pomocí lze testovat, zda hrana uzlu stromu zasahuje do objemu prohledávaného kvádru. Nyní bude odvozena podmínka, kterou lze použít k testování, zda hrana prohledávaného kvádru zasahuje do objemu uzlu stromu.

Uzel stromu je v prostoru reprezentován krychlí. V tomto případě dojde ke zjednodušení z toho důvodu, že stěny těchto krychlí jsou vždy kolmé na souřadnicové osy. Nerovnice poloprostorů, jejichž průnikem je daná krychle, budou mít tedy jednoduchý tvar $n_1 x \geq -d$ (resp. $n_2 y \geq -d$, nebo $n_3 z \geq -d$). Část prostoru, která je vymezena dvěma rovnoběžnými rovinami, pak lze popsat nerovnicí (v případě rovin kolmých na osu x)

$$-d_l \leq n_1 x \leq -d_u. \tag{5.12}$$

Pro úsečky musíme uvažovat obecný předpis (5.1). Nyní bude opět provedeno dosazení rovnic (5.1) do nerovnice (5.12) a výraz následně upraven s tím, že budeme předpokládat, že normálový vektor roviny je jednotkový.

$$\begin{aligned} -d_l &\leq n_1(p_1 + t(q_1 - p_1)) \leq -d_u \\ -d_l &\leq p_1 + t(q_1 - p_1) \leq -d_u \end{aligned} \tag{5.13}$$

$$\frac{-d_l - p_1}{q_1 - p_1} \leq t \leq \frac{-d_u - p_1}{q_1 - p_1} \tag{5.14}$$

Diskuze změny smyslu nerovnosti je v tomto případě stejná, jako pro nerovnici (5.6). Bude provedeno označení

$$b_{max} := max\left(\frac{-d_l - p_1}{q_1 - p_1}, \frac{-d_u - p_1}{q_1 - p_1}\right) \tag{5.15}$$

$$b_{min} := min\left(\frac{-d_l - p_1}{q_1 - p_1}, \frac{-d_u - p_1}{q_1 - p_1}\right). \tag{5.16}$$

Podmínka se tedy opět sestává z kontroly, zda platí výraz

$$(b_{max} \geq 0) \wedge (b_{min} \leq 1). \tag{5.17}$$

V kódu bylo ještě využito skutečnosti, že jmenovatel $q_1 - p_1$ je roven x-ové složce jednotkového směrového vektoru úsečky vynásobené délkou úsečky.

Pro případy rovin kolmých na ostatní souřadnicové osy je odvození identické. Jediným rozdílem je změna dolních indexů p_2, q_2 (resp. p_3, q_3). Celkově je testování průniku kvádru a krychle popsáno v algoritmu 5.1.

Prohledávání válce

Návrh na vyzkoušení metody prohledávání válce přinesl vedoucí práce. Podmínka přidání uzlu na zásobník je v tomto případě založena na testování vzdálenosti uzlu od osy válce. Za účelem sestavení této podmínky je nutné specifikovat pojem „vzdálenost uzlu od úsečky“.

Algoritmus 5.1: Test průniku kvádru a krychle

Input: Axis aligned cube, Arbitrarily oriented rectangle box
forall *cube edges e* **do**
  if *e intersects with rectangle box* **then**
    return true;
  end if
end forall
forall *rectangle boxe's edges e* **do**
  if *e intersects with the cube* **then**
    return true;
  end if
end forall
return false;

Function Cube's edge intersects with rectangle box

Input: Axis aligned edge e, Arbitrarily oriented rectangle box
forall *rectangle boxe's parallel face doubles* **do**
  if *not condition* (5.11) **then**
    return false;
  end if
end forall
return true;

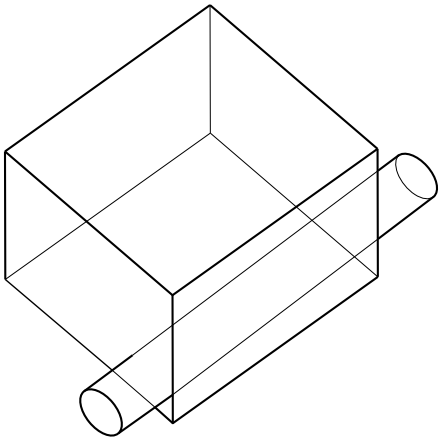
Function Rectangle boxe's edge intersects with cube

Input: Arbitrarily oriented edge e, Axis aligned cube
forall *cube's parallel face doubles* **do**
  if *not condition* (5.17) **then**
    return false;
  end if
end forall
return true;

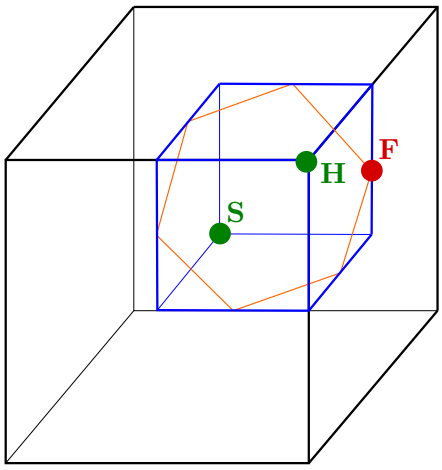
Definice 5.3. Nechť objekt K je zadaný uzel reprezentovaný v prostoru krychlí a objekt U reprezentuje úsečku. Vzdálenost K od U je definována jako $\min \|k - u\|$, kde $k \in K$ a $u \in U$.

V praxi by práce s touto definicí vyžadovala iteraci přes objem krychle a délku úsečky, popřípadě formulaci optimalizační úlohy. Za účelem zjednodušení výpočtu pracuje algoritmus pouze se středem krychle a s jejími vrcholy. Tyto body jsou pak testovány na vzdálenost od osy válce. Vzorec pro počítání vzdálenosti bodu od přímky v prostoru je již dobře známý a dostatečně jednoduchý na výpočet. Úprava vzorce tak, aby počítal vzdálenost od úsečky, bude popsána později.

Je nutné zavést tzv. vzdálenostní toleranční koeficient pro případy, kdy



(a) : Znázornění případu vysvětlujícího zavedení vzdálenostního tolerančního koeficientu



(b) : Bod X, jehož vzdálenost je od vrcholu A a středu S největší

Obrázek 5.3: Ilustrace k prohledávání válce

se testují uzly ve vyšších vrstvách stromu. V tomto případě se totiž mohou vrcholy a střed uzlu nacházet mimo prohledávaný válec. Uzel ale přesto může svým objemem zasahovat do válce. Tato situace je znázorněna na obrázku 5.3a.

Pro stanovení tolerančního koeficientu je potřeba najít takový bod v krychli, jehož vzdálenost od nejbližšího vrcholu nebo středu je největší. Díky symetrii lze tuto úlohu řešit pouze v osmině krychle (modrá část), jak je zobrazeno na obrázku 5.3b. Bod F nejvzdálenější bodům S a H byl nalezen následujícím způsobem. Mějme úsečku mezi body H a S. Jejím středem veďme rovinu kolmou na úsečku. Tato rovina představuje množinu všech bodů, které mají stejnou vzdálenost od S jako od H. Jejím průnikem s modrou krychlí je šestiúhelník (na obrázku znázorněn oranžovou barvou). Jeho vrcholy jsou body, která mají ze všech ostatních bodů v krychli největší vzdálenost od S a H. Tyto vrcholy jsou na hranách krychle umístěny v jejich středech. Jedním z nich je bod F vyznačený na obrázku 5.3b. Jeho vzdálenost od bodu H (resp. S) je rovna

$$|\mathrm{HF}| = |\mathrm{SF}| = \sqrt{\left(\frac{d}{2}\right)^2 + \left(\frac{d}{4}\right)^2} = d\frac{\sqrt{5}}{4}, \tag{5.18}$$

kde d je délka hrany černé krychle. Tím byla získána hodnota vzdálenostního tolerančního koeficientu $\frac{\sqrt{5}}{4}$.

