ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Software pro grafickou demonstraci plánování pohybu kvadrokoptér
Student:	Bc. Tomáš Mahr
Vedoucí:	doc. Dipl.-Ing. Dr. techn. Stefan Ratschan
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je návrh a implementace softwarového systému, který může sloužit ke grafické demonstraci určitého algoritmu pro plánování pohybu kvadrokoptéry, k experimentování s tímto algoritmem a k porovnávání s jinými algoritmy.

1) Vypracujte rešerši softwarových balíků pro simulaci, vizualizaci a plánování pohybu robotů a kvadrokoptér.
2) Ve spolupráci s vedoucím práce proveďte detailní specifikaci a navrhněte uživatelské rozhraní výsledného softwarového systému.
3) Navrhněte softwarovou architekturu, která je modulární ve smyslu, že jednotlivé komponenty (simulace, vizualizace, plánování pohybu) se dají snadno vyměnit.
4) Přizpůsobte algoritmus pro falzifikaci hybridních systémů [1] problému plánování pohybu kvadrokoptéry a implementujte jej jako modul systému.
5) Systém implementujte, otestujte na vhodných příkladech, zdokumentujte a zhodnoťte jeho užitečnost.

Seznam odborné literatury

[1] Kuřátko, Jan, and Stefan Ratschan. "Combined global and local search for the falsification of hybrid systems." Formal Modeling and Analysis of Timed Systems. Springer International Publishing, 2014. 146-160.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 23. září 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Software pro grafickou demonstraci plánování pohybu kvadrokoptér

Bc. Tomáš Mahr

Vedoucí práce: doc. Dipl.-Ing. Dr. techn. Stefan Ratschan

7. ledna 2018

Poděkování

Rád bych na tomto místě poděkoval panu Stefanu Ratschanovi za pomoc při vedení mé diplomové práce. Dále bych chtěl poděkovat své rodině za podporu během studia.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2018 .....

Odkaz na tuto práci

Mahr, Tomáš. *Software pro grafickou demonstraci plánování pohybu kvadrokoptér*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstract

The goal of this thesis is to design and implement modular software for the graphical demonstration of quadrocopter motion planning. The thesis also explains basic theory of quadrocopter motion planning. One of the existing algorithms is implemented as a module.

Keywords Quadrocopter, motion planning, graphical demonstration, modular software.

Abstrakt

Práce se zabývá návrhem a implementací aplikace pro grafickou demonstraci plánování pohybu kvadrokoptér. Je kladen důraz na modulárnost softwaru. Dále je vyložena základní teorie plánování pohybu kvadrokoptér a je implementován jeden z existujících algoritmů pro plánování pohybu.

Klíčová slova Kvadrokoptéra, plánování pohybu, grafická demonstrace, modulární software.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Kvadrokoptéra, nebo také dron, je bezpilotní vrtulník se čtyřmi rotory (aktuátory). Ovládá se na dálku nebo pomocí naprogramovaného mikropočítače. Pohyb kvadrokopéry se řídí nastavením otáček rotorů. Mezi typické vlastnosti kvadrokoptéry patří nízká hmotnost, malé rozměry a dobré manipulační schopnosti. V současné době se drony stávají velmi populárními. Používají se nejen pro zábavné účely (focení, nahrávání videa), své uplatnění mají také v průmyslu a používá je policie, hasiči i armáda.

V této práci se zabývám automatickým ovládáním kvadrokoptér. Konkrétně se jedná o situaci, kdy známe matematický model dronu a popis okolí, ve kterém se pohybuje. Dále máme zadaný počáteční a koncový stav dronu. Cílem je najít plán pohybu, tedy časový vývoj hodnot aktuátorů, který dovede drona z počátečního do koncového stavu, aniž by došlo ke kolizi s okolím. Tato práce se zabývá pouze softwarovou simulací a vizualizací, nikoliv ovládáním reálného dronu. Cílem práce je vytvořit modulární software pro plánování pohybu kvadrokoptér s grafickou vizualizací. Modularita softwaru spočívá v možnosti snadno měnit a přidávat plánovač, simulátor a vizualizaci pohybu. Existuje velké množství algoritmů pro plánování pohybu. V této práci je jeden ([2]) z nich popsán a implementován.

První část této práce obsahuje definice základních pojmů týkajících se plánování pohybu kvadrokoptér a rešerši existujících softwarových balíků. Následující dvě kapitoly se zabývají analýzou a návrhem vlastního modulárního softwaru pro plánování pohybu kvadrokoptér. Dále je popsána implementace, včetně všech tří modulů. Plánovač pohybu je implementací algoritmu [2]. Poslední kapitola je věnována testování a nasazení softwaru.

Kapitola 1

Plánování pohybu kvadrokoptér

1.1 Základní pojmy

Tato sekce vysvětluje několik základních pojmů a definuje problém, kterým se práce zabývá.

- **Aktuátor**
  Pojmem aktuátor označujeme rotor kvadrokoptéry. Hodnota aktuátoru pro nás znamená rychlost otáčení vrtule rotoru.
- **Stav kvadrokoptéry**
  Jedná se o hodnoty stavových proměnných kvadrokoptéry, mezi které patří:
  - pozice
  - rotace
  - rychlost
  - rychlost rotace

  Jelikož uvažujeme trojrozměrný prostor, jsou všechny tři parametry reprezentovány trojrozměrným vektorem reálných čísel.
- **Fyzikální vlastnosti kvadrokoptéry**
  V této práci budeme uvažovat hmotnost a rozměry kvadrokoptéry.
- **Popis okolí**
  Jedná se o geometrický popis prostředí, ve kterém se kvadrokoptéra pohybuje (např. budova). V této práci uvažujeme trojrozměrné prostředí.

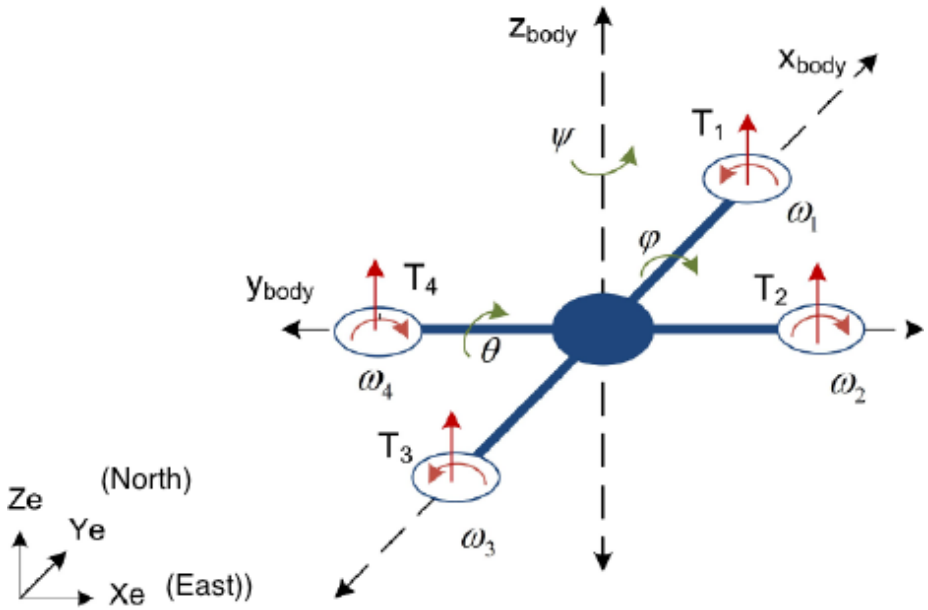
- **Plánovací úloha**
  Vstupními parametry plánovací úlohy jsou:
  - fyzikální vlastnosti kvadrokoptéry
  - popis 3D okolí
  - počáteční stav kvadrokoptéry
  - cílový stav kvadrokoptéry

- **Plán pohybu**
  Plán pohybu je časový vývoj hodnot aktuátorů. Pro zjednodušení budeme uvažovat diskrétní hodnoty na časové ose. Plán splňuje plánovací úlohu, pokud vede kvadrokoptéru z počátečního do cílového stavu, aniž by došlo ke kolizi s okolím.

1.2 Ovládání pohybu kvadrokoptéry

V této kapitole je vyložena základní teorie pohybu kvadrokoptéry. Poznatky jsou využity při implementaci plánovače pohybu.

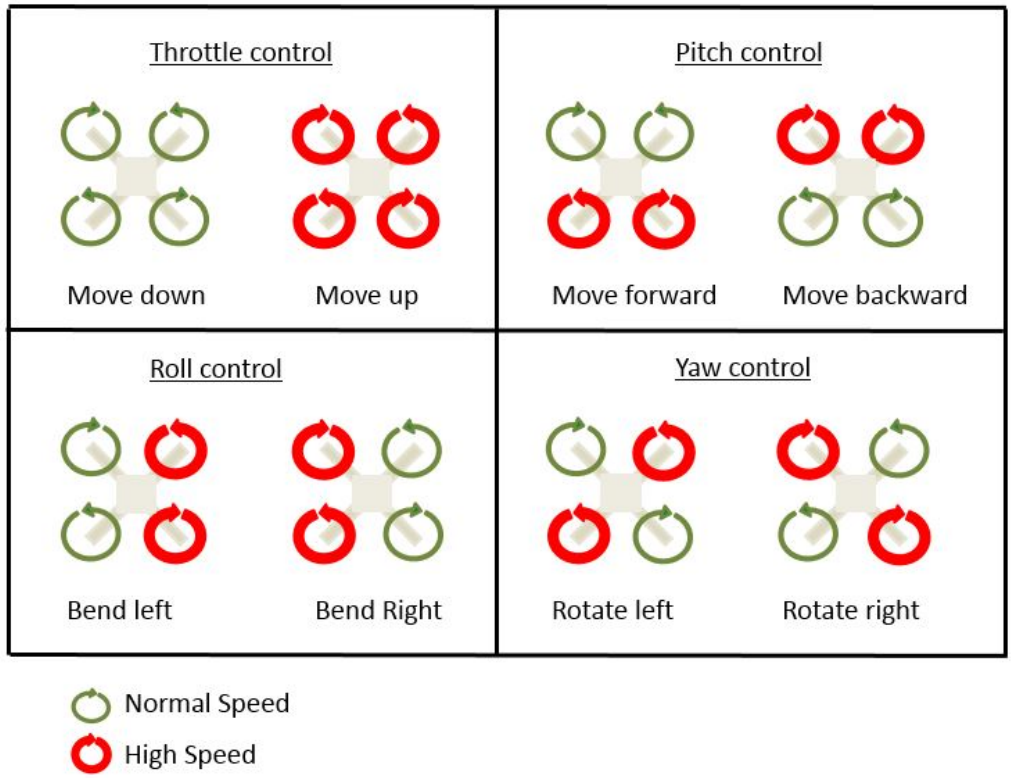
Uvažujme vztažnou soustavu souřadnic, kde osa X_e směřuje na východ, osa Y_e na sever a osa Z_e vzhůru (kolmo k zemi). Obrázek 1.1 znázorňuje model kvadrokoptéry skládající se ze čtyř aktuátorů: $T1$, $T2$, $T3$, $T4$ (přední, pravý, zadní, levý). Pozice kvadrokoptéry je dána souřadnicemi jejího středu vzhledem k počátku vztažné soustavy. Ze středu kvadrokoptéry vycházejí osy x_{body}, y_{body} a z_{body}. Rotace kolem těchto os jsou určeny úhly ϕ, θ a ψ (roll, pitch, yaw). Pokud jsou tyto tři úhly nulové, jsou osy x_{body}, y_{body}, z_{body} rovnoběžné s osami X_e, Y_e, Z_e.



Obrázek 1.1: Model kvadrokoptéry

Zdroj: `https://www.researchgate.net/profile/Mochammad_Ariyanto/publication/275211305/figure/fig2/AS:294491698286599@1447223700736/Fig-2-Quadrotor-coordinate-and-thrust-forces.png`

Pohyb kvadrokoptéry se řídí výhradně nastavením aktuátorů. Následující obrázek zobrazuje základní pohyby kvadrokoptéry a příslušná nastavení aktuátorů.



Obrázek 1.2: Ovládání pohybu kvadrokoptéry

Zdroj: `http://www.socialledge.com/sjsu/index.php?title=File:CmpE244_S14_Quadcopter_Quad_motion1.JPG`

Dron se vznese přímo vzhůru, pokud jsou hodnoty všech čtyř aktuátorů stejné a pokud je vztlaková síla vytvořená aktuátory větší než gravitační síla. Nastavením vyšších hodnot nějakých dvou aktuátorů oproti jiné dvojici lze dosáhnout různých pohybů. Tato teorie je využita při implementaci plánovače pohybu.

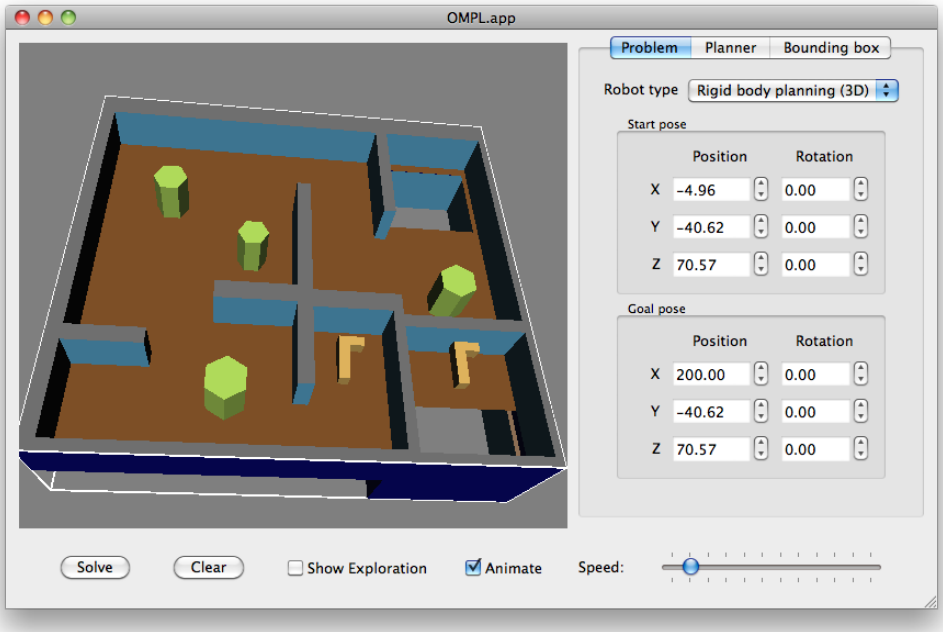
1.3 Rešerše softwarových balíků

Tato část obsahuje rešerši existujících softwarů pro plánování pohybu autonomních robotů.

1.3.1 OMPL

Open Motion Planning Library (OMPL) [10] je open source knihovna pro výpočet plánů pohybu. Obsahuje pouze sampling-based algoritmy pro plánování pohybu, nikoliv popis prostředí, detektor kolizí, vizualizaci, atd. Knihovna je navržena tak, aby ji bylo možné jednoduše integrovat s jinými systémy a frameworky (např. ROS nebo MoveIt!). Implementace je tedy dostatečně abstraktní, aby bylo možné potřebné komponenty implementovat. Knihovna OMPL je napsána v C++ s možností napojení na Python. K dispozici je také grafická aplikace OMPL.app napsaná v Pythonu, která slouží jako front-end k OMPL knihovně. K integraci Pythonu a C++ slouží tzv. Python bindings. Cílem aplikace je demonstrovat integraci s OMPL a poskytnout jednoduchý nástroj pro vyzkoušení plánovacích algoritmů. Plánovat pohyb lze buď pro volná tělesa, která nevyžadují vstupní kontrolní signály, nebo pro roboty vyžadující ovládání. OMPL obsahuje několik předdefinovaných typů robotů, mezi které patří i kvadrokoptéra. Je zde implementován jednoduchý, blíže nespecifikovaný matematický model kvadrokoptéry.

OMPL i grafická aplikace se distribuují ve formě zdrojových kódů. Kompilaci lze provést na OS Linux nebo Mac OS. Kompilace je poměrně náročná - vyžaduje alespoň 4 GB RAM a trvá několik hodin. I přes dodržení přesného postupu se mi nepodařilo grafickou aplikaci zprovoznit.



