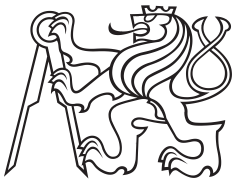
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

# Bezkolizní řízení robotické helikoptéry

**Štěpán Vejvoda**

Vedoucí: Ing. Jan Chudoba
Studijní program: Otevřená informatika
Specializace: Základy umělé inteligence a počítačových věd
Květen 2023

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vejvoda** Jméno: **Štěpán** Osobní číslo: **483527**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Bezkolizní řízení robotické helikoptéry**

Název bakalářské práce anglicky:

**Collision-Free Navigation of Robotic Helicopter**

Pokyny pro vypracování:

Cílem práce je navrhnout řídicí systém pro model robotické helikoptéry, který zajistí bezkolizní přelet do zadané cílové pozice.
Předpokládá se, že úloha bude zjednodušena o řešení lokalizace helikoptéry, která bude řešena motion-capture systémem.
- Seznamte se se senzory používanými v robotice pro měření vzdáleností. Po konzultaci s vedoucím práce vyberte vhodné senzory a navrhněte jejich konfiguraci na robotické helikoptéře za účelem získání informací o překážkách v blízkosti helikoptéry.
- Navrhněte a implementujte modifikaci řídicího systému helikoptéry tak, aby zabránil pohybu helikoptéry vedoucí ke kolizi s překážkou.
Očekávané výstupy projektu:
- rešerše metod pro detekci překážek vhodných pro létající roboty a metod pro předcházení kolizí,
- návrh a implementace řídicího kódu pro simulátor a/nebo reálnou helikoptéru v jazyce C/C++,
- zhodnocení funkčnosti a vlastností implementovaného řešení.

Seznam doporučené literatury:

[1] Fisher, Robert & Konolige, Kurt.. Handbook of Robotics Chapter 22-Range Sensors, (2008).
[2] 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..
[3] Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, Fei Gao - Survey of UAV motion planning - IET Cyber-systems and Robotics, 2020.

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: **11.02.2022** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **30.09.2023**

| 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) |

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

## 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

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

# Poděkování

Tímto bych chtěl poděkovat panu Ing. Janu Chudobovi za jeho vhled, připomínky a hlavně trpělivost při vedení mé bakalářské práce.


# 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 závěrečných prací.

V Praze dne 26. 5. 2023

. . . . . .

# Abstrakt

Tato bakalářská práce se zaměřuje na vývoj bezkolizního systému pro robotickou helikoptéru a jeho integraci do MRS UAV systému. V teoretické části práce jsou rozebrány různé způsoby získávání dat pomocí senzorů a problematika lokalizace helikoptéry. Dále je představena datová reprezentace 3D prostoru - octree a nejpoužívanější algoritmy pro hledání cesty. V rámci praktické části této práce byl vytvořen ROS balíčkek zajišťující bezkolizní navigaci robotické helikoptéry, a to pomocí programovacího jazyka C++.



**Klíčová slova:** UAV, OctoMap, ROS, A* algoritmus, RRT, detekce překážek

**Vedoucí:** Ing. Jan Chudoba



# Abstract

This bachelor's thesis focuses on the development of a collision-free system for a robotic helicopter and its integration into the MRS UAV system. The theoretical part of the thesis discusses various ways of acquiring data from sensors and the issue of helicopter localization. Furthermore, it introduces the 3D data representation - octree, and the most commonly used algorithms for finding a path. In the practical part of the thesis, ROS package for collision-free navigation was developed using the C++ programming language.



**Keywords:** UAV, OctoMap, ROS, A* algorithm, RRT, collison detection

# Obsah

# Obrázky



# Tabulky

# Kapitola 1
# Úvod

Dnešní doba přináší významný technologický pokrok, a to zejména v oblasti robotiky, kybernetiky a informatiky. Jednou z těchto progresivních oblastí je vývoj dronů (také nazývaných multirotor aerial robots, Unmanned Aerial Vehicles - UAV atd.), které se vyznačují schopností autonomního letu jak v přírodním terénu, tak i v urbanizovaném prostředí.V současnosti se drony uplatňují v široké škále aplikací, včetně vojenských a záchranných operací, průzkumu a mapování terénu, natáčení filmů z ptačí perspektivy nebo v zemědělství. Drony v těchto aplikacích musí být schopny efektivně vnímat a interpretovat své okolí, aby se vyhnuly překážkám a zachovaly bezpečnou trajektorii letu. Tuto funkčnost zajišťuje právě bezkolizní systém.

Cílem této bakalářské práce je naimplementovat bezkolizní systém pro robotickou helikoptéru, který umožní plánování bezpečné trajektorie a bude schopen dynamicky reagovat na nečekané překážky. Tento systém bude navržen jako rozšiřující balíček pro řídící systém vyvinutý v laboratoři MRS (Multi-Robot Systems) na Fakultě elektrotechnické Českého vysokého učení technického v Praze (FEL ČVUT).

V teoretické části této práce se nejprve seznámíme se senzory vidění, které se v robotice běžně používají, a prozkoumáme základní principy jejich fungování. Dále se zaměříme na to, jak získaná data z těchto senzorů reprezentovat, aby byla interpretovatelná pro systém dronu. Dalším tématem, kterému se budeme věnovat, je problematika lokalizace helikoptéry a v neposlední řadě si představíme nejpoužívanější algoritmy pro plánování trajektorie. V praktické části této práce se budeme věnovat samotné implementaci bezkolizního systému a jejímu zhodnocení.

## 1.1 Související práce

Vzhledem k rozsáhlému uplatnění v praxi se drony čím dál více stávají předmětem výzkumu a jsou předmětem mnoha vědeckých publikací. Rád bych zmínil některé z těchto prací, které mě inspirovaly a ovlivnily směřování této bakalářské práce.

První z těchto prací je bakalářská práce "Mapování prostoru robotickou helikoptérou" [18] od Ondřeje Masopusta, která se zabývá tématem mapování prostoru pomocí robotické helikoptéry. Dalším důležitým zdrojem inspirace

je práce "Collision-free Navigation System for Robotic Helicopter" [19] od Anastasie Mozgunove, která se zaměřuje na bezkolizní navigační systém pro robotickou helikoptéru. Tyto práce mi poskytly cenný materiál a podněty. Kromě toho jsem také čerpal z práce Jana Cabiara [20], která se zabývá 3D rekonstrukcí prostředí pomocí stereo vidění.

# Kapitola 2

# Senzory a datová reprezentace jejich měření

Pro plánování trajektorie je nezbytné, aby systém disponoval informacemi o svém okolí. K tomuto účelu slouží senzory pro detekci vzdálenosti, které umožňují měření dat až ve třech dimenzích. Měření v jedné dimenzi je užitečné pro zjištění například výšky, ve které se UAV pohybuje. Měření ve třech dimenzích se používá například pro vytvoření 3D rekonstrukce objektu.

V této kapitole se budeme zabývat reprezentací naměřených dat a metodami pro jejich získávání. Mezi tyto metody patří například triangulace nebo Time-of-Flight (ToF).

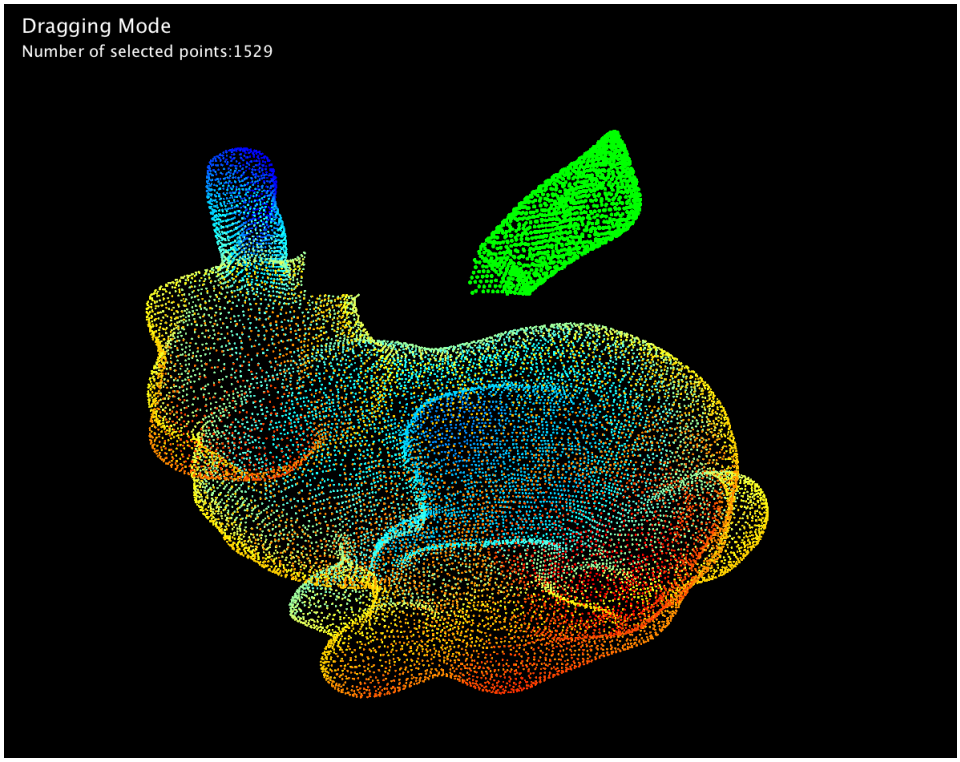
## 2.1 Datová reprezentace měření

Reprezentace informací je klíčovým faktorem v robotice. Tyto informace zahrnují data o okolním prostředí a utvářejí kontext robota, který mu umožňuje jednat a provádět dedukci. Je proto žádoucí umět tyto informace reprezentovat tak, aby byla pro robota snadno interpretovatelná a analyzovatelná. [1]

Pro reprezentaci naměřených dat ze senzorů, které detekují vzdálenost, se standardně používají dva formáty: **mračna bodů** (tzv point clouds nebo point sets) a **obrazy vzdálenosti** (tzv. range images). [2]

### 2.1.1 Mračno bodů

Mračno bodů je diskrétní množina bodů ve 3D prostoru, kde každý bod je reprezentován svými souřadnicemi $(X, Y, Z)$. [2] Tyto body mohou být naměřeny různými senzory, jako jsou laserové snímače nebo stereokamery. Mračna bodů poskytují informaci o tvaru a struktuře objektů ve scéně. Díky nim je možné provádět pokročilé analýzy, jako je segmentace objektů, detekce překážek, rozpoznávání tvarů a mnoho dalších úloh. [4]




**Obrázek 2.1:** Vizualizace mračna bodů [17]

### 2.1.2 Obraz vzdálenosti

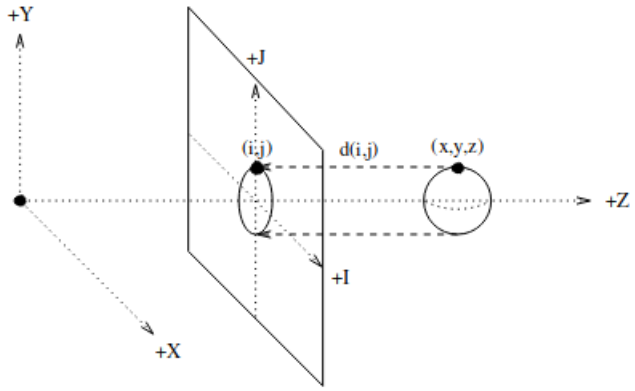
Obraz vzdálenosti je formou zobrazení prostoru, ve kterém je každý pixel $(i, j)$ reprezentován hodnotou $d(i, j)$. Tato hodnota představuje vzdálenost od senzoru k odpovídajícímu bodu ve scéně $(x, y, z)$. Tímto způsobem je možné zachytit geometrii objektů a jejich prostorové uspořádání.