Nyní již tedy lze stanovit podmínku, která poslouží pro rozhodnutí, zda uzel přidat na zásobník pro další prohledávání. Z výše uvedeného je zřejmé, že v nejhorším případě bude vzdálenost vrcholu nebo středu krychle od úsečky rovna hodnotě uvedené ve výrazu 5.18. Lze tedy spočítat hodnotu absolutní tolerance, která bude přičítána k poloměru válce.

$$\tau = \max\left(0,\ d\frac{\sqrt{5}}{4} - r\right), \tag{5.19}$$

kde r je hodnota poloměru válce. Tento výraz zafunguje i v situaci, kdy jsou krychle příliš malé (jsou hluboko ve stromě) a není potřeba přidávat žádnou toleranci.

Nyní bude pojednáno o počítání vzdálenosti vybraného bodu od úsečky reprezentující osu válce. S ohledem na rozsah této práce je zde uveden pouze odkaz [1] na jeden z možných postupů výpočtu vzdálenosti bodu od přímky. Na ten bude navázáno v následující úvaze. Značení je ponecháno stejné jako na odkazované stránce.

V návaznosti na [1] můžeme odvodit vzorec pro hodnotu parametru t.

$$t = \frac{(B - A) \cdot \vec{u}}{\|\vec{u}\|^2} \tag{5.20}$$

Nyní je potřeba omezit výpočet z přímky pouze na požadovanou úsečku. Toho je docíleno tím způsobem, že je kontrolována pozice bodu X. Ze směrového vektoru úsečky je určena tzv. nejvýznamnější dimenze. To je taková dimenze, jejíž složka je ve směrovém vektoru v absolutní hodnotě největší. Označme tuto dimenzi δ a operátoru $X(\delta)$ přisuďme hodnotu složky bodu X v dimenzi δ. Dále nechť A a C jsou koncové body úsečky tak, že platí $A(\delta) < C(\delta)$. Pokud je nerovnice

$$A(\delta) - \tau < X(\delta) < C(\delta) + \tau \tag{5.21}$$

splněna, pak má smysl testovat vzdálenost $\|B - X\|$, protože bod X leží na úsečce prodloužené o hodnotu tolerance. Testování uzlů na průnik s válcem je tedy prováděno algoritmem 5.2.

Implementace funkcí

S pomocí výše popsaných iterátorů byly sestaveny funkce, které zprostředkovávají přístup k funkcím prohledávání prostoru ve směru na vyšší úrovni abstrakce. Byly implementovány dva typy funkcí, které se liší pouze typem návratové hodnoty. První typ vrací souřadnice nejbližšího obsazeného voxelu. Pokud v prohledávaném prostoru není žádný voxel obsazený, vrací se bod o souřadnicích (NAN, NAN, NAN). Druhý typ pak vrací pouze vzdálenost od zadané pozice k nejbližšímu obsazenému voxelu. Vzdáleností je myšlen kolmý průmět na směrový vektor, který udává směr prohledávání. Pro usnadnění práce bylo nastaveno, že pokud se v prostoru nenachází žádný obsazený voxel, vrací se hodnota vzdálenosti, do které je prostor prohledáván. Důvod tohoto rozhodnutí je vysvětlen v podsekci 5.3.1.

5.3 Třída OcTreeManager

Třída OcTreeManager využívá třídu ObstDetOcTree a přidává další funkce, které poskytují interakci s řídícím algoritmem helikoptéry na vyšší úrovni abstrakce. Tyto funkce jsou:

- korekce letového příkazu v závislosti na vzdálenosti překážky ve směru letu helikoptéry

Algoritmus 5.2: Test průniku válce a krychle

Input: cube center S, cube edge size d, line segment L
τ := equation (5.19);
if *S is near L* **then**
    return true;
end if
V := vertices coordinates;
forall *vertices v from V* **do**
    if *v is near L* **then**
        return true;
    end if
end forall
return false;

Function P is near L

Input: point P, line segment L, tolerance τ, cylinder radius r
t := equation (5.20);
X := L(t);
distance := $\|P - X\|$;
if *condition* (5.21) *and distance* $< (r + \tau)$ **then**
    return true;
end if
return false;

- zjištění obsazenosti letové trasy zadané jako sekvence krajních bodů úseček, ze kterých se trasa skládá
- ukládání dat ze senzoru do OcTree

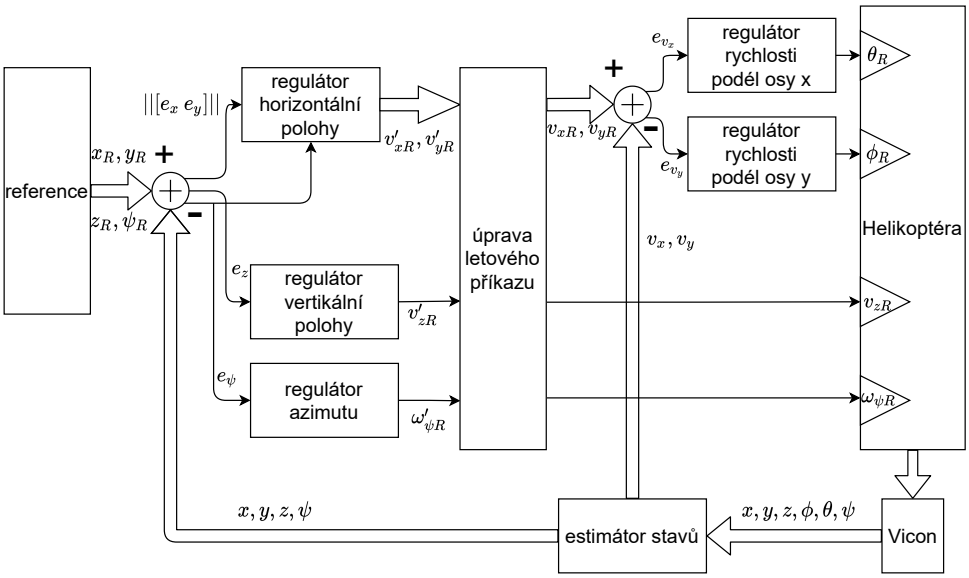
Tyto funkce budou následně podrobně popsány. Pro správnou funkci třídy je nutné periodicky obnovovat interní informaci o poloze a natočení helikoptéry pomocí metody `setPose`.

5.3.1 Korekce letového příkazu

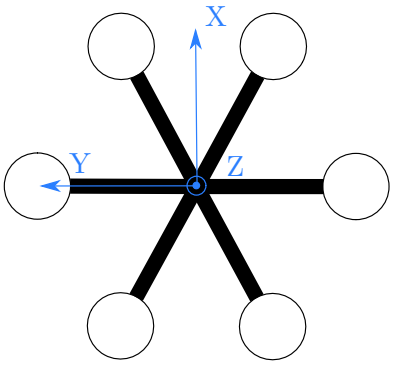
Korekce letového příkazu je zprostředkovávána pomocí metody s názvem `adjustFlightCommand`. Ta pracuje na úrovni rychlostí. Její funkce spočívá pouze ve zpomalení před překážkou. Zasazení do stávajícího řídícího algoritmu v helikoptéře je znázorněno na obrázku 5.4.

Vstupním argumentem je aktuální letový příkaz. Letový příkaz se skládá z jednotlivých rychlostních referencí, mezi které patří

- "elevator": ovlivňuje pohyb v ose X,
- "aileron": ovlivňuje pohyb v ose Y,



Obrázek 5.4: Zasazení korekce letového příkazu do stávajícího řídícího systému, převzato z [10], upraveno a použito se svolením



Obrázek 5.5: Souřadnicový systém spojený s helikoptérou, pohled shora, osa Z dle pravidla šípu směřuje ven z obrázku

- "throttle": ovlivňuje pohyb v ose Z,
- "rudder": ovlivňuje otáčení kolem osy Z.

Jako souřadnicové osy jsou v tomto seznamu použity osy souřadného systému spojeného s helikoptérou, který je zobrazen na obrázku 5.5. Osa X směřuje ve směru vpřed.