Obrázek 1.3: GUI pro OMPL

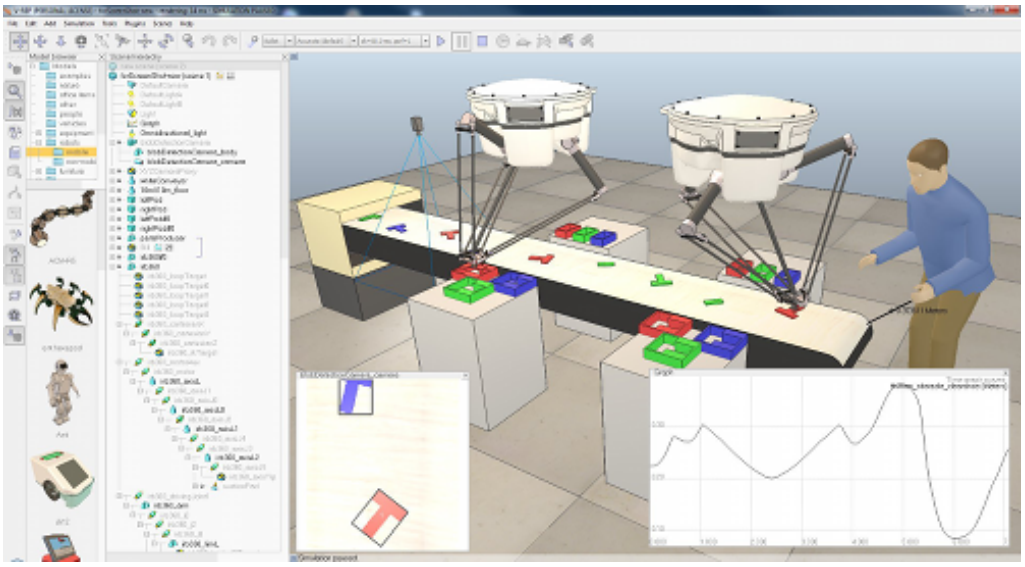
Zdroj: `http://ompl.kavrakilab.org/images/gui_define.png`

1.3.2 MoveIt!

MoveIt! [11] je framework pro plánování pohybu, který funguje nad ROS (Robot Operating System). ROS je open-source operační systém pro roboty, který poskytuje nízkoúrovňovou funkcionalitu, jako například ovladače hardwaru. MoveIt! poskytuje funkcionalitu pro kinematiku, plánování pohybu, detekci kolizí, a další. O plánování pohybu se typicky starají externí knihovny, např. OMPL.

1.3.3 V-REP

V-REP [12] je simulátor robotů, který obsahuje integrované vývojové prostředí (IDE). Plánování pohybu opět zajišťují externí knihovny.



Obrázek 1.4: V-REP IDE

Zdroj: `https://www.ode-wiki.org/wiki/images/e/e5/V-rep.png`

1.3.4 Závěr rešerše

Během hledání softwarových balíků pro plánování pohybu jsem nenašel žádný, který by se zaměřoval pouze na kvadrokoptéry. Výše zmíněné balíky jsou velmi komplexní. Mým cílem je vytvořit software, který bude jednoduchý nejen pro uživatele, ale i pro vývojáře.

Kapitola 2

Analýza

Tato kapitola popisuje funkční a nefunkční požadavky na aplikaci. Dále jsou vyjmenovány scénáře užití a je proveden výběr technologií.

2.1 Funkční požadavky

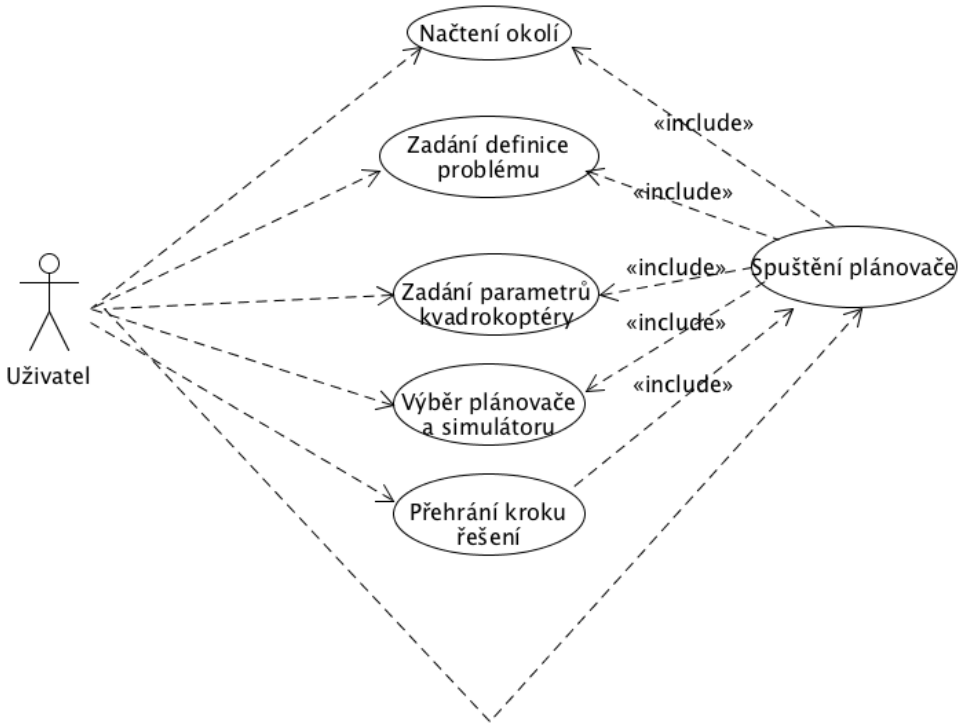
Na základě zadání byly stanoveny následující funkční požadavky, které budou implementovány.

- F1: Uživatel bude mít možnost nahrát popis okolí ze souboru.
- F2: Uživatel bude mít možnost zvolit algoritmus plánování a modul pro simulaci pohybu.
- F3: Aplikace bude obsahovat simulátor pohybu implementovaný podle [4].
- F4: Aplikace bude obsahovat plánovač pohybu implementovaný podle [2].
- F5: Aplikace bude obsahovat modul pro grafickou vizualizaci plánu pohybu.
- F6: Uživatel bude mít možnost zadat definici problému a fyzikální vlastnosti kvadrokoptéry.
- F7: Aplikace bude zobrazovat jednotlivé kroky plánovače.
- F8: Uživatel bude mít možnost zpětně přehrát jednotlivé kroky plánovače.

2.2 Nefunkční požadavky

- N1: Aplikace poběží alespoň pod operačním systémem Linux.

2.3 Scénáře užití



Obrázek 2.1: Scénáře užití

2.4 Výběr technologií

Rozhodoval jsem se mezi dvěma programovacími jazyky: Java a C++. Jelikož má aplikace umožňovat grafickou demonstraci plánování pohybu, bylo potřeba zvolit nějakou grafickou knihovnu nebo framework s podporou 3D grafiky. Jednou z možností bylo vyvinout celou aplikaci na platformě Java s využitím Java 3D API [6]. Druhou možností bylo zvolit jazyk C++ spolu s nějakým frameworkem, kterých existuje celá řada. Nejvíce mě zaujal framework Qt [5]. Oba přístupy (C++ a Java) nabízejí v podstatě stejnou funkcionalitu. Z důvodu osobních preferencí a lepší podpory 3D grafiky byla zvolena druhá možnost, tedy jazyk C++ a framework Qt.

Qt je multiplatformní framework pro vytváření aplikací s grafickým uživatelským rozhraním (GUI). Podporuje nejen desktopové, ale i mobilní platformy. Tento framework nabízí vše, co potřebujeme, tedy tvorbu GUI, podporu 3D grafiky, načítání 3D formátů a tvorbu modulárních aplikací. Qt nabízí své IDE s názvem Qt Creator, které jsem pro vývoj použil. Použil jsem Qt ve verzi 5.7.1. Abych měl jistotu, že aplikace poběží správně pod OS Linux, probíhal vývoj na tomto systému (Ubuntu 14.04.5).

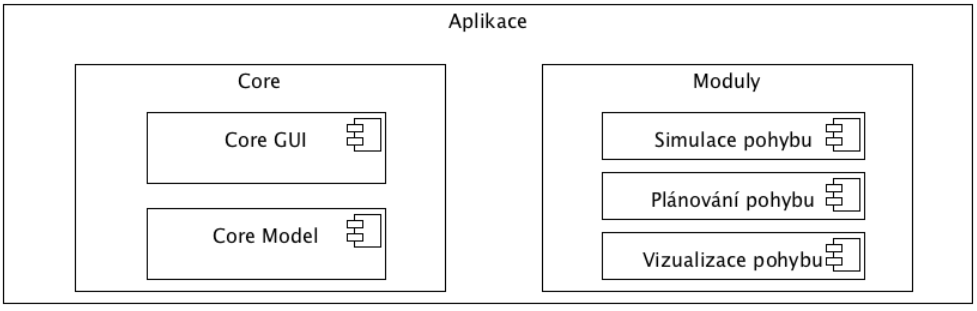
Kapitola 3

Návrh

V této kapitole je proveden návrh aplikace. Nejprve jsou popsány hlavní komponenty a představeny jednotlivé třídy. Následuje popis rozhraní jednotlivých modulů a sekvenční diagram zobrazující spolupráci modulů. Součástí této kapitoly je také návrh uživatelského rozhraní.

3.1 Architektura aplikace

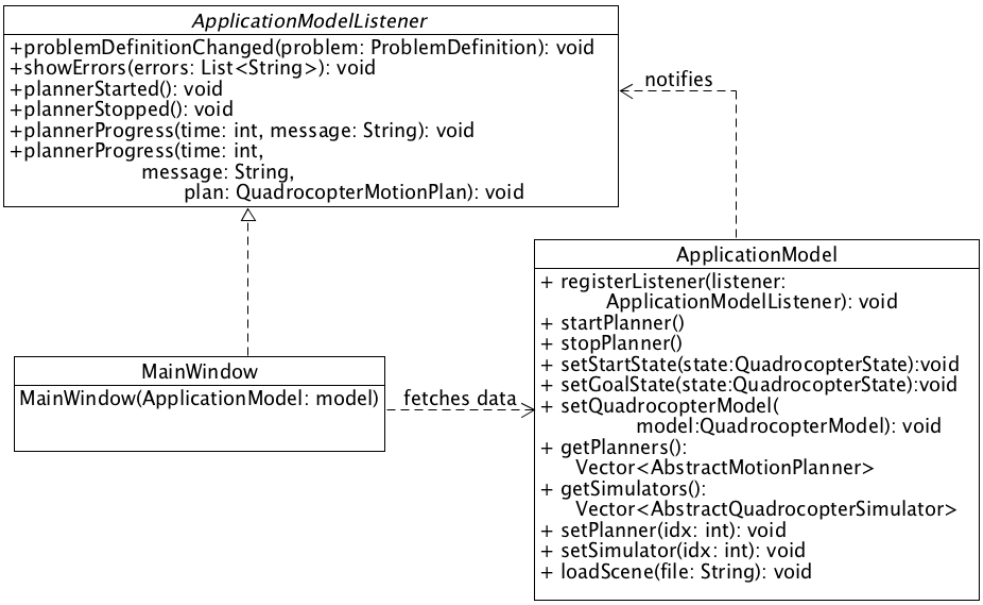
Hlavní komponenty aplikace jsou tzv. core view a core model. Core view, reprezentující hlavní GUI aplikace, slouží k základnímu ovládání aplikace. Aplikační logika je uložena v core modelu. Vedle těchto core komponent jsou součástí architektury také moduly pro simulaci, plánování a vizualizaci pohybu. Moduly jsou řízeny komponentou core model.



Obrázek 3.1: Hlavní architektura aplikace

3.1.1 Model a view

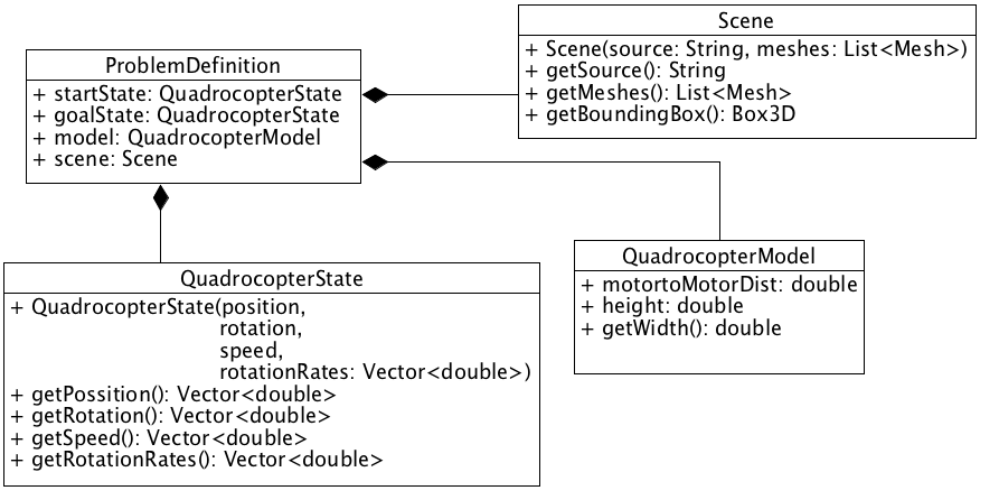
Oddělení aplikační logiky od zobrazování dat je realizováno pomocí tříd `ApplicationModel` reprezentující model a `MainWindow` reprezentující view. View pouze zobrazuje data a přijímá vstupy od uživatele, které následně deleguje do modelu. Model informuje view o změnách pomocí rozhraní `ApplicationModelListener`, čímž je dosaženo nezávislosti modelu na view, jak je vidět v následujícím diagramu.



Obrázek 3.2: Model a view

3.1.2 Reprezentace plánovací úlohy

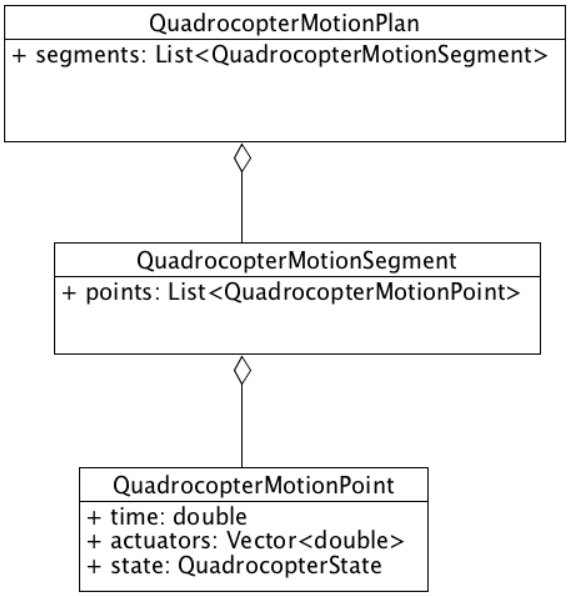
Plánovací úloha je definována startovním a cílovým stavem kvadrokoptéry (`QuadrocopterState`), fyzikálními vlastnostmi kvadrokoptéry (`QuadrocopterModel`) a prostorem, ve kterém se kvadrokoptéra pohybuje (`Scene`).



Obrázek 3.3: Reprezentace plánovací úlohy

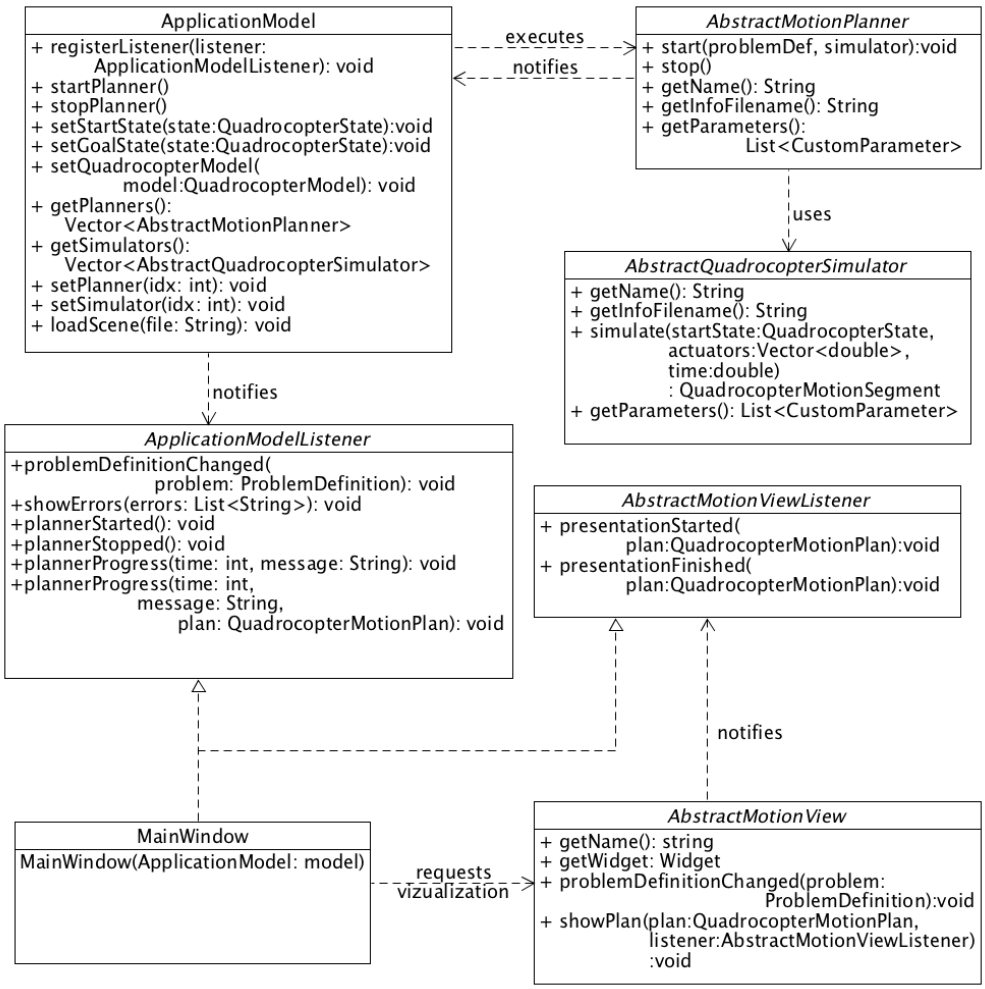
3.1.3 Reprezentace plánu pohybu

Plán pohybu (`QuadrocopterMotionPlan`) se skládá z jednotlivých segmentů (`QuadrocopterMotionSegment`). Segmenty se skládají z bodů (`QuadrocopterMotionPoint`). Každý bod obsahuje čas, nastavení aktuátorů a stav kvadrokoptéry.