Při práci s obrazy vzdálenosti existuje několik metod, jak namapovat data z $(i, j, d(i, j))$ na prostorové souřadnice $(X, Y, Z)$. Nejčastěji používaná mapování, nazývaná také projekce, jsou **ortografická**, **perspektivní**, **cylindrická** a **sférická**. Každé z těchto mapování má své vlastnosti a využití v různých aplikacích.

Při implementaci obrazů vzdálenosti je důležité provést kalibraci senzoru, aby bylo dosaženo přesných měření. Pro kalibraci se často používají parametry $\alpha$ a $\beta$, které jsou specifické pro daný senzor a slouží ke korekci v závislosti na jeho vlastnostech. [2]

- **Ortografická projekce**
  Ortografická projekce je prvním z používaných mapování při práci s obrazy vzdálenosti. Při ortografické projekci se paprsky, které vycházejí z bodů ve scéně, promítají rovnoběžně na rovinu obrazu. Každý bod $(x, y, z)$ je promítnut na bod $(X, Y)$ ve 2D obrazu, přičemž $X$ a $Y$ jsou souřadnice na rovině obrazu. Vzdálenost od roviny obrazu je zachycena jako hodnota obrazového pixelu. Výhoda ortografické projekce je, že zachovává tvar a rozměry objektů ve scéně. Díky tomu je vhodná pro aplikace, které vyžadují přesnou geometrickou reprezentaci prostoru jako je například právě počítačové vidění. [2]

$$(X, Y, Z) = (\alpha i, \beta j, d(i, j))$$

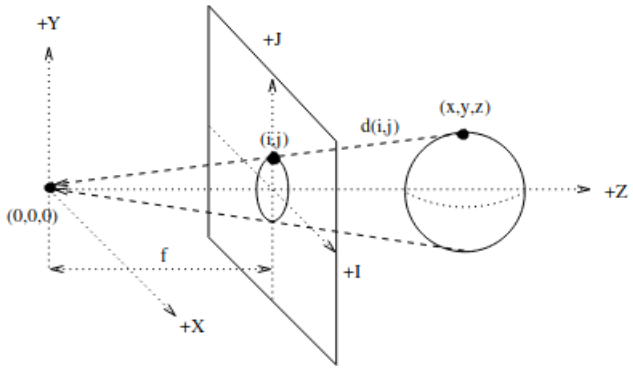



**Obrázek 2.2:** Ortografická projekce [2, str. 2]

- **Perspektivní projekce**
  Perspektivní projekce je dalším využívaným mapováním při práci s obrazy vzdálenosti. Na rozdíl od ortografické projekce se perspektivní projekce snaží zachytit iluzi hloubky a perspektivy. Při perspektivní projekci je každému pixelu $(i, j)$ v obrazu přiřazena vzdálenost $d(i, j)$, která odpovídá vzdálenosti od obrazu k prostorovému bodu $(X, Y, Z)$ ve směru paprsku, který prochází senzorem a daným bodem. Souřadnice senzoru jsou $(0, 0, 0)$. Toto mapování lze vyjádřit pomocí vztahu:

$$(X, Y, Z) = \frac{d(i, j)}{\sqrt{\alpha^2 i^2 + \beta^2 j^2 + f^2}} (\alpha i, \beta j, f),$$

  kde $\alpha$ a $\beta$ jsou kalibrační parametry a $f$ je ohnisková vzdálenost. Tyto parametry ovlivňují tvar a perspektivu projekce a jsou specifické pro daný senzor. [2]



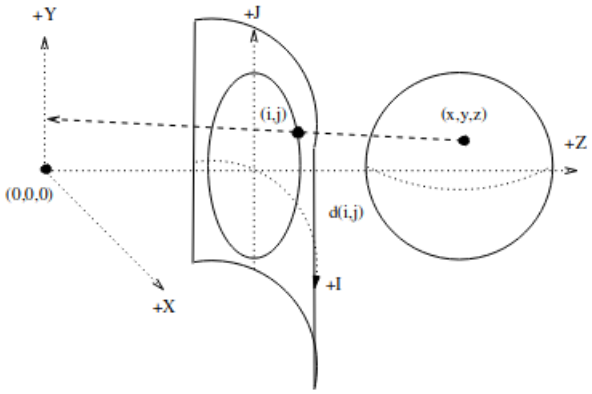

**Obrázek 2.3:** Perspektivní projekce [2, str. 2]

- **Cylindrická projekce**
  Při cylindrické projekci jsou body ve scéně promítány na válcový povrch. Tato projekce lze vyjádřit vztahem

$$(X, Y, Z) = (d(i,j)\sin\alpha i, \beta j, d(i,j)\cos\alpha i),$$

  kde $d(i,j)$ je vzdálenost v obrazu v bodě $(i,j)$, $\alpha$ a $\beta$ jsou kalibrační parametry a $i$, $j$ jsou souřadnice pixelu v obrazu.

  V cylindrické projekci jsou zachovány rovné vertikály, což znamená, že body se stejnými horizontálními souřadnicemi budou mít stejnou projekci. Navíc, délky jsou také zachovány v horizontálním směru, což zajišťuje přesnou reprezentaci vzdáleností mezi body. Při cylindrické projekci se senzor standardně otáčí ve směru osy $x$ a posouvá ve směru osy $y$. [2]



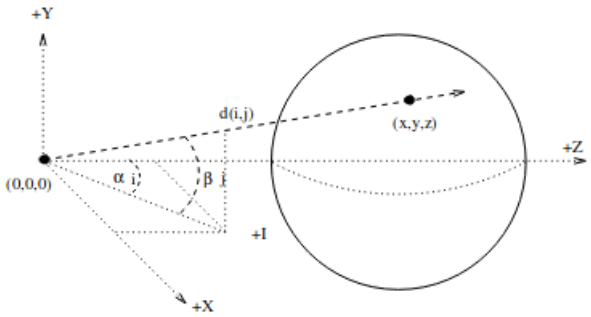

**Obrázek 2.4:** Cylindrická projekce [2, str. 2]

- **Sférická projekce**
  Posledním zmíněným způsobem mapování pro převod bodů do 2D obrazu je sférická projekce. Tato projekce simuluje promítání bodů na povrch sféry, která má své ohnisko v počátku souřadnicového systému. Při sférické projekci rotuje senzor ve směru osy x a y.

Lze vyjádřit vztahem:

$$(X, Y, Z) = d(i, j)(\cos(\beta j)\sin(\alpha i), \sin(\beta j), \cos(\beta j)\cos(\alpha i)),$$

kde $(X, Y, Z)$ jsou prostorové souřadnice promítnutého bodu, $d(i, j)$ je vzdálenost od senzoru v bodě $(i, j)$ v obrazu, $\alpha$ a $\beta$ jsou kalibrační parametry. $(i, j)$ reprezentuje azimut a vyvýšení z pohledu ohniska. [2]




**Obrázek 2.5:** Sférická projekce [2, str. 2]

## 2.2 Metody měření vzdálenosti

### 2.2.1 Triangulace

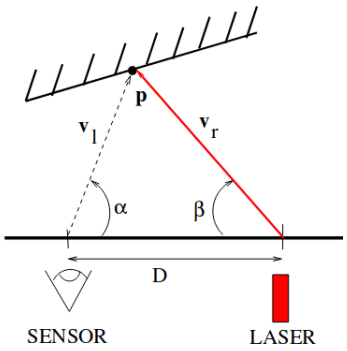
U senzorů, které detekují vzdálenost, se můžeme setkat jak s pasivní, tak i s aktivní triangulací. Obě metody triangulace jsou založeny na geometrických vztazích, které používají pro výpočet vzdálenosti a polohy bodů ve scéně. [2] [5]

#### Aktivní triangulace

Při aktivní triangulaci je povrch měřeného objektu osvícen světelným zdrojem. Senzor detekuje vzor vytvořený světelným zdrojem a pomocí známých parametrů, jako je pozice světelného zdroje, pozice snímače a úhel mezi nimi, se využívá základní trigonometrie k výpočtu polohy bodu.

U této metody se obvykle využívá laserového osvětlení, které přináší několik praktických výhod. Laserové osvětlení je možné řídit a umožňuje jedinečnou identifikaci laserového bodu s konkrétní frekvencí pomocí selektivního spektrálního filtru, který eliminuje jiné světelné zdroje. Dále lze pomocí čoček nebo zrcadel upravovat laserový bod a vytvářet tak více bodů nebo pruhů, což umožňuje měření více trojrozměrných bodů současně. Poslední výhodou je, že laserový paprsek lze přesně řídit pomocí zrcadel a umožňuje tak zaměření a pozorování konkrétní oblasti.