Podle aktuální rychlosti se mění vzdálenost od překážky, pod kterou by se helikoptéra neměla dostat. Její hodnota je určena z intervalu, který je definován krajními hodnotami zadanými v konstruktoru třídy. Konkrétně jde o vztah

$$l = l_{min} + s_c \cdot (l_{max} - l_{min}), \tag{5.22}$$

kde l je limitní vzdálenost od překážky, l_{min} je spodní hranice intervalu, l_{max} je horní hranice intervalu a s_c je tzv. rychlostní poměrný koeficient, který se

počítá podle vztahu

$$s_c = \min\left(\frac{s_a}{s_l}, 1\right), \tag{5.23}$$

kde s_a je aktuální skutečná translační rychlost a s_l je rychlostní limit, který je zatím dle [10] nastaven na 1 ms^{-1}.

Vzdálenost, do jaké se prohledává prostor v zadaném směru, je také ovlivněna aktuální rychlostní referencí. Při vyšších rychlostech je totiž nutná delší doba pro zpomalení. Tato vzdálenost se počítá podle vzorce

$$d = \max(2 \cdot s_a,\ 1.2 \cdot l_{min}). \tag{5.24}$$

Koeficienty 2 a 1.2 byly určeny na základě teoretické úvahy.

Samotná korekce letového příkazu pak spočívá pouze v přenásobení jednotlivých složek tzv. zpomalovacím koeficientem, který se spočítá podle vztahu

$$c = \frac{d_{obst} - l}{d - l}, \tag{5.25}$$

kde d_{obst} je vzdálenost nejbližší překážky. Jak již bylo zmíněno na konci podsekce 5.2.2, hodnota d_{obst} nabývá maximálně hodnoty d. To má za následek, že hodnota koeficientu c může nabýt maximálně hodnoty 1.

Zároveň ale toto jednoduché řešení neodmítá poruchu. Pokud např. vlivem větru bude dron tlačen k překážce, korekční algoritmus nenastaví reference rychlostí tak, aby se efekt poryvu větru anuloval. Jelikož je prozatím s helikoptérou lítáno pouze v laboratoři, není tento fakt zásadním problémem.

5.3.2 Zjišťování obsazenosti letové trasy

Obsazenost trasy se testuje použitím funkce s názvem `obstacleOnRoute`. Ta zejména využívá funkce pro detekci překážky ve válci, kterou poskytuje třída ObstDetOctree (viz podsekci 5.2.2). Vstupním argumentem je pouze vektor[2] souřadnic, který definuje po částech lineární trasu v prostoru. Z těchto souřadnic je pro každý úsek vypočítán počáteční bod, směrový vektor a vzdálenost úseku. S těmito parametry je v cyklu volána funkce pro prohledávání válce. Ta vyžaduje také rozměry prohledávaného tělesa. Ty jsou privátními parametry třídy OcTreeManager a jsou nastavovány v konstruktoru. Výstupní hodnotou je čtveřice hodnot skládající se ze souřadnic nejbližší překážky na trase a vzdálenosti, kterou by po dané trase uletěl dron ze zadaného počátečního bodu, než by k překážce doletěl. Pokud se na trase překážka nevyskytuje, je vrácena čtveřice hodnot NAN.

5.3.3 Ukládání dat ze senzoru

Funkce ukládání dat ze senzoru je využívána pouze vláknem RangeSensorDataManager (viz sekci 5.4). Funkce přijímá data ve formě objektu definovaného

[2] Zde ve smyslu datové struktury `std::vector<>` dostupné ve standartní knihovně jazyka C++.

v knihovně GearBox [3], která zprostředkovává komunikaci se senzorem. Objekt obsahuje pouze hodnoty vzdáleností tak, jak je senzor postupně naměřil[3]. Pro uložení do OcTree je nutné data přepočítat do globálních souřadnic. K tomu je nutné znát

- aktuální pozici helikoptéry (aktualizována přes funkci `setPose`),
- posunutí senzoru vůči měřené poloze helikoptéry,
- počet měření, které senzor provádí v jedné otočce,
- v jakém úhlu začíná senzor měřit a
- směr, jakým se otáčí laserový paprsek v senzoru[4].

Druhý parametr je nutné specifikovat v konstruktoru třídy OcTreeManager. Poslední tři parametry lze načíst přímo ze senzoru. Z toho důvodu jsou poskytnuty dvě verze konstruktorů tak, aby bylo možné tyto parametry specifikovat předem, i je nastavit podle hodnot, které se přečtou ze senzoru při startu programu.

Pro přepočítání naměřených dat ze souřadného systému spojeného se senzorem do souřadného systému spojeného se zemí bylo využito homogenních souřadnic. Pro ty lze definovat transformační matici ze souřadného systému a do souřadného systému b

$$T_a^b = \begin{bmatrix} R_a^b & \vec{t_a^b} \\ 0\ 0\ 0 & 1 \end{bmatrix}, \tag{5.26}$$

kde t_a^b je translační vektor mezi souřadnými systémy a R_a^b je rotační matice daná součinem rotačních matic kolem jednotlivých os v pořadí

$$R_a^b = R_x R_y R_z. \tag{5.27}$$

Tímto způsobem počítá s rotačními maticemi systém Vicon.

Takto byly vytvořeny transformační matice T_s^u ze souřadného systému senzoru, do souřadného systému helikoptéry a T_u^g ze souřadného systému helikoptéry do souřadného systému spojeného se zemí. Výsledná transformační matice $T_s^g = T_u^g T_s^u$ pak byla implementována v kódu.

Použitý senzor disponuje dle specifikace výrobce jmenovitým rozsahem 20 - 5600 mm [17]. Pokud se během měření vyskytne překážka ve větší vzdálenosti, než je jmenovitý rozsah, signalizuje tuto skutečnost senzor uložením měření o hodnotě 0 mm. Do OcTree jsou tedy přidávána pouze měření s hodnotu větší než 50 mm. Byla zvolena vyšší hranice než je udávaných 20 mm za účelem zamezení případného šumu. Nepředpokládá se měření překážek v menší vzdálenosti, než je 50 mm. Zároveň jsou data filtrována tak, jak je popsáno v sekci 3.3. Nová měření jsou ukládána jako paprsky. Voxely na trase paprsku jsou tedy aktualizovány jako neobsazené, jak je popsáno v sekci 3.2.

[3] Tedy data jsou v cylindrickém souřadném systému, který je spojený se senzorem.
[4] po/proti směru hodinových ručiček

5.4 Vlákno RangeSensorDataManger

Procedura ve funkci nazvané RangeSensorDataManager má za úkol získávat data z dálkového senzoru a předávat je ke zpracování třídě OcTreeManager (podsekce 5.3.3). RangeSensorDataManager je určen k volání v samostatném vlákně. Vstupním argumentem je ukazatel na třídu OcTreeManager.

Postup procedury spočívá v navázání spojení se senzorem a nastavením příslušných parametrů v senzoru. Následně přečte ze senzoru parametry, které se používají ke zpracování dat před uložením a tato data předá třídě OcTreeManager. Následuje smyčka, ve které se vždy počká na nové měření a toto měření se v tom samém vlákně uloží do OcTree pomocí metody poskytnuté třídou OcTreeManager. Konec činnosti vlákna je řízen atomickým logickým atributem třídy OcTreeManager. Tato proměnná je nastavena na logickou hodnotu `true` v destruktoru třídy OcTreeManager a tím je činnost vlákna zastavena. Před ukončením vlákna je spojení se senzorem zrušeno.

Kapitola 6

Testování

V této části bude popsána forma testování správnosti a rychlosti implementovaných algoritmů. Testy byly rozděleny na programové, simulační a laboratorní.