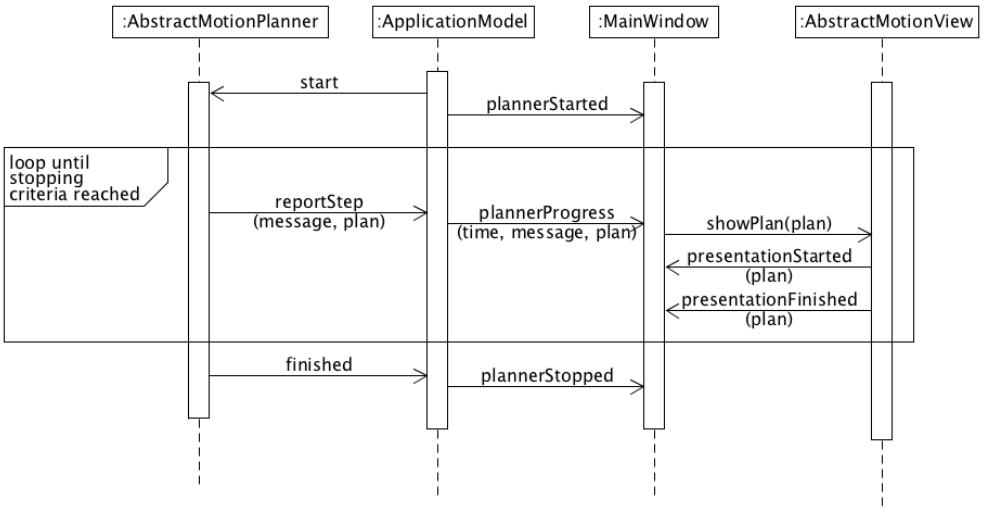
Obrázek 3.4: Reprezentace plánu pohybu

3.1.4 Zapojení plánovače, simulátoru a vizualizace



Obrázek 3.5: Zapojení plánovače, simulátoru a vizualizace

Následuje sekvenční diagram, který zobrazuje spolupráci tříd při spuštění plánovače.



Obrázek 3.6: Sekvenční diagram

3.2 Modul simulátor pohybu

Simulátor pohybu kvadrokoptéry je popsán třídou `AbstractQuadrocopterSimulator`. Její klíčová metoda je `simulate`, která přijímá následující parametry:

- fyzikální vlastnosti kvadrokoptéry (`QuadrocopterModel`)
- počáteční stav kvadrokoptéry (`QuadrocopterState`)
- požadované nastavení jednotlivých aktuátorů (úhlové rychlosti)
- požadovanou délku simulace v sekundách
- požadovanou velikost kroku v sekundách

Cílem této metody je provést simulaci pohybu kvadrokoptéry při daném počátečním stavu a daném nastavení aktuátorů. Metoda vrací segment plánu pohybu. Tato třída by v sobě měla mít zakódovaný matematický model kvadrokoptéry, který je typicky reprezentován soustavou diferenciálních rovnic. Vyřešením soustavy rovnic pro dané počáteční podmínky lze získat časový vývoj stavu kvadrokoptéry.

3.3 Modul plánovač pohybu

Dalším modulem této aplikace je plánovač pohybu, který je popsán třídou `AbstractMotionPlanner`. Cílem plánovače je vyřešit problém plánování, tedy vrátit takový plán, který splňuje uživatelem zadanou úlohu. K tomu slouží metoda `start`, která přijímá dva parametry: definici problému a simulátor pohybu, který se má při řešení použít. `ApplicationModel` má seznam plánovačů. Vybraný plánovač je modelem přesunut na nové vlákno a spuštěn pomocí metody `start`. Třída `AbstractMotionPlanner` definuje následující signály:

- `void reportStep(QString message, QuadrocopterMotionPlan *plan)` Slouží k reportování kroku algoritmu. Součástí je popis kroku a volitelně plán pohybu.
- `void finished(QString message)` Slouží k oznámení o ukončení algoritmu.
- `void error(QString message)` Vyvolán při výskytu chyby (např. když úloha nemá řešení).

Model tyto signály přijímá pomocí svých slotů a předává vytvořený plán vizualizéru.

3.4 Modul vizualizace plánu pohybu

Abstraktní třída `AbstractMotionView` reprezentuje zobrazovač plánu pohybu. Jednou z čistě virtuálních metod je `getWidget`. Pomocí ní třída vrací ukazatel na `QWidget`, což je obecný grafický prvek v Qt. Tento prvek bude zobrazen v pravé části okna aplikace. Metoda `showPlan` je vyvolána v okamžiku, kdy vznikne nový plán pohybu a je potřeba ho zobrazit. O zahájení a ukončení přehrávání je notifikován `AbstractMotionViewListener`. Plán pohybu musí být zobrazen pomocí instance `QWidget`, kterou vrací metoda `getWidget`. Metoda `problemDefinitionChanged` je volána, kdykoliv dojde ke změně definice problému. Lze tak například zobrazit počáteční a cílový stav, vykreslit scénu, atd.

3.5 Parametrizace modulů

Moduly plánovač a simulátor je možné jednoduše parametrizovat. Chce-li modul definovat vlastní parametry, využije k tomu metodu `getCustomParameters`.

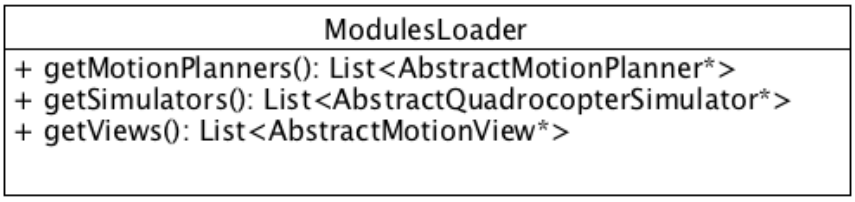
Pomocí ní vrátí seznam obsahující instance třídy `CustomParameter`, které se skládají ze jména, popisu a hodnoty. Parametry jsou zobrazeny uživateli k náhledu a editaci pomocí core GUI.

Kontrolu parametrů lze provést v metodě `check`, která se volá před spuštěním plánovače. Modul tak má možnost zkontrolovat uživatelem zadané parametry a případně vrátit seznam chyb. Plánovač je spuštěn pouze při nulovém počtu chyb.

3.6 Rozšířitelnost pomocí pluginů

Jedním z požadavků na aplikaci byla modularita, tedy snadná výměna a možnost přidávání plánovače, simulátoru a vizualizace. Tento požadavek bude realizován pomocí pluginů. Pluginy jsou v Qt implementovány jako sdílené knihovny, které lze načítat za běhu aplikace. Máme zde 3 druhy modulů, které jsou popsány rozhraními: `AbstractMotionPlanner`, `AbstractQuadrocopterSimulator` a `AbstractMotionView`. Pro vytvoření nového modulu je potřeba vytvořit nový plugin (sdílenou knihovnu), který musí obsahovat třídu implementující jedno ze tří rozhraní. Sestavený plugin má koncovku `.so` na OS Unix/Linux a koncovku `.dll` na OS Windows. Plugin je třeba umístit do složky `plugins`, která se nachází v instalační složce aplikace.

O načítání pluginů se stará třída `ModulesLoader`. Načítání probíhá tak, že je prováděna iterace přes všechny soubory ve složce plugins. Každý soubor je předán třídě `QPluginLoader`, která se z něj snaží získat třídu, kterou plugin exportuje, a dále přetypovat na jedno z rozhraní. Načítání pluginů inicializuje instance třídy `ApplicationModel` během jejího vytváření. Třída `ModulesLoader` vrací výchozí moduly, které jsou součástí aplikace, spolu s nalezenými moduly v pluginech.



Obrázek 3.7: Třída ModulesLoader

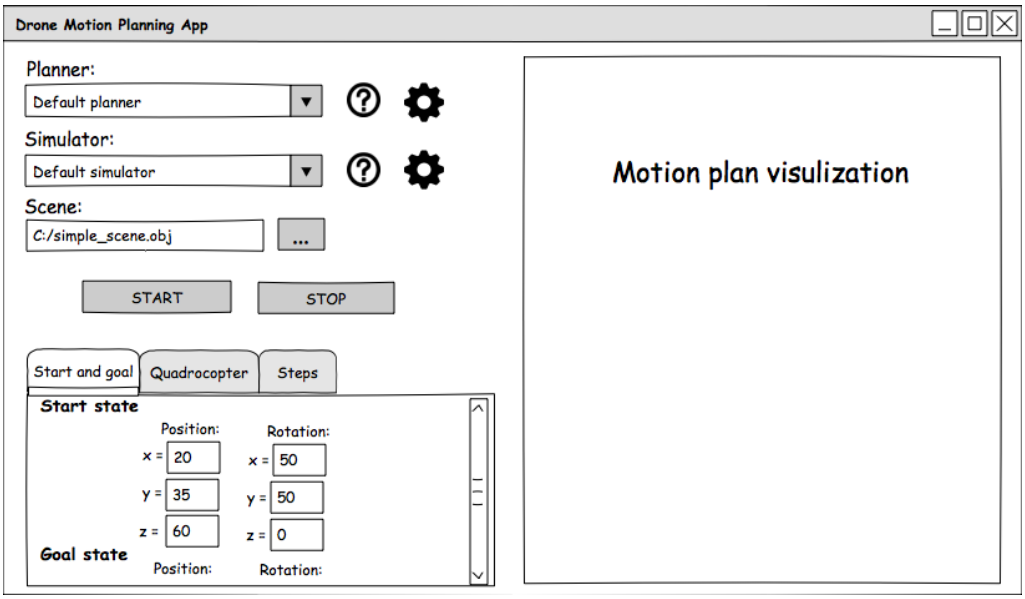
3.7 Uživatelské rozhraní

Jelikož je cílem vytvořit aplikaci pro grafickou demonstraci plánování pohybu, je celé uživatelské rozhraní grafické. Při návrhu uživatelského rozhraní jsem se snažil vycházet z desatera bodů použitelnosti podle Jakoba Nielsena [1]. Jedná se o soubor pravidel, které by měly být dodrženy, aby bylo rozhraní uživatelsky přívětivé a dobře ovladatelné.

Aplikace se skládá z jediného okna, které je rozděleno na levou a pravou část. Levá část slouží k zadání definice problému, parametrů, atd. Konkrétně tato část obsahuje následující ovládací prvky:

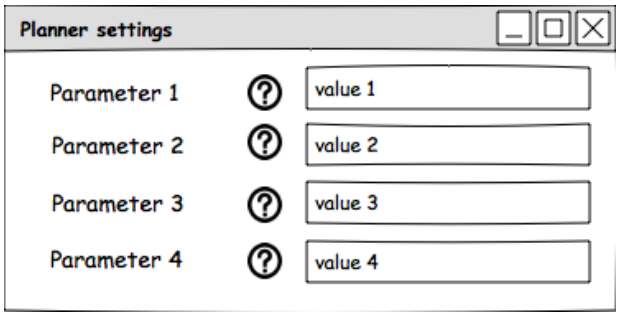
- 2 rozbalovací menu pro výběr plánovače a simulátoru
- tlačítka pro zobrazení informací o plánovači a simulátoru
- tlačítka pro editaci parametrů plánovače a simulátoru
- tlačítko pro výběr scény
- 3 záložky
  - zadání počátečního a cílového stavu kvadrokoptéry
  - zadání fyzikálních vlastností kvadrokoptéry
  - tabulka zobrazující jednotlivé kroky plánovače
- tlačítka start a stop pro spuštění a ukončení plánovače

Celá pravá část okna je vyhrazena pro zobrazování plánu pohybu. Vzhled a chování této části je plně v kompetenci konkrétního modulu pro vizualizaci. Vizualizace nemusí být jen grafická, může být např. textová. Níže se nachází navržený prototyp uživatelského rozhraní. První záložka umožňuje zadat startovní a cílovou pozici kvadrokoptéry.



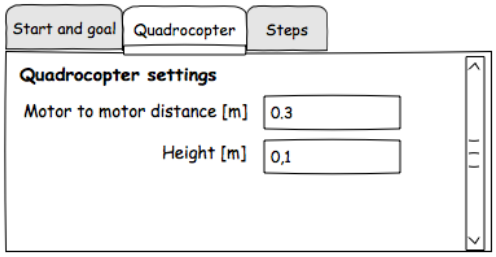
Obrázek 3.8: Uživatelské rozhraní - zadání definice problému

Poklepáním na ikonu otazníku dojte k zobrazení nového okna s popisem daného modulu (plánovače nebo simulátoru). Ikona ozubeného kola slouží k otevření nového okna určeného k editaci parametrů daného modulu. Jednotlivé parametry jsou zobrazeny pod sebou. Každý řádek obsahuje název parametru, textové pole pro zadání hodnoty a ikonu otazníku pro zobrazení popisu parametru.



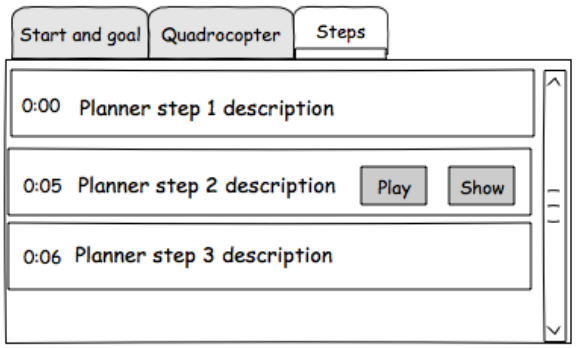
Obrázek 3.9: Editace parametrů modulu

Pro zadání fyzikálních vlastností kvadrokoptéry slouží druhá záložka.



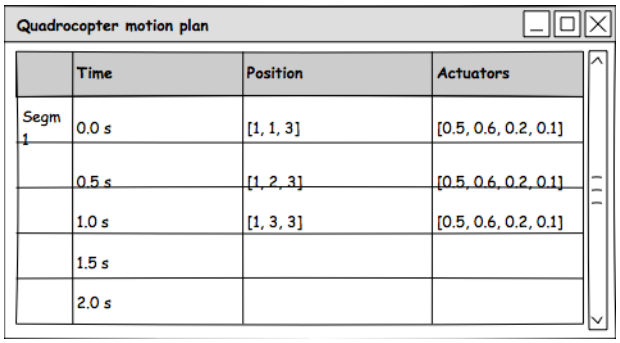
Obrázek 3.10: Uživatelské rozhraní - fyzikální vlastnosti kvadrokoptéry

Kliknutím na tlačítko START dojde ke spuštění plánovače. Tlačítko se deaktivuje a naopak se aktivuje tlačítko STOP pro ukončení plánovače. Ve třetí záložce jsou zobrazování jednotlivé kroky plánovače. Každý krok obsahuje čas, popis a dále, v případě existence plánu pohybu, je zde tlačítko Play pro přehrání plánu a tlačítko Show zobrazení plánu pohybu v novém okně.



Obrázek 3.11: Uživatelské rozhraní - zobrazení kroků plánovače

Kliknutím na tlačítko Show tedy dojde k otevření nového okna s tabulkou zobrazující plán pohybu. Každý řádek tabulky reprezentuje jeden časový okamžik, který se skládá z času, stavových proměnných kvadrokoptéry a hodnot aktuátorů.



Obrázek 3.12: Detailní zobrazení vytvořeného plánu pohybu

Kapitola 4

Implementace

V této kapitole je nejprve vysvětlen způsob načítání scény a detekce kolizí. Zbytek kapitoly je věnován popisu implementace všech tří modulů (simulace, plánování a vizualizace pohybu).

4.1 Načítání scény

Jednou z částí definice problému je scéna, neboli okolí, ve kterém se kvadrokoptéra pohybuje. Definice scény je řešena pomocí 3D grafických formátů. O načítání scény se stará třída `SceneLoader`, která vnitřně využívá knihovnu Assimp (Open Asset Import Library) [7]. Tato knihovna, která je součástí Qt, podporuje širokou škálu 3D formátů, například Collada, Blender 3D, Wavefront Object a mnoho dalších. Kompletní seznam lze najít na `http://www.assimp.org/main_features_formats.html`.