Tato technika má ale i nevýhody. Může být potenciálně nebezpečná pro lidský zrak a při osvícení kovového nebo leštěného povrchu může vzniknout nesprávný odraz, který vede k nesprávnému výpočtu vzdálenosti. [2]



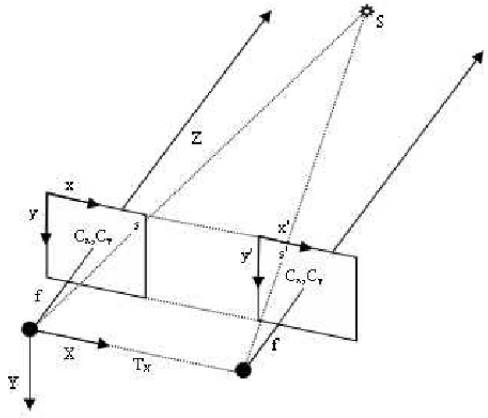

**Obrázek 2.6:** Aktivní triangulace [2, str. 9]

### Pasivní triangulace

Pasivní triangulace se liší od aktivní triangulace tím, že nepoužívá světelný zdroj, ale místo toho využívá dva senzory. Při pasivní triangulaci vytváří hledaný bod spolu se dvěma senzory trojúhelník. Díky známé vzdálenosti mezi senzory a známému úhlu, který svírají paprsky směřující do senzorů, je možné vypočítat vzdálenost bodu. Pasivní triangulace je využívána například ve **stereo vidění**, které typicky používají **hloubkové kamery**. [2] [6]

### Stereo vidění

Při stereo vidění je nezbytné provést kalibraci obou kamer, neboť čočky kamer mohou způsobit zkreslení a jiné komplikace, které ovlivňují správné nalezení odpovídajících bodů ve snímcích. Kalibrací se vytváří tzv. rektifikace původních snímků, což jsou upravené snímky s ideální stereo geometrií oproti původním snímkům 2.7.




**Obrázek 2.7:** Ideální stereo geometrie [2, str. 3] (pozn. bohužel nebyl nalezen obrázek z lepším rozlišením, který by odpovídal popisu)

Níže si představíme vztahy mezi rovinami obrazu a okolním světem reprezentovaným 3D body [2].

Pro projekci 3D bodu na levou a pravou rovinu obrazu kamery slouží projekční matice **P**.

$$\mathbf{P} = \begin{bmatrix} F_x & 0 & C_x & -F_x T_x \\ 0 & F_y & C_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

kde $F_x = fm_x, F_y = fm_y$ jsou ohniskové vzdálenosti v pixelech. Ve vzorci je $f$ ohnisková vzdálenost kamery a $m_x$ a $m_y$ jsou rozměry pixelů. Body $C_x$ a $C_y$ jsou souřadnice optického centra roviny obrazu kamer. Jedná se o ortogonální projekci ohniska, a proto se většinou nachází ve středu promítací roviny. $T_x$ je posunutí kamery relativní k referenčni kameře a pro levou kameru je tato hodnota nulová [2].

Samotná projekce mezi homogenními souřadnicemi z 3D do 2D odpovídá rovnici:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

kde $(x/w, y/w)$ jsou souřadnice ve výsledné rovině obrazu [2].

Pokud je naším cílem trojrozměrná rekonstrukce dvourozměrného obrazu, pak můžeme zjistit hloubku bodů díky reprojekční matici **Q**.

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & F_x \\ 0 & 0 & -1/T_x & \frac{C_x - C'_x}{T_x} \end{bmatrix}$$

Pokud $x, y$ a $x', y$ jsou body obrazů, které korespondují se stejným bodem ve 3D a definujeme si disparitu $d = x - x'$, pak platí

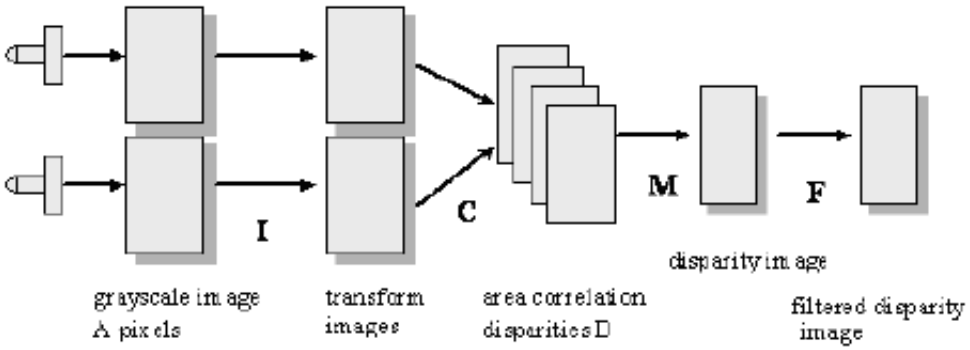
$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \mathbf{Q} \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix}$$

kde $(X/W, Y/W, Y/W)$ jsou souřadnice bodu ve trojrozměrné scéně. [2]

### Stereo metody zpracování obrazu

Důležitým problémem stereo vidění je nalézení shodných bodů na rovinách obrazů kamer, které reprezentují stejný objekt nebo bod ve scéně. K řešení tohoto problému slouží několik metod. Nejúčinnější a nejvíce praktickou metodou je **korelace plochy** (area correlation).

Korelace plochy porovnává malé plochy obrazů z kamer a snaží se najít shodu použitím korelace. Velikost těchto ploch není daná. Menší plochy mají výhodu, že jsou většinou více podobné, a tak se rychleji nalezne shoda. Výhoda velkých ploch je zase větší S/N (signal-to-noise ratio). [2] [8]




**Obrázek 2.8:** Základní stereo zpracování [2, str. 5]

Klasická metoda korelace plochy má standartně 5 kroků:

1. **Korekce geometrie**. Oprava zkreslení snímků
2. **Transfromace obrazů**. Normalizace podle průměrné lokální intenzity.
3. **Korelace plochy**. V tomto kroku probíhá samotná korelace, kde se porovnávají malé plochy obrazů.
4. **Získání extrémů**. Nalezne se extrém korelace pro každý pixel a vytvoří se obraz disparity. Každá hodnota pixelu je disparita mezi levým a pravým snímkem v nalezené nejlepší shodě (extrém).
5. **Dodatečné filtrování**. Odstranění šumu s použitím různých filtrů.

Konkrétních metod korelace plochy existuje několik. Příklady naleznete v [2].

### 2.2.2 Time of Flight (ToF)

Princip této metody je zřejmý: senzor vysílá laserový paprsek, který se odrazí od překážky a je zachycen senzorem. Senzor měří dobu, za kterou paprsek urazí tuto vzdálenost, a díky znalosti konstantní rychlosti světla je možné vypočítat vzdálenost, kterou paprsek urazil. Tento princip je podobný tomu, který využívají radary nebo sonary, a proto je senzor využívající metodu ToF nazýván **LIDAR** (Light Detection and Ranging).

Většina LIDARů je jednorozměrná (vysílají pouze jeden paprsek). V aplikacích robotiky jsou často doplněny zrcadly, která rozptylují paprsky do různých směrů, nebo se senzory otáčejí. Velkou nevýhodou LIDARů je jejich omezení na krátké vzdálenosti kvůli nedostatečnému časovému rozlišení. V některých případech je proto vhodnější využít jiné alternativy. [2]

### 2.2.3 Fázová modulace

Senzory založené na fázové modulaci buď modulují amplitudu nebo frekvenci paprsku. Tyto senzory fungují v podstatě na stejném principu jako senzory využívající metodu ToF. Nicméně, na rozdíl od nich, senzory založené na fázové modulaci neměří čas přímo, ale detekují fázový posun mezi vyslaným a přijatým paprskem. Tímto způsobem jsou schopny odhadnout čas, který paprsek potřeboval k uražení vzdálenosti k překážce, a vypočítat tak vzdálenost překážky. [2]

# Kapitola 3

# Lokalizace helikoptéry

V předchozí kapitole jsme se seznámili s principem fungování senzorů pro detekci vzdálenosti používaných v robotice. Nicméně, aby byl systém schopný plánovat bezkolizní trajektorii, nestačí pouze znát informace o okolních překážkách, ale je také potřebná informace o aktuální poloze UAV. Tímto vzniká problém lokalizace helikoptéry, který je předmětem této kapitoly.

## 3.1 SLAM

V robotice můžeme při lokalizaci UAV spoléhat na technologie, které nejsou součástí systému, ale pouze se systémem komunikují, jako například GPS nebo motion capture technologie. Při absenci těchto technologií, ale nastává problém, a proto za účelem vytvoření opravdu autonomní robotiky vznikl SLAM (Simultaneous localization and mapping), který současně vytváří mapu okolního prostoru a určuje polohu UAV. [9]

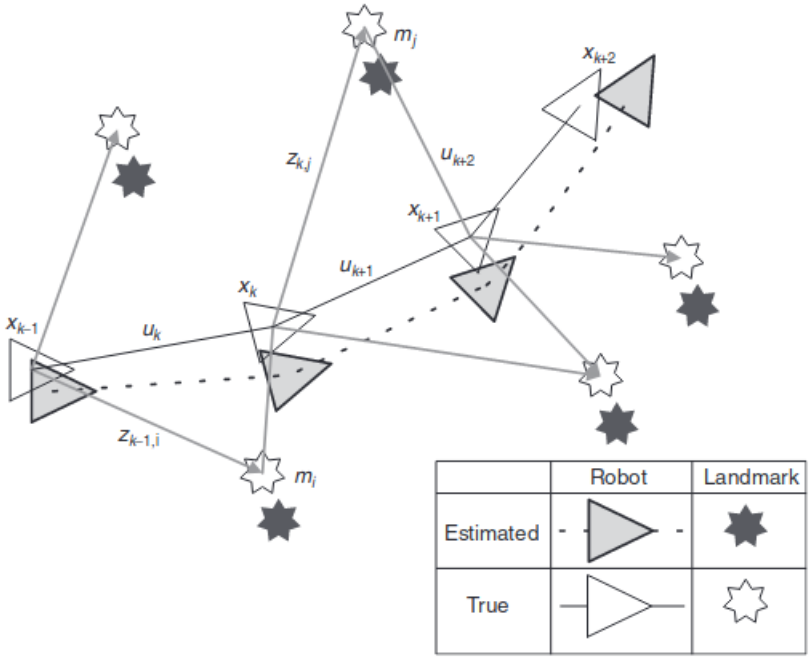
### 3.1.1 Formulace problému

Mějme mobilního robota, který pomocí senzoru zaznamenává okolí (obrázek 3.1). V čase $k$ definujeme [9]:

- $\mathbf{x}_k$: pozice a orientace, ve které se robot v čase $k$ nachází
- $\mathbf{u}_k$: řídící vektor zadaný v čase $k-1$ určený k přesunu robota do stavu $\mathbf{x}_k$ v čase $k$
- $\mathbf{m}_i$: pozice $i$-tého orientačního bodu (předpokládá se nezávislost na času)
- $\mathbf{z}_{ik}$: zpozorování $i$-tého orientační bodu v čase $k$ robotem

Dále také definujeme množiny [9]:

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k\} = \{\mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: historie polohy robota
- $\mathbf{U}_{0:k} = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_k\} = \{\mathbf{U}_{0:k-1}, \mathbf{u}_k\}$: historie řídících příkazů
- $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_n\}$: množina všech orientačních bodů
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_k\} = \{\mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: množina všech zpozorování orientačních bodů




**Obrázek 3.1:** SLAM [9, str. 100]

### 3.1.2 Pravděpodobnostní SLAM

Problematiku SLAM můžeme také vyjádřit v pravděpodobnostní formě. Je zapotřebí, abychom pravděpodobnostní rozdělení [9]

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{3.1}$$

vypočítali pro každé $k$. Pravděpodobnostní rozdělění výše popisuje pravděpodobnost pozic orientačních bodů a současné pozice robota za předpokladu, že známe startovní pozici robota, všechny řídící příkazy, které robot vykonal a všechna provedená pozorování orientačních bodů.