6.1 Programové testy

6.1.1 Správnost implementovaných algoritmů

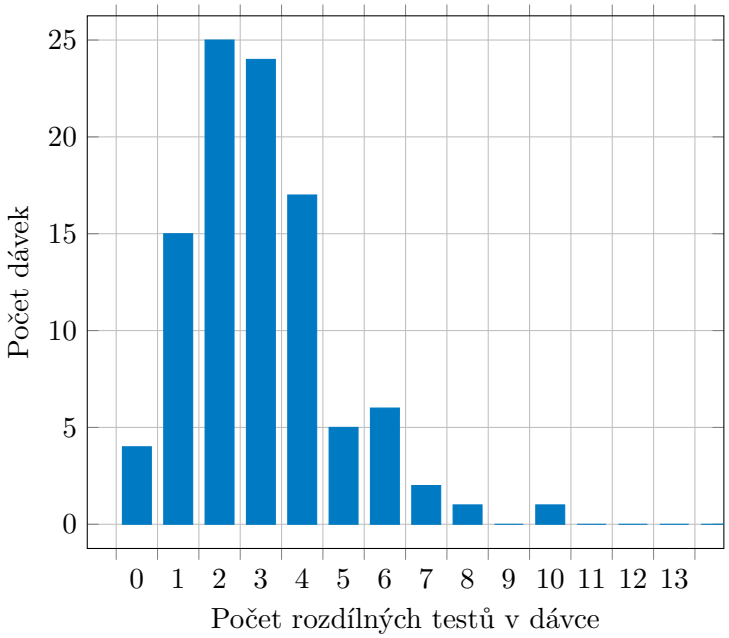
Manuálně sestavené testy

Pro kontrolu správnosti metod prohledávání prostoru byly sestaveny automatické testy. Tyto testy jsou konfigurovány z externího `json` souboru. Bylo manuálně namodelováno několik jednoduchých prostředí, ve kterých se testuje. Informace o scéně je uložena do octree. Poté jsou nastaveny parametry prohledávání. Výsledek obdržený z prohledávací funkce je porovnán se správným výsledkem. Testy funkcí třídy ObstDetOctree jsou nastaveny tak, aby byly přijaty obdržené výsledky, pokud se liší od správných výsledků pouze v povoleném rozmezí daném rozlišením octree.

Metody prohledávání válce, kvádru i koule splňují všechny manuálně sestavené testy.

Porovnávací testy

Byly implementovány i porovnávací testy, u kterých není předem specifikován správný výsledek. Pouze se porovnávají výsledky obdržené z metod prohledávání válce a kvádru. Tyto testy využívají data naměřená autory knihovny OctoMap. Tato data je možné nalézt v [14]. Pro každý test je vygenerována náhodná pozice a náhodný směrový vektor. Porovnávací testy ovšem zřejmě vykazují občasné odlišnosti v obdržených výsledcích. To je způsobeno jednak rozdílným tvarem válce a kvádru a také tím, že vzdálenost překážky se vyhodnocuje jako průmět do směrového vektoru. Z toho důvodu se stává, že obě metody dávají rozdílné výsledky. Pro eliminaci těchto rozdílností bylo během porovnávacích testů ustoupeno od počítání vzdálenosti k překážce jako průmětu do směrového vektoru. Zároveň rozměry válce a kvádru byly



Graf 6.1: Počet dávek, v nichž se objevil daný počet rozdílných testů

nastaveny tak, aby si objemy přibližně odpovídaly. K výsledkům pak bylo přistoupeno z pohledu statistiky.

Bylo provedeno 3000 porovnávacích testů. Ty byly rozděleny do 100 dávek po 30 testech. Rozdělení počtu rozdílných výsledků v dávkách je zobrazeno v grafu 6.1. Daný výsledek připomíná hustotu Poissonova rozdělení. Můžeme pozorovat, že dominuje počet 2 až 3 rozdílných testů v jedné dávce. V žádné dávce nebylo více jak 10 rozdílných testů. Celkově bylo z 3000 testů 298 rozdílných. To dává přibližně 90% shodu obou metod.

Na základě výše uvedeného bylo rozhodnuto, že obě metody jsou správně naimplementované z hlediska správnosti výsledků, které dávají.

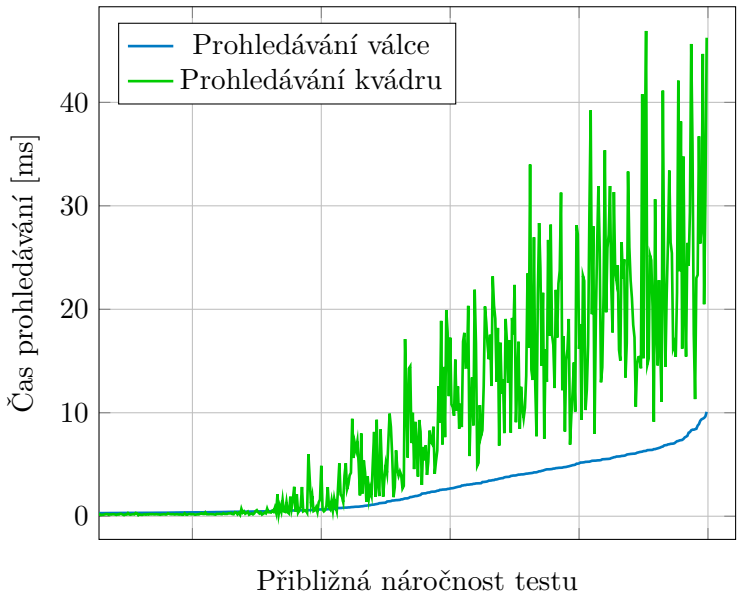
6.1.2 Testování rychlosti

Byly také provedeny testy ověřující rychlost algoritmů. K tomu bylo použito funkce `clock` ze standardní knihovny, která počítá čas strávený programem na procesoru.[1] Rychlostní testy byly prováděny v prostředí zmapovaném autory knihovny OctoMap, stejně jako u porovnávacích testů.

Testování rychlosti prohledávání v celém okolí helikoptéry

Bylo provedeno 1000 testů na náhodných pozicích. Nejdelší prohledávání činilo 21.4 ms.

[1] Testy byly prováděny na stroji Macbook Pro, Mid 2010, 2.4 GHz Intel Core 2 Duo, 8 GB 1067 MHz DDR3.



Graf 6.2: Výsledky náhodných rychlostních testů seřazené vzestupně podle doby prohledávání válce

Testování rychlosti prohledávání v zadaném směru

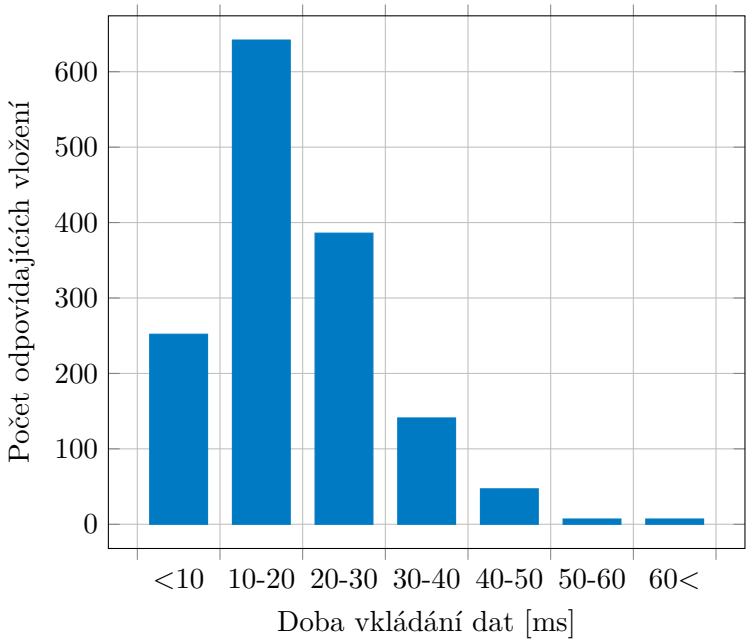
Testy byly provedeny pro poloměr válce 45 cm, hrany základny kvádru 80 cm a délku prohledávání 1.5 m. Jde tedy o hodnoty reálně použitelné v praxi při nasazení algoritmu na středně velké drony.

První část testů rychlosti byla provedena ve třech specifických pozicích podle míry obsazenosti prostoru. Bylo provedeno 100 volání funkce pro každou z pozic a výsledný čas spadající na jedno volání byl následně dopočten vydělením. Výsledky těchto testů jsou uvedeny v tabulce 6.1. Z těchto výsledků je přibližně zřejmá časová náročnost obou prohledávacích algoritmů.