Třída `SceneLoader` má metodu `loadScene`, která pro daný soubor ve 3D formátu použije knihovnu Assimp pro jeho načtení a vrátí instanci třídy `Scene`. Knihovna Assimp vrací strukturu `aiScene`, která popisuje celou scénu. Scéna se skládá mimo jiné z částí označených jako mesh. Mesh popisuje geometrii dané části - tedy souřadnice vrcholů, jejich propojení, atd. Aby naše třída `Scene` nebyla závislá na knihovně Assimp, překopírujeme potřebná data ze struktury `aiScene`. Třída `Scene` by měla popisovat především geometrii. V našem případě bude scéna popisovat interiér budovy, případně venkovní oblast, kde můžeme jednotlivé objekty pro zjednodušení vnímat jako kvádry. Kopírovat budeme tedy jen meshe a vrcholy, propojením vrcholů se nebudeme zabývat. Toto rozhodnutí souvisí i s detektorem kolizí, který je popsán v následující sekci.

Byla vytvořena třída `Mesh`, která obsahuje seznam vrcholů a metodu `getBoundingBox` pro získání ohraničujícího kvádru. Třída `Scene` tedy ob-

sahuje zdrojový soubor a seznam instancí třídy `Mesh`. Třídu `Scene` je možné v budoucnu vylepšit tak, aby detailněji popisovala geometrii scény.

I přes to, že by Qt 3D mělo podporovat všechny formáty dostupné z `http://www.assimp.org/main_features_formats.html`, jsem měl s některými formáty problém. Scéna se nevykreslila vůbec, případně se vykreslila jen její část. Velmi záleželo i na programu, pomocí kterého byly 3D scény vytvářeny. Nakonec se osvědčil program FreeCAD [14] a formát Collada (.dae). Tento program a formát jsou tedy doporučeny pro tvorbu scén.

4.2 Detekce kolizí

V průběhu řešení plánovací úlohy je potřeba detekovat kolize kvadrokoptéry s okolím. Použití detektoru kolizí plně závisí na plánovači. V rámci této práce byl implementován jednoduchý detektor kolizí `DefaultCollisionChecker`, který přijímá model kvadrokoptéry, její aktuální stav a scénu. Tento detektor funguje tak, že projde všechny `Mesh` objekty, které scéna obsahuje, a pro každý z nich zjistí, zda koliduje s kvadrokoptérou. Pro zjednodušení se porovnávají ohraničující boxy rovnoběžné s osami souřadnic (axis-aligned bounding box). Třída `Mesh` má metodu `getBoundingBox`. Ohraničující box kvadrokoptéry je pro zjednodušení krychle, jejíž hrana má délku rovnou nejdelší hraně kvadrokoptéry vynásobené $\sqrt{2}$. To zaručí, že se kvadrokoptéra do boxu vejde při jakékoliv rotaci.

Způsob získání ohraničujícího boxu je snadný. Projdeme všechny body meshe a pamatujeme si nejmenší souřadnice (x_{min}, y_{min}, z_{min}) a největší souřadnice (x_{max}, y_{max}, z_{max}). Ohraničující box je potom definován vrcholy $[x_{min}, y_{min}, z_{min}]$ a $[x_{max}, y_{max}, z_{max}]$.

Dva ohraničující boxy A a B spolu kolidují, pokud platí všechny následující podmínky:

- $A.x_{min} <= B.x_{max} \wedge A.x_{max} >= B.x_{min}$
- $A.y_{min} <= B.y_{max} \wedge A.y_{max} >= B.y_{min}$
- $A.z_{min} <= B.z_{max} \wedge A.z_{max} >= B.z_{min}$

4.3 Simulátor pohybu

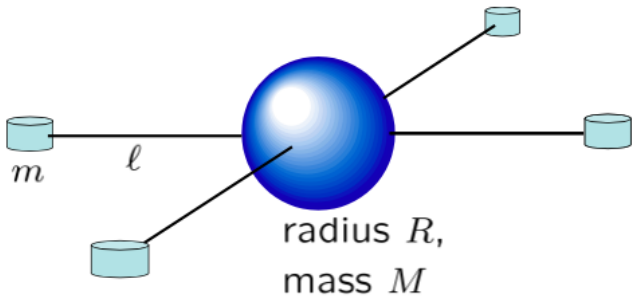
Tato kapitola popisuje implementaci modulu pro simulování pohybu. Nejprve je představen matematický model kvadrokoptéry, na kterém je simulátor založen. Následně je popsána implementace simulace pohybu s použitím řešiče diferenciálních rovnic.

4.3.1 Matematický model kvadrokoptéry

Implementovaný simulátor pohybu je založen na matematickém modelu [4]. Model pracuje s následujícími dvanácti stavovými proměnnými:

Stavová proměnná	Popis
p_n	pozice X
p_e	pozice Y
h	pozice Z
u	rychlost X
v	rychlost Y
w	rychlost Z
ϕ	roll
θ	pitch
ψ	yaw
p	úhlová rychlost otáčení kolem osy X
q	úhlová rychlost otáčení kolem osy Y
r	úhlová rychlost otáčení kolem osy Z

Kvadrokoptéra je idealizována tak, že její střed má tvar koule s poloměrem R a hmotností M. Rotory se nachází ve vzdálenosti l a mají hmotnost m.



Obrázek 4.1: Model kvadrokoptéry

Zdroj: [4]

Aktuátory δ_f, δ_r, δ_b a δ_l se přepočítávají na točivé momenty τ_ϕ, τ_θ a τ_ψ podle rovnice

$$\begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} k_1 & k_1 & k_1 & k_1 \\ 0 & -lk_1 & 0 & lk_1 \\ lk_1 & 0 & lk_1 & 0 \\ -k_2 & k_2 & -k_2 & k_2 \end{pmatrix} \begin{pmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{pmatrix}.$$

Dalšími vstupními proměnnými jsou J_x, J_y, J_z a lze je spočítat podle vzorců:

$$J_x = J_y = \frac{2MR^2}{5} + 2l^2m$$
$$J_z = \frac{2MR^2}{5} + 4l^2m.$$

Samotná soustava diferenciálních rovnic popisující pohyb kvadrokoptéry vypadá následovně:

$$\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{h} \end{pmatrix} = \begin{pmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ s\theta & -s\phi c\theta & -c\phi c\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \begin{pmatrix} -q\sin\theta \\ q\cos\theta\sin\phi \\ q\cos\theta\cos\phi \end{pmatrix} + \frac{1}{m}\begin{pmatrix} 0 \\ 0 \\ -F \end{pmatrix}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{pmatrix} + \begin{pmatrix} \frac{1}{J_x}\tau_\phi \\ \frac{1}{J_y}\tau_\theta \\ \frac{1}{J_z}\tau_\psi \end{pmatrix}$$

Kompletní popis matematického modelu je uveden v dokumentu [4] v kapitolách 2 a 3. Vyřešením ODE systému pro nějaké počáteční hodnoty stavových proměnných získáme časový vývoj těchto proměnných.

4.3.2 Řešič soustavy diferenciálních rovnic

Pro vyřešení ODE systému byla zvolena C++ knihovna ODEINT [8], která je součástí Boost C++ knihoven. ODEINT s pracuje s polem stavových proměnných (náš matematický model se skládá z 12 proměnných, pole má tedy

délku 12), které musí být na začátku inicializovány. Dále je potřeba reprezentovat ODE systém a předat ho řešiči. To lze provést implementováním funkce se signaturou

```
void ode_system(const state_type &x,
                state_type &dxdt,
                const double t).
```

Datový typ `state_type` je v našem případě pole délky 12. Proměnná `t` reprezentuje čas. Tělo funkce musí obsahovat přepis daného ODE systému. Jako příklad uveďme rovnici

$$s'(t) = f(s, t).$$

Její přepis bude vypadat následovně:

`dxdt[i] = f(x[i], t)`,
kde `i` je index proměnné `s`.

Pro získání výsledků je potřeba naimplementovat funkci se signaturou

`void observer(const state_type &x , const double t)`.

Funkce je volána během integrování, kdykoliv jsou k dispozici nové hodnoty. Řešič lze spustit pomocí funkce `integrate`, které je potřeba předat funkci reprezentující ODE systém, počáteční hodnoty proměnných, počáteční a koncový čas a observer.

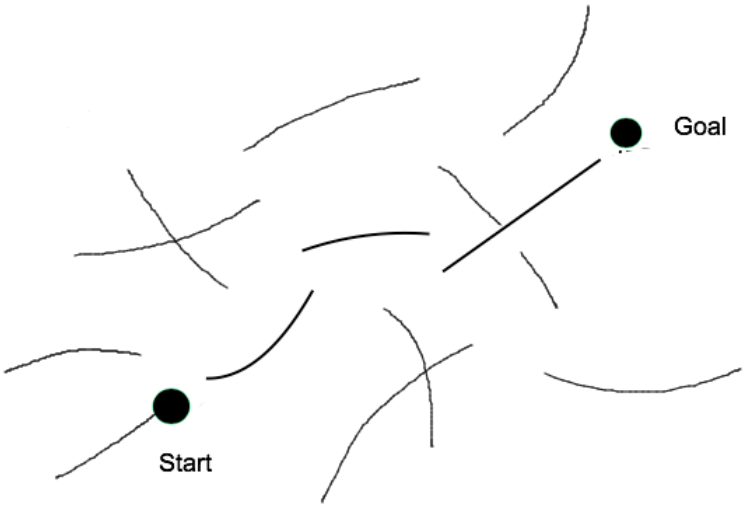
4.4 Plánovač pohybu

V rámci této aplikace byl implementován algoritmus pro plánování pohybu podle dokumentu [2]. Algoritmus byl implementován jako modul aplikace. Zmíněný dokument popisuje algoritmus pro falzifikaci hybridního systému, neboli nalezení takové trajektorie, která vede systém z počátečního do nebezpečného stavu. V tomto algoritmu provedeme drobnou úpravu: místo nebezpečného stavu budeme uvažovat cílový stav kvadrokoptéry.

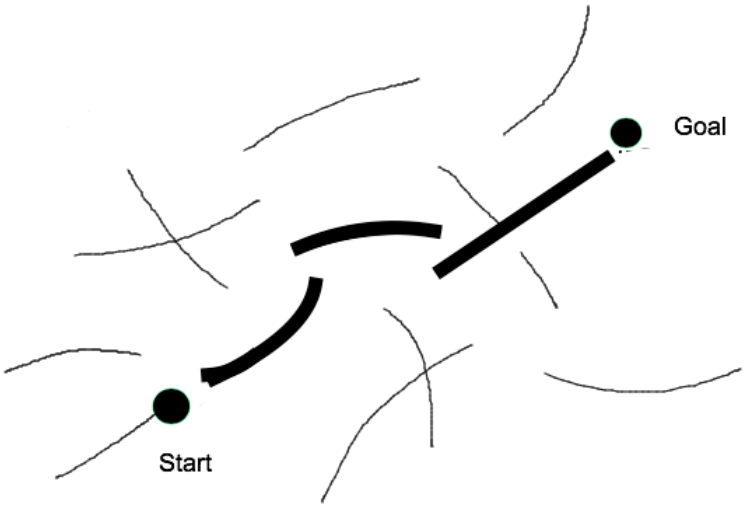
4.4.1 Algoritmus pro plánování pohybu

Tato sekce popisuje algoritmus převzaný z [2]. Nejprve je uvedena základní myšlenka algoritmu a jsou definovány potřebné pojmy. Následně je algoritmus popsán formou pseudokódu.

Základní myšlenku algoritmu znázorňují následující 2 obrázky:



Obrázek 4.2: Algoritmus plánování pohybu - generování segmentů



Obrázek 4.3: Algoritmus plánování pohybu - nalezení nejkratší cesty

Nejprve je spuštěno velké množství simulací, které jsou náhodně rozmístěny v prostoru. Výsledkem tedy bude velké množství krátkých, nespojených segmentů (obrázek 4.2).

Z nich jsou vybrány takové segmenty, které tvoří nejkratší cestu ze startu do cíle (obrázek 4.3). Následně se opakují dvě fáze:

- Na nejkratší cestu je aplikována některá z metod pro lokální optimalizaci s cílem vytvořit nový plán s nižší cenou
- Přidání nového segmentu do prostoru

Definice 1 *Stav kvadrokoptéry je čtveřice trojrozměrných vektorů reálných čísel:*

- *pozice*
- *rotace*
- *rychlost*
- *rychlost změny rotace*

Definice 2 *Segment řešení je dvojice (x, t), kde x je stav kvadrokoptéry a t je nezáporné reálné číslo. Bod x se nazývá startovní bod segmentu řešení. Číslo t značí délku segmentu řešení, která je měřena v časových jednotkách (např. sekundy).*

Definice 3 *Stav kvadrokoptéry y nazýváme koncovým bodem segmentu (x, t), pokud je y koncovým bodem simulace o délce t z bodu x.*

Abychom mohli označit dva stavy kvadrokoptéry jako stejné, je nutné, aby se rovnaly všechny 4 stavové proměnné. Představme si stavové proměnné jako vektor reálných čísel. Potom můžeme definovat vzdálenost dvou stavů kvadrokoptéry *stateDist* jako Euklidovu vzdálenost.

Množinu segmentů řešení S si lze představit jako ohodnocený graf, jehož vrcholy jsou:

- startovní body segmentů
- koncové body segmentů
- počáteční stav kvadrokoptéry
- cílový stav kvadrokoptéry

Váhu hrany mezi vrcholy v_1 a v_2 označme $w(v_1, v_2)$. Definujeme $w(v_1, v_2) = 0$, pokud v_2 je koncový bod segmentu řešení se startovním bodem v_1. V opačném případě platí $w(v_1, v_2) = stateDist(v_1, v_2)$.

Definice 4 *Kandidátní trajektorie je posloupnost segmentů (x_1, t), ... (x_n, t).*

Chceme najít trajektorii z počátečního stavu do cílové stavu. Kandidátní trajektorie musí splňovat následující podmínky:

- startovní bod prvního segmentu je roven počátečnímu stavu kvadrokoptéry
- koncový bod segmentu je roven startovnímu bodu následujícího segmentu
- koncový bod posledního segmentu je roven cílovému stavu kvadrokoptéry

Obecný algoritmus pro plánování pohybu autonomního robota lze popsat následujícím pseudokódem:

Algorithm 1 Plánování pohybu

1: $S \leftarrow \{\}$
2: Vygeneruj velké množství sekvencí a přidej je do množiny S.
3: *cand* $\leftarrow$ trajektorie s nejnižší cenou z S
4: **while** optimalizace trajektorie *cand* pomocí minimalizace cenové funkce neprodukuje dostatečně dobré řešení **do**
5: přidej nový segment do množiny S
6: *cand* $\leftarrow$ trajektorie s nejnižší cenou z S
7: **end while**

4.4.2 Implementace algoritmu pro plánování pohybu

V této sekci jsou popsány implementační detaily výše zmíněného algoritmu, který byl upraven pro potřeby plánování pohybu kvadrokoptér. Algoritmus je nejprve představen ve formě pseudokódu. Následně jsou jeho zásadní kroky slovně vysvětleny.

Algorithm 2 Plánování pohybu kvadrokoptér

1: $optimalActuators \leftarrow$ najdi hodnoty aktuátorů, které způsobí přibližné vznášení se kvadrokoptéry
2: $S \leftarrow \{\}$
3: Vygeneruj velké množství náhodných segmentů s použitím $optimalActuators$ a přidej je do množiny S.
4: Odstraň segmenty, které kolidují s okolím.
5: $cand \leftarrow$ segmenty z S tvořící nejkratší cestu ze startu do cíle
6: **while** není splněno žádné ukončovací kritérium **do**
7: optimalizuj $cand$ pomocí lokální optimalizace
8: vygeneruj nový segment s použitím $optimalActuators$ a přidej ho do množiny S
9: $cand \leftarrow$ trajektorie s nejnižší cenou z S
10: return $cand$
11: **end while**

4.4.2.1 Zjištění optimálního nastavení aktuátorů

Aby byl let kvadrokoptéry realistický, je potřeba zjistit optimální nastavení otáček rotorů. To závisí nejen na matematickém modelu (který je zakódován v simulátoru), ale také na fyzikálních parametrech kvadrokoptéry (hmotnost, rozměry, ...). Cílem je tedy zjistit, jaké nastavení aktuátorů způsobí vznášení se kvadrokoptéry. Hledání je realizováno pomocí binárního půlení s použitím modulu simulátor. Hodnoty aktuátorů jsou nastaveny na stejnou hodnotu, která se následně snižuje, resp. zvyšuje, podle toho jestli kvadrokoptéra stoupá, resp. klesá. Plánovač si nalezenou hodnotu a zapamatuje a následně ji využívá v dalších krocích algoritmu.