Pro tuto problematiku je žádoucí, aby řešení bylo rekurzivní. Mějme rozdělení $P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$ v čase $k-1$. Pak po provedení řídícího příkazu $\mathbf{u}_k$ a získání pozorování $\mathbf{z}_k$ můžeme vypočítat pravděpodobnost 3.1 pomocí Bayesova vzorce. Pro výpočet Bayesova vzorce ale musíme znát model pozorování (observation model) a model pohybu (motion model).

Model pozorování popisuje pravděpodobnost zpozorování orientačního bodu za předpokladu, že víme pozici orientačních bodů a pozici robota. [9]

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \tag{3.2}$$

Model pohybu zase popisuje pravděpodobnost pozice robota v čase $k$ za předpokladu, že známe předchozí pozici robota a řídící příkaz, který robot vydal v čase $k-1$. [9]

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{3.3}$$

SLAM algoritmus nyní můžeme implementovat jako rekurzi o dvou krocích. [9]
**Aktualizace času**

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \times P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \tag{3.4}$$

**Aktualizace měření**

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \tag{3.5}$$

Pro řešení probabilistického SLAMu je nutné najít vhodnou reprezentaci modelu pozorování a modelu pohybu. Dvě nejznámější reprezentace jsou Extended Kalman Filter (EFK) a Rao-Blackwellized filter, které jsou podrobněji rozebrány v [9, str. 103-106].

# Kapitola 4

# Datová reprezentace okolního prostoru

Systém si musí při plánování trajektorie uchovávat naměřená data. Je tedy žádoucí vytvořit mapu, která bude co nejvěrněji reprezentovat okolí UAV. Nejpopulárnější a současně nejvíce intuitivní datovou reprezentací okolního prostoru je 3D mřížka (grid). Mřížku si můžeme rozdělit na dva typy.

**Uniform grid map** je taková datová reprezentace, ve které jsou elementární body mřížky rozděleny rovnoměrně. Tyto body budeme dále nazývat voxely. (Jedná se o analogii k pixelu ve 2D mřížce). Výhodou tohoto typu mřížky je rychlé vyhledávání. Zásadní nevýhodou ale jsou vysoké nároky na paměť, neboť u každého voxelu musíme ukládat informaci o jeho stavu. Další nevýhodou je častý výskyt redundantních dat. Pokud bychom měli velký prázdný prostor mezi překážkami, nepotřebujeme přece vědět, že každý voxel v tomto prostoru je prázdný. Tento problém řeší non-uniform grid map.

**Non-uniform grid map** je druhým typem mřížky. Ten umožňuje nerovnoměrné rozdělení voxelů. V našem případě to znamená, že v okolí objektů hustota voxelů bude vyšší, než v prázdných prostorech. Díky tomu dosáhneme daleko efektivnějšího zaplnění paměti a redukce redundancí. Oproti prvnímu typu je ale prohledávání takové mřížky složitější a časově náročnější.

V další sekci si představíme knihovnu OctoMap, která implementuje právě non-uniform grid map. [11]

## 4.1 OctoMap

Při vytváření mapy pro robotickou aplikaci většinou požadujeme tři věci: pravděpodobnostní reprezentaci, efektivitu řešení a modelování volných, obsazených a nezmapovaných ploch. [11]

**Pravděpodobnostní reprezentace** prostoru je pro robotické aplikace velice důležitá. Při senzorickém měření vzdálenosti často dochází k chybám, které jsou zapříčeněné například odrazy nebo pohybujícími se předměty. To vede k určité nejistotě a je tedy žádoucí, abychom reprezentovali data jako pravděpodobnost výskytu objektu nebo prázdného prostoru (hit or miss). Dochází tak k eliminaci šumu a zpřesnění měření. [11]

**Modelování nezmapovaných ploch** je pro mobilního robota stejně důležité jako modelování volných a obsazených ploch. Robot může naplánovat bezkolizní trajektorii pouze ve známem prostředí. Pokud má robot informaci

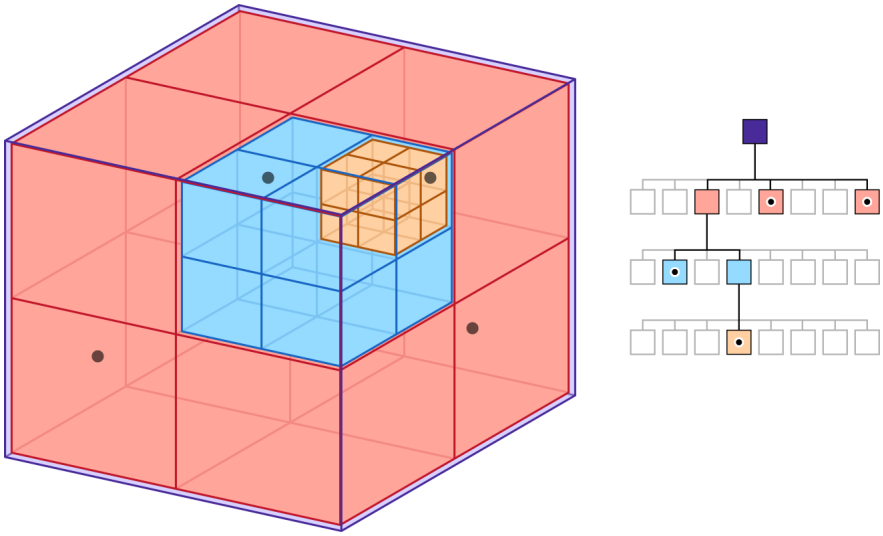
o neprozkoumaných oblastech, může se jim poté vyhnout a eliminovat riziko kolize. Při průzkumu okolí je naopak informace o nezmapovaném prostředí zásadní a robot pak tyto oblasti může prioritizovat. [11]

**Efektivita** řešení je zásadní pro jakékoli plánování. Je proto žádoucí aby prohledávání modelu bylo časově nenáročné a aby řešení nemělo vysoké paměťové nároky.

Knihovna OctoMap je volně dostupná open-source knihovna, která implementuje octree a splňuje všechny výše uvedené nároky.

### 4.1.1 Octree

Octree je stromová datová struktura, která se v robotice často používá k reprezentaci prostoru 4.1. Uzel tohoto stromu reprezentuje 3D prostor. Potomoci uzlu vznikají rozdělením dosavadního prostoru na oktanty.



**Obrázek 4.1:** Vizualizace octree [12]

Vzhledem k tomu, že octree je hierarchická datová struktura, tak dělení na oktanty můžeme rekurzivně opakovat do té doby, dokud nedosáhneme požadovaného rozlišení.

Listy octree nesou informaci o obsazenosti voxelu. Pokud senzory zachytí objekt, tak octree inicializuje příslušný voxel a označí ho za obsazený. Neinicializované voxely se považují za neprozkoumané. Knihovna OctoMap navíc inicializuje voxely mezi senzorem a obsazeným polem a označuje je za volné. [12]

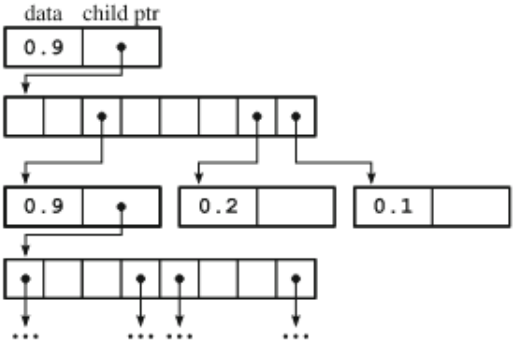
### 4.1.2 Detaily implementace

#### Implementace uzlů

Každý uzel stromu potřebuje čtyři informace: obsazenost (volný nebo obsazený), souřadnice středu, velikost voxelu a ukazatele na potomky.

Souřadnice a velikost voxelu mohou být získány při procházení stromu, takže není potřeba, aby si je uzel explicitně pamatoval.

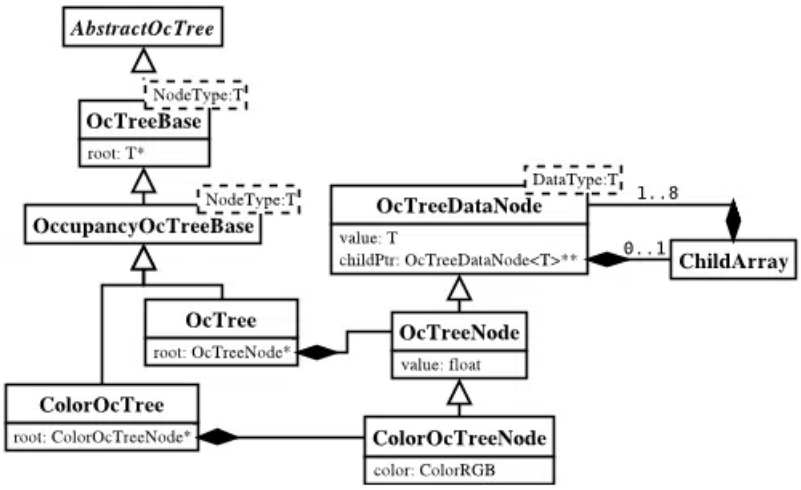
I přesto, že uzel potřebuje informaci o potomcích, se ukázalo, že by bylo velice neefektivní, kdyby každý uzel obsahoval seznam osmi ukaztelů. Místo toho si uzel uchovává ukazatel na pole ukazatelů, které je ale alokováno pouze pro vnitřní uzly stromu.