	Prohledávání kvádru [ms]	Prohledávání válce [ms]
Prázdný prostor	0.2	0.4
Středně plný prostor	9.4	3.5
Více plný prostor	37.6	11.3

Tabulka 6.1: Výsledky testování rychlosti prohledávacích metod

Pro lepší představu byly provedeny ještě testy rychlosti na náhodných pozicích do náhodných směrů. Pro každý test se opět počítal čas jako průměr ze 100 běhů. Celkem byl měřen čas na 1000 náhodných pozicích. Výsledek je zobrazen na grafu 6.2. Časové výsledky byly seřazeny vzestupně podle času prohledávání válce. Z grafu vyplývá podstatná informace, že metoda prohledávání válce je přibližně čtyřikrát rychlejší, než metoda prohledávání kvádru. Vodorovná osa nenese žádnou konkrétní informaci. Pouze přibližně ukazuje na složitost prohledávacího testu. Z grafu lze také odhadnout, že pravděpodobně žádné prohledávání nepřesáhne značně 10 ms pro prohledávání



Graf 6.3: Počet vkládání dat v závislosti na délce trvání

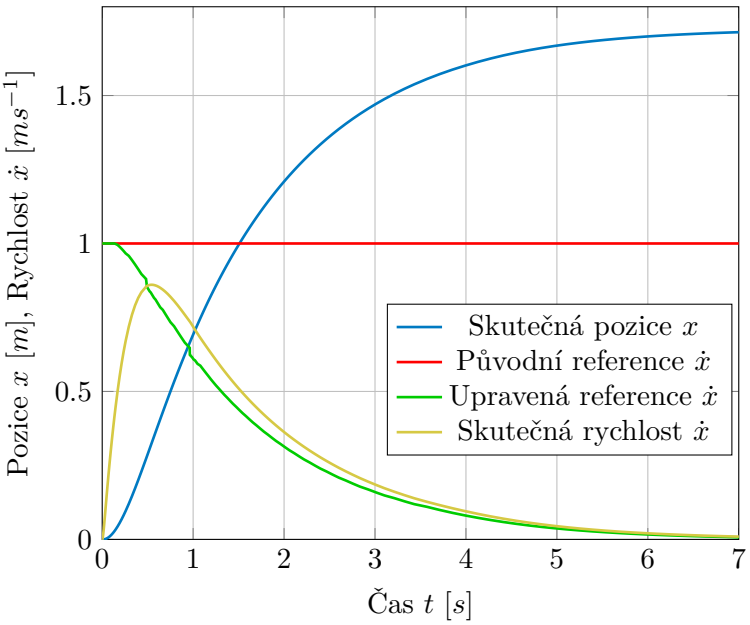
válce a 45 ms pro prohledávání kvádru.

Testování rychlosti vkládání nových měření

Byly provedeny i rychlostní testy vkládání nových měření. Byla uložena sekvence měření ze senzoru Hokuyo do textového souboru. Tento soubor byl poté v měřicím programu zpětně čten a byly měřeny doby vkládání novým měření. Začínalo se s prázdným octree. Výsledky testů vkládání jsou zobrazeny v grafu 6.3. Maximální doba vkládání jednoho skenu činila 66.8 ms. Z grafu tedy lze vyčíst podstatnou informaci, že vkládání dat do octree trvá přibližně 20 ms. Jelikož pořízení jednoho skenu pomocí použitého senzoru Hokuyo trvá 100 ms, je doba vkládání dostatečně nízká na to, aby neblokovala pořizování nově naměřených dat.

6.2 Simulační testy

Úprava letového příkazu byla nejprve testována v simulačním prostředi Simulink. Byl použit matematický model, který identifikoval Martin Jiroušek, a jím navržené regulátory [10]. Na rozdíl od skutečnosti byly regulátory ponechány v časové doméně. Dále bylo vygenerováno virtuální simulační prostředí, ve kterém byla namodelována zeď na pozici $x = 2$. Helikoptéra startovala z nulových počátečních podmínek s referencí polohy $x_{ref} = 4$. Výsledek simulace je zobrazen v grafu 6.4. Je vidět, že helikoptéra zastavila před zdí. Zároveň lze pozorovat, že původní reference rychlosti (červený průběh) se stále drží na maximu 1 ms^{-1}. Tedy bez korekce by helikoptéra do zdi narazila.



Graf 6.4: Průběh simulačního testu úpravy letového příkazu

6.3 Laboratorní testy

Laboratorní testy byly prováděny v laboratoři skupiny IMR v CIIRC[2]. Nejprve byly prováděny zkoušky pouze se senzorem v ruce, následně pak se senzorem umístěným na helikoptéře.

6.3.1 Testování zpracování naměřených dat

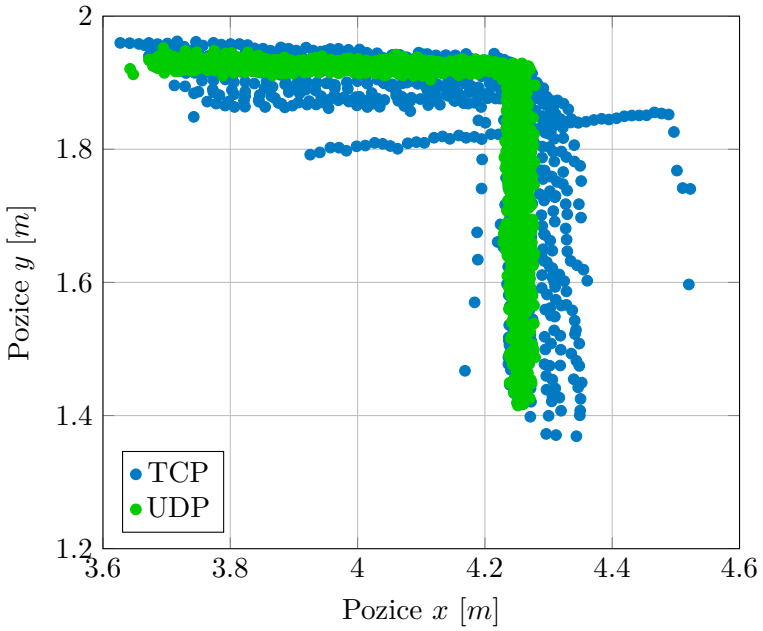
Testování protokolu

Během měření bylo zjištěno, že informace o získávání polohy z pozičního systému je nezanedbatelně zpomalena. Intuitivně bylo odhadnuto, že problém by mohl být v zahlcení sítě, přes kterou se data o poloze přenášejí. Za účelem ověření této hypotézy byl proveden test, ve kterém bylo pohybováno senzorem pouze v horizontálním směru po přímce přibližnou rychlostí 0.5 ms^{-1}. Nejdříve byl test proveden se získáváním dat o poloze přes TCP. Následně byl proveden podobný test s použitím protokolu UDP. Rozdíl v naměřených datech je zobrazen v grafu 6.5. Je vidět, že data uložená se získáváním polohy přes UDP jsou více kompaktní. Volba použitého protokolu tedy má vliv na měření.

Ověření naměřené scény

Následující testy byly provedeny s rozlišením octree 5 cm. Za účelem ověření správnosti mapování prostředí byla sestavena scéna z krabic tak, jak je

[2]Český institut informatiky, robotiky a kybernetiky, Jugoslávských partyzánů 1580/3, 160 00 Praha 6, Dejvice



Graf 6.5: Porovnání horizontálního testu při použití TCP a UDP

zobrazeno na obrázku 6.1. Ta byla naskenována se senzorem v ruce. Výsledek zmapování je zobrazen na obrázku 6.2. Barvy voxelů odpovídají výšce nad podlahou laboratoře. Přibližně uprostřed obrázku můžeme pozorovat tři zelené oblasti, která odpovídají naskenovaným krabicím.

Stejná scéna byla také zmapována senzorem umístěným na helikoptéře. Zmapované prostředí je zobrazeno na obrázku 6.3.