4.4.2.2 Počáteční generování segmentů

Plánovač definuje parametry, pomocí kterých lze nastavit počet segmentů a délku segmentu v sekundách. Generování segmentů je randomizované. Plánovač nejprve zjistí ohraničující box scény, ve kterém náhodně volí počáteční pozice segmentů. Simulace by ideálně měly odpovídat pohybům popsaným v sekci 1.2. Je tedy potřeba zajistit, aby kvadrokoptéra nejen stoupala přímo vzhůru a klesala přímo dolu, ale aby se pohybovala i jinými směry. K tomu je potřeba změnit hodnoty některých aktuátorů. Jako základ je použita hodnota $optimalActuators$, nalezená v přechozím odstavci. Dále bylo experimentálně zjištěno, že stačí změnit aktuátory o jedno procento, aby se změnil směr pohybu. Při simulacích se tedy střídavě používají následující hodnoty

aktuátorů, kde $OPT = optimalActuators$, $low = optimalActuators \cdot 0.99$ a $high = optimalActuators \cdot 1.01$:

- $(high, high, high, high)$
- (low, low, low, low)
- $(high, high, OPT, OPT)$
- $(OPT, high, high, OPT)$
- $(OPT, OPT, high, high)$
- $(high, OPT, OPT, high)$
- $(high, OPT, high, OPT)$
- $(OPT, high, OPT, high)$

4.4.2.3 Odstranění segmentů kolidujících s okolím

Segment je reprezentován třídou `QuadrocopterMotionSegment` a skládá se z jednotlivých bodů (`QuadrocopterMotionPoint`). Jedním z atributů každého bodu je stav kvadrokoptéry (`QuadrocopterState`). Plánovač iteruje přes všechny stavy segmentu a pokud některý z nich koliduje s okolím, je takový segment odstraněn. Pro kontrolu kolizí je použit `DefaultCollisionChecker`.

4.4.2.4 Nalezení kandidátní trajektorie s nejnižší cenou

Problém nalezení trajektorie s nejnižší cenou je převeden na problém hledání nejkratší cesty v ohodnoceném grafu (graf je popsán v sekci 4.4.1). Pro implementaci byl zvolen Dijkstrův algoritmus.

4.4.2.5 Lokální optimalizace

Lokální optimalizace je klíčový krok algoritmu. Jejím cílem je upravit seznam segmentů tvořících kandidátní trajektorii tak, aby byla minimalizována cílová funkce a zároveň byla splněna omezení. Postup řešení tohoto problému je převzat z [3]. Využíváme matematickou optimalizaci s použitím derivací. Analytické vyjádření derivací ovšem neznáme, protože plánovač využívá simulátor jako black-box. Musíme tedy provádět konečné derivace. Nejprve uveďme potřebná značení:

- $x \in \mathbb{R}^{12}$ ... stav kvadrokoptéry

- $u \in \mathbb{R}^4$ ... hodnoty aktuátorů
- $t \in \mathbb{R}$ ... délka segmentu v sekundách
- $x_0^i \in \mathbb{R}^{12}$ ... počáteční stav i-tého segmentu
- $\Phi(t, x_0, u)$ ... koncový stav segmentu s počátečním stavem x_0, aktuátory u a dobou simulace t
- $s \in \mathbb{R}^{12}$ ... startovní stav kvadrokoptéry (podle definice plánovací úlohy)
- $g \in \mathbb{R}^{12}$ ... cílový stav kvadrokoptéry (podle definice plánovací úlohy)

Cílová funkce optimalizační úlohy

Každý segment je tedy tvořen 17 proměnnými ($x_0 \in \mathbb{R}^{12}$, $u \in \mathbb{R}^4$, $t \in \mathbb{R}$). Uvažujme, že chceme optimalizovat N segmentů. Potom bude mít vstupní vektor pro optimalizaci velikost $17 \cdot N$ a cílová funkce bude mít signaturu

$$\mathbb{R}^{17 \cdot N} \to \mathbb{R}.$$

Vstupní parametry jsou $\chi = [x_0^1, u^1, t^1, \ldots, x_0^N, u^N, t_N] \in \mathbb{R}^{17 \cdot N}$. Definice cílové funkce vypadá následovně:

$$F(\chi) := \frac{1}{2} \sum_{i=1}^{N} (t^i)^2 \tag{4.1}$$

Smyslem minimalizace cílové funkce je tedy minimalizovat celkový čas plánu pohybu. Zdroj [3] ovšem neřeší kolize s překážkami. Během testování bylo zjištěno, že výsledné optimalizované plány častokrát kolidují s okolními překážkami. Pro odstranění tohoto problému byl k cílové funkci přidán term, který rychle roste v případě, že se trajektorie přiblíží k překážce, a naopak klesá, pokud se trajektorie vzdálí od překážky. Nechť $d \in \mathbb{R}$ je nejmenší vzdálenost mezi segmentem a překážkou. Škálováním d mezi hodnoty 0 a 1vznikne d' ($0 < d' \leq 1$). Výsledný term je potom roven $1/d'$. Cílovou funkci zohledňující kolize lze potom zapsat jako

$$F'(\chi) := \frac{1}{2} \sum_{i=1}^{N} (t^i)^2 + \sum_{i=1}^{N} 1/d'^i. \tag{4.2}$$

Co se týká derivací cílové funkce, jsme schopni časovou složku spočítat analyticky. Pro vzdálenostní složku musíme použít konečné derivace.

$$\frac{\partial F'(\chi)}{\partial t} = t + \frac{\partial 1/d'}{\partial t},$$
$$\frac{\partial F'(\chi)}{\partial x_0} = \frac{\partial 1/d'}{\partial x_0},$$
$$\frac{\partial F'(\chi)}{\partial u} = \frac{\partial 1/d'}{\partial u}.$$

Všechny konečné derivace jsou v této práci počítány pomocí vzorce

$$f'(x) = \frac{f(x+h) - f(x)}{h}.$$

Jako h byla zvolena hodnota $\sqrt{\epsilon} \approx 3.45 \cdot 10^{-4}$.

Omezení optimalizační úlohy

Vedle cílové funkce je potřeba definovat také omezení. Nejprve definujme funkci $d(a, b) := \frac{1}{2}\|a - b\|^2 = \frac{1}{2}(a - b)^T(a - b) \in \mathbb{R}$, která měří vzdálenost mezi dvěma body $a \in \mathbb{R}^{12}$ a $b \in \mathbb{R}^{12}$. V našem případě existují 3 druhy omezení. První omezení vyžaduje, aby vzdálenosti mezi startovním stavem kvadrokoptéry a prvním bodem z prvního segmentu byla nulová:

$$d(x_0^1, s) = 0. \tag{4.3}$$

Derivace tohoto omezení lze vyjádřit analyticky bez nutnosti provádět konečné derivace:

$$\frac{\partial d(x_0^1, s)}{\partial x_0^1} = (x_0^1 - s)^T \in \mathbb{R}^{1\times 12},$$
$$\frac{\partial d(x_0^1, s)}{\partial u} = 0 \in \mathbb{R}^{1\times 4},$$
$$\frac{\partial d(x_0^1, s)}{\partial t} = 0 \in \mathbb{R}.$$

Druhé omezení vyžaduje, aby vzdálenost mezi cílovým stavem a posledním bodem posledního segmentu byla nulová:

$$d(\Phi(t_N, x_0^N, u^N), g) = 0. \tag{4.4}$$

Derivace druhého omezení vypadají následovně:

$$\frac{\partial d(\Phi(t,x_0,u),g)}{\partial x_0} = (\Phi(t,x_0,u)-g)^T \frac{\partial \Phi(t,x_0,u)}{\partial x_0} \in \mathbb{R}^{1\times 12},$$
$$\frac{\partial d(\Phi(t,x_0,u),g)}{\partial u} = (\Phi(t,x_0,u)-g)^T \frac{\partial \Phi(t,x_0,u)}{\partial u} \in \mathbb{R}^{1\times 4},$$
$$\frac{\partial d(\Phi(t,x_0,u),g)}{\partial t} = (\Phi(t,x_0,u)-g)^T \frac{\partial \Phi(t,x_0,u)}{\partial t} \in \mathbb{R}.$$

Třetí a poslední druh omezení se týká sousedních segmentů. Pro N segmentů tedy máme $N-1$ takovýchto omezení. Omezení vyžadují, aby rozdíly koncových stavů sousedních segmentů byly nulové. Pro k-tý segment tedy definujeme omezení a jeho derivace následovně:

$$x_0^{k+1} - \Phi(t_k, x_0^k, u^k) = 0. \tag{4.5}$$

$$\frac{\partial (x_0^{k+1} - \Phi(t_k, x_0^k, u^k))}{\partial x_0^k} = -\frac{\partial \Phi(t_k, x_0^k, u^k)}{\partial x_0^k} \in \mathbb{R}^{12\times 12},$$
$$\frac{\partial (x_0^{k+1} - \Phi(t_k, x_0^k, u^k))}{\partial u^k} = -\frac{\partial \Phi(t_k, x_0^k, u^k)}{\partial u^k} \in \mathbb{R}^{12\times 4},$$
$$\frac{\partial (x_0^{k+1} - \Phi(t_k, x_0^k, u_k))}{\partial t_k} = -\frac{\partial \Phi(t_k, x_0^k, u^k)}{\partial t_k} \in \mathbb{R}^{12}.$$

Jelikož má stav kvadrokoptéry dimenzi 12, je toto omezení dekomponováno na 12 omezení. Jinými slovy, každá z 12-ti stavových proměnných posledního bodu segmentu k, musí být rovna příslušné stavové proměnné prvního bodu segmentu $k+1$.

Implementace optimalizační úlohy

Pro implementaci lokální optimalizace byla zvolena knihovna NLopt [13]. Jedná se o open-source knihovnu pro nelineární optimalizace, podporující různé optimalizační algoritmy. Knihovna podporuje jazyky C, C++, Matlab, Python a další.

Implemetace začíná vytvořením instance `nlopt_opt` zavoláním `nlopt_create(nlopt_algorithm algorithm, unsigned int n)`, kde `algorithm` je požadovaný optimalizační algoritmus a `n` je dimenze úlohy (v našem případě $N \cdot 17$). Z dostupných algoritmů založených na derivacích je použitelný pouze SLSQP (sekvenční kvadratické programování), protože jako jediný podporuje nelineární rovnostní omezení. Dalším krokem implementace je

nastavení spodních, resp. horních hranic pomocí `nlopt_set_lower_bounds(nlopt_opt opt, const double *lb)`, resp. `nlopt_set_upper_bounds(nlopt_opt opt, const double *ub)`. NLopt zaručuje, že hodnoty vstupního vektoru budou vždy mezi definovanými hranicemi.

Následně je potřeba předat funkci, která má být minimalizována, pomocí `nlopt_set_min_objective`. Předaná funkce poskytuje funkční hodnotu (dle 4.2) a příslušné derivace. Omezení jsou opět reprezentována funkcemi, které vrací hodnotu daného omezení a příslušné derivace. Vznikly tak celkem 3 funkce - jedna pro každé omezení. K předání omezení se používá volání `nlopt_add_equality_constraint`.

4.4.2.6 Přidání nového segmentu

V případě, že optimalizací kandidátní trajektorie nevznikne dostatečně dobré řešení, přichází na řadu rozšíření množiny segmentů o nový segment. Máme spoustu možností, jak přidat nový segment. Bylo zvoleno řešení, kdy je vygenerován náhodný bod v prostoru a z něj provedena simulace náhodným směrem.

4.4.3 Parametry plánovače

Níže je seznam parametrů, které tento plánovač přijímá.

- Délka jedné simulace (segmentu) v sekundách
- Počet segmentů na metr krychlový (pro počáteční generování)
- Maximální počet iterací hlavního algoritmu
- Maximální doba lokální optimalizace
- Tolerance pro omezení lokální optimalizace

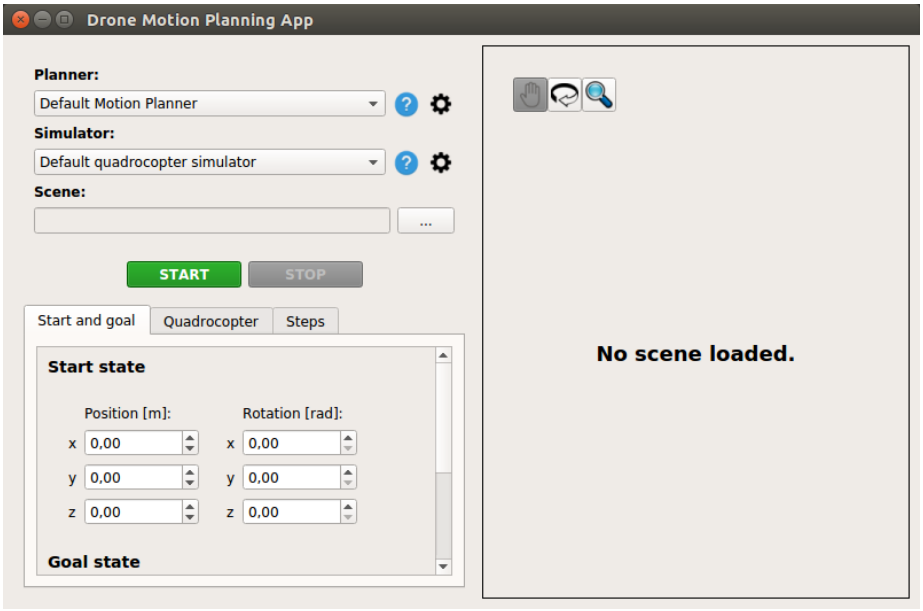
4.5 3D vizualizace plánu pohybu

V rámci této práce byla implemetována třída `Simple3DView`, jako jeden z modulů pro vizualizaci pohybu. Třída dědí z `AbstractMotionView` a slouží pro 3D vizualizaci plánů pohybu. Samotné zobrazování má na starosti pomocná třída `Simple3DViewWidget`, na kterou jsou delegovány veškeré požadavky na zobrazení plánu.

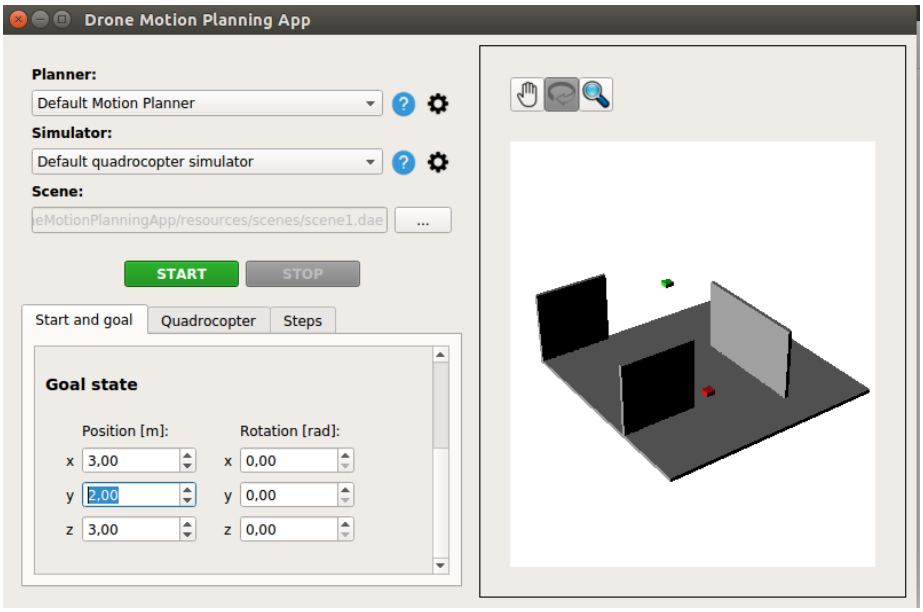
Pro práci s 3D grafikou byly použity Qt 3D moduly (3dcore, 3drender, 3dinput a 3dextras). Základním elementem v Qt 3D je `Qt3DCore::QEntity`.

Při volání `problemDefinitionChanged` je pomocí třídy `Qt3DRender::QSceneLoader` načtena 3D scéna. `Qt3DRender::QSceneLoader` vnitřně používá knohovnu Assimp, takže stačí předat cestu k nějakému souboru ve 3D formátu (.dae, .obj, atd.). Dále je vykreslen počáteční a cílová pozice kvadrokoptéry. Kvadrokoptéra je pro jednoduchost reprezentována jako kvádr. Uživatel tak může vidět, jak vypadá definice problému před spuštěním plánovače. Kdykoliv je definice problému změněna (např. posunutí startovní pozice), je tato změna okamžitě reflektována ve 3D zobrazovači.

Většina plochy zobrazovače slouží k prezentaci 3D scény. V horní části se nachází tlačítka pro ovládání scény: posouvání, rotace, přibližování/oddalování.