**Obrázek 4.2:** První uzly v octree [11, str. 196]

Všechny tyto opatření přispívají k tomu, že paměťové nároky jsou o 60-65% nižší. [11, str. 196]

#### Architektura octree a důležité metody




**Obrázek 4.3:** UML diagram nejběžnějších tříd octree a uzlů [11, str. 196]

Základní funkčnost octree je implementovaná v ve třídě *OcTreeBase* a základní funkčnost uzlu ve třídě *OcTreeDataNode*. *OccupancyOcTreeBase* přidává informaci o obsazenosti a nové funkčnosti jako například vkládání skenů. Hlavní třídou je *OcTree*, který vychází z *OccupancyOcTreeBase* a *OcTreeNode*. Struktura poskytuje možnost snadného rozšíření. Jako příklad jsou na obrázku4.3 uvedeny třídy, které funkčnost octree doplňují o uchovávání barvy naskenovaných objektů. [11, str. 196]

Naměřená data můžeme do octree vkládát pomocí metod třídy *OcTree*. Metoda *insertRay()* aktualizuje naměřený bod (označí ho za obsazený) a současně uvolní všechny mezibody.

Pro vkládání point cloudů je ve třídě *OcTree* připravena metoda *insertScan()*, která je zoptimalizována, aby byla efektivnější než vkládání jednotlivých paprsků. [11]

# Kapitola 5

# Plánování trajektorie

V minulých kapitolách byly představeny různé senzory pro detekci objektů a způsoby jak zmapovat okolí. Tato kapitola bude pojednávat o plánování konkretní trasy na základě získaných informací. Opět existuje spoustu možností jak k této problematice přistupovat. V robotice se využívá zejména metod prohledávání grafu a RRT (Rapidly-exploring random tree). V naší aplikaci využijeme algoritmus A*, který slouží k efektivnímu prohledávání grafu, i algoritmus RRT.

### 5.0.1 RRT

Tento algoritmus je v robotice velice popoulární. V podstatě se jedná o generování náhodných bodů, které propojujeme a vytváříme tak strom. Kořenem tohoto stromu je vždy výchozí pozice zařízení. Ve vytvořené mapě si zvolíme náhodný bod. K tomuto bodu nalezneme bod stromu, jehož souřadnice jsou nejblíže. Na spojnici těchto dvou bodů zvolíme bod podle předem definované délky kroku (např. desetina vzdálenosti mezi dvěma body). Dále kontrolujeme, jestli se na spojnici nového bodu a bodu ve stromu nachází překážka a pokud ne, tak přídáme nový bod do stromu. Kroky opakujeme a tímto způsobem nalezneme cestu k požadovanému bodu. Při použití vhodných heurestik se jedná o velice efektivní způsob nalezení trasy. [13] [14]

**Algorithm 1** RRT

$V \leftarrow \{x_{init}\}; E \leftarrow \emptyset$
**for** $i = 1, ..., n$ **do**
  $x_{rand} \leftarrow SampleFree();$
  $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
  **if** $ObstacleFree(x_{nearest}, x_{new})$ **then**
    $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\};$
  **end if**
**end for**
**return** $G = (V, E);$

### 5.0.2 A* algoritmus

Tento algorimtus patří do rodiny algoritmů, které řeší plánování trasy prohledáváním grafu. Jedná se o rozšíření klasického BFS (Breadth-first search) nebo DFS (Depth-first search). Cílem tohoto algoritmu je nalézt optimální cestu a minimalizovat její cenu

$$c(x) = g(x) + h(x)$$

,kde $x$ je vrchol, $g(x)$ je vzdálenost od výchozí pozice a $h(x)$ je heurestika. [13]

**Algorithm 2** A* algorithm

$open \leftarrow \{node_{start}\}; closed \leftarrow \emptyset;$
**while** $open \neq \emptyset$ **do**
  $node_{current} \leftarrow findLowestCost(open);$
  $pop(open, node_{current});$
  $neighbours \leftarrow getNeighbours(node_{current});$
  **for** $n \in neighbours$ **do**
    **if** $n = goal$ **then**
      **return**
    **end if**
    $n.cost \leftarrow distance() + heurestics()$
    **if** $closedHigherCost(n, closed)$ or $openHigherCost(n, open)$ **then**
      $open \leftarrow open \cup \{n\};$
    **end if**
  **end for**
  $closed \leftarrow closed \cup \{node_{current}\}$
**end while**
**return**

### Heurestiky

V některých případech je prohledávání všech vrcholů grafu velice časově náročné. V takových případech využíváme hurestiky. Definice heurestiky je jednoduchá. Jedná se vlastně o "návod", díky kterému algoritmus ví jaký další vrchol grafu navštívit. Při využití vhodné heurestiky tak můžeme výrazně snížit počet navštívených uzlů. Heurestik existuje celá řada. Pro naši úlohu použijeme eukleidovskou a manhattanskou vzdálenost od koncového bodu. Algoritmus tak bude přednostně navštěvovat body, které se blíží k cíli. [13]

# Kapitola 6

# Implementace bezkolizního systému

Hlavním cílem této bakalářské práce je rozšíření řídícího systému robotické helikoptéry, které bude zajišťovat bezkolizní přelet do cílové destinace. Po konzultaci s vedoucím a oponentem práce bylo rozhodnuto, že řešení této úlohy bude zjednodušeno o implementaci komponenty (octomap_server), která odebírá data ze senzorů a vytváří octree, a namísto toho se použije již naimplementovaný ROS balíček mrs_octomap_server, který nabízí stejnou funkčnost. Výstupem této práce je ROS balíček **path_planner**, který zahrnuje implementaci plánovacích algoritmů a funkčnost detekce překážek. Díky tomu poskytuje řídícímu systému schopnost plánovat bezkolizní trajektorii. Kód je napsán v programovacím jazyce C++.

## 6.1 Použité nástroje a technologie

- **MRS UAV system**
  Tato platforma je především určena pro výzkum a nabízí celou škálu funkčností jako řízení helikoptéry, plánování trajektorie, počítačové vidění nebo sledování. K MRS UAV system byl vyvinut i simulátor (na bázi Gazeba), jehož hlavní předností je, že modely dronů v simulátoru odpovídají reálným dronům dostupných v laboratoři. Díky tomu, je přechod ze simulace do reálného světa velice plynulý. [16]

- **ROS**
  Robot Operating System je navzdory názvu open-source middleware. Jinými slovy se jedná o sadu knihoven a nástrojů určených k vývoji robotických aplikací. Poskytuje služby jako abstrakci hardwaru, ovladání nízkoúrovňových zařízení, předávání zpráv mezi procesy, správa balíčků atd. V praxi se jedná o peer-to-peer síť procesů, které spolu komunikují. ROS jako takový se skládá z šesti objektů: [15]

  - Uzel (node) - základní výpočetní jednotka
  - Zpráva (message) - základní datový typ, který ROS využívá k publikaci a odběru
  - Kanál (topic) - uzly publikují do kanálu zprávy, nebo je z něj odebírají

- Master - statá se o organizaci a vzájemnou komunikaci mezi uzly
- rosout - Ekvivalent ROS ke standartnímu vstupu a výstupu.
- roscore - Master + rosout + parameter server

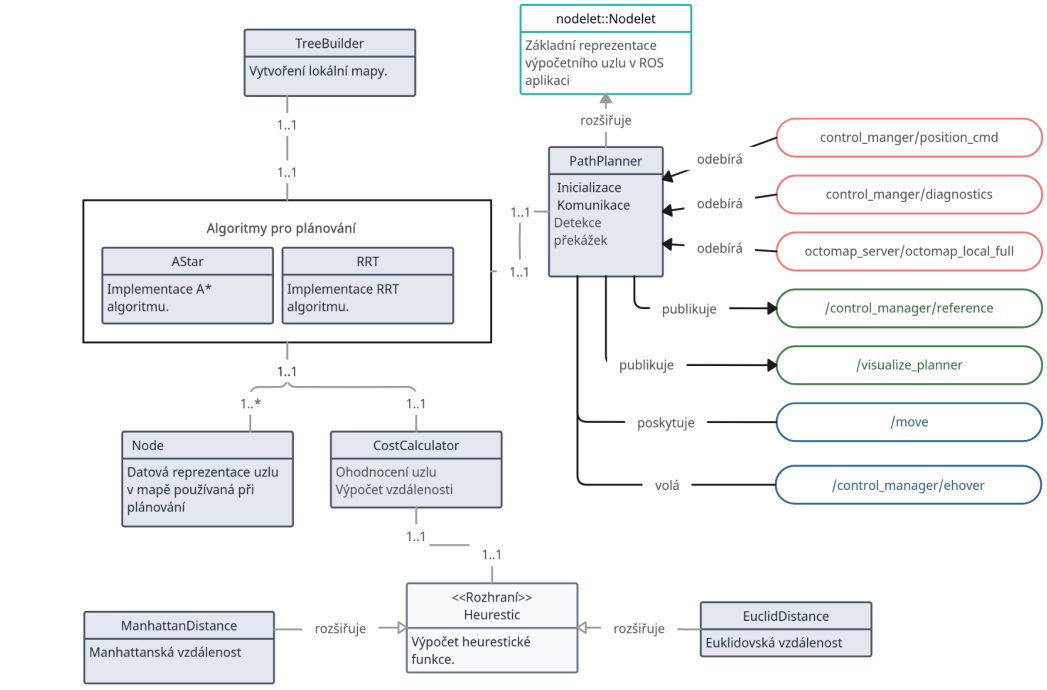
Seznam použitých ROS balíčků:

- mrs_octomap_server - vytváří octree na základě přijmaných dat ze senzoru
- octomap - implementace octree
- octomap_msgs - poskytuje zprávy a serializaci pro knihovnu OctoMap
- octomap_ros - poskytuje konverzi mezi základními typy ROS a OctoMap
- octomap_rviz_plugins - vizualizace
- octomap_tools - další nástroje pro knihovnu OctoMap

- **OctoMap**
- **RViz**
  Nástroj k 3D vizualizaci dat určený pro ROS.
- **Další použité C++ knihovny**
  - Eigen - vysokoúrovňová knihovna pro lineární algebru, maticové a vektorové operace, geometrické transformace atd.
  - dynamicEDT3D - knihovna poskytující EDT (Euclidean Distance Transform) pro 3D.
- **Ostatní nástroje**
  - Tmux - terminálový multiplexor
  - Bash, Python - skriptování
  - Singularity - kontejnerizace
  - Ubuntu 20.04 LTS (Focal Fossa)

## 6.2 Architektura balíčku path_planner

Při návrhu a implementaci této softwarové architektury bylo dbáno na dodržování zásad dobrého kódu. Cílem bylo vytvořit čistý, čitelný a dobře zdokumentovaný kód, který splňuje splňuje standarty softwarového vývoje. Důraz byl kladen na modularitu, zapouzdření a škálovatelnost balíčku, což by mělo vést k jednoduché údržbě a případnému budoucímu rozšíření. Konvence typické pro programovací jazyk C++ (pojmenování proměnných, funkcí, tříd atd.) byly zachovány, aby se zajistila konzistence kódu.