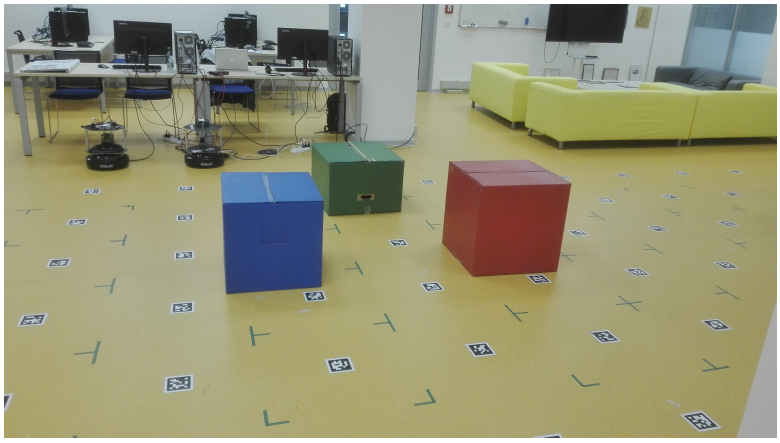
Za účelem exaktnějšího ověření správnosti naměřených dat byly porovnány rozměry ve skutečné scéně a v naměřených datech. Schéma naměřené scény při pohledu shora je zobrazeno na obrázku 6.4. Barvy krabic jsou zachovány. Naměřené hodnoty jsou k porovnání uvedeny v tabulce 6.2. Hodnoty uvedené

	a	b	c	d	e	f	g	h	i
Skut. [cm]	57.5	59.0	58.5	53.0	64.0	60.0	94.0	66.5	66.0
Ruč. [cm]	55	60	60	55	60	60	85	75	68
Heli. [cm]	60	60	65	60	65	65	96	65	70
	Celková průměrná absolutní chyba $\approx$ 3.4 cm								

Tabulka 6.2: Porovnání naměřených vzdáleností skutečných, ručně naskenovaných a naskenovaných senzorem na helikoptéře

jako skutečné byly naměřeny ručním metrem s milimetrovým rozlišením. Z dat vyplývá, že hodnoty si odpovídají. Průměrná absolutní chyba činí přibližně 3.4 cm, což je méně než použité rozlišení octree. Větší odchylky u některých měření mohou být dány šumem v měření, kvůli kterému není vždy jednoznačné, mezi kterými dvěma body měřit.

Během laboratorních testů bylo také zjištěno, že šum v datech může být způsoben i lesklostí podlahy v laboratoři, případně skly ve stěnách laboratoře.



Obrázek 6.1: Sestavená měřená scéna

Tento šum není vždy plně odstraněn filtrací dat, jelikož v některých případech jde o více než jeden sousední bod.

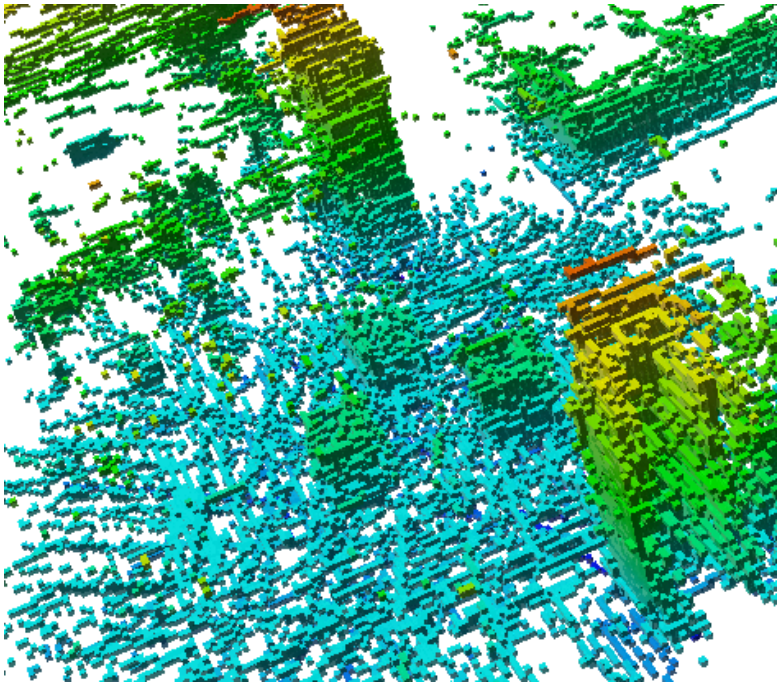
6.3.2 Testování korekce letového příkazu

Test v laboratoři byl proveden bez spuštěných motorů. Helikoptéra byla ručně přenášena a letový příkaz simulován nastavením řídících páček na vysílačce. Část průběhu testu je zobrazena v grafu 6.6. Lze pozorovat, že letový příkaz se upravuje podle vzdálenosti od překážky. Bylo ovšem zaznamenáno také nesprávné chování. Více je popsáno v závěru.

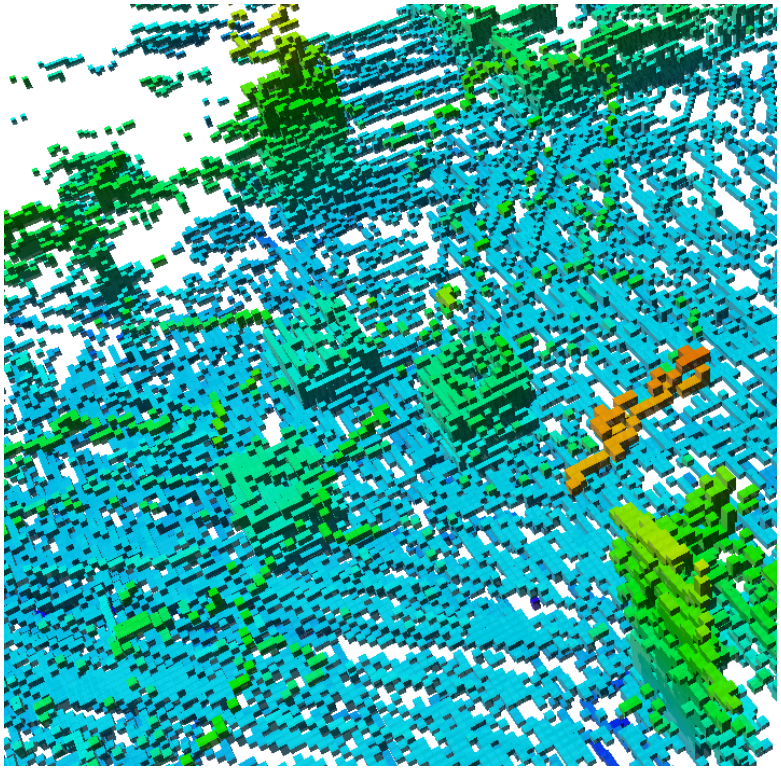
6.3.3 Testování prostorového omezení mapovaného prostoru

V laboratoři byla také nasimulována situace, kdy pozice helikoptéry přesáhne limit mapy. Toho bylo docíleno nastavením velmi jemného rozlišení mapy. Celkový objem, který lze do octree uložit, se tak zmenšil na rozměr vhodný pro laboratorní podmínky. Ukázalo se, že knihovna OctoMap řeší tuto situaci tím způsobem, že přestane ukládat nově naměřená data a do terminálu vypisuje varování `WARNING: coordinates ( (3.8 1.8 0.5) -> (6.8 6.2 0.6)) out of bounds in computeRayKeys`. Program ovšem neukončí svou činnost. Pokud má tedy helikoptéra jiný zdroj měření aktuální pozice, má možnost se vrátit do daných mezí.

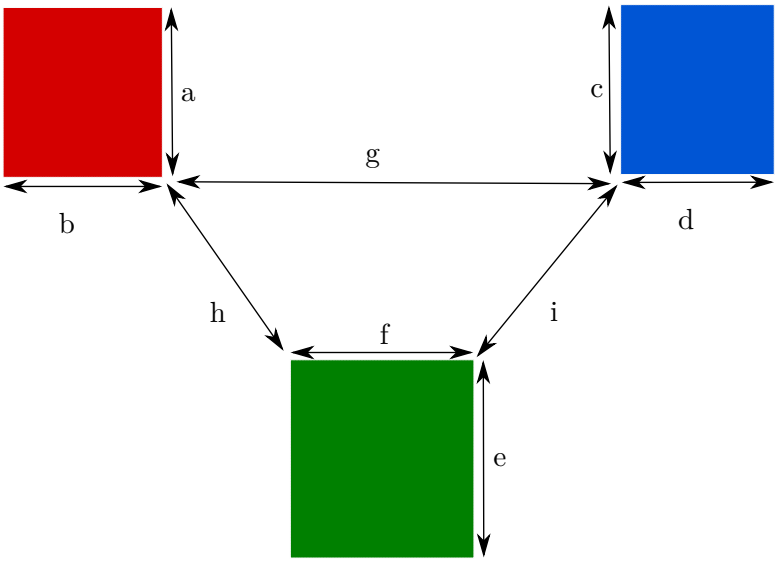
Pokud nastane situace, kdy bude zapotřebí prohledávat větší prostor, než je dáno původní hloubkou octree a použitého rozlišení, lze v kódu tuto hloubku zvýšit. Knihovna OctoMap ovšem nepodporuje dynamické zvyšování mezí mapy za běhu programu. Úprava musí být provedena před kompilací knihovny.