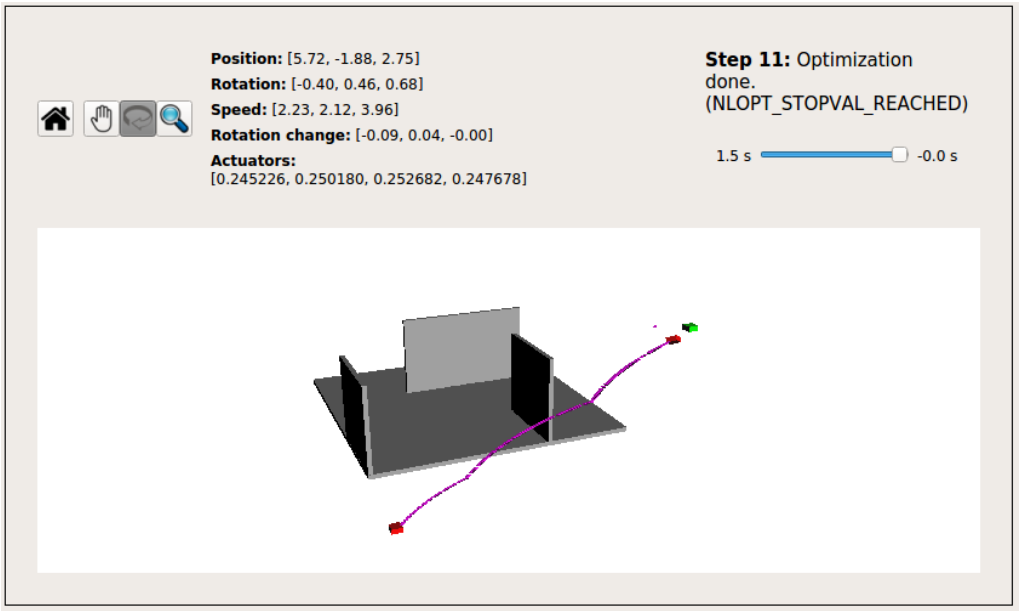


Obrázek 4.4: Vizualizace plánu pohybu před načtením scény



Obrázek 4.5: Vizualizace plánu pohybu po načtení scény

Trajektorie plánu pohybu je animována fialovou barvou. Během animace jsou zobrazovány stavové informace kvadrokoptéry (pozice, rotace, rychlost, rychlost rotace) a hodnoty aktuátorů. Pohyb kvydrokoptéry je také reprezentován časovou osou, na které je možno vidět uběhnutý a zbývající čas. Po dokončení animace lze ručně měnit polohu posuvníku na časové ose a tím procházet daný plán pohybu.



Obrázek 4.6: Vizualizace plánu pohybu

Kapitola 5

Testování

5.1 Testování simulace

Abych ověřil správnou funkčnost simulace, vytvořil jsem ve třídě `DefaultQuadrocopterSimulatorTest` několik jednotkových testů. Testy provádějí simulace s různými nastaveními akturátorů. Výstupem testů je textový výpis vývoje pozic a rotací kvadrokoptéry v čase. Tyto vývoje jsem porovnával s předpokládaným chováním popsaným v sekci 1.2. Jednotlivé testy simulují pohyby podle schématu 1.2. Počáteční hodnoty stavových proměnných jsou pro všechny simulace nulové. Nastavení parametrů je uvedeno v následující tabulce.

Parametr	**Hodnota**
R	0.15 m
M	0.5 kg
m	0.1 kg
l	0.15 m
k_1	1
k_2	1

Tabulka 5.1: Nastavení parametrů simulátoru

Přední aktuátor $T1$ tedy směřuje na východ a levý aktuátor $T4$ směřuje na sever, jak je vidět na obrázku 1.1. Výsledky jednotlivých simulací jsou prezentovány v tabulkách. Hodnota 0 značí přesnou nulu. Ostatní hodnoty jsou zaokrouhleny na příslušný počet desetinných míst.

5.1.1 Vznášení se kvadrokoptéry

Nejprve bylo potřeba zjistit, jaké hodnoty aktuátorů způsobí alepoň přibližnou levitaci kvadrokoptéry, tedy vznášení se nehybně v jednom bodě. Způsob, jak toto zjistit, je popsán v kapitole 4.4.2.1. Byla nalezena hodnota 0.244938, kterou označím jako OPT. Tato hodnota je použita jako základ v následujících testech.

5.1.2 Pohyb vzhůru

Abychom dosáhli pohybu vzhůru, stačí nastavit všechny aktuátory na stejnou hodnotu, která musí být větší než hodnota pro vznášení se. Zjistil jsem, že stačí aktuátory zvýšit o 1 procento. Hodnoty aktuátorů pro tento test jsou tedy $T1 = T2 = T3 = T4 = OPT \cdot 1.01$. Jak lze vyčíst z tabulky 5.1.2, kvadrokoptéra se pohybuje směrem vzhůru, aniž by docházelo k rotaci. To je očekávané chování.

Čas [s]	0	1	2	3	4	5	6	7	8	9	10
Pozice x	0	0	0	0	0	0	0	0	0	0	0
Pozice y	0	0	0	0	0	0	0	0	0	0	0
Pozice z	0	0.04	0.19	0.44	0.78	1.22	1.76	2.4	3.1	3.9	4.9
Rotace x	0	0	0	0	0	0	0	0	0	0	0
Rotace y	0	0	0	0	0	0	0	0	0	0	0
Rotace z	0	0	0	0	0	0	0	0	0	0	0

Tabulka 5.2: Simulace pohybu směrem vzhůru

5.1.3 Pohyb dolů

Nastavením aktuátorů na hodnotu nižší než je hodnota pro vznášení dosáhneme klesání kvadrokoptéry. Provedl jsem snížení o 2 procenta. Hodnoty aktuátorů pro tento test jsou $T1 = T2 = T3 = T4 = OPT \cdot 0.98$. Z tabulky 5.1.3 lze vidět, že souřadnice z se zmenšuje, kvadrokoptéra tedy klesá.

Čas [s]	0	1	2	3	4	5	6	7	8	9
Pozice x	0	0	0	0	0	0	0	0	0	0
Pozice y	0	0	0	0	0	0	0	0	0	0
Pozice z	0	−0.09	−0.39	−0.88	−1.56	−2.45	−3.53	−4.8	−6.27	−7.94
Rotace x	0	0	0	0	0	0	0	0	0	0
Rotace y	0	0	0	0	0	0	0	0	0	0
Rotace z	0	0	0	0	0	0	0	0	0	0

Tabulka 5.3: Simulace pohybu směrem dolů

5.1.4 Levá rotace kolem osy z

Pro dosažení rotace je potřeba zvýšit otáčky jednoho páru protilehlých rotorů. Chceme-li způsobit levou rotaci, musí se jednat o rotory točící se proti směru hodinových ručiček, což jsou v našem případě $T1$ a $T3$. Hodnoty těchto aktuátorů jsem zvýšil o 1 procento, takže $T1 = T3 = OPT \cdot 1.01$ a $T2 = T4 = OPT$. Velikost úhlu rotace začíná na hodnotě nula a postupně se zvyšuje, což odpovídá levé rotaci podle obrázku 1.1. Kvůli zvýšeným otáčkám rotorů dochází také k mírnému stoupání kvadrokoptéry.

Čas [s]	0	1	2	3	4	5	6	7	8	9	10
Pozice x	0	0	0	0	0	0	0	0	0	0	0
Pozice y	0	0	0	0	0	0	0	0	0	0	0
Pozice z	0	0.02	0.09	0.22	0.39	0.61	0.88	1.2	1.57	1.98	2.4
Rotace x	0	0	0	0	0	0	0	0	0	0	0
Rotace y	0	0	0	0	0	0	0	0	0	0	0
Rotace z	0	0.18	0.72	1.63	2.9	4.54	6.54	8.9	11.62	14.71	18.16

Tabulka 5.4: Simulace levé rotace kolem osy z

5.1.5 Pravá rotace kolem osy z

Zde je situace obdobná jako v případě levé rotace. Jediný rozdíl je ten, že zvýšení hodnot proběhne na aktuátorech $T2$ a $T4$. Tedy $T2 = T4 = OPT \cdot 1.01$ a $T1 = T3 = OPT$. Hodnoty v tabulkách 5.1.4 a 5.1.5 jsou stejné, až na směr rotace kolem osy z.

Čas [s]	0	1	2	3	4	5	6	7	8	9
Pozice x	0	0	0	0	0	0	0	0	0	0
Pozice y	0	0	0	0	0	0	0	0	0	0
Pozice z	0	0.02	0.09	0.22	0.39	0.61	0.88	1.2	1.57	1.98
Rotace x	0	0	0	0	0	0	0	0	0	0
Rotace y	0	0	0	0	0	0	0	0	0	0
Rotace z	0	−0.18	−0.72	−1.63	−2.9	−4.54	−6.54	−8.9	−11.62	−14.71

Tabulka 5.5: Simulace pravé rotace kolem osy z

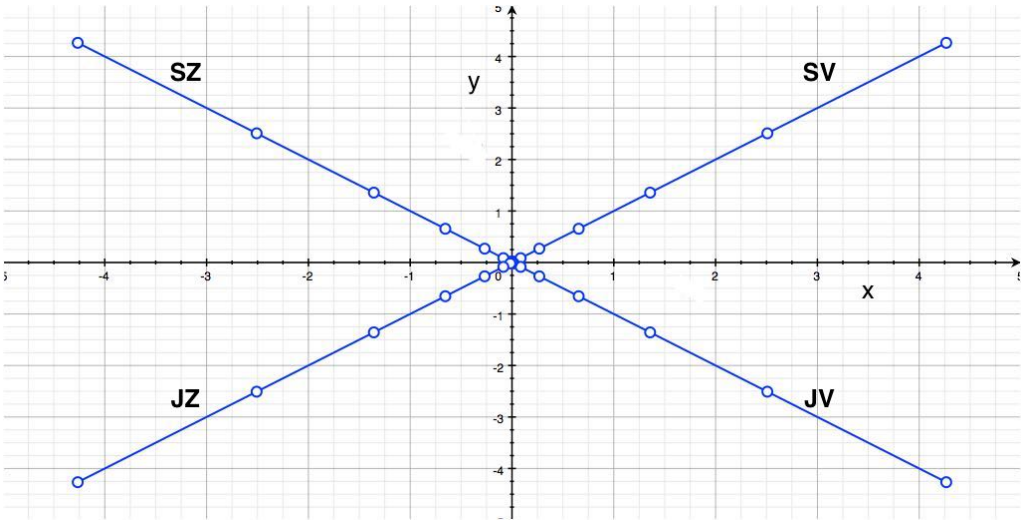
5.1.6 Pohyby do světových stran

Aby se kvadrokoptéra pohybovala do světových stran, je potřeba zvýšit hodnoty dvou sousedních aktuátorů. Výsledný pohyb potom bude ve směru opačné dvojice sousedních aktuátorů. Máme-li výchozí stav dle obrázku 1.1 a zvýšíme-li hodnoty $T1$ a $T2$, měla by se kvadrokoptéra pohybovat ve směru $T3$ a $T4$, tedy severozápadně. Následující tabulka zachycuje nastavení aktuátorů pro jednotlivé směry.

	$T1$	$T2$	$T3$	$T4$
severozápad (SZ)	$OPT \cdot 1.01$	$OPT \cdot 1.01$	OPT	OPT
severovýchod (SV)	OPT	$OPT \cdot 1.01$	$OPT \cdot 1.01$	OPT
jihovýchod (JV)	OPT	OPT	$OPT \cdot 1.01$	$OPT \cdot 1.01$
jihozápad (JZ)	$OPT \cdot 1.01$	OPT	OPT	$OPT \cdot 1.01$

Tabulka 5.6: Nastavení aktuátorů pro pohyb do světových stran

Simulace začínají v bodě [0, 0] a trvají 4 sekundy. Velikost kroku je 0.25 sekund.



Obrázek 5.1: Pohyby do světových stran

Vývoj pozice ve směru osy z je ve všech čtyřech případech stejný.

Čas [s]	0	0.5	1	1.5	2	2.5	3	3.5	4
Pozice z	0	0.006	0.02	0.05	0.08	0.11	0.12	0.04	−0.16

5.2 Testování lokální optimalizace

V této kapitole je popsáno testování lokální optimalizace, která je součástí modulu pro plánování pohybu. Nejprve je ověřena správnost jednotlivých funkcí zapojených do lokální optimalizace: cílové funkce a tří druhů omezení. Následně je provedeno testování lokální optimalizace jako celku.

Pro účely testování byla vytvořena třída `PlanOptimizationTest`, ve které se nachází veškeré jednotkové testy pro lokální optimalizaci. Následující sekce jen prezentují výsledky testování.

5.2.1 Ověření správnosti funkcí spojených s lokální optimalizací

V této sekci je ověřena správnost funkcí použitých v lokální optimalizaci: cílová funkce, omezení mezi startovním stavem a prvním segmentem, omezení mezi sousedními segmenty a omezení mezi posledním segmentem a cílovým stavem. Ve všech případech je použita následující definice problému:

Tabulka 5.7: Parametry úlohy použité pro testování lokální optimalizace

Vzdálenost motorů kvadrokoptéry	0.3 m
Výška kvadrokoptéry	0.15 m
Scéna	Jediná překážka s ohraničujícím boxem definovaným vrcholy [0, 0, 0] a [3, 3, 3]
Startovní stav kvadrokoptéry	pozice: [4, 4, 4], ostatní proměnné nulové
Cílový stav kvadrokoptéry	pozice: [10, 10, 10], ostatní proměnné nulové

Plán pohybu, který budeme v testování používat, se skládá ze dvou segmentů:

- **Segment 1:**
  Čas: 3s
  Počáteční bod (pozice, rotace, rychlost, změna rotace):
  (3.2, 3.2, 3.2), (0, 0, 0), (0, 0, 0), (0, 0, 0)
  Koncový bod:
  (4.553, 4.553, 3.264), (−0.182, 0.185, −0.017), (1.755, 1.755, 0.567), (−0.122, 0.122, 0)
  Aktuátory:
  (0.244938, 0.244938, 0.247387, 0.247387)

- **Segment 2:**
  Čas: 3s
  Počáteční bod (pozice, rotace, rychlost, změna rotace):
  (6, 6, 6), (0, 0, 0), (0, 0, 0), (0, 0, 0)
  Koncový bod:
  (7.353, 7.353, 6.064), (−0.182, 0.185, −0.017), (1.755, 1.755, 0.567), (−0.122, 0.122, 0)
  Aktuátory:
  (0.244938, 0.244938, 0.247387, 0.247387)

5.2.1.1 Ověření správnosti cílové funkce

Nyní přejdeme k testování cílové funkce (4.2). První sčítanec cílové funkce je suma čtverců délek segmentů, vydělená dvěma. V našem případě mají oba segmenty délku 3, takže očekávaná hodnota prvního sčítance je $1/2 \cdot (3^2 + 3^3) = 9$. Druhý sčítanec je suma vzdálenostních termů, která je větší nebo rovna nule. Hodnota cílové funkce v našem případě je 13.0619, což odpovídá této konkrétní situaci.

Tabulka 5.8: Gradient cílové funkce

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(−5.1031, −5.1031, −5.1031)	Pozice	(0, 0, 0)
Rotace	(0, 0, 0)	Rotace	(0, 0, 0)
Rychlost	(0, 0, 0)	Rychlost	(0, 0, 0)
Změna rotace	(0, 0, 0)	Změna rotace	(0, 0, 0)
Aktuátory	(0, 0, 0, 0)	Aktuátory	(0, 0, 0, 0)
Čas	3	Čas	3

Derivace cílové funkce pro proměnnou čas je 3 v obou segmentech. To odpovídá analytickému spočtení derivace. Derivace podle pozice prvního segmentu je záporné číslo. Důvod je ten, že zvětšením souřadnic pozice prvního segmentu dojde ke zvětšení vzdálenosti prvního segmentu od překážky, čímž se sníží vzdálenostní term a tím pádem dojde ke snížení hodnoty cílové funkce. Derivace pro ostatní proměnné je rovna nula, protože tyto proměnné nemají vliv na hodnotu cílové funkce.

5.2.1.2 Ověření správnosti omezení mezi startovním stavem a prvním segmentem

Nyní budeme ověřovat správnost implementace prvního omezení. Hodnota tohoto omezení je 0.96, což odpovídá vzorci (4.3).

Tabulka 5.9: Gradient pro omezení mezi startovním stavem a prvním segmentem

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(−0.8, −0.8, −0.8)	Pozice	(0, 0, 0)
Rotace	(0, 0, 0)	Rotace	(0, 0, 0)
Rychlost	(0, 0, 0)	Rychlost	(0, 0, 0)
Změna rotace	(0, 0, 0)	Změna rotace	(0, 0, 0)
Aktuátory	(0, 0, 0, 0)	Aktuátory	(0, 0, 0, 0)
Čas	0	Čas	0

Parametry druhého segmentu nemají žádný vliv na vzdálenost prvního segmentu od startovního stavu, proto jsou derivace nulové. Hodnoty v prvním segmentu odpovídají rozdílu stavu prvního segmentu a startovního stavu.