**Obrázek 6.1:** Architektura implementovaného rozšíření

Nejprve si krátce shrneme architekturu znazorněnou na diagramu 6.1 a konkrétní třídy si rozebereme v následující sekci.

Hlavní součástí balíčku je třída **PathPlanner**, která má na starosti inicializaci a komunikaci s ostatními uzly systému. Komunikace probíhá prostřednictvím kanálů, ze kterých třída odebírá zprávy a do kterých je publikuje, a pomocí služeb, které poskytuje, nebo volá. Tato třída uchovává informaci o stavu dronu a na základě tohoto stavu spouští plánování a řídí chování dronu. Kromě toho také zajišťuje detekci nečekaných překážek.

Další důležitou komponentou je implementace plánovacích algoritmů. V praktické části byly naimplementovány třídy **AStar** (A*) a **RRT**, které tyto algoritmy reprezentují. Tyto třídy přijímají mapu získanou z třídy PathPlanner, vytvářejí její lokální kopii a upravují ji tak, aby brala v úvahu bezpečnou vzdálenost dronu od překážky. Poté vypočítávají trajektorii (v případě A* optimální) pro bezkolizní let k cílové destinaci. Pokud je cíl mimo zmapovaný prostor, generují dočasný cíl a v případě nedosažitelného cíle AStar aplikuje "zavírání"regionů. K vytvoření lokální kopie mapy využívají algorimty třídu **TreeBuilder**. Navštívené uzly jsou reprezentovány třídou **Node** a k výpočtu jejich ceny slouží třída **CostCalculator**.

## 6.3 Jednotlivé části balíčku path_planner

### 6.3.1 Třída PathPlanner

Během celého procesu přeletu může dron zaujmout několik různých stavů. Tyto stavy jsou reprezentovány pomocí výčtové třídy (tzv. enum class) **UAVState**. Zde je stručný popis jednotlivých stavů:

- **INITIALIZING** - Probíhá inicializace uzlu.
- **READY** - Dron se vznáší na místě a je připraven na plánování.
- **HOVERING** - Dron se vznáší na místě.
- **FLYING** - Dron je v pohybu a míří k dalšímu bodu trajektorie.
- **PLANNING** - Dron plánuje trajektori.
- **NO_PATH_TO_FOLLOW** - Žádná cesta nenalezena.
- **REACHED_GOAL** - Dron dosáhl cíle.

Ve těchto stavech třída **PathPlanner** plní své klíčové funkce, které zajišťují správnou komunikaci, řízení dronu a detekci překážek.

Pro komunikaci s ostatními uzly v systému třída využívá konkrétně pěti kanálů a dvou služeb:

- *control_manager/position* – Z tohoto kanálu odebírá třída zprávy o současné pozici dronu. Při příchodu zprávy se probouzí zpětné volání (tzv. callback), které aktualizuje atribut **start_**. Tento atribut se používá jako startovací pozice pro jakékoli plánování a reprezentuje aktuální polohu dronu.
- *control_manager/diagnostics* – Z tohoto kanálu třída odebírá zprávy o diagnostice řídícího systému. Při příchodu zprávy se probouzí zpětné volání, které kontroluje jestli je dron ve stavu FLYING a zároveň jestli řídící systém nemá cíl (příkaz k pohybu nastaví atribut diagnostiky have_goal jako pravdivý, dokud dron nedosáhne požadovaného bodu). Tím se zajistí, že dron nejprve dorazí do požadovaného bodu trajektorie a až potom následuje další. Při splnění těchto podmínek je stav dronu změněn na HOVERING.
- *octomap_sever/octomap_local_full* – Z tohoto kanálu třída odebírá zprávy, které obsahují octree. Při příchodu zprávy se probouzí zpětné volání, které vytvoří kopii octree a uloží ji do atributu **octree_global_**.
- *control_diagnostics/reference* – Na tento kanál třída publikuje referenci bodu, na jehož souřadnice se má dron přesunout. Vyvolává tím příkaz k pohybu.

- */visualize_planner* - Na tento kanál se publikují data, která se mají vizualizovat ve RViz.
- */move* - Toto je služba která spouští celý proces přeletu do uživatelem zadané cílové destinace. Při volání probíhá kontrola připravenosti systému a jeho inicializace. Následně uloží cíl do atributu **goal_** a mění stav dronu z INITIALIZING na READY.
- */control_manager/ehover* - Tato služba se volá v případě detekce nečekané překážky. Dron tak nouzově zastaví, zahodí současnou trajektorii, změní stav dronu na HOVERING a čeká na přeplánování.

Samotný proces je řízen časovačem **timerMain(timerEvent)**, který se spouští s frekvencí nastavenou v konfiguračním souboru. Nejprve se kontroluje, zda je uzel inicializován, a zda se odebírají zprávy ze všech kanálů. Pokud ano, nastavuje se atribut **ready_to_plan_** na pravdu, čímž se signalizuje připravenost systému.

Pokud je dron ve stavu READY vytváří instanci plánovacího algoritmu a volá metodu této instance **findPath(start_coord, goal_coord, mapping_tree)**, která jako parametry přijímá souřadnice startu, cíle, mapu a ukládá si její výstup, kterým je naplánovaná trajektorie. Pokud cíl není nalezen dron přechází do stavu NO_PATH_TO_FOLLOW. Pokud je dron v cíli přechází do stavu GOAL_REACHED. Ve všech ostatních případech přechází dron do stavu HOVERING.

Pokud je dron ve stavu HOVERING, nemá žádnou trajektorii, kterou by následoval a přesto není v cíli, tak to znamená, že dosáhl dočasného cíle a tím pádem přechází do stavu READY a čeká ho naplánování dalšího úseku trajektorie. Pokud je dron ve stavu HOVERING a má naplánovanou trajektorii, tak se provede příkaz k pohybu a dron mění svůj stav na FLYING.

Detekce nečekaných překážek je řízena časovačem **timerSafety(timerEvent)**, který se spouští s frekvencí nastavenou v konfiguračním souboru. Pokud se dron nachází ve stavu FLYING, je v mapě vyslán paprsek, jehož zdroj je v současné pozici dronu a směřuje k bodu, ke kterému dron letí. K tomu slouží metoda **castRay(origin, direction, end, ignoreUnknownCells, max_range)** třídy **octomap::OcTree**. Pokud paprsek zachytí kolizi, znamená to, že se na naplánované trajektorii, která byla původně bezkolizní, vyskytla nečekaná překážka. V takovém případě je volána služba *control_manager/ehover*, která provede nouzové zastavení. Naplánovaná trajektorie se zahazuje a dron přechází do stavu HOVERING, kde čeká na naplánování nové trajektorie.

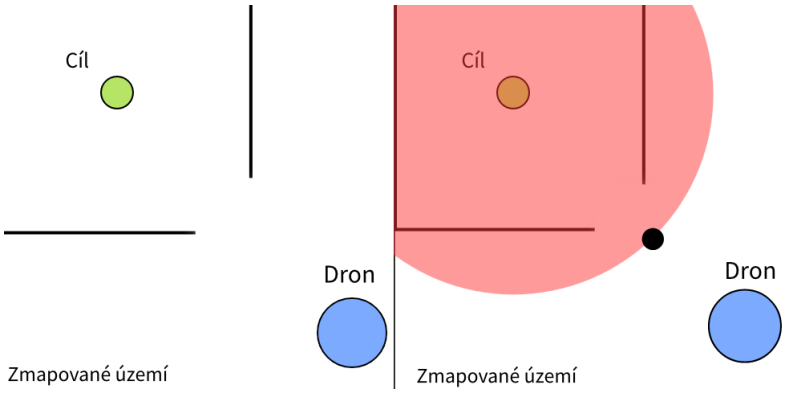
### 6.3.2 Třída AStar

Tato třída implementuje vyhledávací algoritmus A*. Hlavní metodou této třídy je metoda **findPath(start_coord, goal_coord, mapping_tree)**. Tato metoda slouží k zahájení procesu plánování trajektorie. Přijímá počáteční souřadnice, cílové souřadnice a mapovací strom. Jejím úkolem je spustit algoritmus pro hledání optimální cesty mezi počátečním a cílovým bodem v

trojrozměrném prostředí (pozn. v této aplikaci je optimální cestou myšlena cesta v mapovacím stromu, nikoli v reálném prostředí, kvůli omezením měření). Tímto způsobem je umožněno naplánovat trajektorii pro pohyb v tomto prostoru.

Na začátku jsou importovány nezbytné knihovny a třídy, a následně jsou inicializovány potřebné datové struktury. Metodou **init()** se inicializuje A* algoritmus nastavením počátečního bodu a vytvořením mapovacího stromu. V případě nedosažitelnosti cíle (cesta vede mimo zmapované území) metoda **generateTemporaryGoal(goal_key)** generuje dočasný cíl.

Generace dočasného cíle neprobíhá náhodně, algoritmus vybírá volný uzel nejlbižší cíli. Při generaci nového cíle může dojít k tomu, že algoritmus vybere viditelný uzel, který ale není pro dron dosažitelný. Během procesu plánování si algoritmus uchovává vzdálenost mezi nejbližším navštíveným bodem (k cíli) a cílem a v případě nedosažitelnosti cíle následuje volání metody **closeRegion(goal_key)**. Metoda iteruje přes mapovací strom a aktualizuje hodnoty všech uzlů, které spadají do regionu definovaného touto vzdáleností, na hodnotu Occupancy::OCCUPIED. Tento proces se opakuje, dokud není nalezen dosažitelný dočasný cíl, nebo pokud nepřekročí konfigurační parametr **replanning_limit_**., který reprezentuje limit počtu přeplánování.




**Obrázek 6.2:** Zavírání regionů

Metoda **findNeighbours(current_node,goal_key)** slouží k nalezení volných sousedů aktuálního uzlu a aktualizaci otevřené množiny uzlů. Expanze probíhá ve všech 26 směrech trojrozměrné mřížky.