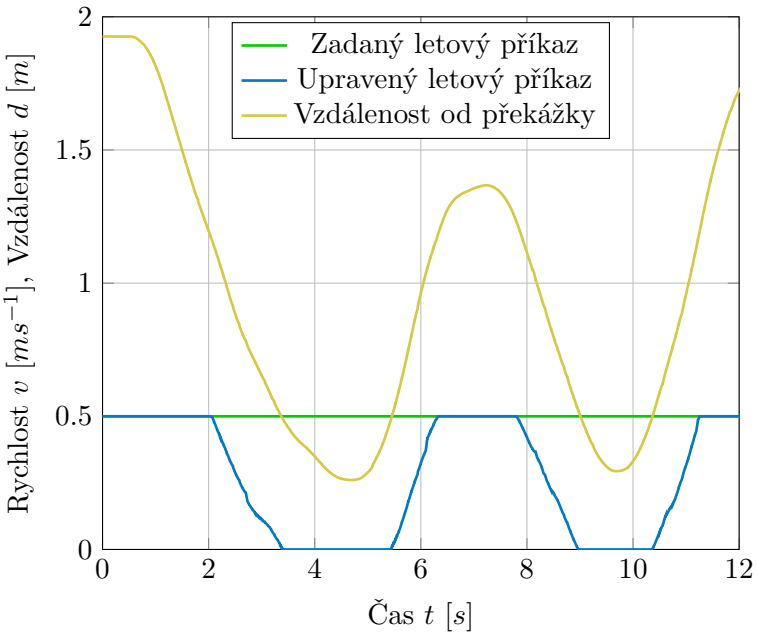
Obrázek 6.2: Výsledek měření se senzorem v ruce



Obrázek 6.3: Výsledek měření se senzorem na helikoptéře



Obrázek 6.4: Pohled na měřenou scénu shora



Graf 6.6: Testování korekce letového příkazu v laboratoři

Kapitola 7
Závěr

7.1 Dosažené výsledky

V této práci byla popsána implementace softwarového systému pro mapování okolí robotické helikoptéry. Systém zprostředkovává sběr dat ze senzoru, jejich filtraci a ukládání do octree. Je k dispozici metoda, pomocí které lze zjistit, zda je letová trasa zadaná jako sekvence lineárních úseků volná pro průlet. Nad rámec zadání byla implementována metoda pro úpravu letového příkazu podle přítomnosti překážky v přímém směru letu.

Ukládání dat je dostatečně rychlé v porovnání s množstvím dat, které jsou přijímány ze senzoru. Všechny skeny jsou uloženy rychleji, než je pořízeno nové měření. Většina skenů je uložena za dobu menší než 30 ms. Přesnost naměřených dat je dostatečná. Měření ukázalo průměrnou absolutní chybu menší, než bylo použité rozlišení octree. Metoda prohledávání válce vykazuje dostatečnou rychlost na to, aby bylo možné včas zareagovat na překážku ve směru letu. Během měření nepřekročila dobu 10 ms.

7.2 Možná rozšíření práce

Laboratorní experimenty ukázaly, že metoda pro úpravu letového příkazu není v této chvíli zcela vyladěna tak, aby bylo možné provádět testy v zapojené řídící smyčce. Bylo pozorováno, že občas letový příkaz není zkorigován tak, aby helikoptéra zastavila před překážkou. Dále bylo zřídka pozorováno, že letový příkaz byl naopak posílen, což je nechtěné chování. Za účelem detekce zdroje tohoto problému byla uložena letová data z několika laboratorních scénářů. Byl sestaven vizualizační skript v programu Matlab, který zobrazuje pozici helikoptéry, prohledávaný prostor před helikoptérou, naměřená data a graf vývoje zadaného a zkorigovaného letového příkazu. V datech byly identifikovány momenty nesprávného chování algoritmu. Naměřená data byla poté znovu zpracována algoritmem korekce letového příkazu za účelem ladění. V nově zpracovaných datech ovšem nebyly pozorovány žádné chybné zásahy do letového příkazu. Z časových důvodů byla takto analyzována pouze jedna dávka naměřených dat. Nicméně zdroj problému, který se v laboratoři objevil, zatím nebyl nalezen. Hlubší porozumění tomuto chování může být předmětem

dalšího vývoje.

Z časových a konstrukčních důvodů nebyla vyzkoušena metoda mapování okolí pomocí nakloněného 2D senzoru přesně tak, jak bylo navrženo v části 2.5.2. Scény naměřené senzorem na helikoptéře byly naměřeny v konfiguraci, kdy senzor zabírá pouze spodní část prostoru před helikoptérou.

Dále bylo při laboratorních testech zjištěno, že frekvence, s jakou použitý dálkoměr pořizuje data, není dostatečně vysoká, aby bylo možné spolehlivě mapovat okolí při letové rychlosti okolo 1 ms^{-1}. Z toho důvodu bych doporučil v budoucím vývoji vyzkoušet mapování pomocí 3D lidaru, který by mohl poskytovat větší množství dat. Zároveň by bylo nutné optimalizovat ukládací proces tak, aby vyhovoval většímu množství pořízených dat v čase.

7.2.1 Autonomní lokalizace

Jedním z dalších kroků, které budou navazovat na tuto práci je úloha autonomní lokalizace. Prozatím je detekce polohy řešena pozičním systémem Vicon v laboratoři. Ze znalosti okolního prostředí lze ovšem aktuální pozici odhadovat. Tato úloha se obecně označuje zkratkou SLAM[1].

Kuramachi a spol. [11] představují systém lokalizace pomocí metody „ICP SLAM“[2]. K tomu používají pouze 3D lidar a inerciální senzor. Autoři uvádějí, že jejich metoda je robustní pro libovolný pohyb.

Jiné řešení úlohy autonomní lokalizace uvádí Hess a spol. [8]. Ti představili systém pro 2D mapování v reálném čase. Jejich práce by mohla být rozšířena do 3D pro úlohu lokalizace helikoptéry v prostoru.

Ocando a spol. publikovali článek [13], ve kterém se zabývají prostorovou SLAM úlohou s použitím 2D lidaru. K mapování používají pozemního robota. Rozmítání laserového paprsku do třetí dimenze řeší umístěním senzoru na otočnou plošinu.

Z výše uvedeného je zřejmé, že pro 3D SLAM lokalizaci je výhodnější využít 3D lidar. Články zabývající se 3D SLAM s použitím 2D senzoru umístěného pevně bez možnosti náklonu nebyly nalezeny.

7.2.2 Prohledávání trasy

Možným rozšířením metody pro prohledávání letové trasy by bylo změnit typ vstupního argumentu. Místo po částech lineární trasy by mohlo být možné specifikovat libovolnou křivku popsanou v prostoru. Bylo by nutné najít vhodnou formu reprezentace takové trasy. V takovém případě by bylo nutné sestavit další iterátor, který by prohledával okolí zadané křivky v prostoru.

7.2.3 Plánování trasy

Dalším z kroků, které vedou k vyšší samostatnosti UAV je schopnost plánovat trasu. S tím je spojen i úkol trasu opravovat po zmapování dříve neznámé překážky na trase.

[1] SLAM z anglického „simultaneous localization and mapping“

[2] ICP z anglického “iterative closest point”

Jednou z instancí úloh plánování trasy je „Dubins touring problem“ (DTP). Tímto problémem se v poslední době zabývali např. Faigl a spol. [7]. DTP je původně formulován jako optimalizační problém nalezení nejkratší trasy v rovině s omezením na zakřivenost trajektorie. Agent se navíc může pohybovat pouze vpřed. Úlohu lze použít i pro plánování trasy helikoptéry, pokud by se uvažovalo mapování a pohyb pouze ve dvou dimenzích.

Přehled metod pro plánování trasy ve 3D speciálně pro UAV poskytuje Yang a spol [22]. Ve svojí práci shrnují a porovnávají algoritmy a přístupy v té době publikované.

Příloha A

Bibliografie

[1] *Analytická geometrie*. URL: www2.karlin.mff.cuni.cz/~portal/analyticka_geometrie/prostor.php?kapitola=vzdalenost.

[2] *Award Winning Motion Capture Systems*. URL: www.vicon.com.

[3] Geoffrey Biggs a Alex Makarenko. *GearBox*. URL: gearbox.sourceforge.net.

[4] Jan Cabiar. „Rekonstrukce 3D modelu prostředí helikoptérou“. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2016. URL: http://hdl.handle.net/10467/64650.

[5] D. De Gregorio a L. Di Stefano. „SkiMap: An efficient mapping framework for robot navigation“. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, s. 2569–2576. DOI: 10.1109/ICRA.2017.7989299.

[6] Alberto Elfes. „Occupancy grids: A stochastic spatial representation for active robot perception“. In: *Proceedings of the Sixth Conference on Uncertainty in AI*. Sv. 2929. Morgan Kaufmann. 1990, s. 6.

[7] Jan Faigl et al. „On solution of the Dubins touring problem“. In: *2017 European Conference on Mobile Robots (ECMR)*. 2017, s. 1–6. DOI: 10.1109/ECMR.2017.8098685.

[8] Wolfgang Hess et al. „Real-time loop closure in 2D LIDAR SLAM“. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, s. 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

[9] Armin Hornung et al. „OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees“. In: *Autonomous Robots* (2013). Software available at http://octomap.github.com. DOI: 10.1007/s10514-012-9321-0. URL: octomap.github.com.

[10] Martin Jiroušek. „Řízení polohy robotické hexakoptéry“. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2020. URL: dspace.cvut.cz/handle/10467/87812.

[11] Ryo Kuramachi et al. „G-ICP SLAM: An odometry-free 3D mapping system with robust 6DoF pose estimation“. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015, s. 176–181. DOI: 10.1109/ROBIO.2015.7418763.

[12] Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Tech. zpr. Rensselaer Polytechnic Institute, Image Processing Laboratory, říj. 1980.

[13] Manuel González Ocando et al. „Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS“. In: *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. 2017, s. 1–6. DOI: 10.1109/SBR-LARS-R.2017.8215333.

[14] *OctoMap 3D scan dataset*. URL: ais.informatik.uni-freiburg.de/projects/datasets/octomap/.

[15] Pavel Petráček. „Návrh, lokalizace a stabilizace specializované bezpilotní helikoptéry pro dokumentaci historických objektů“. Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2019.

[16] William Pugh. „Skip Lists: A Probabilistic Alternative to Balanced Trees“. In: *Commun. ACM* 33.6 (červ. 1990), s. 668–676. ISSN: 0001-0782. DOI: 10.1145/78973.78977. URL: https://doi.org/10.1145/78973.78977.

[17] *Scanning Rangefinder Distance Data Output/URG-04LX-UG01 Product Details*. URL: www.hokuyo-aut.jp/search/single.php?serial=166.

[18] P. Štibinger, T. Báča a M. Saska. „Localization of an Ionizing Radiation Source by a Formation of Unmanned Aerial Vehicles“. In: *IEEE IROS (Second Workshop on Multi-robot Perception-Driven Control and Planning)*. 2018.

[19] *The Point Cloud Library (PCL)*. URL: pointclouds.org.

[20] R. Triebel, P. Pfaff a W. Burgard. „Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing“. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, s. 2276–2282. DOI: 10.1109/IROS.2006.282632.

[21] *Ultrasonic sensors UM30*. URL: www.sick.com/ag/en/distance-sensors/ultrasonic-sensors/um30/c/g185672.

[22] Liang Yang et al. „A literature review of UAV 3D path planning“. In: *Proceeding of the 11th World Congress on Intelligent Control and Automation*. 2014, s. 2376–2381. DOI: 10.1109/WCICA.2014.7053093.

Příloha B

Obsah přiloženého DVD

```
|-- Makefile
|-- README.md
|-- data
|   |-- Hornung-data
|   |   `-- fr_079.bt
|   |-- scans-octovis
|   |   |-- boxes-hand-scan.bt
|   |   `-- boxes-helicopter-scan.bt
|   `-- simulation-data
|       `-- wall-2-X.bt
|-- simulation
|   |-- adjustFlightCommand.cpp
|   |-- adjustFlightCommand_wrapper.cpp
|   `-- dynamic_model.slx
|-- src
|   |-- LineDistanceIterator.hxx
|   |-- ObstDetOcTree.cpp
|   |-- ObstDetOcTree.h
|   |-- OctreeManager.cpp
|   |-- OctreeManager.h
|   |-- RangeSensorDataManager.cpp
|   |-- RangeSensorDataManager.h
|   `-- RectangleBoxIterator.hxx
|-- tests
|   |-- ObstDetOcTreeBenchmark.cpp
|   |-- ObstDetOcTreeTest.cpp
|   |-- ObstDetOcTreeTest.json
|   |-- OctreeManagerTest.cpp
|   |-- OctreeManagerTest.json
|   `-- TestLaunch.cpp
`-- vicon
    |-- tclient.cc
    |-- vicon_client.cc
    `-- vicon_client.h
```

Makefile

Přiložený Makefile obsahuje cíle pro kompilaci přiložených souborů. Pro korektní fungování je nutné mít k dispozici zkompilovanou verzi knihovny OctoMap a Gearbox. Umístění hlavičkových a knihovních souborů je nutné uvést v proměnných „LINK_DIR“ a „INCLUDE_DIR“, pokud nejsou ve standardně prohledávaných adresářích.

Cíle v přiloženém Makefile mají následující funkce.

- Cíl „all“ zkompiluje program napsaný v souboru „TestLauch.cpp“. Ten je určen pro testovací spouštění systému. V programu se inicializuje objekt „OctreeManager“. Na konci programu se uloží naměřená data do souboru s příponou „.bt“, která slouží pro vizualizaci v programu Octovis.

- Cíl „test“ zkompiluje manuálně sestavené a porovnávací testy tříd „ObstDetOcTree“ a „OctreeManager“. Pro správnou funkci tohoto cíle je nutné vypnout počítání skalárního součinu při počítání vzdálenosti. To lze provést nastavením makra „COMPUTEDOT“ v souboru „ObstDetOcTree.cpp“ na hodnotu 0.

- Cíl „bench“ zkompiluje rychlostní testy třídy „ObstDetOcTree“.

Složka „simulation“

Tato složka obsahuje dynamický model vytvořený v prostředí Simulink. Model obsahuje matematický popis helikoptéry a regulátory navržené kolegou Jirouškem [10]. Do modelu byl implementován blok simulující funkci úpravy letového příkazu. Pro správnou činnost je nutné nastavit adresářové cesty v bloku s názvem „adjustFlightCommand“ v záložce „Libraries“ a adresářovou cestu k modelu prostředí v záložce „Start“. Model prostředí použitý k simulaci v této práci je ve složce „data/simulation-data“ a obsahuje zeď kolmou na osu x na pozici $x = 2$.

Složka „src“

Tato složka obsahuje veškeré zdrojové soubory, které jsou potřebné pro implementaci systému do řídícího algoritmu helikoptéry. Stručný návod na implementaci je poskytnut v souboru „README.md“.

Složka „tests“

V této složce jsou programy sloužící pro testování naimplementovaných algoritmů. Některé z nich používají prostředí naměřené autorem knihovny OctoMap. Soubor s tímto prostředím je poskytnut s licencí CC BY 3.0 a nachází se ve složce „data/Hornung-data“.

Složka „vicon“

V této složce jsou kódy implementované vedoucím práce. Slouží k získávání dat z pozičního systému Vicon.