5.2.1.3 Ověření správnosti omezení mezi cílovým stavem a posledním segmentem

Hodnota druhého omezení je spočtena podle vzorce 4.4. V našem případě je hodnota rovna 18.0412.

Tabulka 5.10: Gradient pro omezení mezi cílovým stavem a posledním segmentem

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(0, 0, 0)	Pozice	(−2.646, −2.646, −3.935)
Rotace	(0, 0, 0)	Rotace	(58.547, −58.484, −0.015)
Rychlost	(0, 0, 0)	Rychlost	(−6.139, −6.139, −11.894)
Změna rotace	(0, 0, 0)	Změna rotace	(27.262, −27.162, −0.05)
Aktuátory	(0, 0, 0, 0)	Aktuátory	(−372.804, −384.299, −30.190, −41.68)
Čas	0	Čas	−2.658

Parametry prvního segmentu neovlivňují vzdálenost druhého segmentu od cílového stavu, proto jsou derivace nulové. Zbytek gradientu ověříme pomocí konečných derivací.

Tabulka 5.11: Ověření gradientu pro omezení mezi cílovým stavem a posledním segmentem pomocí konečných derivací

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(0, 0, 0)	Pozice	(−2.646, −2.646, −3.935)
Rotace	(0, 0, 0)	Rotace	(59.035, −57.996, −0.015)
Rychlost	(0, 0, 0)	Rychlost	(−6.138, −6.138, −11.893)
Změna rotace	(0, 0, 0)	Změna rotace	(27.934, −26.49, −0.045)
Aktuátory	(0, 0, 0, 0)	Aktuátory	(−87.221, −96.868, 131.432, 118.246)
Čas	0	Čas	−2.61

Jak je vidět v předchozích dvou tabulkách, hodnoty spočtené pomocí konečných derivací zhruba odpovídají hodnotám spočteným pomocí vzorců.

5.2.1.4 Ověření správnosti omezení mezi segmenty

Mezi sousedními segmenty se nachází 12 omezení - jedno pro každou stavovou proměnnou. Zde bude prezentováno omezení jen pro jednu proměnnou - a sice první souřadnici pozice (pozice x). Označme pozici x koncového

bodu prvního segmentu jako x_1. Dále označme pozici x počátečního bodu druhého segmentu jako x_2. Hodnota omezení se spočte jako $x_2 - x_1$, tedy $6 - 4.553 = 1.447$.

Tabulka 5.12: Gradient pro omezení mezi segmenty (pozice x)

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(−1, 0, 0)	Pozice	(1, 0, 0)
Rotace	(1.028e-11, −44.209, −1.353)	Rotace	(0, 0, 0)
Rychlost	(−3, -1.028e-11, -1.028e-11)	Rychlost	(0, 0, 0)
Změna rotace	(−0.049, −44.145, −2.029)	Změna rotace	(0, 0, 0)
Aktuátory	(104.112, 432.801, 140.631, −668.961)	Aktuátory	(0, 0, 0)
Čas	−1.801	Čas	0

Obecně platí, že změna pozice počátečního bodu segmentu má za následek stejnou změnu koncového bodu segmentu. Derivaci $x_2 - x_1$ podle pozice počátečního bodu prvního segmentu si lze představit jako derivaci $x_2 - x_1$ podle x_1. Výsledek je tedy -1. Jediná nenulová derivace pro druhý segment je derivace podle pozice x, protože $\frac{\partial x_2 - x_1}{\partial x_2} = 1$.

Správnost gradientu ověříme pomocí konečných derivací.

Tabulka 5.13: Ověření gradientu pro omezení mezi segmenty (pozice x) pomocí konečných derivací

Segment 1		Segment 2	
Proměnná	Derivace	Proměnná	Derivace
Pozice	(−1, 0, 0)	Pozice	(1, 0, 0)
Rotace	(1.028e-11, −44.209, −1.353)	Rotace	(0, 0, 0)
Rychlost	(−3, -1.028e-11, -1.028e-11)	Rychlost	(0, 0, 0)
Změna rotace	(−0.049, −44.145, −2.029)	Změna rotace	(0, 0, 0)
Aktuátory	(104.112, 432.801, 140.631, −668.961)	Aktuátory	(0, 0, 0)
Čas	−1.801	Čas	0

Gradient spočtený pomocí konečných derivací je úplně stejný jako gradient spočtený pomocí vzorců.

5.2.1.5 Testování lokální optimalizace jako celku

V této sekci provedeme testování lokální optimalizace jako celku, kdy budeme zkoumat výstupní plány pohybu, které optimalizace produkuje. Pa-

rametry úlohy a segmenty jsou stejné jako v předchozích čtyřech sekcích, kde jsme zkoumali správnost cílové funkce a jednotlivých omezení.

Knihovně NLopt lze nastavit velké množství parametrů. Níže je uveden seznam použitých parametrů a jejich hodnot.

Tabulka 5.14: Testování lokální optimalizace: nastavení parametrů NLopt

NLopt parametr	**Hodnota**
Maximální doba běhu	300 s
Tolerance pro všechna omezení	polovina počáteční hodnoty, ne méně než 0.1
Spodní/horní mez na pozici	$-2/+2$ vůči počáteční hodnotě
Spodní/horní mez na rotaci	$-0.5/+0.5$ vůči počáteční hodnotě
Spodní/horní mez na rychlost	$-2/+2$ vůči počáteční hodnotě
Spodní/horní mez na změnu rychlosti	$-1/+1$ vůči počáteční hodnotě
Spodní/horní mez na aktuátory	$-1\%/+1\%$ vůči počáteční hodnotě
Spodní/horní mez na čas	třetina/dvojnásobek počáteční hodnoty

V této sekci budeme používat následující značení:

- **A:** omezení mezi startovním stavem a prvním segmentem
- **Bi:** omezení mezi segmenty i a $i + 1$
- **C:** omezení mezi cílovým stavem a posledním segmentem

Počáteční hodnoty omezení jsou uvedeny v následující tabulce:

Tabulka 5.15: Testování lokální optimalizace: počáteční hodnoty omezení

Omezení	**Hodnota**
A	0.96
B (pozice)	(1.44664, 1.44664, 2.73522)
B (rotace)	(0.182658, −0.185784, 0.0170635)
B (rychlost)	(−1.75594, −1.75594, −0.567733)
B (změna rotace)	(0.122469, −0.122469, 0)
C	18.0412

Počáteční plán pohybu byl třikrát po sobě optimalizován, přičemž výstup první optimalizace byl použit jako vstup druhé optimalizace, atd. Tento způsob byl zvolen z důvodu, že i plánovač pohybu spouští optimalizaci iterativně. Navíc je zde snaha snížit hodnoty omezení o polovinu v každé iteraci. Hodnoty omezení po jednotlivých spuštěních jsou vypsány v

následujících tabulkách. Každá tabulka také obsahuje informaci o změně hodnoty omezení oproti předchozímu běhu.

Tabulka 5.16: Testování lokální optimalizace: hodnoty omezení po prvním běhu optimalizace

Omezení	**Hodnota**	**Změna v %**
A	0.0862168	-91
B (pozice)	(0.142536, 0.111496, 0.0927674)	(−90.1, −92.2, −96.6)
B (rotace)	(0.00839522, −0.00786998, −0.00223361)	(−95.4, −95.7, −86.9)
B (rychlost)	(−0.0738183, −0.0921237, −0.00461698)	(−95.7, −94.7, −99.1)
B (změna rotace)	(−0.0104463, 0.0107143, −0.00159801)	(−91.4, −91.2, x)
C	1.83792	-89.8

Tabulka 5.17: Testování lokální optimalizace: hodnoty omezení po druhém běhu optimalizace

Omezení	**Hodnota**	**Změna v %**
A	0.0302797	−64.8
B (pozice)	(0.0920606, 0.0749626, −0.0017729)	(−35.4, −32.7, −98.0)
B (rotace)	(0.0039898, −0.00416922, 0.00136358)	(−52.4, −47.0, −38.9)
B (rychlost)	(0.00712617, 0.0223196, −0.000911526)	(−90.3, −75.7, −80.2)
B (změna rotace)	(−0.00043401, 0.000567084, −0.000749334)	(−95.8, −94.7, −53.1)
C	0.867517	−52.7

Tabulka 5.18: Testování lokální optimalizace: hodnoty omezení po třetím běhu optimalizace

Omezení	**Hodnota**	**Změna v %**
A	0.0229543	-24.1
B (pozice)	(0.0562184, 0.00734736, −0.00791327)	(−38.9, −90.1, 346.3)
B (rotace)	(0.00209889, −0.00269521, −0.000717343)	(−47.3, −35.3, −47.3)
B (rychlost)	(−0.0579516, −0.0480813, 0.045627)	(713.2, 115.4, 4905.5)
B (změna rotace)	(0.000386454, 0.000634187, −0.000717748)	(−10.9, 11.8, −4.2)
C	0.420881	−51.4

Níže jsou uvedeny doby jednotlivých běhů optimalizace.

Tabulka 5.19: Testování lokální optimalizace: doba běhu

	Čas
1. běh	21.6 s
2. běh	31.3 s
3. běh	25.5 s
Součet	78.4 s

Následující tabulka zobrazuje hodnoty omezení před optimalizací a po třech kolech optimalizace spolu s procentuálním zlepšením či zhoršením.

Tabulka 5.20: Testování lokální optimalizace: závěrečné shrnutí

Omezení	**Počáteční hodnota**	**Koncová hodnota**	**Změna v %**
A	0.96	0.02	−97.7
B (pozice x)	1.44	0.05	−96
B (pozice y)	1.44	0.007	−99
B (pozice z)	2.73	−0.007	−99.7
B (rotace x)	0.18	0.002	−98.9
B (rotace y)	−0.18	−0.002	−98.5
B (rotace z)	0.01	−0.0007	−93
B (rychlost x)	−1.75	−0.05	−96.7
B (rychlost y)	−1.75	−0.04	−97.2
B (rychlost z)	−0.56	0.04	−91.9
B (změna rotace x)	0.12	0.0003	−99.6
B (změna rotace y)	−0.12	0.0006	−99.6
B (změna rotace z)	0	−0.0007	zhoršení
C	18.0412	0.42	−97.6

Ve všech případech došlo k více než 90-ti procentnímu zlepšení. Ovšem žádné omezení není splněno na stoprocentně. Jediná proměnná, kde došlo ke zhoršení, je změna rychlosti rotace kolem osy z. Velkou roli zde hraje způsob nastavování aktuátorů, kdy používáme konstantní hodnoty pro daný segment. Abychom dosáhli lepších spojení segmentů, bylo by potřeba měnit aktuátory polynomiálně. Po konzultaci s vedoucím práce bylo rozhodnuto, že takové vylepšení je mimo rámec této práce a proto bude ponecháno stávající řešení.

Snížení tolerance omezení

Nyní provedeme snížení tolerance jednotlivých omezení z hodnoty 0.1 na hodnotu 0.01 a provedeme test znovu. Maximální doba běhu optimalizace byla zvýšena na 900 s.

Tabulka 5.21: Testování lokální optimalizace s tolerancí omezení 0.01: doby běhů

	Čas
1. běh	900+ s
2. běh	85 s
3. běh	900+ s
Součet	1885 s

Tabulka 5.22: Testování lokální optimalizace s tolerancí omezení 0.01: závěrečné shrnutí

Omezení	**Počáteční hodnota**	**Koncová hodnota**	**Změna v %**
A	0.96	0.08	−91.5
B (pozice x)	1.44	−0.08	−94.1
B (pozice y)	1.44	−0.08	−94.4
B (pozice z)	2.73	0.35	−86.9
B (rotace x)	0.18	−0.004	−97.4
B (rotace y)	−0.18	0.01	−93.8
B (rotace z)	0.01	0.009	−41.5
B (rychlost x)	−1.75	0.59	−66.3
B (rychlost y)	−1.75	0.27	−84.4
B (rychlost z)	−0.56	−0.59	4.2
B (změna rotace x)	0.12	0.001	−98.8
B (změna rotace y)	−0.12	−0.002	−97.7
B (změna rotace z)	0	0.003	zhoršení
C	18.04	0.05	−99.7

Celková doba běhu se zvýšila ze 78.4 s na 1885 s, tedy zhruba 24 krát. V prvním a třetím běhu byla optimalizace ukončena kvůli překročení maximální doby běhu. Co se týká splnění omezení, výrazné vylepšení se nedostavilo.

Testování lokální optimalizace s použitím omezení jen pro pozice

Splnit všechna omezení s konstantní změnou aktuátorů je obtížné. Proto vyzkoušíme spojovat jen pozice. Ostatní 3 proměnné (rotace, rychlost, změna rotace) budeme ignorovat.

Celkový čas tří běhů je 5.8 s. Hodnoty omezení po třetím běhu optimalizace jsou zobrazeny níže v tabulce.

Tabulka 5.23: Testování lokální optimalizace: omezení jen na pozici

Omezení	**Počáteční hodnota**	**Koncová hodnota**	**Změna v %**
A	0.96	0.001	−99.7
B (pozice x)	1.44	−0.001	−99.9
B (pozice y)	1.44	−0.0008	−99.9
B (pozice z)	2.73	0.002	−99.9
C	18.04	0.01	−99.9

Pokud spojujeme jen pozice, je čas běhu mnohem menší a zároveň je dosaženo mnohem lepších výsledků.

5.3 Testování plánovače pohybu

V této kapitole bude provedeno ověření funkčnosti plánovače pohybu jako celku. Testování bude zkoumat kvalitu výsledných plánů pohybu v jednotlivých krocích algoritmu, čas běhu a vliv parametrů plánovače. Nejprve si algoritmus plánování pohybu připomeňme. Pro lepší čitelnost označme hlavní kroky algoritmu jako K1, K2 a K3:

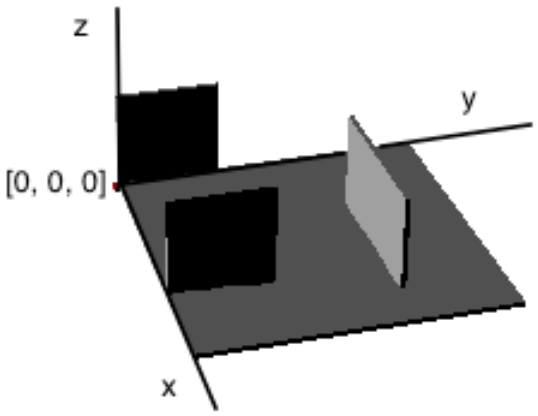
- **K1:** Počáteční generování segmentů a nalezení nejkratší cesty ze startu do cíle.
- **K2:** Lokální optimalizace.
- **K3:** Přidání nového segmentu.

Krok K1 je proveden pouze jednou, a to na začátku. Kroky K2 a K3 se v cyklu opakují, dokud není dosaženo ukončovacího kritéria.

Během testování lokální optimalizace bylo zjištěno, že algoritmus funguje mnohem rychleji, pokud při lokální optimalizaci klademe omezení jen pozici kvadrokoptéry. Proto budeme v tomto testování používat implementaci lokální optimalizace, která spojuje jen pozice.

Nastavení plánovací úlohy

Při testování plánovače pohybu budeme používat scénu obsahující čtyři překážky, jejichž ohraničující boxy jsou definovány souřadnicemi: (4, 0, 0) a (4.1, 2, 2), (0, 0, 0) a (6, 6, 0.1), (2, 4, 0) a (5.1, 4.1, 2), (0, 0, 0) a (0.1, 2, 2). Scéna je zachycena na následujícím obrázku:



Obrázek 5.2: Testovací scéna

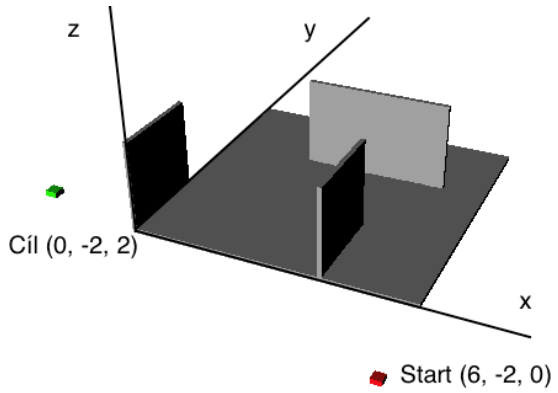
Kvadrokoptéra má vzdálenost motorů 0.3 m a výšku 0.1 m. Modulu simulátor pohybu bylo ponecháno defaultní nastavení, tedy průměr středu: 0.15 m, hmotnost středu: 0.5 kg, hmotnost rotoru: 0.1 kg, koeficienty k1 a k2: 1.

Plánovač pohybu je nastaven následovně: délka segmentů při počátečním generování: 3 sekundy, počet segmentů na metr krychlový: 3, maximální počet iterací: 30, maximální doba běhu lokální optimalizace: 300 sekund.

5.3.1 Plánovací úloha bez překážek

Nejprve otestujeme plánovač pohybu v situaci, kdy v okolí přímé spojnice mezi startovní a cílovou pozicí není žádná překážka. Zvolíme startovní pozici

(6, −2, 0) a cílovou pozici (0, −2, 2).