Metoda **reconstructPath(goal_node)** rekonstruuje nalezenou cestu z koncového uzlu zpět k počátečnímu uzlu.

Metody **visualizePoint(node_key)** a **visualizePath(Path)** jsou zodpovědné za vizualizaci cíle a trajektorie.

### 6.3.3 Třída RRT

Tato třída implementuje vyhledávací algoritmus RRT (Rapidly Exploring Random Tree) a je navržena tak, aby umožňovala generování bezkolizních cest mezi zadaným startovním a cílovým bodem. Hlavní metodou třídy RRT je stejně jako ve třídě AStar metoda **findPath(start_coord, goal_coord,**

**mapping_tree)**, která přijmá jako parametry startovní bod, cílový bod, ke kterému má být nalezena cesta, a mapovací strom.

V prvním kroku metoda inicializuje strom RRT, který obsahuje pouze startovací bod. Následně probíhá iterativní proces, ve kterém se postupně rozšiřuje strom směrem k cílovému bodu.

V každé iteraci se generuje náhodný bod v prostoru pomocí metody **getRandomNode()**. Dále algorimtus nalezne nejbližší bod ve stromu k tomuto náhodnému bodu metodou **getClosestNode()** a na spojnici těchto dvou bodů vybere nový bod pomocí metody **selectNodeBetweenPoints( closest_node_coord, random_node_coord, step)**, která příjmá jako parametry tyto dva body a délku kroku, která se definuje v konfiguračním souboru. Následně algoritmus testuje prostor mezi nejbližším bodem a novým bodem na kolizi užitím metody **isFreeRay(closest_node_coord, new_node_coord)**.

Tato metoda vysílá paprsek v mapovacím stromu s původem v nejbližším vybraném bodě a s cílem v nově nalezeaném bodě. Pokud je spojnice mezi těmito body bezkolizní, je nový bod přidán do stromu. Tím dochází k postupnému rozšiřování stromu ve snaze najít cestu do cílového bodu.

Tento proces pokračuje až do dosažení cílového bodu nebo dosažení maximálního počtu iterací. Pokud je cílový bod dosažen, je sestavena cesta zpět k startovnímu bodu pomocí metody **reconstructPath(current_node)**.

K vizualizaci se kromě stejných metod jako ve třídě Astar používá metoda **visualizeRay(ray)**, která vizualizuje hrany stromu.

### 6.3.4 Třída TreeBuilder

Tato třída implementuje návrhový vzor Builder, který umožňuje jednoduché vytváření mapy s ohledem na bezpečnou vzdálenost od překážek.

Při vytváření mapy se nejprve přidá původní mapa pomocí metody **addMappingTree(octree)** a následně se přidá konfigurace pomocí metody **addConfig(tree_config)**. Zavoláním metody **build()** se vytvoří samotná mapa s odpovídajícím rozlišením a postupně prochází všechny uzly v původním stromu. Pro každý uzel je vypočítána vzdálenost od nejbližšího obsazeného uzlu pomocí **EDT** (Euclidean Distance Transform). Pokud je tato vzdálenost menší nebo rovna bezpečné vzdálenosti, je hodnota uzlu ve vytvořeném mapovacím stromu nastavena na **Occupancy::OCCUPIED**. V opačném případě je hodnota uzlu nastavena na **Occupancy::FREE**.

### 6.3.5 Ostatní třídy

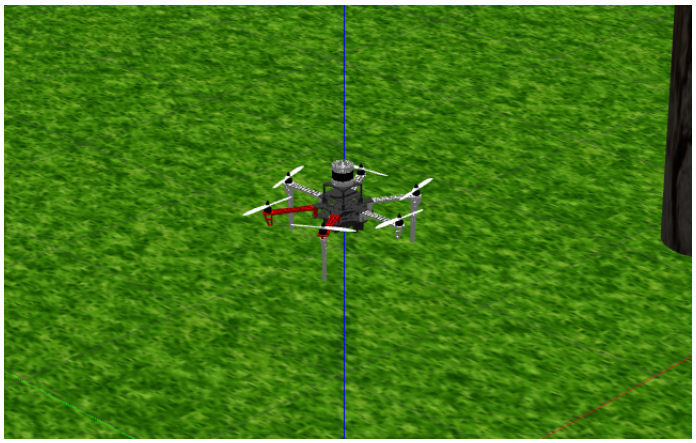
Třída **CostCalculator** příjmá v konstruktoru instanci třídy libovolné heurestiky (naimplementovány jsou euklidovská a manhattanská vzdálenost). Tato třída obsahuje metody, které vrací celkovou vzdálenost, vzdálenost mezi dvěma uzly a hodnotu heurestické funkce.

# Kapitola 7

# Výsledky a diskuze

## 7.1 Výsledky

Pro ověření funkčnosti implementovaných algoritmů byly provedeny testy v zalesněném prostředí a na nerovném terénu. Cílem testování bylo získat bezkolizní trajektorii pro helikoptéru a dosáhnout požadovaného cíle. Testování probíhalo v simulovaném prostředí s použitím modelu dronu **DJI f550**. Dron byl vybaven jednorozměrovým lidarem Garmin směřujícím dolů a 3D lidarem OS0-128.

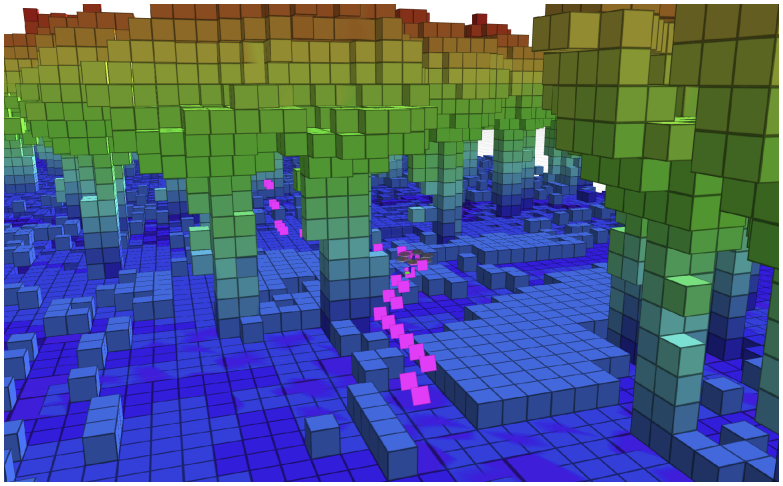


**Obrázek 7.1:** Model dronu DJI f550

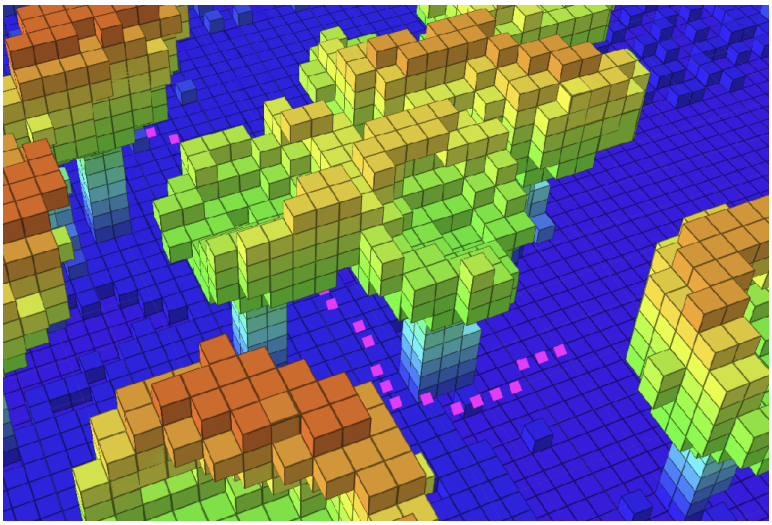
Během testování byly sledovány různé parametry, včetně času potřebného k nalezení cesty, délky nalezené trajektorie a úspěšnosti nalezení cesty. Průběh plánování trasy a vyhýbání se překážkám byl vizualizován v reálném čase pomocí vizualizačního nástroje RViz určeného pro ROS. Tím bylo umožněno detailní sledování chování algoritmů a ověření jejich správnosti.

Během testování detekce dynamických překážek byl detekční radius v konfiguračním souboru zvýšen na hodnotu vyšší než je bezpečná vzdálenost od překážek. Tím bylo dosaženo toho, že pokud se dron přiblížil k překážce, byla tato překážka detekována a trajektorie byla přeplánována. I když výsledná trajektorie nebyla optimální, testování potvrdilo správné chování detekce. V dalších testech byl detekční radius opět snížen na původní hodnotu.

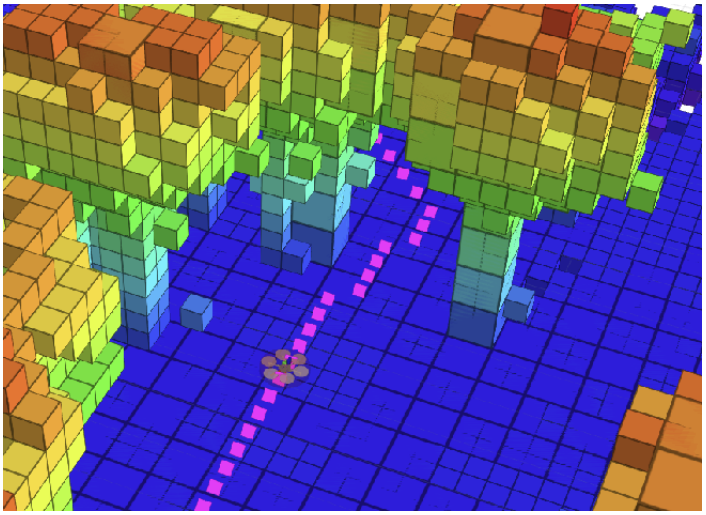
Výsledky těchto testovacích scénářů potvrzují funkčnost implementovaného RRT a A* algoritmu při navigaci helikoptéry. Helikoptéra byla schopna nalézt bezpečnou trajektorii, vyhýbat se překážkám a úspěšně dorazit ke svému cíli.



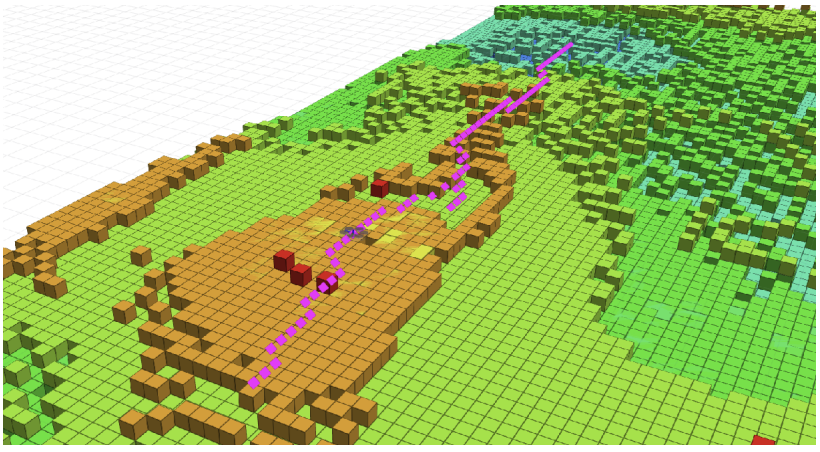
**Obrázek 7.2:** Vizualizace trajektorie nalezené A* algoritmem 1



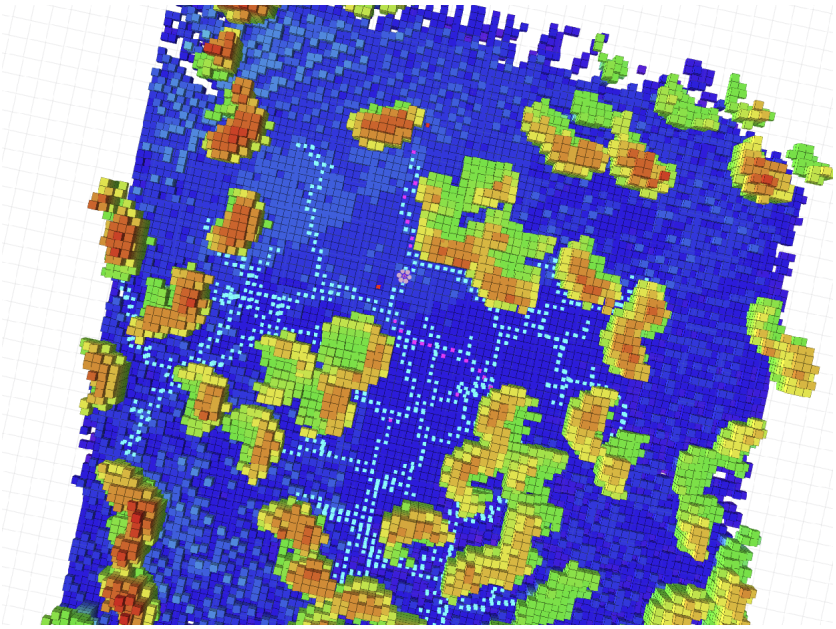
**Obrázek 7.3:** Vizualizace trajektorie nalezené A* algoritmem 2



**Obrázek 7.4:** Vizualizace trajektorie nalezené A* algoritmem 3



**Obrázek 7.5:** Vizualizace trajektorie v nerovném terénu



**Obrázek 7.6:** Vizualizace růstu RRT a naplánované trajektorie.

## 7.2 Diskuze

V této diskuzi se zaměříme na zhodnocení efektivity implementovaných algoritmů a na návrhy pro jejich rozšíření a vylepšení.

Kromě vylepšení (uvedených níže) stávajících algoritmů je také možné zvážit implementaci nových algoritmů a využití vhodnějších heuristik. Implementovaný systém je navržen s ohledem na modularitu a škálovatelnost, což umožňuje snadné rozšíření funkcionality. Pro složitější aplikace může být tato možnost velmi přínosná, a pomocí implementace nových algoritmů nebo jejich kombinace je možné dosáhnout efektivnějšího řešení.

### 7.2.1 Rychlost plánování

Prvním zkoumaným parametrem je rychlost plánování. Časová náročnost implementace A* algoritmu odpovídá očekáváním. I přestože je pozorovatelné mírné zpomalení v komplexním prostředí, není nijak zásadní. Významné zpoždění lze pozorovat při opakovaném přeplánování. V implementaci jsou použity vhodné datové struktury a výpočetně nenáročné heurestiky. Pro dosažení dalšího zlepšení efektivity a rychlosti plánování bychom mohli zvážit paralelizaci algoritmu.

Rychlost plánování algoritmu RRT se výrazně zpomaluje s rostoucí mapou. Pro zlepšení tohoto problému lze využít vhodných heuristik pro výběr bodů, které nejsou zcela náhodné. (např. bod nejbližší k cíli). Tímto způsobem by se čas plánování mohl výrazně zkrátit.

Oba algoritmy mohou využít přístupu, kde se plánuje trajektorie na menším úseku mapy a předešlá mapa se postupně zapomíná. Tento přístup může být výhodný pro snížení výpočetních a paměťových nároků.

### 7.2.2 Délka trajektorie

A* algoritmus je schopen nalezení optimální trajektorie v dané mapě, zatímco RRT algoritmus toto nezaručuje.

Pro zlepšení RRT algoritmu je možné aplikovat vhodné heuristiky pro výběr bodů, zvolit vhodnou délku kroku v závislosti na konkrétním prostředí nebo implementovat RRT* algoritmus [14], který se snaží dopočítávat nejkratší cestu v grafu k cíli. Tímto způsobem se může zvýšit pravděpodobnost nalezení optimální trajektorie.

### 7.2.3 Rychlost přeletu

Co se týče rychlosti přeletu, hlavním problémem A* algoritmu je, že generuje body blízko sebe, což může vést k častému zastavení dronu. Možností, jak tento problém eliminovat, je ignorovat body ležící na spojnici mezi dvěma body, čímž by se snížil počet bodů v trajektorii a dron by méně často zastavoval.

Dalším potenciálním vylepšením je úprava rychlosti letu. V rovných úsecích by dron mohl zvýšit rychlost a rychleji je tak překonat. Pro oba algoritmy je

také možné implementovat vyhlazení zatáček, které by umožnilo plynulejší vyhýbání se překážkám.

# Kapitola 8

# Závěr

V této bakalářské práci jsme se zaměřili na vývoj bezkolizního systému pro robotickou helikoptéru. Nejprve jsme se seznámili s teoretickými znalostmi nezbytnými pro tento úkol. Popsali jsme nejčastěji používané technologie pro měření vzdálenosti objektů a věnovali se také problému lokalizace helikoptéry.

Dále jsme se zabývali reprezentací zmapovaného okolí a základními algoritmy pro plánování cesty. Pro implementaci těchto konceptů jsme vytvořili ROS balíček v programovacím jazyce C++, který rozšířil funkčnost MRS UAV systému o možnost plánování bezkolizní trajektorie a detekce překážek.

Během testování jsme ověřili správnou funkčnost našeho balíčku a potvrdili jeho schopnost úspěšně plánovat bezkolizní trajektorie a detekovat překážky.

I přes určité obtíže, zejména při poznávání fungování MRS UAV systému a ROS, jsme byli schopni dosáhnout cílů této práce. Pro autora to byla cenná zkušenost, která ho posunula vpřed v oblasti robotiky, kybernetiky a softwarového inženýrství.

# Literatura


[1] *Context Representation and Reasoning in Robotics-An Overview* Dimitropoulos, Konstantinos and Hatzilygeroudis, Ioannis. 2022. isbn: 978-3-030-80570-8. doi:10.1007/978-3-030-80571-5_6.

[2] *Handbook of Robotics Chapter 22-Range Sensors,.* 2008. Fisher, Robert Konolige, Kurt.

[3] *3D is here: Point Cloud Library (PCL).* R. B. Rusu and S. Cousins: 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 1-4, doi: 10.1109/ICRA.2011.5980567.

[4] *Research on 3D Point Cloud Object Detection Algorithm for Autonomous Driving*, Haiyang Jiang and Yuanyao Lu and Shengnan Chen, Mathematical Problems in Engineering, 2022.

[5] *Active triangulation technique.* Kalová, I.; Lisztwan, M. In 5th International Conference of PhD Students. Miskolc: University of Miskolc, 2005. p. 99-104. ISBN: 963-661-673- 6.

[6] *A A R C H I V E S The use of optical scanning for analysis of casting shape.* Wieczorowski, Michal and Grzelka, Mirosław and Budzik, Grzegorz and Augustyn-Pieniazek, J. 05/2023

[7] *Determining the Epipolar Geometry and its Uncertainty: A Review.* Zhang, Z. International Journal of Computer Vision 27, 161–195 (1998). https://doi.org/10.1023/A:1007941100561

[8] *Area-based correlation and non-local attention network for stereo matching.* Li, X., Fan, Y., Lv, G. et al. Vis Comput (2021). https://doi.org/10.1007/s00371-021-02228-w

[9] *Simultaneous localization and mapping: part I.* Durrant-Whyte and T. Bailey, in IEEE Robotics  Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.

[10] *The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles.* Baca, T., Petrlik, M., Vrba, M. et al. J Intell Robot Syst 102, 26 (2021). https://doi.org/10.1007/s10846-021-01383-5

[11] *OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees* in Autonomous Robots, Hornung A., Wurm K.M., Bennewitz M., Stachniss C., and BurgardW. 2013, DOI: 10.1007/s10514-012-9321-0.

[12] *Octree Construction and Nearest Neighborhood Search(NNS)* Dulal, Saurab. 14 Mar. 2018,https://dulalsaurab.github.io/computerscience/octree-construction-and-nns/. Accessed 11 Aug. 2022.

[13] *Survey of UAV motion planning - IET Cyber-systems and Robotics,* Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, Fei Gao. 2020.

[14] *Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions.* Iram Noreen, Amna Khan, Zulfiqar Habib. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 11, 2016

[15] Dokumentace ROS, URL - http://wiki.ros.org/

[16] Dokumentace MRS UAV system, URL - https://ctu-mrs.github.io/

[17] 3D Point Cloud Editor, Caltech 2012, URL - https://www.paradise.caltech.edu/ yli/software/pceditor.html

[18] Ondřej Masopust: *Mapování prostoru robotickou helikoptérou.* Bakalářská práce, ČVUT v Praze, 2021

[19] Anastasiia Mozgunova: *Collision-free navigation system for robotic helicopter.* Bakalářská práce, ČVUT v Praze, 2021

[20] Jan Cabiar: *Rekonstrukce 3D modelu prostředí helikoptérou.* Bakalářská práce, ČVUT v Praze, 2016