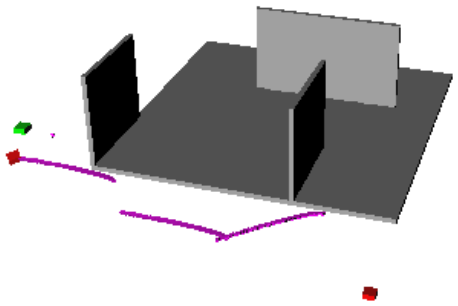
Obrázek 5.3: Konfigurace plánovací úlohy bez překážek

Prvních 6 kroků algoritmu je uvedeno v tabulce níže.

Tabulka 5.24: Výsledky testování plánovače pohybu s tolerancí omezení 0.1, bez překážek

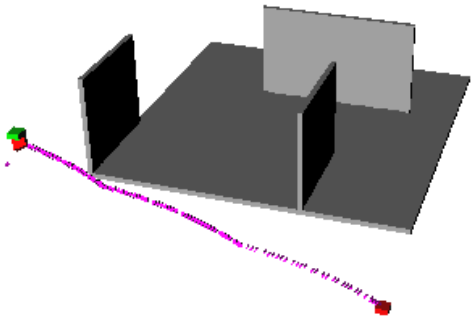
Krok **Čas**	**Omezení (abs. hodnoty)**	**Součet omezení /** **Změna**	**Délka plánu** [sec]	**Omezení v toleranci**
1 (K1) 00:15	A: 1.239 B1: (0.12, 0.673, 0.466) B2: (0.138, 0.097, 0.547) C: 0.561	3.841 / -	9	NE
2 (K2) 00:20	A: 0.03 B1: (0.002, 0.004, 0.002) B2: (0.001, 0.001, 0.0008) C: 0.23	0.27 / −0.99 %	3	NE
3 (K3) 00:25	A: 0.004 B1: (0.0005, 0.0001, 0.0001) B2: (0.0002, 0.0003, 0.00007) B3: (0.042, 0.1006, 0.059) C: 0.455	0.66 / 144 %	3	NE
4 (K2) 00:28	A: 0.002 B1: (0.072, 0.004, 0.059) B2: (0.071, 0.018, 0.055) C: 0.09	0.371 / −43 %	1.52	ANO
5 (K3) 00:33	A: 0.002 B1: (0.072, 0.004, 0.059) B2: (0.071, 0.018, 0.055) C: 0.09	0.371 / 0 %	1.52	ANO
6 (K2) 01:04	A: 0.0003 B1: (0.045, 0.002, 0.003) B2: (0.026, 0.0003, 0.003) C: 0.008	0.087 / −76 %	1.11	ANO

První krok trval 15 sekund a bylo při něm vygenerováno velké množství segmentů a následně byla nalezena nejkratší cesta ze startu do cíle. Cestu tvoří 3 segmenty, jak je vidět na obrázku níže.



Obrázek 5.4: Testování plánovače pohybu s tolerancí omezení 0.1, výsledek po prvním kroku

Po následujících třech krocích (po 28 sekundách) plánovač vytvořil plán, jehož omezení splňují stanovenou toleranci 0.1. Zároveň byla výrazně snížena délka plánu. Z obrázku níže vidíme, že lokální optimalizace má tendenci spojit startovní a cílový bod přímou čarou.



Obrázek 5.5: Testování plánovače pohybu s tolerancí omezení 0.1, výsledek po šestém kroku

5.3.1.1 Vliv počtu generovaných segmentů na dobu běhu plánovače

Nyní se podíváme, jaký vliv má parametr počet segmentů na metr krychlový na dobu běhu celého algoritmu. Zajímá nás, za jak dlouho bude vytvořen plán s hodnotami omezení s tolerancí 0.1. Byly vyzkoušeny různé hodnoty

parametru a pro každou z nich bylo provedeno 10 měření, ze kterých se spočetl aritmetický průměr.

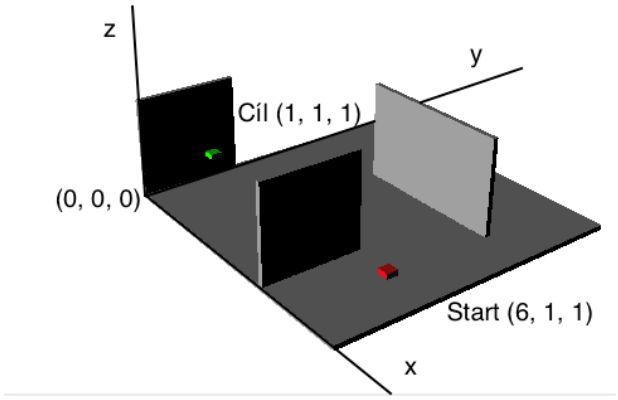
Z testování vyplynulo, že optimální počet segmentů pro náš případ je 1 nebo 2 na metr krychlový. Větší počet segmentů prodlouží dobu hledání nejkratší cesty. Příliš malý počet segmentů sice zkrátí dobu hledání cesty v grafu, ale za to prodlouží dobu běhu lokální optimalizace.

Tabulka 5.25: Vliv počtu generovaných segmentů na dobu vytvoření plánu s tolerancí pro omezení 0.1

Počet segmentů na metr krychlový	**Doba běhu plánovače**
0.3	74 s
0.5	43 s
1	24 s
2	24 s
3	43 s
4	38 s
5	83 s
6	81 s
7	98 s

5.3.2 Plánovací úloha s překážkami

Nyní otestujeme plánovač pohybu v situaci, kdy se mezi startovní a cílovou pozicí nachází překážka. Nastavíme startovní pozici na (6, 1, 1) a cílovou na (1, 1, 1).

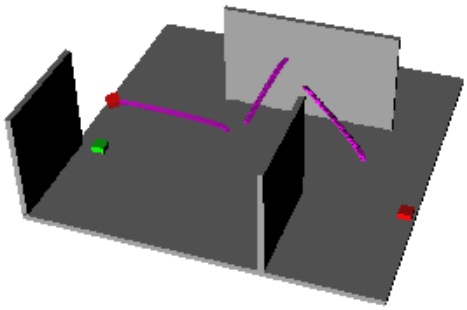


Obrázek 5.6: Konfigurace plánovací úlohy s překážkami

Tabulka 5.26: Výsledky testování plánovače pohybu s tolerancí omezení 0.1, s překážkami

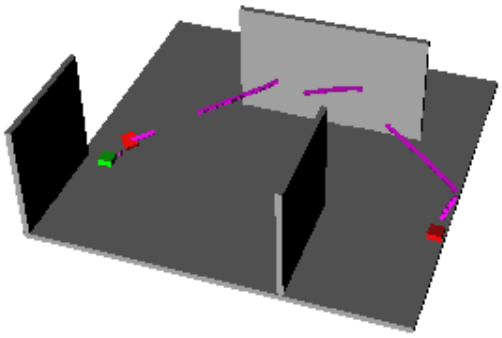
Krok Čas	**Omezení (abs. hodnoty)**	**Součet omezení / Změna**	**Délka plánu [sec]**	**Omezení v toleranci**
1 (K1) 00:09	A: 0.662 B1: (0.350, 0.039, 0.434) B2: (0.372, 0.146, 0.360) C: 0.310	2.68 / -	9	NE
2 (K2) 00:37	A: 0.196 B1: (0.058, 0.082, 0.025) B2: (0.016, 0.036, 0.042) C: 0.025	0.48 / −82 %	3.03	NE
3 (K3) 00:39	A: 0.091 B1: (0.205, 0.887, 0.623) B2: (0.174, 0.264, 0.498) B3: (0.372, 0.077, 0.136) B4: (0.217, 0.136, 0.526) C: 0.025	4.23 / 781 %	13	NE
4 (K2) 05:41	A: 0.069 B1: (0.047, 0.202, 0.061) B2: (0.565, 0.555, 0.135) B3: (0.482, 0.005, 0.093) B4: (0.517, 0.6, 0.061) C: 0.119	3.511 / −16.9 %	4.38	NE
5 (K3) 05:44	A: 0.069 B1: (0.047, 0.202, 0.061) B2: (0.321, 0.381, 0.058) B3: (0.243, 0.173, 0.026,) B4: (0.482, 0.005, 0.093) B5: (0.517, 0.6, 0.061) C: 0.119	3.458 / −1.5 %	7.38	NE
6 (K2) 06:53	A: 0.007 B1: (0.004, 0.082, 0.037) B2: (0.037, 0.03, 0.047) B3: (0.014, 0.011, 0.028) B4: (0.02, 0.001, 0.015) B5: (0.022, 0.012, 0.006) C: 0.022	0.395 / −88.5 %	3.53	ANO

V prvním kroku byly vybrány 3 segmenty tvořící nejkratší cestu mezi startem a cílem.



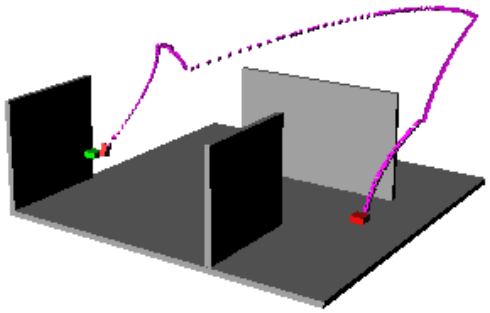
Obrázek 5.7: Testování plánovače pohybu s tolerancí omezení 0.1, s překážkami, výsledek po prvním kroku

Lokální optimalizace ve druhém kroku způsobila kolizi segmentu s okolím. Kolizní segment byl odstraněn a v následujícím kroku byly vybrány nové segmenty tvořící nejkratší cestu.



Obrázek 5.8: Testování plánovače pohybu s tolerancí omezení 0.1, s překážkami, výsledek po čtvrtém kroku

Po šestém kroku (v čase 06:53) máme k dispozici plán, jehož hodnoty omezení splňují stanovenou toleranci 0.1.



Obrázek 5.9: Testování plánovače pohybu s tolerancí omezení 0.1, s překážkami, výsledek po šestém kroku

5.4 Testování uživatelského rozhraní

Testování uživatelského rozhraní bylo pojato jako testování s experty. Zúčastnili se ho dva softwaroví inženýři, kteří si aplikaci vyzkoušeli. Z diskuze vyplynuly následující nápady na vylepšení, které byly implementovány:

- Vizualizace pohybu by měla obsahovat časovou osu, aby měl uživatel představu o délce plánu pohybu. Tato funkce byla implementována spolu se zobrazováním stavových proměnných kvadrokoptéry.

- Kameru (pohled na 3D scénu) by mělo být možné resetovat do původního nastavení. Bylo tedy přidáno nové tlačítko, které toto umožňuje.

- Trajektorie plánu pohybu by měla mít jinou barvu, než je barva kvadrokoptéry. Barva byla změněna z červené na fialovou.

- Ikony pro zobrazení informací o plánovači a simulátoru by měly být výstižnější (původně byly použity ikony s písmenem 'i'). Nově jsou použity ikony s otazníkem.

5.5 Struktura projektu

```
DronMotionPlanningApp.pro ..... konfigurační soubor Qt projektu
src/
    gui ..... adresář obsahující zdrojové kódy pro GUI
    model ..... adresář obsahující zdrojové kódy pro business logiku
    test ..... adresář obsahující zdrojové kódy unit testů
doc/
    doxygenconfig ..... konfigurační soubor pro Doxygen
    html .... výstup programu Doxygen obsahující HTML dokumentaci
distribution ..... adresář sloužící jako šablona pro distribuci aplikace
    lib ..... sdílené knihovny potřebné pro běh aplikace
    plugins ..... sdílené Qt knihovny + moduly aplikace
    resources/
        scenes ..... ukázkové scény
    README.txt ..... Instrukce ke spuštění aplikace
    run.sh ..... Skript ke spuštění aplikace.
```

5.6 Jednotkové testy

Pro účely ověření správné funkčnosti simulátoru pohybu a lokální optimalizace byly napsány unit testy (jednotkové testy). Zdrojové kódy testů jsou součástí hlavní aplikace a nachází se v adresáři `src/test`. Framework Qt zajišťuje podporu pro unit testy, čehož jsem využil. Pro každou testovanou třídu byla vytvořena nová třída, jejíž název má příponu `Test`. Pro každou testovanou metodu byla vytvořena nová metoda s předponou `test`. Vstupním bodem pro spuštění testů je funkce `main` v souboru `test_main.cpp`, která postupně spouští všechny testové třídy pomocí `QTest::qExec`.

Projektový soubor `DronMotionPlanningApp.pro` obsahuje sekci pro unit testy:

```
test {
    message(Test build)
    QT += testlib
    TARGET = UnitTests
    SOURCES -= src/main.cpp
    HEADERS += <vycet_hlavickovych_souboru>
    SOURCES += <vycet_implementacnich_souboru>
} else {
    message(Normal build)
```

```
}
```

Projekt lze sestavit dvěma způsoby:

- **Build hlavní aplikace**: V tomto případě není potřeba upravovat projektový soubor. Výsledkem bude spustitelná hlavní aplikace. Testy v ní nebudou obsaženy.
- **Build testů**: V tomto případě je potřeba přidat řádek `CONFIG += test`. Výsledkem bude spustitelná aplikace obsahující pouze testy.

Závěr

Cílem diplomové práce bylo vytvořit software pro grafickou demonstraci plánování pohybu kvadrokoptér. Nejprve jsem vysvětlil základní terminologii v oblasti plánování pohybu, nadefinoval potřebné pojmy a provedl rešerši existujících řešení.

Dále jsem provedl návrh software pro plánování pohybu kvadrokoptér, který je dostatečně modulární a udržovatelný. Jeho architektura tedy splňuje zadání - moduly plánovač, simulátor a vizualizaci lze snadno měnit a přidávat. Díky distribuci jednotlivých modulů ve formě sdílených knihoven (pluginů) je možné moduly přidávat a odebírat za běhu aplikace. Jako programovací jazyk jsem zvolil C++, a to mimo jiné z důvodu lepší podpory 3D grafiky. Pro usnadnění implementace GUI, pluginů a 3D grafiky jsem vybral framework Qt, jehož výhodou je mimo jiné podpora více platforem (Windows, Linux, Mac OS). Pomocí zmíněných technologií jsem vytvořil aplikaci s grafickým rozhraním, která může sloužit pro běh a porovnávání plánovačů pohybu kvadrokoptér. Uživatel může vytvářet vlastní 3D scény a používat je v aplikaci.

Nastudoval jsem matematický model kvadrokoptéry, vybral vhodný řešič soustav diferenciálních rovnic a provedl implementaci modulu pro simulaci pohybu kvadrokoptér. Funkčnost simulace pohybu byla řádně otestována.

Dále jsem provedl implementaci plánovače pohybu podle algoritmu [2], který je plně funkční. Provedl jsem testování lokální optimalizace, jakožto hlavního kroku algoritmu. Na závěr jsem otestoval plánovač pohybu jako celek. Plánovač funguje velmi dobře v případě, kdy mezi startovní a cílovou pozicí není žádná překážka. Jelikož původní algoritmus neřeší kolize s překážkami, musel jsem vymyslet způsob, jak kolizím zabránit. Přestože kolizím nebylo zabráněno stoprocentně, algoritmus produkuje uspokojivá řešení.

Literatura

[1] Nielsen, J. *10 Usability Heuristics for User Interface Design*. 30.4.2017. Dostupné z http://www.designprinciplesftw.com/collections/10-usability-heuristics-for-user-interface-design

[2] Kuřátko, J., Ratschan, S. *Combined global and local search for the falsification of hybrid systems. Formal Modeling and Analysis of Timed Systems*. Springer International Publishing, 2014. 146-160.

[3] Kuřátko, J., Ratschan, S. *Solving Reachability Problems by a Scalable Constrained Optimization Method*.

[4] Beard, Randal *Quadrotor Dynamics and Control Rev 0.1*. 2008. All Faculty Publications. Paper 1325. http://scholarsarchive.byu.edu/facpub/1325 146-160.

[5] *Qt framework*. https://www.qt.io

[6] *Java 3D API*. http://www.oracle.com/technetwork/java/javase/overview/index-jsp-138252.html

[7] *Open Asset Import Library*. http://www.assimp.org

[8] *ODEINT - řešič diferenciálních rovnic pro C++*. http://headmyshoulder.github.io/odeint-v2/

[9] Gamma, E. *Design patterns : elements of reusable object-oriented software*. 1995. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA

[10] *The Open Motion Planning Library*. http://ompl.kavrakilab.org

[11] *MoveIt!*. http://moveit.ros.org

[12] *V-REP*. http://www.coppeliarobotics.com

[13] Steven G. Johnson *The NLopt nonlinear-optimization package*. http://ab-initio.mit.edu/nlopt

[14] *FreeCAD*. https://www.freecadweb.org

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

ODE Ordinary Differential Equations

OMPL Open Motion Planning Library

IDE Integrated Development Environment

API Application Programming Interface

3D Three-dimensional

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF