Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Metody plánování trajektorie pro UAV

Plzeň, 2019 Josef Navrátil

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni, dne 24.5.2019 ……

vlastnoruční podpis

Poděkování

Touto cestou bych rád poděkoval vedoucímu mé bakalářské práce Ing. Miroslavovi Flídrovi, Ph.D. za jeho cenné rady, čas a vstřícnost, který mi během vypracování bakalářské práce věnoval.

Anotace

Tato bakalářská práce vznikla za účelem představení a porovnání různých algoritmů pro plánování trajektorie v prostoru. Uvažovány jsou jak algoritmy pro diskrétní prostor tak i pro prostor spojitý. Mezi tyto algoritmy patří RRT, RRT*, A*, Basic Theta*, Lazy Theta*. Všechny zmíněné algoritmy jsou naprogramované pro 2D a pro 3D prostředí. Veškeré provedené kroky a nastavení v algoritmech budou odůvodněny.

Klíčová slova

Plánování trajektorie, prohledávání grafu, rrt, A*, UAV

Abstract

This bachelor thesis was created to present and compare variol algoritmh for path planning. Considered are discrete space and continous space. These algorithms include RRT, RRT*, A*, Basic Theta*, Lazy Theta*. All these algorithms are programmed for 2D and 3D environments. Every steps and settings made in the algorithms will be justified.

Keywords

Path planning, graph searching, rrt, A*, UAV

OBSAH

Kapitola 1

Úvod

Inteligentní vozidla se stávají čím dál tím více automatizovanými. V některých případech až absolutně, to se týká především robotů v kontrolovaném prostředí. Informace a data, které tyto prostředky potřebují pro bezproblémový pohyb v prostoru, jsou získávána ze senzorů na vlastním těle, ze senzorů v prostoru či od jiných robotů v systému. Hlavními cíli této problematiky je zvýšení bezpečnosti, komfortu a v neposlední řadě i šetření spotřeby energie.

Aplikování těchto metod inteligentního cestování výrazně napomáhá řidičům, kteří se tak nemusí soustředit na spoustu úkolů, které za ně zpracovávají právě tyto inteligentní podpůrné systémy [1]. Tyto systémy již nyní pomáhají zvýšit bezpečnost například na dálnicích [2]. Dalšími sektory užití je například automatické a velmi precizní parkování, sledování slepých úhlů, nouzové brzdění a další. Tyto systémy se nazývají *pokročilé asistenční systémy pro řidiče* (ADAS). Plně automatizovaná vozidla jsou pak jen rozšířením systému ADAS. Myšlenkou plně automatizovaných systémů je absence lidského faktoru a tedy zamezení lidských ztrát v případě nehod, či mnohem rychlejší možnosti rozhodování. I přes to, že již nyní existují téměř plně automatizované systémy a jejich funkčnost je velmi přesná, bude ještě nějakou dobu trvat, než budou existovat plně automatizovaní a důvěryhodní roboti či vozidla.

Ovšem tato problematika se netýká jen řidičů na povrchu, ale také dopravy ve vzduchu či pod vodou. Různé systémy pro let jsou již nasazeny v reálném použití a další snahy pokračují. Hlavním tématem těchto dnů je provoz UAV[1]. Provoz je prováděn v 3 - dimenzionálním prostoru [3]. Možností použití těchto dronů je nezměrné množství. Hlavní výhodou oproti pozemním dopravním prostředkům je rychlost změny pozice. To je samozřejmě dovoleno letem po přímce do cílového bodu.

Po celém světě se jedná o velmi diskutované téma, avšak do této doby existuje jen hrstka států, které automatizované stroje povolují. Ve většině zemí se prozatím jedná jen

[1] UAV – z anglického Unmanned aerial vehicle, přeloženo jako bezpilotní letoun. Detailněji bude popsán v kapitole 2.1. Co je to UAV

o projekty, které mají povolené demonstrování svých strojů, ale není dovolen jejich prodej a nasazení v reálném prostředí.

Jedním ze systému pro autonomní provoz robota je plánování trajektorie v závislosti na prostředí, cenu cesty, tedy její délku či náročnost, a vyhnutí se překážkám. Tento problém popisuje právě tato bakalářská práce. V druhé kapitole této práce bude popsán samotný UAV. To bude provedeno jak popisem využití UAV tak i matematickým modelem. Ve třetí kapitole budou představeny některé z metod plánování trajektorie v prostoru. Představeny budou základní algoritmy a i jejich modifikace, které mají za úkol zlepšit parametry vygenerovaných trajektorií. Ve čtvrté kapitole budou implementovány některé z těchto algoritmů a pomocí série testů budou tyto algoritmy zhodnoceny. V poslední, tedy páté, kapitole bude shrnuta celá tato práce.

Kapitola 2

UAV

2.1. Co je to UAV

Bezpilotní letouny, jak se UAV překládá do českého jazyka, jsou typy letounů, které neovládá pilot na palubě. Mohou to být letouny ovládané pilotem na dálku či samovolně se rozhodující stroje. Dalším možným, a velmi používaným, označením je dron.

Drony jsou v dnešní době velmi využívané ve spoustě oblastí. To je zapříčiněno hlavně díky možnosti připevnění velkého množství různých senzorů od klasické kamery pro soukromé účely až po multispektrální kamery či LIDAR pro snímání prostoru. Jsou také hojně využívány v sektorech integrovaných záchranných systémů, pro monitorování prostředí a situaci, pro vyhledávání osob pomocí termokamery, nebo pro poskytnutí léků ve špatně dostupných místech či během rychlého zásahu.

Obecně UAV nemusí být jen kvadroptéra, může mít více rotorů [4]. Například hexakoptéra má rotorů šest. Bezpilotním letounem je ovšem míněno například i deltakřídlo, které je velmi podobné klasickému letounu a má jeden rotor.

Cílem této kapitoly bude představení principů pohybu právě kvadrotorové helikoptéry a uvedení rovnic pro matematický model.

2.2. Poloha v prostoru

Pro odvození matematického modelu musí být definovány 2 druhy souřadnicového systému:

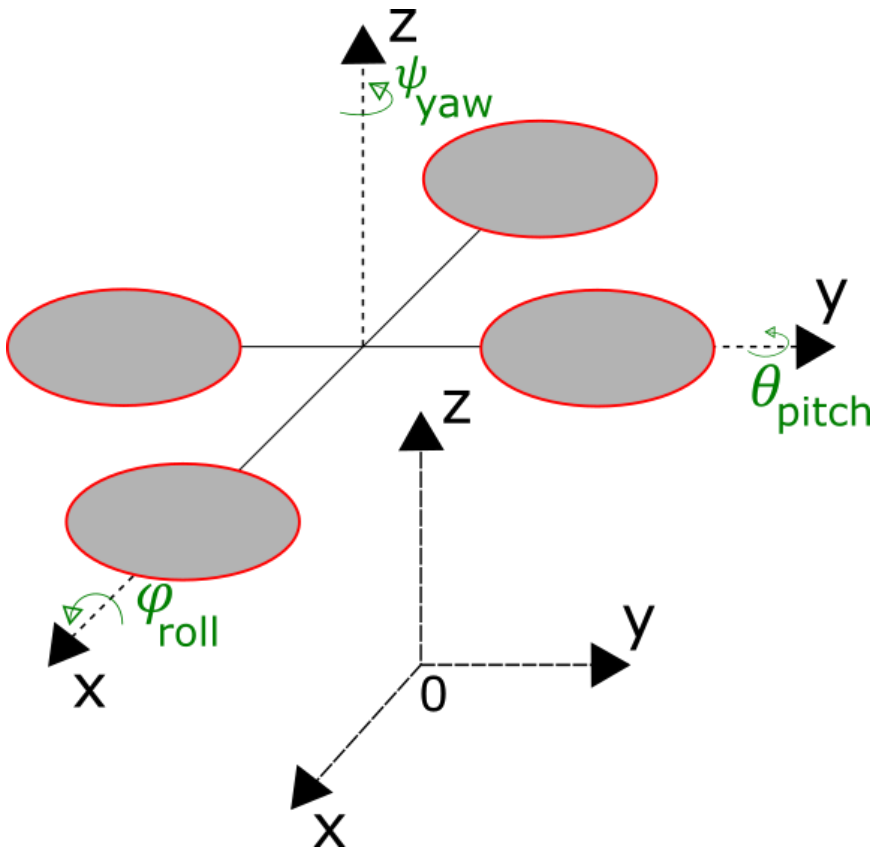
- $\mathcal{F}^E$ - vůči zemi
- $\mathcal{F}^B$ - vůči tělu helikoptéry

Pro určení pozice kvadrotorové helikoptéry v prostoru lze použít inerciální souřadnicový systém s osami x, y, z vztažný vůči zemi [5]. Kladný směr osy z je směřován od země vzhůru. Úhel natočení helikoptéry je popsán trojicí Eulerových úhlů φ, θ, ψ s respektem k souřadnicovému systému země. Rotaci kolem osy x, v terminologii popsána jako roll, značí úhel φ. Rotace kolem osy y, nazvaná pitch, je popsán a úhlem θ. A rotaci kolem osy z, neboli yaw, představuje ψ [4].

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \boldsymbol{\eta} = \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix}, \mathbf{q} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix} \tag{2.1}$$

Obdobně pro helikoptéru

$$\mathbf{v}^{\mathbf{B}} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \qquad \boldsymbol{\omega}^{\mathbf{B}} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \boldsymbol{\nu} = \begin{bmatrix} \mathbf{v}^{\mathbf{B}} \\ \boldsymbol{\omega}^{\mathbf{B}} \end{bmatrix} \tag{2.2}$$



Obr. 1 Zobrazení UAV v prostoru [4]

2.3. Kinematika kvadroptéry

Následující kapitola vychází ze zdroje [5].

Kinematika tuhého tělesa se šesti stupni volnosti (6 DOF) je předepsána

$$\dot{q} = P\,\nu, \tag{2.3}$$

kde

- $\dot{q}$ je zobecněný vektor rychlosti pro $\mathcal{F}^E$
- J je zobecněná matice rotace a transformace
- ν je zobecněný vektor rychlosti pro $\mathcal{F}^B$

Zobecněná matice rotace a transformace pro přepočet rychlostí z $\mathcal{F}^B$ do $\mathcal{F}^E$ je složena ze 4 submatic

$$P = \begin{bmatrix} R & O_{3x3} \\ O_{3x3} & T \end{bmatrix} \tag{2.4}$$

Matice rotace má tvar

$$\boldsymbol{R} = \begin{bmatrix} C_\psi C_\theta & C_\theta S_\theta S_\varphi - S_\psi C_\varphi & C_\psi S_\theta C_\theta + S_\psi S_\varphi \\ S_\psi C_\theta & S_\psi S_\theta S_\varphi + C_\psi C_\varphi & S_\psi S_\theta C_\varphi - C_\psi S_\varphi \\ -S_\theta & C_\theta S_\varphi & C_\theta C_\varphi \end{bmatrix}, \tag{2.5}$$

kde $C_x = \cos(x)$, $S_x = \sin(x)$.

Matice $\boldsymbol{R}$ je ortogonální, platí tedy $\boldsymbol{R^T = R^{-1}}$.

Matice transformace pro přepočet úhlové rychlosti z $\mathcal{F}^B$ do $\mathcal{F}^E$ má tvar

$$\boldsymbol{T} = \begin{bmatrix} 1 & S_\varphi T_\theta & C_\varphi T_\theta \\ 0 & C_\varphi & -S_\varphi \\ 0 & S_\varphi / C_\theta & C_\varphi / C_\theta \end{bmatrix} \tag{2.6}$$

kde $C_x = \cos(x)$, $S_x = \sin(x)$, $T_x = \tan(x)$.

2.4 Dynamika kvadroptéry

Následující kapitola vychází ze zdroje [5].

Dynamika je popsána diferenciálními rovnicemi, které byly odvozeny pomocí Newton-Eulerovy metody. Bere v úvahu hmotnost tělesa *m* a setrvačnost těla *J*. Předpokladem je, že má kvadroptéra symetrické tělo se čtyřmi nosiči rotorů. Hlavní osy setrvačnosti se shodují s osami $\mathcal{F}^B$, matice setrvačnosti je pak diagonální maticí

$$\mathbf{J} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{2.7}$$

Dynamika je pak popsána rovnicí

$$\begin{bmatrix} m\mathrm{I}_{3x3} & O_{3x3} \\ O_{3x3} & \mathrm{J} \end{bmatrix} \begin{bmatrix} \dot{\mathrm{v}}^B \\ \dot{\omega}^B \end{bmatrix} + \begin{bmatrix} \omega^B \mathrm{x}(m\mathrm{v}^B) \\ \omega^B \mathrm{x}\,(\mathrm{J}\omega^B) \end{bmatrix} = \begin{bmatrix} \mathrm{f}^{\mathrm{B}} \\ \tau^{\mathrm{B}} \end{bmatrix} \tag{2.8}$$

kde

- $\mathbf{I_{3x3}}$ je identická matice
- $\dot{\mathbf{v}}^{\boldsymbol{B}}$ je lineární vektor akcelerace
- $\dot{\boldsymbol{\omega}}^{\boldsymbol{B}}$ je úhlový vektor akcelerace
- $\mathbf{f^B}$ je silový vektor působící na helikoptéru
- $\boldsymbol{\tau}^{\mathbf{B}}$ je vektor točivého momentu působící na helikoptéru

2.5. Systém kvadroptéry

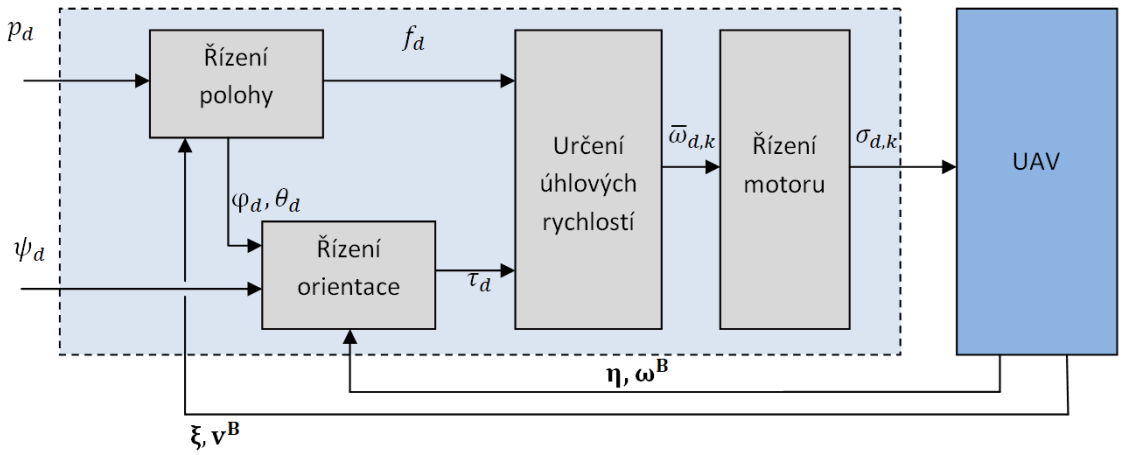
Obecně pro řízení jakéhokoliv robota je zapotřebí systému, který bude určovat potřebnou sílu či tah jednotlivých motorů v závislosti na požadované rychlosti či poloze v prostoru. Řízení tohoto systému pak může probíhat buďto dopředným řízením a nebo řízením se zpětnou vazbou.

Pro autonomní let kvadroptéry je zapotřebí vytvořit systém se zpětnou vazbou [6] pro sledování aktuální polohy, aby byla možná eliminace nárazu do překážek. Schéma takového systému je možné vidět na obrázku 2.

Vstupem do celého systému jsou informace o požadované trajektorii p_d a o požadovaném natočení kvadroptéry (yaw) ψ_d. Dále budou uvedeny informace o tom, co má jaký blok za úkol:

- Řízení polohy zpracovává informaci o požadované trajektorii. Dále pak zasílá požadavek na tah motorů f_d do bloku Určení úhlových rychlostí a požadovaný náklon (roll) φ_d a sklon (pitch) θ_d do regulátoru Řízení orientace.
- Řízení orientace má na vstupu právě požadovaný náklon spolu s požadovaným sklonem a požadovaným natočením. Ze zpracovaných signálů pak regulátor zasílá požadované momenty τ_d do bloku Určení úhlových rychlostí.
- Určení úhlových rychlostí zpracovává informaci o požadovaném tahu motorů spolu s požadovanými momenty. Z regulátoru pro určení úhlových rychlostí je pak poslán požadavek pro úhlovou rychlost vrtulí $\bar{\omega}_{d,k}$ do regulátoru pro řízení motoru.
- Řízení motoru pak vyhodnotí požadavek na úhlovou rychlost a výsledkem je povel pro motory $\sigma_{d,k}$, který je odeslán přímo na řídící jednotku UAV.

Z UAV je pak při zpětnovazebním řízení odeslána informace o poloze a o orientaci s respektem k zemi. Aktuální polohu zpracovává opět blok Řízení polohy a informaci o aktuálním natočení zpracovává Řízení orientace.



Obr. 2 Schéma zpětnovazebního systému kvadroptéry [6]

Kapitola 3

Přehled algoritmů pro nalezení trajektorie

Aby se UAV mohl autonomně dostat z bodu A do bodu B, je zapotřebí vygenerovat takzvané uzly v konfiguračním prostoru[2]. Tyto uzly je možné chápat jako body v prostoru, díky kterým je pak možné vytvořit trajektorii pro let právě z bodu A do bodu B. Generování těchto uzlů v prostoru může být čistě náhodné nebo přednastavené dle charakteristiky prostoru, který je aktuálně prohledáván.

Příklady algoritmů, které se zabývají právě problematikou generování trajektorie v prostoru pro UAV, budou popsány v této kapitole.

Představeny budou jak základní algoritmy, tak i jejich modifikace. U všech těchto algoritmů bude hodnocena doba zpracování prostoru i délka či tvar nalezené trajektorie.

V této bakalářské práci budou popsány dva typy prostoru a těmi je prostor diskrétní a prostor spojitý. Diskrétní prostor je definován tak, že má konečný počet akcí a konečný počet stavů. V případě této práce je konečným počtem stavů a akcí myšlen konečný počet uzlů v prohledávaném prostoru. Tyto uzly jsou popsány jako vrcholy mřížky[3] popisující daný prostor. U spojitého prostoru může být počet stavů a akcí roven nekonečnu. Popisy těchto prostorů budou detailněji popsány v kapitolách 3.2. a 3.3.

3.1. Konfigurační prostor

Pro úspěšný průlet pomocí UAV v prostoru, je zapotřebí určit takzvané *waypointy*[4] *(*body v prostoru), které určují trajektorii letu a také je zapotřebí znát potřebnou orientaci UAV pro průlet prostorem.

Pro popsání polohy objektu v prostoru je zapotřebí znát souřadnice tohoto objektu. Potřebný počet souřadnic dostačujících pro určení polohy určuje dimenze prostoru. V případě 2D jsou zapotřebí souřadnice x, y, v případě 3D prostoru pak již souřadnice x, y, z [7]. Dále je nutné popsat, zda daný robot dokáže měnit svou orientaci. Orientace robota je pak popsány pomocí

[2] Bude popsán v kapitole 3.1
[3] Bude vysvětleno v kapitole 3.2. Algoritmy plánování trajektorie v diskrétním prostoru
[4] Wayipoint – přeloženo z anglického jazyka jako bod trasy, v diskrétním prostoru definován jako vrchol grafu, ve spojitém prostoru definován jako uzel prostoru

úhlům které korespondují s dimenzí prostoru. V případě popisu orientace v prostoru 2D je zapotřebí jednoho úhlu φ. V případě 3D prostoru je pak počet potřebných úhlů pro popis orientace roven třem. Obecně se pak používají označení φ, θ, ψ. Tyto proměnné jsou nazývány stupni volnosti (DOF, *degrese of freedom*). Konfiguračním prostorem je pak chápán koordinační systém vyjádřený právě pomocí těchto stupňů volnosti.

Pro popis polohy a orientace pevného tělesa, kterému je umožněn pohyb, v prostoru, je tedy potřeba popsat systém se třemi stupni volnosti (x, y, φ). Pro obdobný popis v prostoru třetí dimenze je pak systém nutné popsat šesti stupni volnosti (x, y, z, φ, θ, ψ).

Takto nadefinovaného konfiguračního prostoru se používá právě v rámci plánování trajektorie. Dalšími definicemi, které konfigurační prostor obsahuje, jsou například omezení robota. Těmi mohou být myšleny váha, rozměry, maximální rychlost a další. Hlavním předpokladem pro úspěšné nalezení trajektorie v prostoru je přesný popis konfiguračního prostoru. Tím jsou míněny omezení robota a popis prostoru samotného, tedy rozměry či překážky.

3.2. Algoritmy plánování trajektorie v diskrétním prostoru

Jeden z rozdílů oproti spojitému prostoru je ten, že každý krok, každou cestu z jednoho uzlu do druhého, můžeme ohodnotit libovolnou hodnotou [8]. Jedná se tedy o takzvaný ohodnocený prostor. Tento prostor může být ohodnocen buďto kladně nebo i záporně. V praxi může být záporně ohodnocená cesta mezi uzly chápána jako nežádoucí či ztrátová. Příkladem může být například ekonomická aplikace, kde zápornou cestou mezi uzly je chápána ztráta a kladnou cestou naopak zisk. Pro potřeby této práce bude použito ohodnocení kladné. Každá cesta mezi uzly bude ohodnocena pomocí hodnoty vypočtené z Eulerovy rovnice (3.1) pro n-dimenzionální prostor.

$$d(x, y) = \sqrt[n]{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.1}$$

Nebo je možné každou cestu z uzlu do uzlu druhého, tedy každou hranu prostoru, ohodnotit libovolnou hodnotou. Tato hodnota samozřejmě musí být předem popsaná v algoritmu.

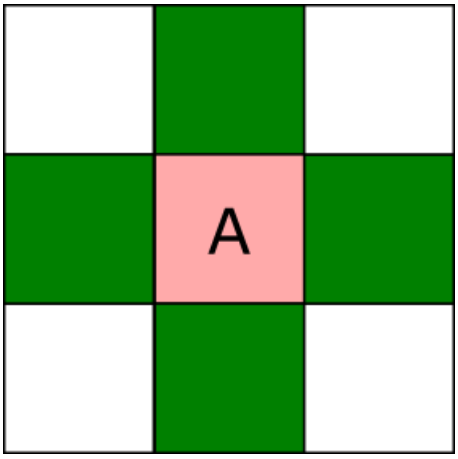
Diskrétní prostor je pak potřeba rozlišit pomocí popisu prostředí [9]. V této bakalářské práci budou použity následující dva popisy.

- **Any angle:** Prostor popsaný jako any angle obsahuje počáteční uzel, cílový uzel a veškeré body všech překážek, které se v simulovaném prostředí nachází. Jednotlivé uzly, které byly během procesu algoritmu nalezeny, jako optimální cesta mohou být spojené přímou cestou a to jen v případě, že *line-of-sight* není přerušena, respektive pokud tato cesta neprotíná jakoukoliv překážku. To znamená, že původní uzel nemusí být *parent,* respektive rodič, ale může být takzvaný *grandparent*, tedy prarodič. Takto nalezená cesta je také kratší než nalezené náhodné cesty ve spojitém prostředí. Ovšem algoritmy psané pro hledání optimálních cest ve viditelném grafu jsou velmi náročné na kapacitu paměti a jsou tím pádem i velmi pomalé. To je způsobeno nutností procházet každý sousední uzel, což se v trojdimenzionálním prostoru velmi výkonnostně prodraží.
- **Grid:** Plánování trajektorie v prostředí popsané pomocí mřížky (grid), je rychlejší než plánování trajektorie v any angle. Každopádně takto vytvořená cesta je méně optimální a méně realistická než cesta vytvořená ve viditelném grafu. To je způsobeno tím, že každý dále rozšířený uzel musí být uzel viditelný a současně musí být přímým potomkem původního uzlu.

Dále se tyto prostory mohou dělit do dalších struktur. Mezi ně patří:

- **Čtvercová mřížka**: Jedná se o strukturu prostředí v prostoru druhé dimenze.
- **Mesh**: Struktura skládající se z libovolných tvarů. Na rozdíl od čtvercového popisu, kde je jediná možná struktura a tou je čtverec, v mesh může být strukturou libovolný polygon. Tento prostor je ovšem potřeba popsat před samotným prohledáváním.
- **Kubická mřížka**: Alternativa čtvercového popisu pro prostor třetí dimenze.

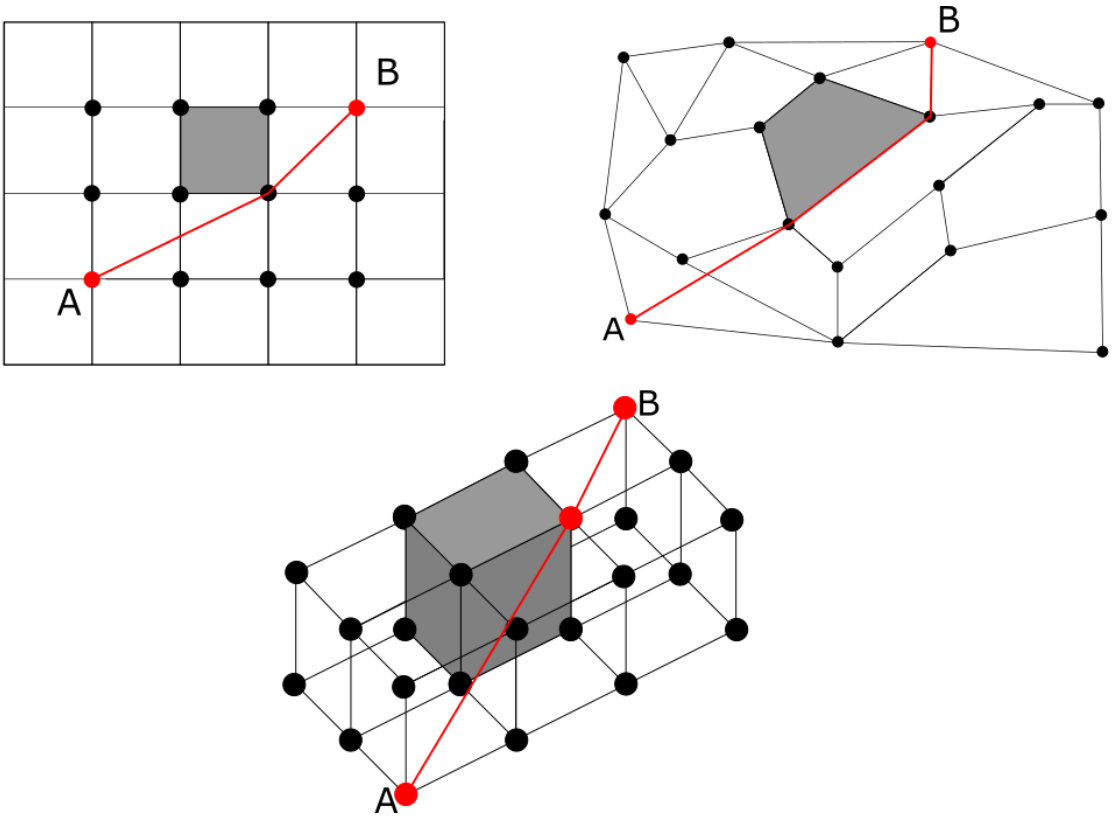
Dále je potřeba rozlišovat vyhodnocování přímých sousedů pomocí tzv. čtyř-okolí či osmi-okolí pro 2D prostor. Čtyř-okolí bodu A je na obrázku 2 znázorněno zelenou barvou a osmi-okolí barvou zelenou i bílou. Podobně je pak ve 3D prostoru rozlišováno šesti-okolí a dvacetišesti-okolí. Rozdíl mezi čtyř-okolím a osmi-okolím je znázorněn na obrázku 3.



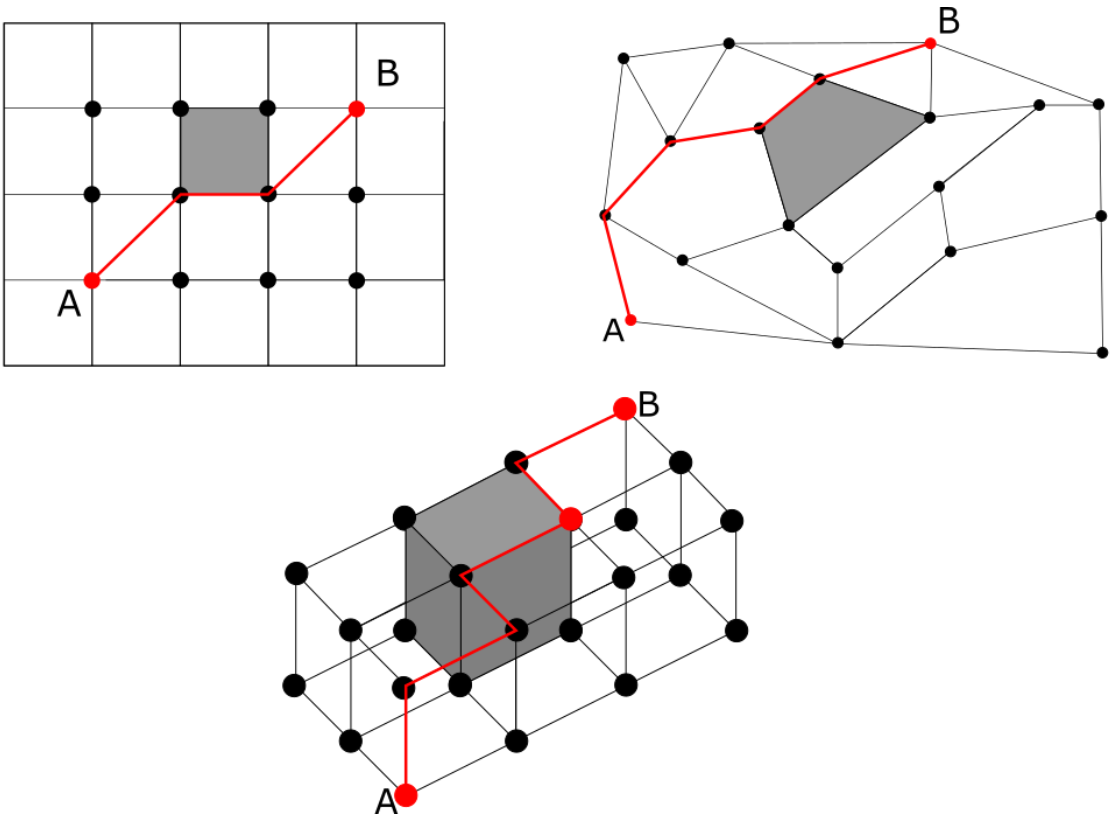
Obr. 3 Čtyř-okolí a osmi-okolí bodu A

Na následujících obrázcích 4 - 5 jsou znázorněny právě popsané struktury prostoru pomocí čtyř-okolí ve 2D prostoru a šesti-okolí ve 3D. Obrázek číslo 3 popisuje Any angle prostor, kde první prostor popisuje čtvercová mřížka, druhý pak mesh a poslední představuje kubickou mřížku. Analogicky je představen i prostor Grid na obrázku 4. Při porovnání těchto dvou obrázku je možné vidět, že délka trasy mezi body A a B jsou v prostoru Any angle značně kratší.

Dalším rozdílem mezi prostorem 2D a 3D je počet zkoumaných vrcholů v případě skutečnosti, že algoritmus prozkoumá kompletní konfigurační prostor. V případě 2D konfiguračního prostoru o rozměrech PxP je zapotřebí prohledat P^2 vrcholů. V případě 3D je pak zapotřebí P^3 vrcholů, čímž razantně roste velikost ukládané paměti i časová náročnost běhu algoritmu.



Obr. 4 Struktura prostoru – Any angle [9]



Obr. 5 Struktura prostoru - Grid

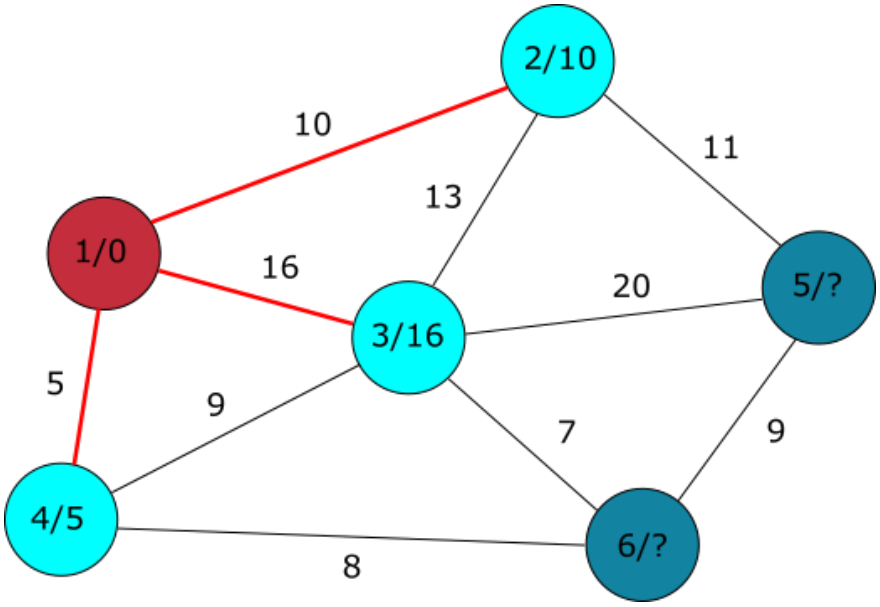
3.2.1. Dijkstrův algoritmus

Algoritmus byl poprvé popsán nizozemským informatikem Edsgerem Dijkstra v roce 1959 [10].

Dijkstrův algoritmus slouží k nalezení nejkratší cesty v grafu. Jedná se o konečný algoritmus, tedy algoritmus, který pro konečný počet vstupů v každém případě skončí. To je způsobené tím,že v každé iteraci se do množiny navštívených uzlů přidá právě jeden uzel podle předem sepsaných pravidel. Dle tohoto popisu je jisté, že maximální počet cyklů, respektive iterací, je stejný jako počet jednotlivých vrcholů grafu.

Tento algoritmus pracuje pouze s kladně ohodnocenými grafy. Na samém začátku algoritmu jsou známy dvě množiny a těmi jsou množina vrcholů a množina hran. Dále je potřeba vytvořit množiny s názvem OPEN a CLOSED. Množina OPEN obsahuje veškeré dosud nenavštívené uzly. Na začátku algoritmu jsou to tedy všechny uzly kromě startovní pozice. Množina CLOSED obsahuje každý uzel, který již byl navštívený. Na začátku algoritmu tedy pouze startovní uzel. V každé iteraci je spočítán nejbližší další možný uzel, který je spojený s předchozím uzlem hranou, jedná se tedy o viditelného souseda, a ten je přesunut do množiny CLOSED. V případě, že je cílem algoritmu i zajištění rekonstrukce nejkratší cesty, je k tomuto uzlu uložen i uzel předcházející, nazývající se předek. Takto algoritmus zpracovává příkazy do té doby, dokud nenajde cestu k cílovému uzlu nebo do doby, než projde celou množinu uzlů, tedy kompletní konfigurační prostor.

Obrázek číslo 6 nastiňuje právě procházení ohodnoceným grafem. Červený uzel je míněn jako počáteční a červenými linkami jsou myšleny trasy právě z tohoto uzlu ke svým sousedům. Číslo u každé hrany pak popisuje hodnotu (například délku) trasy. První číslo v každém uzlu je identifikátor a druhé číslo je cena dosažení tohoto uzlu z počátku. Světle modrou barvou jsou pak označeny uzly, které jsou přímými sousedy počátečního uzlu a jsou tedy vyhodnocovány. V tomto případě by další zkoumaný uzel byl uzel 4, jehož dráha od počátku je ohodnocena hodnotou 5. Tento uzel by pak byl přesunut do množiny CLOSED a ihned na to rozšířen o veškeré své sousedy, které jsou v množině uzlů OPEN. V tomto případě tedy uzly číslo 3 a 6. Dále by následoval opět proces vyhodnocování dokud by nebylo dosaženo vyprázdnění množiny OPEN.



Obr. 6 Dijkstrův algoritmus – prohledávání prostoru

Následující pseudokód popisuje Dijkstrův algoritmus [11].

Algoritmus 1) Dijkstrův algoritmus

```
Main()
U, H                                    //inicializace(Uzly, hrany)
s, c                                    //inicializace(start, cíl)
foreach u in U
        d[u] := infinity                // vzdálenost všech uzlů je neznámá
        p[u] :=undefinied               // nedefinovaní předchůdci
end
d[s] := 0                               // vzdálenost startu od startu
OPEN := U                               // množina nenavštívených uzlů
CLOSED := []                            // množina navštívených uzlů
OPEN.remove(s)
CLOSED.insert(s)
while OPEN is not empty:                // vyhodnocování a procházení prostoru
        min := get_min(OPEN)            // vybrání „nejlepšího uzlu“
        foreach soused u of min:
                alt = d[min] + l(min, u)    // nalezení nejbližšího souseda
                if alt < d[u]
                        d[u] := alt
                        p[u] := min
                        OPEN.remove(min)
                        CLOSED.insert(min)
                end
        end
end
```

3.2.2. Bellman-Fordův algoritmus

Algoritmus se velmi podobá předešlému Dijkstrovovu algoritmu. Tento algoritmus opět prochází ohodnoceným grafem. Jeho rychlost zjištění nejkratší vzdálenosti je oproti Dijkstrovovu algoritmu pomalejší, respektive doba zpracování celého prostoru a nalezení trajektorie je delší. To je způsobeno principem tohoto algoritmu a tím je možnost pracování i s negativně ohodnoceným grafem.

Tento algoritmus byl poprvé popsán Alfonsem Shimbelem v roce 1955, ale je pojmenován po autorech publikace z roku 1956 a 1958 pány Lesterem Fordem a Richardem Bellmanem [12].

Dijkstrovovův a Bellman-Fordův algoritmus v této práci nebudou implementovány, jelikož výsledky z těchto dvou algoritmů nejsou optimální pro řízení UAV. Následující pseudokód popisuje Bellman-Fordův algoritmus [13].

Algoritmus 2) Bellman-Fordův algoritmus

```
Main()
U, H                                    // inicializace(Uzly, hrany)
s, c                                    // inicializace(start, cíl)
foreach u in U
    d[u] := infinity                    // vzdálenost všech uzlů je neznámá
    p[u] :=undefinied                   // nedefinovaní předchůdci
end
d[s] := 0
OPEN := U                               // množina nenavštívených uzlů
CLOSED := []                            // množina navštívených uzlů
while OPEN is not empty:                // vyhodnocování a procházení prostoru
    min := get_min(OPEN)                // vybrání „nejlepšího uzlu“
    foreach soused u with ohodnocení w in H do:
        alt = d[min] + w
        if alt < d[u]:                  // nalezení nejbližšího souseda
            d[u] := alt
            p[u] := min
            OPEN.remove(min)
            CLOSED.insert(min)
        end
    end
end
foreach soused u with value w in H:
    if d[u] + w < d[V]:
        error “Graf obsahuje negativně ohodnocený cyklus“
    end
end
```

3.2.3. A*

Tento algoritmus používá pro hledání optimální cesty heurestiku. Prostor ve kterém tento algoritmus pracuje, je diskrétní a takzvaně kladně ohodnocený. To znamená, že každý vrchol má určitou hodnotu a to 0 a 1. 0 znamená překážka a 1 znamená volný vrchol.

Jedná se o heuristický algoritmus, který používá stejné principy jako Dijkstrův algoritmus. A* [14] ale navíc počítá heuristický prvek. Tedy hodnotu vzdálenosti od aktuálního vrcholu po vrchol výsledný. Dijkstrův algoritmus počítá jen hodnotu vzdálenosti od startovního vrcholu k aktuálnímu vrcholu.

A* patří do skupiny algoritmů nazvané *Greedy algorithm* [15], v překladu chamtivý algoritmus. Jedná se o způsob řešení optimalizačních úloh především v matematice a informatice. Dle definice programu či aplikace, která je předem známá, hladový algoritmus v každé své iteraci hledá minimum. V případě hledání optimální cesty je definicí rozuměno nejkratší vzdálenost. A* má za úkol najít suboptimální cestu ze startovního uzlu, tedy počáteční bod, do uzlu požadovaného, tedy do cíle cesty. Optimální cestou je v tomto případě míněno nejkratší a nejlevnější cesta, respektive vzdálenost, kterou dokáže algoritmus najít.

K nalezení suboptimální cesty je použita funkce značená $f(x)$ [17], jejímž úkolem je ohodnotit každý uzel, kterým A* projde. Toto ohodnocení pak indikuje, v jakém pořadí a jestli vůbec mají být tyto uzly do optimální cesty zahrnuty.
Funkce $f(x)$ je složena ze dvou dalších funkcí:

$$f(x) = g(x) + h(x), \tag{3.2}$$

kde

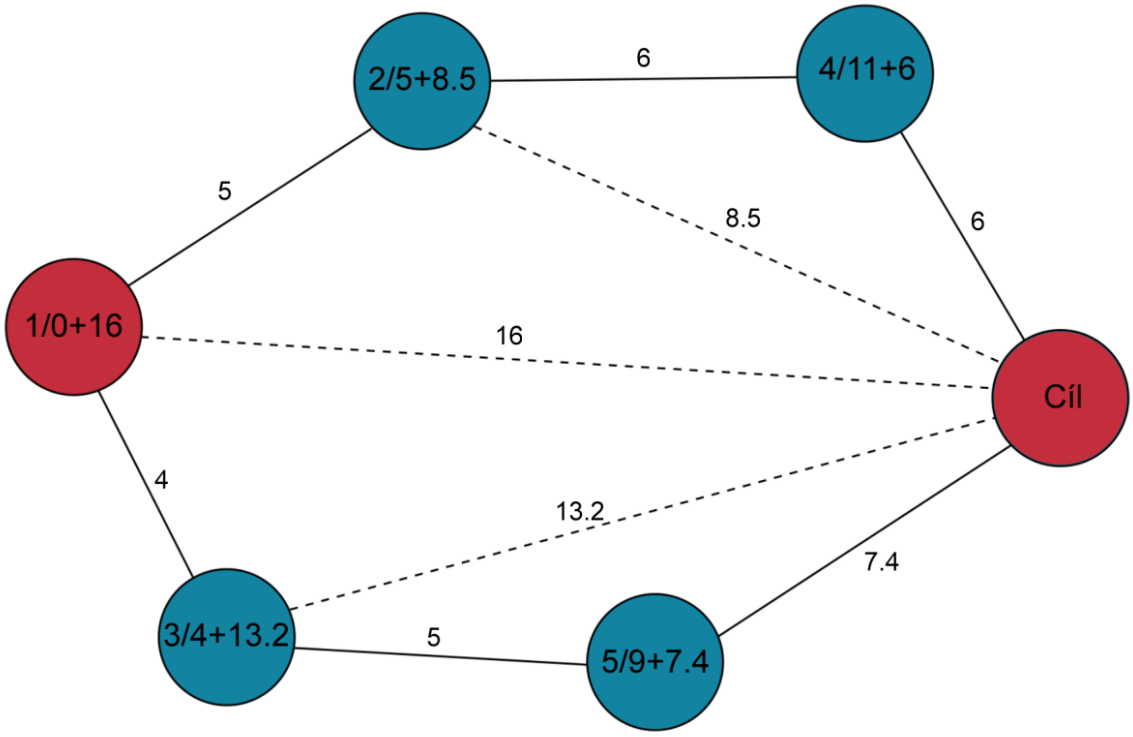
- $g(x)$ - funkce pro ohodnocení ceny pro dosažení aktuálního uzlu z uzlu, v tomto případě tedy vzdálenost těchto dvou uzlů
- $h(x)$ – heuristická funkce

Tato heuristická funkce má za úkol ohodnotit cestu z aktuálního uzlu do uzlu cílového. Pro tenhle účel výpočtu je použita takzvaná vzdušná čára směřující k cíli nehledě na to, jestli prochází libovolnou překážkou či nikoliv. Vzdušná čára je logicky nejkratší možnou cestou do cíle.

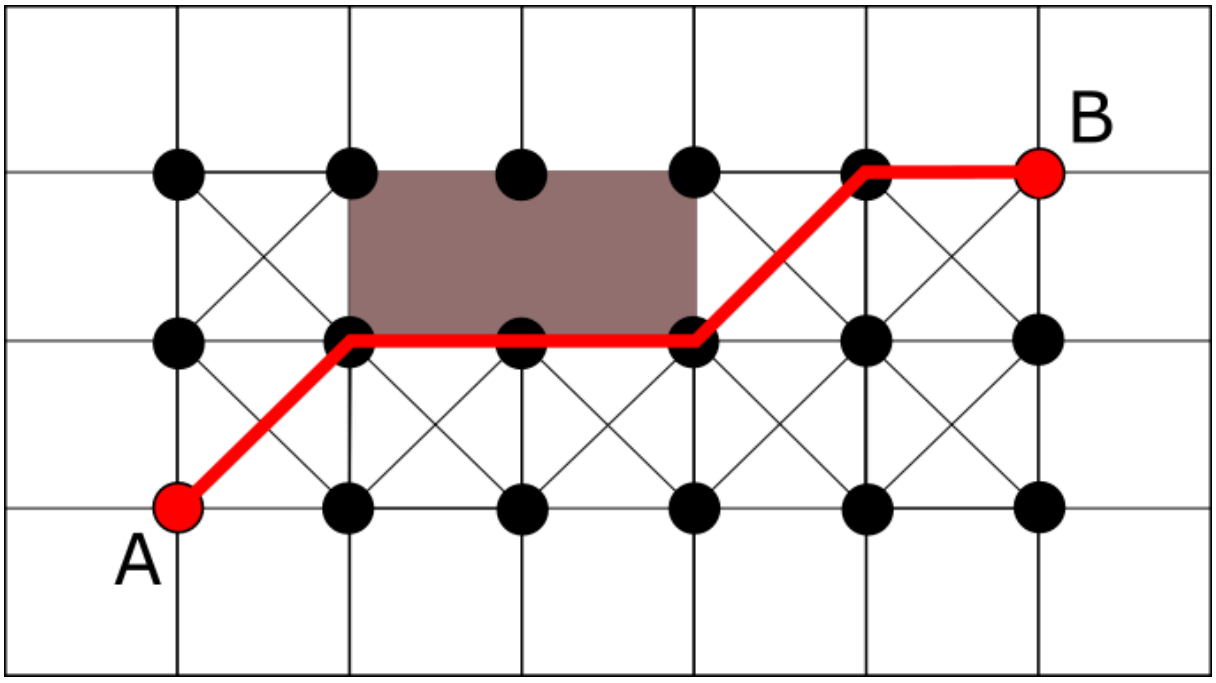
Na samém začátku algoritmu se vytvoří pole dosud nenavštívených uzlů. Tyto uzly jsou uloženy do fronty OPEN, jsou tedy zatím otevřené. Druhým polem je pole již navštívených uzlů. Toto pole se nazývá CLOSED. To, do jakého uzlu algoritmus vstoupí, tedy jaký bod vybere jako následný, ohodnocuje právě funkce $f(x)$. Čím je hodnota této funkce menší, tím příslušný uzel této hodnotě má vyšší prioritu a je tedy odebrán z pole OPEN a je uložen do pole CLOSED. Následně jsou k tomuto přeloženému uzlu spočteny hodnoty funkcí $f(x)$ a $h(x)$. Funkci $g(x)$ počítat nemusíme, tu již známe z předešlého kroku. Takto algoritmus pokračuje až do doby, kdy dosáhne cílového uzlu, nebo naopak cíle nedosáhne, ale již nemá kam pokračovat. Takto zpracovaný algoritmus tedy dokáže najít nejkratší cestu od počátečního uzlu do toho cílového, ovšem zná jen jeho cenu, respektive délku projeté dráhy. Pokud by bylo zapotřebí si cestu i pamatovat, je zapotřebí ukládat si ke každému uzlu i jeho předchůdce, ze kterého byl právě tento uzel rozšířen.

Obrázek 7 demostruje procházení ohodnoceným grafem. Hodnocení je analogické vůči Djikstrovova algoritmu s tím rozdílem, že je zde zakomponována heuristická funkce. Ta je představena jako letecká (přímá) čára ze všech uzlů do cílového uzlu. První číslo v kruhu označuje identifikátor. Druhá hodnota (za lomítkem) je součet funkcí $g(x)$ a $h(x)$, kde $h(x)$ je právě přímá linka do cílového uzlu a $g(x)$ je délka dráhy potřebná k dosažení aktuálního uzlu.

Na obrázku 8 je pak znázorněno nalezení nejkratší možné trajektorie z bodu A do bodu B pomocí A*. Konfigurační prostor je složen



Obr. 7 A* - heuristická funkce



Obr. 8 A* - nalezení nejkratší cesty

Následující pseudokód popisuje algoritmus A* [18].

Algoritmus 3) A*

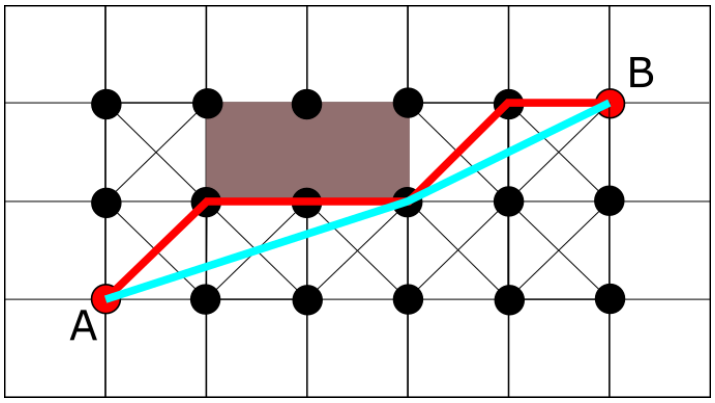
```
Main()
U, H                                    // inicializace(Uzly, hrany)
s, c                                    // inicializace(start, cíl)
OPEN := CLOSED := ∅
g[s] := 0
parent(s) := s                          // startovní uzel je sám sobě předchůdce
while OPEN is not empty :               //dokud neprojdemeveškeré uzly
    u :=open.Pop()
    if u := c:                          //  dosažení cílového uzlu
        return  "cesta nalezena"
    end
    CLOSED.insert(u)
    foreach sousedn ofu do:             //zjištění informací pro všechny sousedy
        if n is not in CLOSED:
            if n is not in OPEN:
                g[n] := infinite
                parent(n) := NULL
            end
            UpdateVertexes(u, n)
        end
    end

    return "cesta nenalezena"
end
UpdateVertexes(u, n)                    // aktualizování funkcí vzdálenosti
    g_old = g(n)
    ComputeCosts(u, n)
    if g(n) <g_old :
        ifn is inOPEN
            OPEN.remove(n)
        OPEN.insert(n, g(n) + h(n))
        end
    end
end
ComputeCosts(u, n)                      // zjištění vzdálenosti mezi uzly
    if g(u) + c(u, n) < g(n):
        parent(n) := u
        g(n) := g(u) + c(u, n)
    end
end
```

3.2.4. Basic Theta*

Tento algoritmus vychází ze stejných principů jako předchozí algoritmus A*. Dá se tedy říct, že se jedná o další vylepšení A algoritmů. Basic Theta* byla vytvořena především proto, že původní algoritmus A* sice dokáže najít suboptimální cestu, ale vytvořená cesta skrze zkoumaný prostor je velmi ostrá a složitá pro takzvaný *post smoothing*[5], tedy vyhlazení cesty. A* totiž používá grid, tedy mřížku, a dostane se tedy jen do uzlů, které aktuální uzel vidí. Vylepšení Basic Theta* [18] je právě v možnosti spojit libovolné dva uzly nehledě na jejich vzdálenost nebo jestli jsou vůbec sousedy. Tento algoritmus totiž probíhá v grafu any angle, následným uzlem tedy nemusí být jen uzel, který je na dohled od původního uzlu, ale může uzly libovolně přeskakovat.

Tato funkce se nazývá `lineofsight` a byla již nastíněna v kapitole 3.1. Funkce `lineofsight` již počítá s tím, že algoritmus si pamatuje veškeré rozšířené uzly. To jsou uzly, ze kterých již algoritmus pokračoval a rozrůstal se tím tedy již porovnávaný prostor. K této informaci si také musí pamatovat předka, ze kterého byl uzel rozvinut. Předkem aktuálně porovnávaného uzlu, tedy počítání funkce $g(x)$ a $h(x)$, může být libovolný uzel, ze kterého se lze do aktuálního uzlu dostat. Toto je zásadní rozdíl oproti A*, kde následným uzlem může být jen takový, který je viditelným sousedem.

Toto je znázorněno na obrázku 9, kde je nalezená trajektorie z bodu A do bodu B pomocí metody A* znázorněna červeně a pomocí metody Basic Theta* modře.



Obr. 9 Basic Theta*(modře) v porovnání s A*(červeně)

[5] Metoda používaná pro zoptimalizování trajektorie

Následující pseudokód popisuje algoritmus BasicTheta* [18].

Algoritmus 4) BasicTheta*

```
Main()
U, H                                    // inicializace(Uzly, hrany)
s, c                                    // inicializace(start, cíl)
OPEN := CLOSED := ∅
g[s] := 0
parent(s) := s                          // startovní uzel je sám sobě předchůdce
while OPEN is not empty :               //dokud neprojdemeveškeré uzly
    u :=open.Pop()
    if u := c                           //  dosažení cílového uzlu
        return  "cesta nalezena"
    end
    CLOSED.insert(u)
    foreachsousedn ofu do:              //zjištění informací pro všechny sousedy
        if n is not in CLOSED:
            if n is not in OPEN:
                g[n] := infinite
                parent(n) := NULL
            end
            UpdateVertexes(u, n)
        end
    end
    return "cesta nenalezena"
end
UpdateVertexes(u, n)                    // aktualizování funkcí vzdálenosti
    g_old = g(n)
    ComputeCosts(u, n)
    if g(n) <g_old :
        if n is inOPEN
            OPEN.remove(n)
        OPEN.insert(n, g(n) + h(n))
end
ComputeCosts(u, n)                      // zjištění vzdálenosti mezi uzly
    if lineofsight(parent(u), n):       //zjištění viditelnosti předchůdce
        if g(parent(u)) + c(parent(u), n) < g(n):
            parent(n) := parent(u)
            g(n) := g(parent(u)) + c(parent(u), n)
        end
    else
        if g(u) + c(u, n) < g(n)
            parent(n) := u
            g(n) := g(u) + c(u, n)
        end
    end
end
```

Line-of-sight

Tato metoda kontroluje viditelnost uzlů a jejich předků. Vstupem do této funkce mohou být jen celočíselná data. Důvod k tomuto pravidlu je takový, že používání této funkce je stejné jako zjišťování jaké body mají být vykresleny a zaznamenány na rastrovém zobrazovacím zařízení během vykreslování přímé čáry mezi dvěma body. Body, které jsou pak vykreslovány jako cesta, jsou zároveň i body, které funkce `lineofsight` [18] porovnávala.

Funkce `lineofsight` zjišťuje ve spolupráci s funkcí grid, zdali uzly, které jsou právě v cestě z prvního uzlu do uzlu druhého, patří do blokovaných bodů, tedy do uzlů, které jsou označeny jako překážka. `Lineofsight` počítá, jaké všechny uzly bude potřeba právě takto porovnat a funkce grid pak zjišťuje právě onu blokaci, jedná se tedy o vyčítání z námi známého prostoru.

Funkce grid vrací jen hodnoty ano či ne, respektive true nebo false, kde hodnota true bude vrácena v případě, že daný bod je blokován, tedy je označený jako překážka.

Ray casting

Aby bylo možné zjistit všechny uzly, které má `lineofsight` projít a vyhodnotit pomocí funkce grid, je nutné použít metodu pro změnu souřadnic v závislosti na trase z prvního uzlu do uzlu druhého. Jednou z takovýchto metod je například takzvaný *Ray casting* [19], který je použit i v této bakalářské práci. [2]

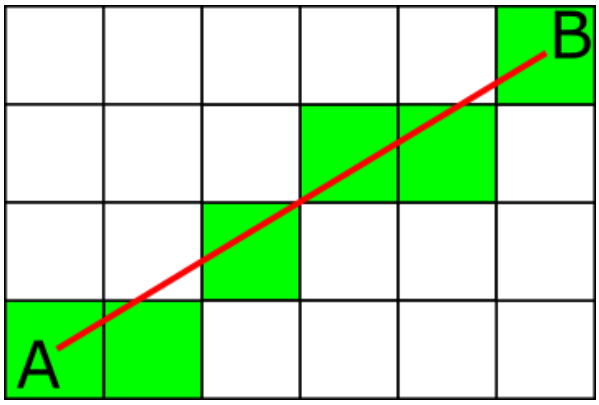
Tato metoda je i hojně využívána v počítačové grafice a počítačové geometrii. Má mnoho dalších použití a různých stylů, pro potřeby této práce bude využita pro zjištění přerušení paprsku.

Konkrétním algoritmem, který patří do skupiny algoritmů fungující jako ray casting, je Bresenhamův algoritmus. Další možností je například použití algoritmus Wu. [3]

Bresenhamův algoritmus

Bresenhamův algoritmus [20] je přímku vytvářející algoritmus v rastrovém, respektive diskrétním prostředí, a jehož úkolem je aproximovat výběr bodů v rastru tak, aby co nejlépe tato aproximace popisovala okolí přímky, která spojuje dva body.

Průchod hodnoceným prostorem je představen na následujícím obrázku 10.

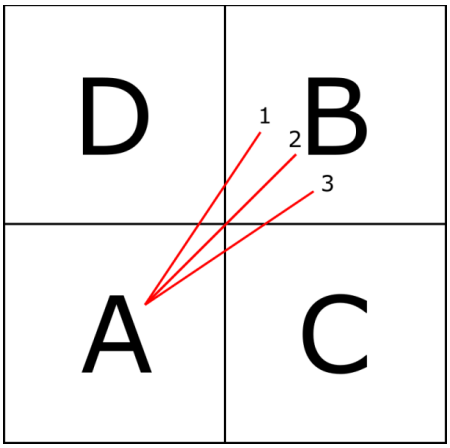


Obr. 10 Bresenhamův algoritmus

Bresenham-based supercover line algoritmus

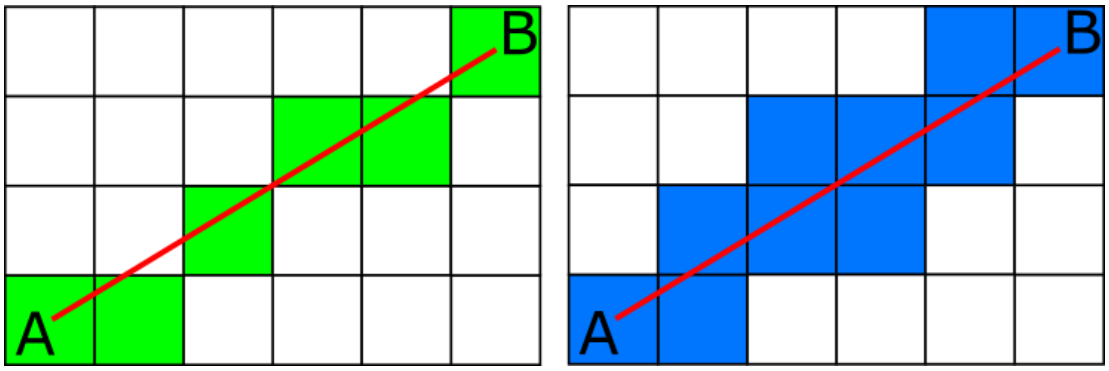
Jak je možné vidět na obrázku výše, Bresenhamův algoritmus ne vždy projde skrze všechny potřebné uzly. Pro potřeby přesnějšího určení všech uzlů, které jsou potřeba k ohodnocení pomocí funkce `grid`, byl použit upravený princip Bresenhamova algoritmu [21].

Bresenhamův algoritmus bere v úvahu vždy jen uzly, které na sebe přímo navazují, nebo v případě změny souřadnice na ose i bod úhlopříčně připojený. Pro potřeby této práce je ovšem zapotřebí vědět i to, zdali je bod mezi dvěma body, při změně souřadnice taktéž volný či označený jako překážka.



Obr. 11 Bresenham - problematika přechodu mezi body

Tato úprava vychází z obrázku 11. Pokud by byla uvažována libovolná cesta z bodu A do bodu B, Bresenhamův algoritmus by počítal jen právě s těmito dvěma body. Pro účely této práce je však zapotřebí počítat i s body D při cestě číslo 1, s bodem C při cestě číslo 2 a s oběma body při využití cesty číslo 3. Toto je základní rozdíl oproti klasickému a původnímu algoritmu. Tento rozdíl je znázorněn na následujícím obrázku 12.



Obr. 12 Rozdíl mezi Bresenhamovým algoritmem a supercovered algoritmem

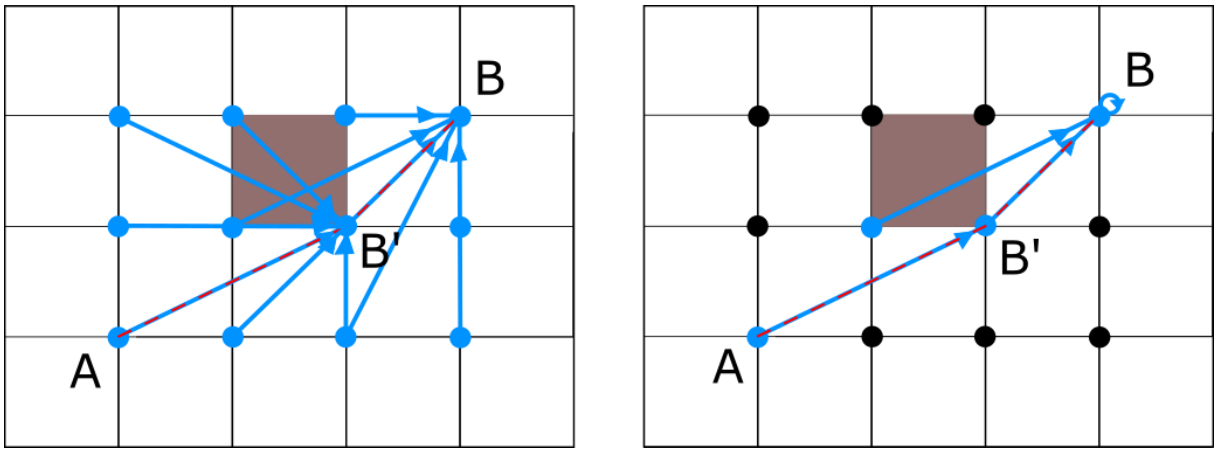
3.2.5. Lazy Theta*

Jak již bylo zmíněno v předchozí kapitole o algoritmu Theta*, rozdílem Thety* a A*byla možnost provést spojení dvou uzlů, které nejsou přímými sousedy. Toto spojení bylo vyhodnocováno funkcí `lineofsight`, kde výstupem této funkce byla boolean informace. Tato informace udávala, zdali je propojení, respektive přehodnocení předchůdců, možné.

Ačkoliv Theta* umožňuje nalezení optimálnější cesty než A* metoda, tím pádem i realističtější trajektorii, je výpočetní náročnost tohoto algoritmu vysoká. Hlavním problémem je, že Theta* používá funkci `lineofsight` i pro takové uzly, které již vůbec nemusí být dále expandovány. Tím právě výpočetní a paměťová náročnost roste.

Z tohoto důvodu byl představen nový algoritmus z rodiny A algoritmů a tím je Lazy Theta* [9]. Lazy Theta* je rychlejší než právě Basic Theta*, neboť neprovádí funkci `lineofsight` pro každý viditelný vrchol, ale vybere pouze nejbližší. Tento vrchol není ovšem vybrán jako u Theta*. V případě, kdy právě vyhodnocovaný uzel je uzel B‘, všechny vzdálenosti jeho sousedů jsou počítány k předchůdci uzlu B‘, tedy k A. Právě z těchto vzdáleností je vybrána

ta minimální a až pak je provedena kontrola `lineofsight`. Pokud kontrola proběhne negativně, jsou veškeré vzdálenosti přepočteny pro vzdálenosti mezi uzlem B‘ a jeho viditelnými sousedy. Analogicky je proces proveden pro bod B. Toto je možné vidět na obrázku číslo 13. Zatímco Theta* vyhodnocuje `lineofsight` celkem 15x, Lazy Theta* tuto funkci spustí pouze 4x. Tím je celý algoritmus mnohem rychlejší.



Obr. 13 Basic Theta*(vpravo) a Lazy Theta*(vlevo)

Následující pseudokód popisuje algoritmus Lazy Theta* [9].

Algoritmus 5) Lazy Theta*

```
Main()
U, H                                    // inicializace(Uzly, hrany)
s, c                                    // inicializace(start, cíl)
OPEN := CLOSED := ∅
g[s] := 0
parent(s) := s                          // startovní uzel je sám sobě předchůdce
while OPEN is not empty :               //dokud neprojdemeveškeré uzly
    u :=open.Pop()
    SetVertexes(s)
    if u := c                           //  dosažení cílového uzlu
        return  "cesta nalezena"
    end
    CLOSED.insert(u)
    foreach sousedn ofu do:             //zjištění informací pro všechny sousedy
        if n is not in CLOSED:
            if n is not in OPEN:
                g[n] := infinite
                parent(n) := NULL
            end
            UpdateVertexes(u, n)
        end
    end
    return "cesta nenalezena"
end
UpdateVertexes(u, n)                    // aktualizování funkcí vzdálenosti
    g_old = g(n)
    ComputeCosts(u, n)
    if g(n) <g_old
        if n is inOPEN
            OPEN.remove(n)
        OPEN.insert(n, g(n) + h(n))
end
ComputeCosts(u, n)                      // zjištění vzdálenosti mezi uzly
    if g(parent(u)) + c(parent(u), n) < g(n):
        parent(n) := parent(u)
        g(n) := g(parent(u)) + c(parent(u), n)
    end
end
SetVertex(u)                            //provedení změn pro uzel
    if NOT lineofsight(parent(u), u):
        parent(u) :=argmin_{n ∈nghbr_vis(u)∩closed}(g(n) + c(n, u))
        g(u) := min_{n ∈nghbr_vis(u)∩closed}(g(n) + c(n, u))
    end
end
```

3.3. Algoritmy plánování trajektorie ve spojitém prostoru

Pro popis prostředí a cenu cesty z uzlu A do uzlu B bude používána Eulerova rovnice vzdálenosti 3.1. Jelikož cenu cesty mezi uzly ve spojitém prostoru nelze přímo definovat určitou hodnotou jako v prostoru diskrétním, rovnice 3.1 poskytne dostačující informaci v podobě vzdálenosti těchto dvou uzlů.

3.3.1. Probabilistic roadmap

Metoda prohledávání prostoru zvaná Probabilistic roadmap [22], zkráceně PRM, je první metodou pro spojitý prostor s náhodným generováním uzlů, který bude v této práci představena.

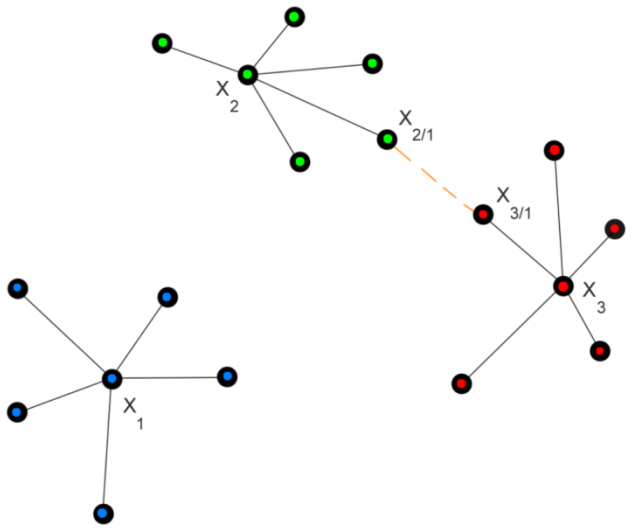
Základní ideou tohoto algoritmu je náhodné generování uzlů v konfiguračním prostoru a následné další pracování s těmito uzly a prostorem. Pokud je každý nově vygenerovaný uzel v neblokovaném prostoru[6], není tedy lokalizován uvnitř překážky, pak je rozšířen o další spojnice k n-nejbližším sousedům. Počet těchto sousedů *n*, ke kterým se z nového uzlu strom rozšiřuje, je předem známý. Propojení probíhá dvěma způsoby. V případě, že vzdálenost nejbližších uzlů je menší nebo rovna maximální možné vzdálenosti, která je zadána programátorem nebo specialistou na prostředí, ve kterém je trasa plánována, pak jsou oba uzly spojeny. V případě, že tato maximální vzdálenost není splněna, je vytvořen nový uzel v právě maximální možné vzdálenosti od rozrůstajícího se uzlu. Jakmile algoritmus propojí n-nejbližších sousedů, vygeneruje pak nový náhodný uzel a zopakuje předchozí proces, tedy propojování. Toto generování uzlů v konfiguračním prostoru je v algoritmu PRM [24] nazýváno jako první fáze. Tato fáze pokračuje do splnění podmínky o konfiguračním prostoru.

Těmi může být:

1. dosažení maximálního počtu vygenerovaných uzlů,
2. dosažení maximálního počtu propojených uzlů,
3. dosažení maximální doby běhu programu.

[6] Neblokovaným nebo volným prostorem je myšlen prostor, který není označený jako překážka.

Na následujícím obrázku 14 je znázorněna tvorba jednotlivých shluků bodů právě po náhodném vygenerování. Každý shluk je odlišen jinou barvou. Znázorněno je i spojení právě shluku X_2 a X_3.



Obr. 14 PRM – tvorba mapy uzlů

Ve druhé fázi se hledá nejkratší vzdálenost uzlů, které jsou předmětem zájmu. Cesta je hledána pomocí Dijkstrovova algoritmu, jenž byl popsán ve stejnojmenné kapitole.

Následující pseudokód popisuje algoritmus PRM [22]. Pod tímto kódem jsou dále uvedeny popisy k hlavním metodám z PRM.

Algoritmus 6) PRM

τ:=**initializeTree**() // *vytvoření prostoru*

τ:=**insertNode**($node_{init}$, τ) // *zapsání startovního uzlu*
for i := 1 **to** N // *N = maximální počet uzlů*
    $z_{rand} :=$ **generateRandomNode**(i) // vytvoření náhodného uzlu
    if obstacleFree(z_{rand}): // *zjištění překážky*
        for j:=1 **to** K: // *K nejbližších sousedů*
            $z_{nearest} :=$ **nearest**(z_{rand}, τ)
            if obstacleFree($z_{nearest}$):
                connect($z_{nearest}$, z_{rand}) // *spojení uzlů*
            end
        end
    end
end

Vysvětlení hlavních funkcí algoritmu

- **GenerateRandomNode**: Tato funkce generuje náhodné pozice uvnitř prohledávaného prostoru.
- **Nearest**: Funkce, která vrací nejbližší nod z prostoru T k nodu, který byl dříve již vygenerován, v závislosti na předem nastavené konstantě maximální vzdálenosti.
- **Steer**: Tato funkce vyhodnocuje, jakým způsobem je zachována maximální vzdálenost mezi zkoumaným nodem a nodem, který vrátila předešlá funkce Nearest. V případě, kdy je maximální aktuální vzdálenost menší nebo rovná konstantě, kterou má algoritmus vnořenou již od začátku, pak se nic neděje. V případě, kdy je vzdálenost větší než maximální povolená, hledá se nový bod, který leží v maximální předepsané vzdálenosti na trajektorii ze zkoumaného nodu do nalezeného nejbližšího nodu.
- **ObstancleFree**: Funkce zkoumající, zdali je nalezený nejbližší uzel, nebo jeho náhradník vygenerovaný z funkce Steer, v prostoru mimo veškeré překážky, které by byly v možné trajektorii.

3.3.2. Rapidly – exploring random tree

RRT [23] neboli náhodně rostoucí strom, je další algoritmus, který pracuje ve spojitém prostředí. Jedná se o autonomní algoritmus používaný pro plánování tras využívaných pak mobilními roboty v n-dimenzionálním prostoru. RRT bylo vytvořeno Stevenem M. LaVallem v roce 1998.

Algoritmus může fungovat jako dynamický program[7], kdy se bude vyhodnocovat a rozšiřovat dle aktuálních, nově získaných informací. Nebo funguje ve stacionárním prostředí, tedy neměnným a předem známým prostředím. U takovéto problematiky je velmi důležité předem a detailně popsat celý prostor, ve kterém se bude autonomní robot pohybovat.

Tento algoritmus patří do skupiny algoritmů nazvaných *Sample based planning* [24]. Tyto algoritmy jsou známé tím, že poskytují rychlé výsledky pro komplexní a několika dimenzionální prostory. Toto prohledávání je prováděno vytvářením náhodně generovaného stromu.

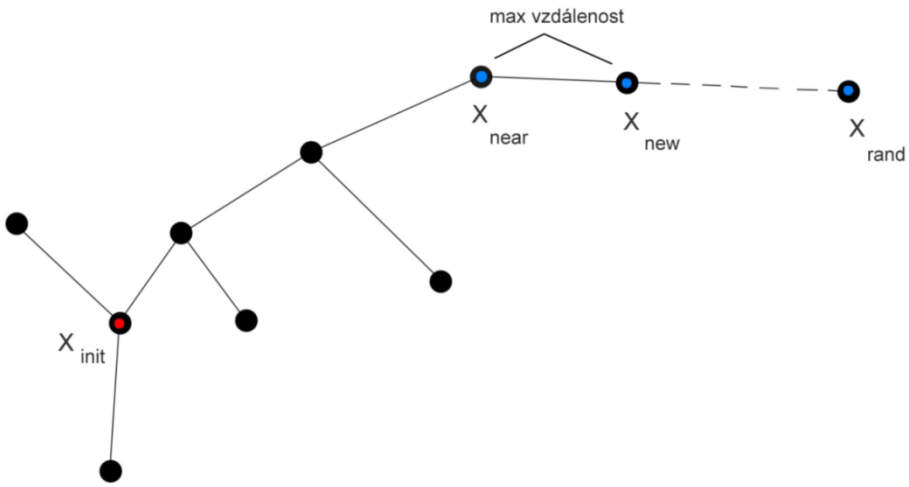
Rapidly-exploring random tree začíná v předem známém uzlu, který je nazýván *init node* a postupně se rozšiřuje do cílového uzlu neboli *goal node*. V každé iteraci se pak náhodně generuje nový uzel. Náhodnou generací uzlu je myšleno náhodné generování souřadnic v závislosti na dimenzi prostoru, velikosti prostoru, respektive rozměry oblasti, ve které se bezpilotní letoun bude pohybovat, a přesnosti vytváření mapy. Tím je myšleno, v jakém formátu budou generovány souřadnice. Ty můžou být generovány jako celá čísla či desetinná čísla na určitý počet desetinných míst.

Během každé iterace, kdy je náhodně vygenerován nový uzel, neboli *rand node*, je zároveň zjištěn a připojen nejbližší uzel k celému již vygenerovanému stromu. Připojení uzlu k trajektorii je také ovlivněno parametrem maximální vzdálenosti přípustného nodu. Tento parametr se volí v závislosti na přesnosti generování či v závislosti na přesnosti informací, které jsou o prohledávaném prostoru známé. Pokud jsou tyto podmínky splněny, včetně podmínky, že nově vygenerovaný uzel neleží v místě překážky, je pak *rand node* připojený k n nejbližšímu uzlu, *nearest node*. Pokud se v dané maximální vzdálenosti nenachází žádný z nejbližších uzlů, i tak je vygenerována nová cesta. Tato cesta si vytvoří

[7] Odvětví optimalizace, kdy je algoritmus schopný reagovat na nové stavy či změny a přizpůsobit jim své další procesy

záchytný bod, který je lokalizován na úsečce mezi aktuálním a nejbližším nodem. Tento záchytný bod je pak vzdálen od aktuálního uzlu, právě o maximální délku, která je zapsána v algoritmu jako parametr. Tímto způsobem je dosaženo faktoru, kdy náhodné vzorky mohou být považovány za řízení směru růstu stromu, zatímco maximální vzdálenost určuje rychlost tohoto růstu.

Na obrázku 15 je vyobrazeno právě ono vytvoření nového uzlu při překročení maximální vzdálenosti náhodného uzlu a k němu nejbližšímu sousedovi.



Obr. 15 RRT – nástin běhu algoritmu

Takto algoritmus pokračuje do té doby, než dosáhne cílového uzlu, *goal node*.

Možnosti ukončení algoritmu:

1. Algoritmus skončí ihned po dosažení cílového nodu a nebude již generovat node uzly a ani generovat novou cestu, i přes to, že by mohla být levnější ve smyslu délky dráhy.
2. Algoritmus neskončí ani po dosažení cílového uzlu, ale pokračuje dál v generování nových nodů a prohledávání prostoru pro získání nové a možná i optimálnější cesty a končí až po dosažení určité doby běhu algoritmu. Tento parametr času běhu je předem známý.

3. RRT opět neskončí ani po dosažení cílového uzlu, ale pokračuje v procesu až do dosažení určitého počtu iterací. Tento parametr počtu iterací je předem známý.
4. Algoritmus skončí ihned na začátku. To je způsobeno tím, že je *goal node* lokalizován v prostoru překážky.
5. RRT nikdy neskončí i při nedosažení cílového uzlu. To může být způsobeno tím, že přesnost generování jednotlivých uzlů je nastavena špatně.

První případ je v celku v pořádku. U druhého a třetího případu existuje pravděpodobnost, že RRT vůbec cestu z počátečního uzlu do cíle nenajde právě kvůli omezené době či počtu iterací. Tento problém je ovšem řešitelný dobrými znalostmi o prostoru a vždy se nastavují na míru. Čtvrtý problém by měl být samozřejmě ošetřen již na začátku algoritmu, aby se předešlo zbytečnému mrhání paměti stroje, který zadanou úlohu zpracovává. Pátý a zároveň poslední problém musí opět řešit specialista na dané prostředí a upravit parametry programu tak, aby k takovémuto problému nedošlo.

Algoritmus RRT může být také ovlivněn tím, že růst prohledávacího stromu prostoru bude směřován do určité oblasti. Toho se dosáhne zvýšením pravděpodobnosti generování nových nodů v určité oblasti. Tímto procesem může být také algoritmus urychlen.

Následující pseudokód popisuje algoritmus RRT [24]. Metody z tohoto algoritmu jsou již popsány v předchozí kapitole PRM.

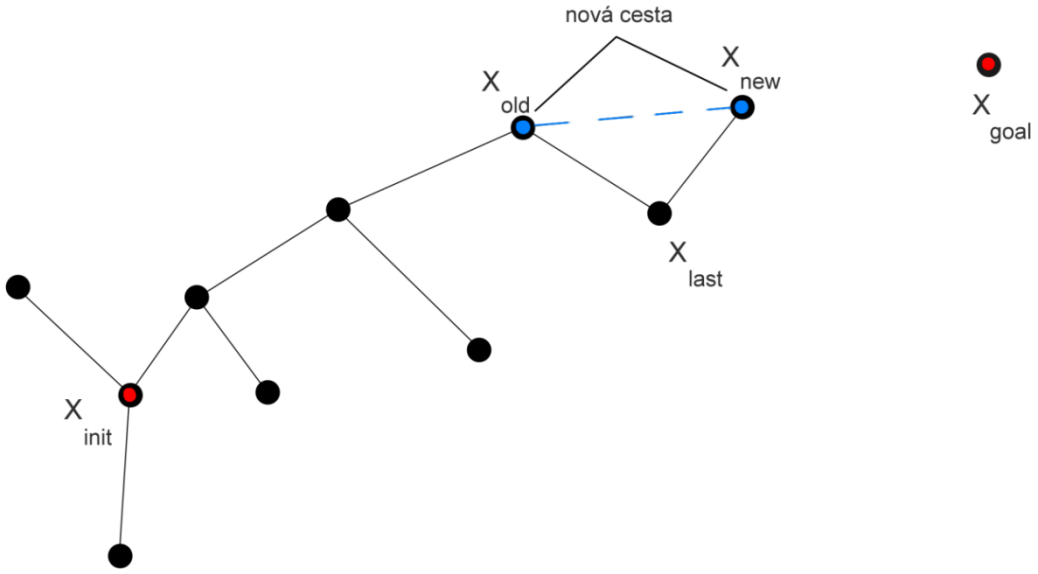
Algoritmus 7) RRT

```
τ:=initializeTree()                          // vytvoření prostoru
τ:=insertNode(node_init, τ)                  // zapsání startovního uzlu
for i := 1 to N                              // N = maximální počet uzlů
      z_rand := generateRandomNode(i)
      z_nearest := nearest(z_rand, τ)
      (z_new, u_new, T_new) := steer(z_nearest, z_rand)
      if obstacleFree(x_new):
            τ := insertNode(z_new, τ)
      end
end
```

3.3.3. Rapidly – exploring random tree Star

Metodu RRT* vytvořili Karaman a Frazzoli [24].

Princip tohoto algoritmu je velmi podobný principu RRT, který byl popsán v předešlé kapitole. Popis RRT* obsahuje veškeré prvky a metody, které používá i předchůdce tohoto algoritmu, rapidly-exploring random tree, a navíc má dvě metody. Tyto metody se nazývají `nearNeighbor` a `rewiringTree`. V překladu blízký soused a obnovení stromu.



Obr. 16 RRT* - Optimalizace trajektorie

Obrázek číslo 16 popisuje metody `nearNeighbor` a `rewiringTree`. Uzel x_{old} je předchůdcem uzlu x_{last}, který je nejbližším sousedem pro x_{new}. Pro optimalizaci trajektorie algoritmu RRT* spojí právě x_{old} a x_{new}.

Tyto dvě metody dovolují RRT* algoritmu provádět optimalizaci cesty, která je nalezená algoritmem RRT. Optimalizace nejenže zkrátí celkovou výslednou cestu algoritmu, ale také ji upraví tak, že výsledná trajektorie vypadá mnohem realističtěji, co se týče reálného pohybu robota.

Následující pseudokód popisuje algoritmus RRT* [25]. Pod tímto kódem jsou dále uvedeny popisy k hlavním metodám, které jsou do tohoto algoritmu přidány.

Skript 8) RRT*

τ:=**initializeTree**()
τ:=**insertNode**($node_{init}$, τ)
for i := 1 **to** N
    z_{rand} := **generateRandomNode**(i)
    $z_{nearest}$:= **nearest**(z_{rand}, τ)
    (z_{new}, u_{new}, T_{new}) := **steer**($z_{nearest}$, z_{rand})
    if obstacleFree(x_{new}):
        z_{near} := **neararest**(τ, z_{new})
        z_{min} := **chooseParent**(z_{near}, $z_{nearest}$, z_{new})
        τ := **insertNode**(z_{min}, z_{new}, τ)
        τ := **rewire**(z_{near}, z_{min}, z_{new}, τ)
    end
end

Vysvětlení hlavních funkcí algoritmu. Metody shodné s předchozím algoritmem RRT jsou vysvětlené v předchozí kapitole.

- **Rewire**: Tato funkce kontroluje, zdali je cesta z blízkého uzlu přes nově vytvořený uzel levnější, tedy kratší než cena minulá. Pokud funkce vrátí hodnotu *true*, jako předek tohoto uzlu je označený nový uzel.
- **ChooseParent**: Funkce vracející nejbližšího souseda k aktuálnímu uzlu, tedy k uzlu, který byl zrovna vytvořen.

3.3.4. Rapidly – exploring random tree with fixed nodes

Rozdíl mezi RRT* a RRT*FN [26] je pouze maximální počet povolených uzlů v paměti.

Omezení paměti zvyšuje rychlost výpočtu vzdáleností nového uzlu ke zbytku uzlů. Tím se radikálně zvýší rychlost celého algoritmu. Před každým generováním nového náhodného uzlu je nejprve vyhodnocena podmínka, zdali maximální povolený počet uzlů je splněn. V případě, že maximální počet přesáhl povolenou hodnotu, je nějaký uzel vymazán. Tento uzel musí ovšem splňovat jistá kritéria. Hlavním a vždy povinným kritériem je, že tento uzel nesmí mít žádné potomky. V případě, že by toto nebylo splněno, nemůže být smazán. Další kritéria pro smazání uzlu jsou již čistě na programátorovi. Politika pro vyčlenění uzlu může být například nejvzdálenější uzel od cílového uzlu. Či nejvyšší prioritu pro odstranění mají ty uzly, které se nacházejí vně překážky.

Následující pseudokód popisuje algoritmus RRT*FN [26].

Skript 9) RRT*FN

τ:=**initializeTree**()
τ:=**insertNode**($node_{init}$, τ)
M
for i := 1 **to** *N*
    if *numberOfNodes > M* // *kontrola počtů uzlů ve stromě*
        τ.removeNode*()*
        τ_{old} .insertNode*()*
    end
    $z_{rand} :=$ **generateRandomNode**(i)
    $z_{nearest} :=$ **nearest**(z_{rand}, τ)
    (z_{new}, u_{new}, T_{new}) := **steer**($z_{nearest}$, z_{rand})
    if obstacleFree(x_{new}):
        $z_{near} :=$ **nearest**(τ, z_{new})
        $z_{min} :=$ **chooseParent**(z_{near}, $z_{nearest}$, z_{new})
        τ := **insertNode**(z_{min}, z_{new}, τ)
        τ := **rewire**(z_{near}, z_{min}, z_{new}, τ)
    end
end

Vysvětlení hlavních funkcí algoritmu. Metody shodné s předchozím algoritmem RRT* jsou vysvětlené v předchozí kapitole.

- **removeNode**: Funkce, která zjišťuje, jestli alespoň jeden uzel nemá svého potomka. Pokud je toto pravidlo splněno, je právě tento uzel smazán. V případě, že je vrácena hodnota *false*, pak žádný uzel odstraněn není.

Kapitola 4

Testy

4.1. Testy vybraných algoritmů

Veškeré simulace probíhali na stolním počítači s operačním systémem Windows 10 64 bit, s procesorem i5-9600k s frekvencí 3.7 GHz. Vývojovým a simulačním prostředím byl MATLAB.

Nejdříve budou testovány algoritmy navržené pro použití v diskrétním prostoru a až poté ve spojitém prostoru. Veškerá prostředí jsou shodná pro diskrétní i spojitý prostor a veškerá nastavení parametrů pro jednotlivé metody budou vysvětleny v jim příslušných kapitolách.

Na začátku každé kapitoly bude uvedena simulace pro 2D prostor. Jelikož 2D prostor není předmětem této bakalářské práce, bude uveden pouze jeden test. Po té budou provedeny testy v 3 různých prostředích pro 3D prostor. Obtížnost, respektive obsah překážek, se bude v každém prostředí stupňovat. Třetí test bude tedy nejobtížnější.

Simulování jednotlivých metod bude probíhat celkem 1000 krát pro každé simulované prostředí a bez veškerých vizuálních operací. Tento počet simulací byl vybrán za účelem eliminace nežádoucích vedlejších procesů, které by se mohli během simulace v počítači spustit. Čas i délka optimální cesty pak bude průměrována a v každém prostoru procentuálně vyčíslena vůči ostatním metodám.

Výsledy jednotlivých simulací budou zaznamenány v obrázcích a v tabulkách. Tabulky budou zobrazovat absolutní a relativní hodnoty trajektorie a doby zpracování. Kompletní zhodnocení všech testů pro diskrétní či spojitý prostor bude uveden na konci dané podkapitoly.

4.1.1. Testování v diskrétním prostoru

Pro implementaci byly vybrány algoritmy A*, Basic Theta* a Lazy Theta*, především kvůli užívání heuristiky ve svých procesech.

Dle informací prezentovaných v teoretické části a z popisu těchto algoritmů se dá předpokládat, že nejvíce optimální trajektorii nalezne Basic Theta*, ovšem s nejdelší dobou

zpracovávání. Naopak A* proběhne nejrychleji, její trajektorie nebude už tak optimální, co se týče délky a tvaru. Lazy Theta* by měla být někde mezi těmito předpoklady.

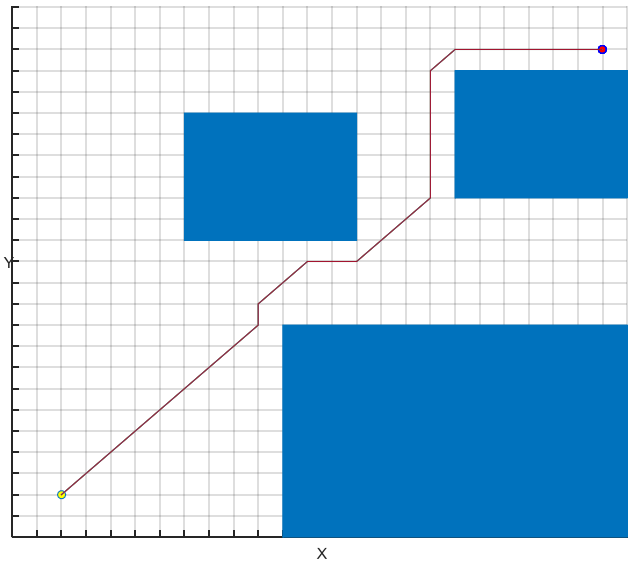
V relativním porovnání bude jako základ, tedy 100%, považována Basic Theta* a od ní se budou odvíjet ostatní procentuální hodnoty algoritmů.

Veškeré testování v diskrétním prostoru bude prováděno pro osmi-okolí ve 2D prostoru a pro dvacetišesti-okolí ve 3D prostoru. Díky tomuto procesu pak již není potřeba používat takzvaný post processing, který se u A* metody používá právě pro spojení dvou vrcholů grafu pomocí úhlopříčky pod úhlem 45°. U zbylých metod ve většině případů postprocessing není potřeba, jelikož berou v úvahu i uzly, které nejsou přímými sousedy.

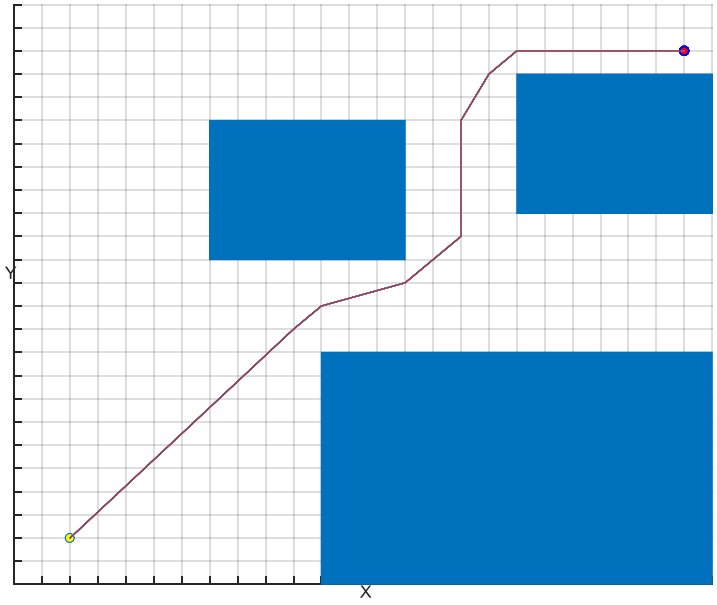
Testování ve 2D prostoru

První simulace probíhala v prostředí se třemi překážkami. Rozměry tohoto konfiguračního prostoru byly nastaveny na 25x25 metrů. Minimální vzdálenost mezi viditelnými uzly je tedy jeden metr a mezi vzdálenějšími sousedy pak 1.4142 metru. Počátečním bodem byl zvolen vrchol [3, 3] a cílovým [24, 23].

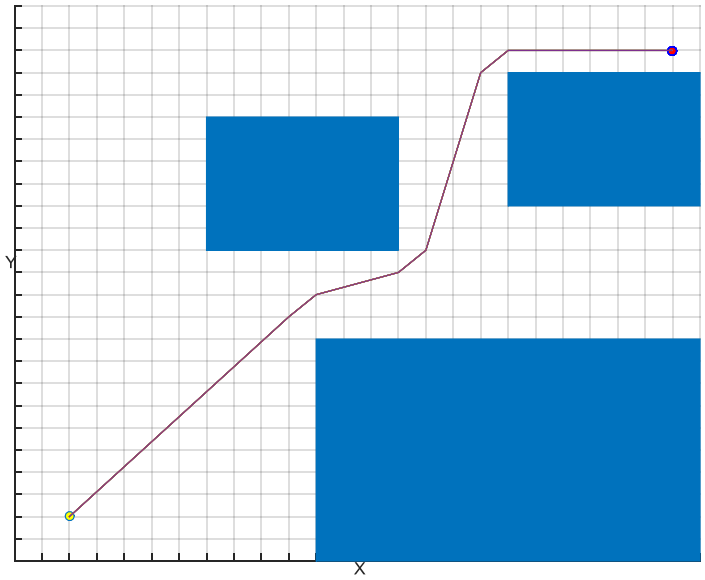
Výsledky prvního testování:



Obr. 17 A* algoritmus



Obr. 18 Basic Theta* algoritmus



Obr. 19 Lazy Theta* algoritmus

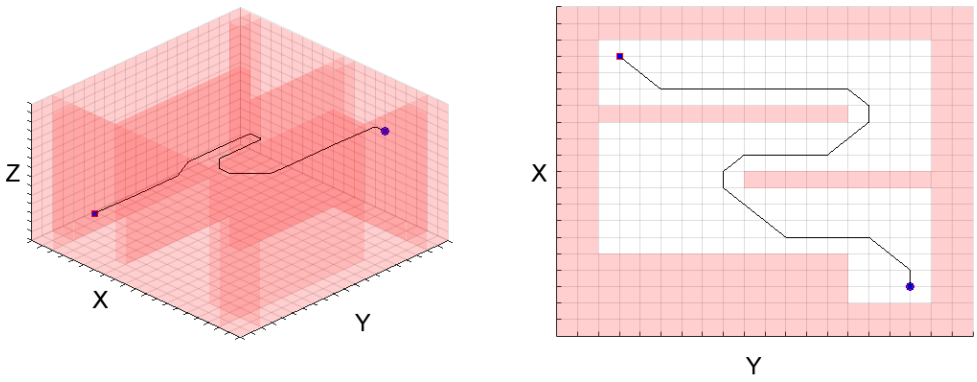
2D – prostor 1	A*		Basic Theta*		Lazy Theta*	
Výsledná trajektorie [m]	34.7990	103.28%	33.6927	100%	34.0968	101.20%
Čas [s]	0.0420	58.66%	0.0716	100%	0.0674	94.13%
`Lineofsight` kontroly[počet]		-		653		278

Tab. 1 Porovnání výsledků pro metody A*, Basic Theta* a Lazy Theta* - 1. Prostor

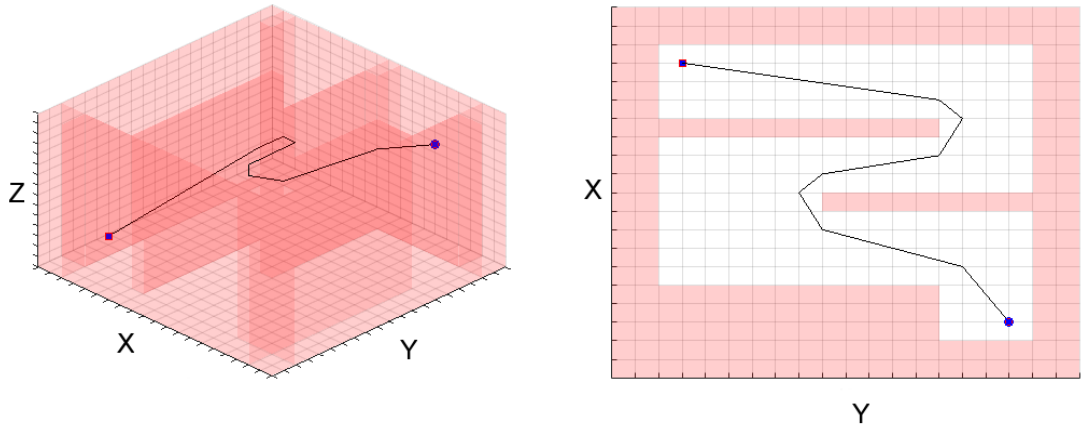
Jak je možné poznat z vygenerovaných trajektorií, nejvíce ostrá trajektorie je vygenerována metodou A*. Algoritmy Basic Theta* a Lazy Theta * jsou si, až na jeden úsek, velmi podobné. Kontrola viditelnosti jednotlivých vrcholů grafu potvrdila teoretické poznatky a Basic Theta* má ve 2D prostoru téměř třikrát vyšší počet kontroly viditelnosti.

Testování ve 3D diskrétním prostoru

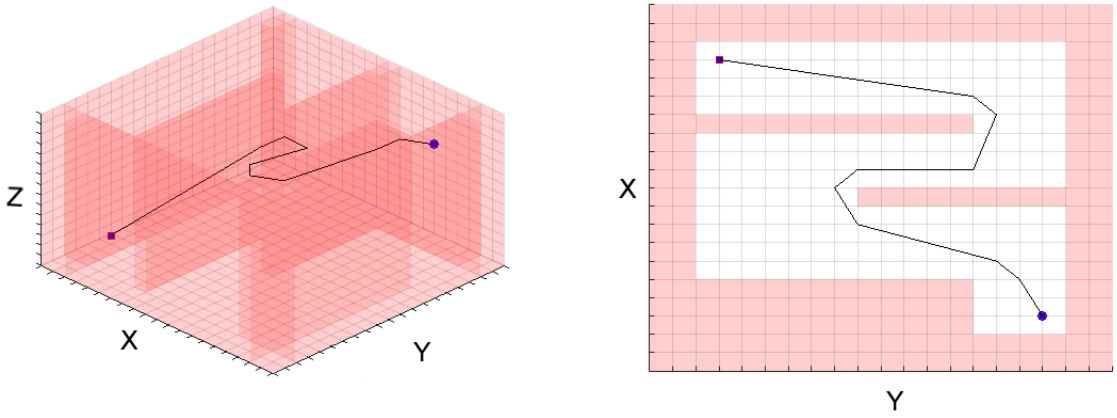
Prvním konfiguračním prostorem pro testování ve 3D byla zvolena mřížka o rozměrech 20x20x15 metrů se dvěma překážkami uvnitř a s kompletně ohraničeným prostorem překážkou. Počátečním vrcholem byl zvolen [3, 3, 3] a cílovým vrcholem [17, 17, 15].



Obr. 20 A* - 3D – prostor 1



Obr. 21 Basic Theta* - 3D – prostor 1



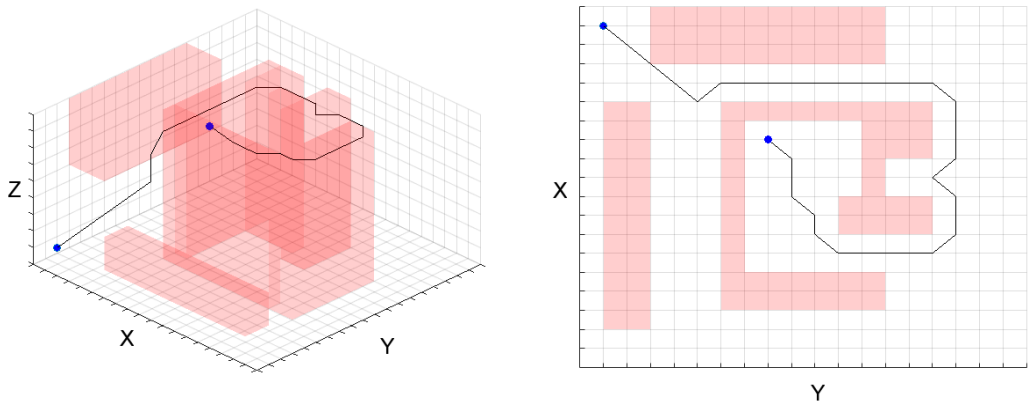
Obr. 22 Lazy Theta* - 3D prostor 2

3D – prostor 1	**A***		**Basic Theta***		**Lazy Theta***	
Výsledná trajektorie [m]	35.5563	101.45%	35.0471	100%	36.0436	102.84%
Čas [s]	0.5531	93.76%	0.5899	100%	0.5604	94.99%
`Lineofsight` kontroly [počet]		-		12323x		1201x

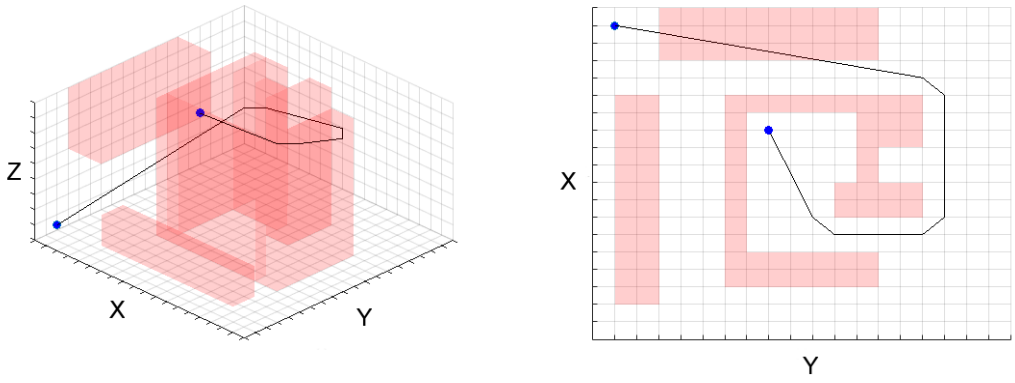
Tab. 2 Porovnání výsledků pro metody A*, Basic Theta* a Lazy Theta* - 2. prostor

Druhým konfiguračním prostorem pro testování ve 3D byla zvolena mřížka o rozměrech 20x20x10 metrů s počátečním vrcholem [2, 19, 2] a cílovým vrcholem [9, 13, 9]. Tento prostor byl vygenerován za účelem pozorování, jak se budou tyto algoritmy s heurestickou funkcí chovat v případě, kdy startovní a cílový vrchol jsou si relativně blízko a odděleny

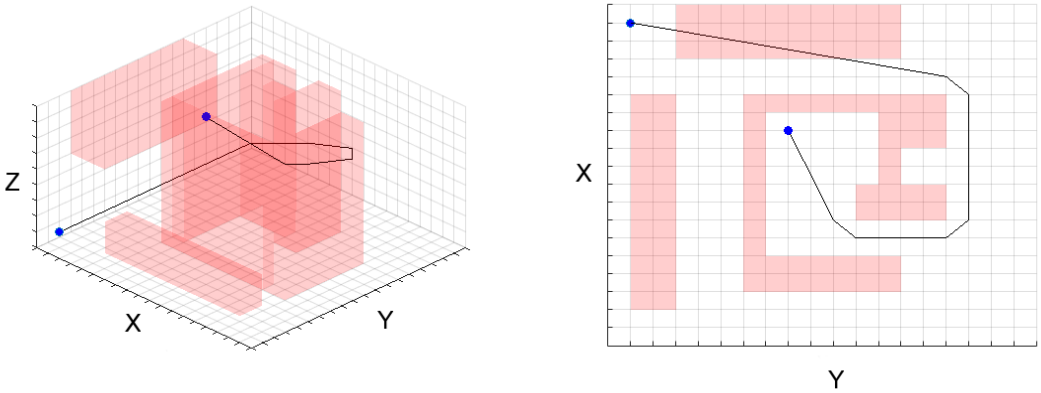
překážkami. Převážná většina volného prostoru, který bude zapotřebí prozkoumat, je na druhé straně konfiguračního prostoru.



Obr. 23 A* - 3D prostor 2



Obr. 24 Basic Theta* - 3D prostor 2

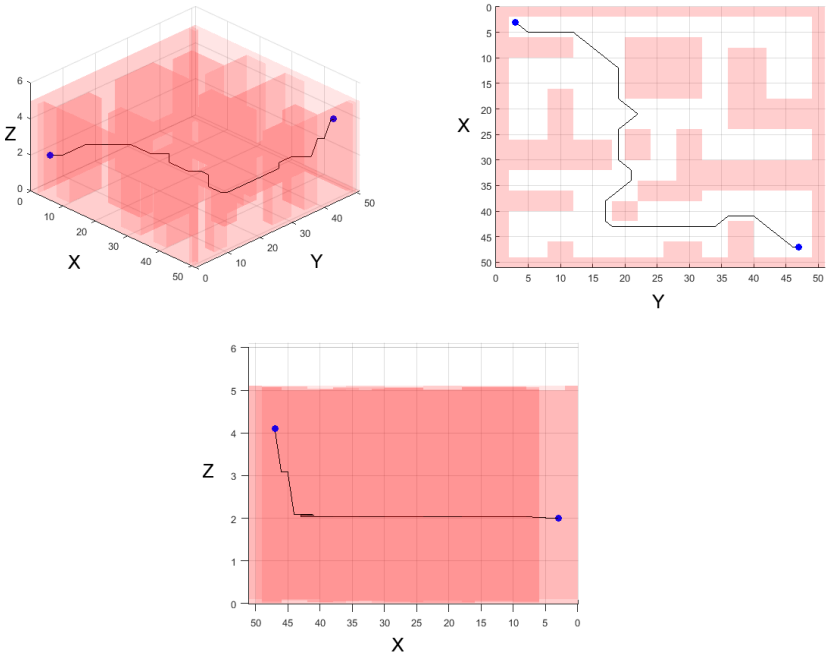


Obr. 25 Lazy Theta* - 3D prostor 2

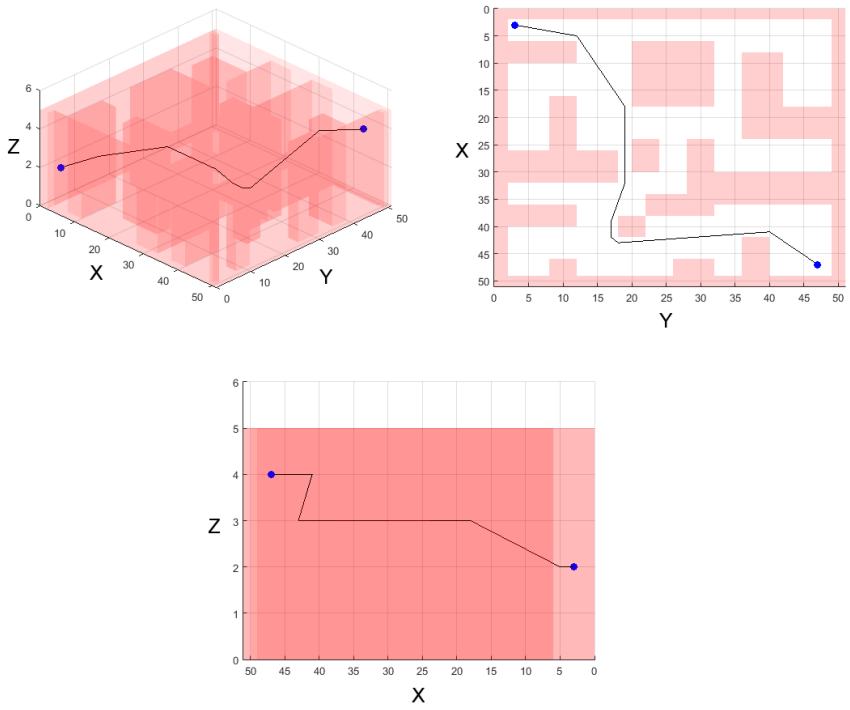
3D – prostor 2	A*		Basic Theta*		Lazy Theta*	
Výsledná trajektorie [m]	37.9706	106.12%	35.7801	100%	35.8525	100.20%
Čas [s]	0.8372	98.38%	0.8510	100%	0.8699	102.22%
`Lineofsight` kontroly [počet]		-		20700x		2094x

Tab. 3 Porovnání výsledků pro metody A*, Basic Theta* a Lazy Theta* - 3. Prostor

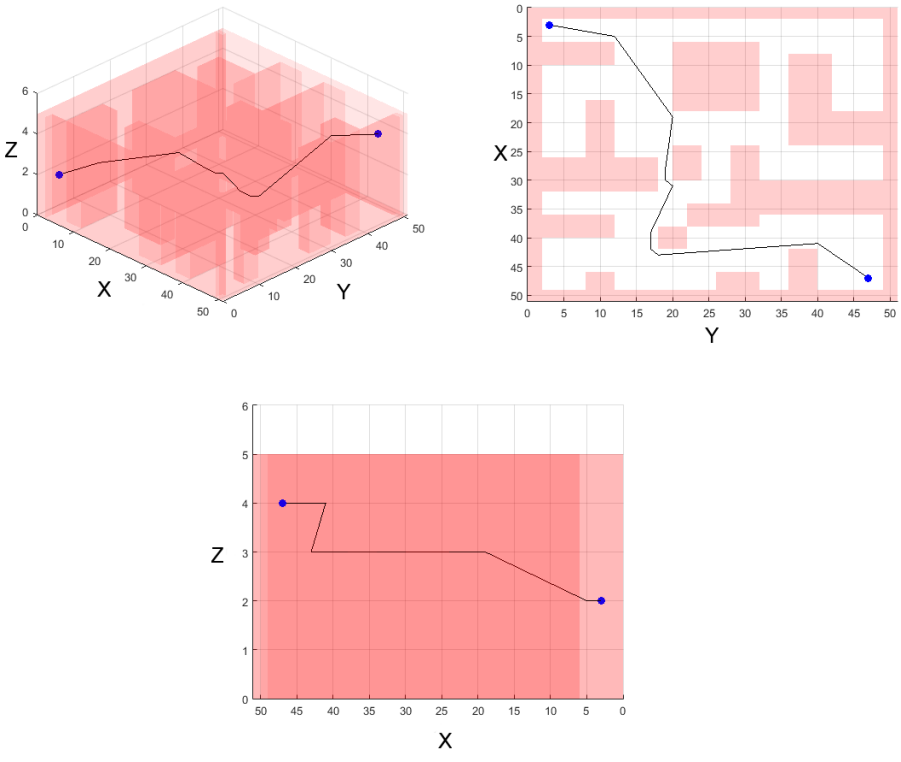
Třetí konfigurační prostor byl vygenerován s nejvíce překážkami a největšími rozměry 50x50x5 metrů. Test v tomto prostoru má za úkol ukázat chování algoritmů v problematickém prostoru, co se počtu a rozmístění překážek týče. Počáteční vrchol byl zvolen o souřadnicích [3, 2, 2] a cílový vrchol [47, 47, 4]. Pouhý skok dva metry na ose Z je zvolen z důvodu porovnání především algoritmu A* se zbytkem algoritmů a proto je na obrázcích 26 – 28 zobrazena i osa Z.



Obr. 26 A* - 3D prostor 3



Obr. 27 Basic Theta* - 3D prostor 3



Obr. 28 Lazy Theta* - 3D prostor 3

3D - prostor 3	A*		Basic Theta*		Lazy Theta*	
Výsledná trajektorie [m]	88.4264	109.10%	81.0454	100%	82.1357	101.35%
Čas [s]	2.9822	109.32%	2.7280	100%	3.2075	117.58%
`Lineofsight` kontroly [počet]	-		41108x		4054x	

Tab. 4 Porovnání výsledků pro metody A*, Basic Theta* a Lazy Theta* - 4. Prostor

Z uvedených výsledků lze soudit, že nejlépe fungující algotimus pro prohledávání diskrétního konfiguračního prostoru je Basic Theta*. A* algoritmus byl užit především z důvodu rychlosti zpracování konfiguračního prostoru, ale jeho výsledná trajektorie není příliš realizovatelný pro let UAV a to především kvůli problému s ostrostí přechodu mezi vrcholy grafu. Lazy Theta* potvrdila předpoklad menšího počtu kontrol viditelnosti mezi jednotlivými vrcholy. Tím je ovšem zapříčiněna možnost chybné predikce předka pro jednotlivé vrcholy a v komplikovaném prostoru může mít takto chybná predikce fatální následky na dobu běhu algoritmu. Z teoreticky rychlejšího algoritmu se pak stává algoritmus pomalejší a to z důvodu nutnosti přepočítání všech sousedů z právě hodnoceného vrcholu. Trajektorie vygenerovaná právě pomocí metody Lazy Theta* je pak ve více problematickém prostoru také ne příliš proveditelná UAV.

Dále je z testů možné zjistit fakt, že teorie popsaná na prostor ve 2D není příliš aplikovatelná na prostředí 3D. Ve 2D prostoru dosahoval A* algoritmus téměř poloviční doby zpracování vůči algoritmu Basic Theta*. Ve 3D prostoru jsou jejich časy zpracování velmi podobné. Délka trajektorie a její tvar ovšem stále hraje ve prospěch algoritmu Basic Theta*. Z těchto důvodu bude v poslední kapitole testována trajektorie vygenerovaná právě algoritmem Basic Theta*.

Velkou výhodou algoritmů v této kapitole je rychlost nalezení optimální trajektorie při změně cíle. To je způsobeno tím, že při prohledávání grafu si tyto algoritmy ponechávají informaci o předchůdci pro každý vrchol grafu. Pro tuto funkci okamžitého nalezení nové trajektorie, je ovšem potřeba nejprve prohledat celý graf. V případě že se tak nestane a cíl bude změněn během procesu algoritmu, může být nový cíl mimo již prohledané vrcholy a musí být pak nejprve nalezen. Pokud je již graf celý prohledán, pak je nalezení nové trajektorie téměř okamžité.

4.1.2. Testování ve spojitém prostoru

Pro prohledávání spojitého prostoru byly vybrány pro testování algoritmy RRT, RRT* a RRT*FN. Důvodem je částečná podobnost s algoritmy testovanými v diskrétním prostoru. I v tomto případě je algoritmus RRT základním algoritmem, RRT* je jeho rozšířením s částečnou optimalizací trajektorie a RRT*FN stejně jako Lazy Theta* zkracuje dobu prohledávání určitou eliminací. V případě Lazy Thety* to byla regulace počtu spuštění funkce `lineofsight`, v případě RRT*FN se jedná o vytěsnění a smazání nevhodných uzlů.

Tyto algoritmy jsou implementovány tak, aby prostor zpracovávaly pomocí dvou módů. Prvním módem je vygenerování maximálního počtu uzlů nehledě na to, zda už bylo dosaženo cílového uzlu. Druhým parametrem je ukončení algoritmu ihned po dosažení cílového uzlu nehledě na nedosažení maximálního počtu vygenerovaných uzlů.

Předpoklad ve spojitém prostoru je podobný tomu v diskrétním. Tedy původní algoritmus (RRT) bude nejrychleji prohledávat prostor, ovšem jeho trajektorie nebude příliš optimální z hlediska délky a tvaru. To bude řešit právě RRT*, jenž by měl mít mnohem reálněji vypadající trajektorii, čas zpracování bude ovšem nejdelší. Kompromisem je opět až další modifikace předešlé metody a tou je metoda RRT*FN. Tato metoda by díky promazávání stromu měla být časově mezi těmito dvěma algoritmy a trajektorie by se měla blížit trajektorii RRT*. Pomocí náhodně generovaných uzlů mohou být ovšem výsledky trajektorií velmi podobné u metod označených jako Star.

U první simulace bude zobrazen vývoj prohledávání, tedy vývoj po určitém počtu iterací. Z důvodu nepřehlednosti ve 3D prostoru již tento vývoj zobrazen nebude.

První set simulací probíhal v módu 1, tedy ukončení ihned po nalezení první možné trajektorie. Vstupní parametry pro metody:

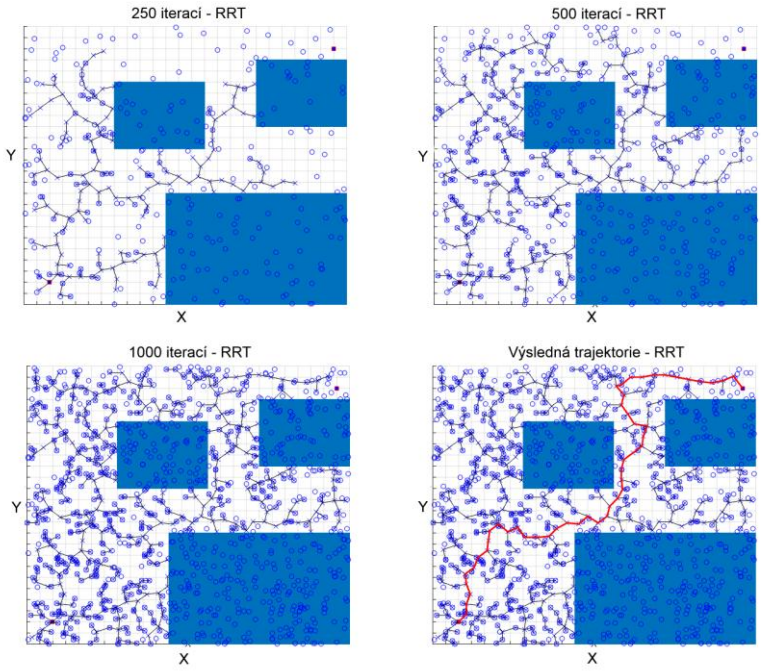
1. Maximální vzdálenost mezi uzly: 1 metr
2. Počet vygenerovaných uzlů: 1000
3. Maximální povolená vzdálenost pro optimalizaci: 3 metry

Maximální povolený počet uzlů ve stromě je jen pro algoritmus RRT*FN a je zobrazen v jednotlivých sloupcích právě u tohoto algoritmu.

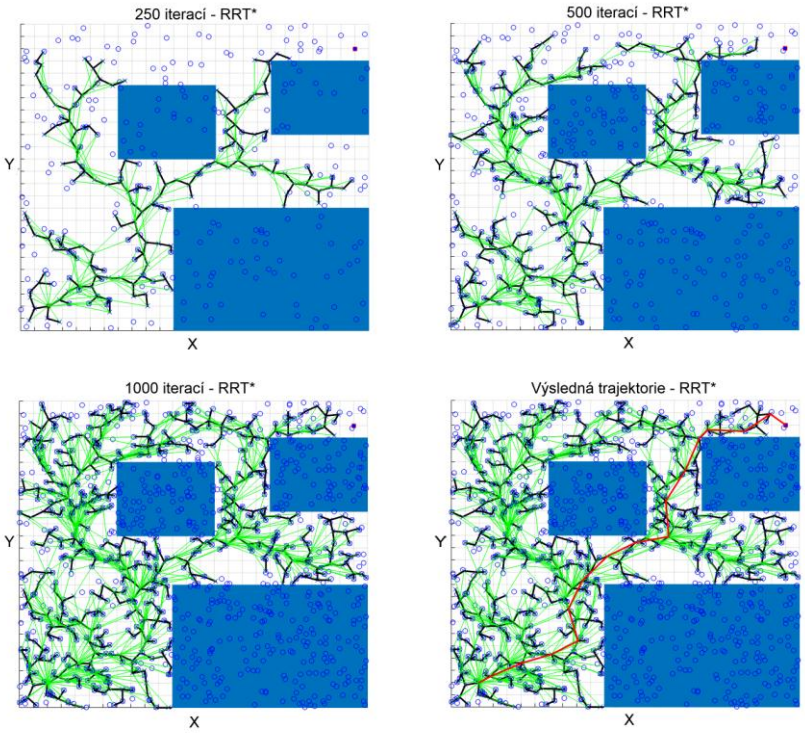
Druhý set simulací probíhal v módu 2 a měl stejné vstupní parametry jako simulace během módu 1.

Z obrázků 30 – 31 je možné vidět, že vygenerované trajektorie i průběh generování uzlů je u algoritmů RRT* a RRT*FN velmi podobný. Na obrázcích 29-31 jsou pak modrými kruhy znázorněny generované uzly, černými linkami pak aktuální spoje mezi uzly, zelenými linkami pak spoje pro optimalizace trajektorie a červenou čarou pak výsledná trajektorie.

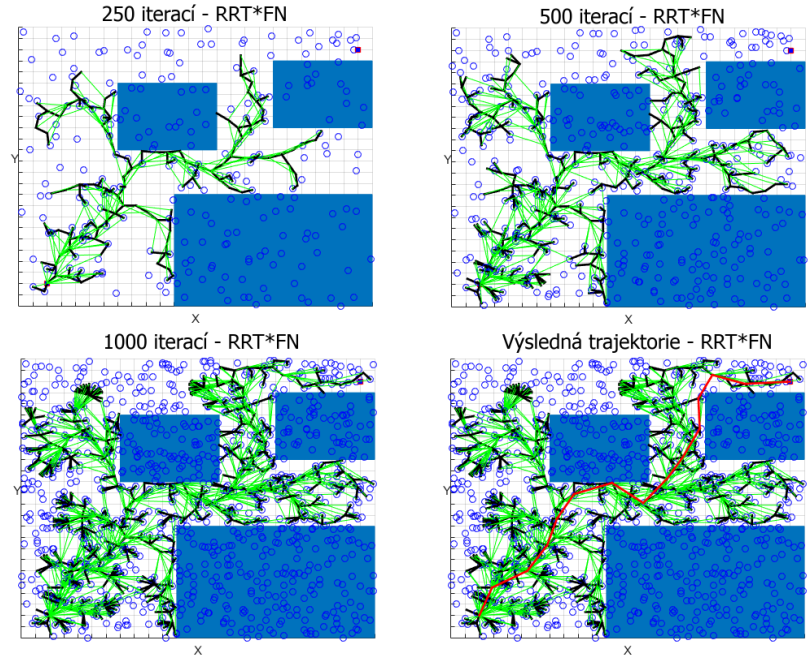
Jak je možné vidět z tabulek 4 - 5, původní předpoklady nebyl zcela potvrzen. Předpoklad pro algoritmus RRT byl správný, díky pouhému generování a hledání nejbližšího uzlu bez optimalizování trajektorie pomocí prohledávání okolí pro získání optimálnější cesty, byl čas tohoto algoritmu nejrychlejší. Trajektorie, jak je možné vidět na obrázcích 4, není příliš optimální a je také nejdelší. Optimálnější trajektorie je ovšem vidět na obrázku číslo 5, kde algoritmus prohledává právě i okolí nově vygenerovaného uzlu. RRT metoda tedy ve 2D prostoru splnila teoretické předpoklady.



Obr. 29 Vývoj RRT stromu 2D



Obr. 30 Vývoj RRT* stromu 2D



Obr. 31 Vývoj RRT*FN stromu 2D

Mód 1	RRT	RRT*	RRT*FN			
			200	250	350	500
Výsledná trajektorie [m]	43.1433	37.1015	36.9609	36.5054	35.9965	36.4896
	116.3%	100%	98.6%	98.4%	97%	96.4%
Čas [s]	1.4430	2.9678	2.7561	2.8569	3.1148	3.5057
	48.6%	100%	92.9%	96.3%	104.9%	118.1%

Tab. 4 Porovnání RRT, RRT*, RRT*FN – prostor 1, mód 1

Mód 2	RRT	RRT*	RRT*FN			
			200	250	350	500
Výsledná trajektorie [m]	43.1846	37.2061	36.1307	36.2358	35.9978	36.6239
	116.1%	100%	97.1%	97.4%	96.8%	98.4%
Čas [s]	2.4456	4.9230	3.7790	4.2684	5.9635	6.5718
	49.7%	100%	76.8%	86.7%	121.1%	133.5%

Tab. 5 Porovnání RRT, RRT*, RRT*FN – prostor 1, mód 2

U metody RRT*FN bylo předpokládáno, že vygenerovaná trajektorie bude, co se týče vzdálenosti, mezi RRT a RRT* a to především z důvodu promazávání stromu. Mělo by tedy být pravděpodobné, že bude smazán právě uzel, který v pozdějším procesu algoritmu může být mnohem lépe umístěný než nově vygenerované uzly a z toho důvodu nebude trajektorie tolik optimální. Tento fakt může být ovšem eliminován právě pravidlem pro mazání uzlů. V těchto simulacích bylo použito pravidlo, že nejvzdálenější uzly od cílového uzlu, které nemají přímého následníka, jsou tedy nerozvětvené (pravidlo popsané v kapitole 3.2.3.), budou smazány. Toto pravidlo by pravděpodobně měl navrhovat specialista na dané prostředí, například s pomocí zakomponování heuristiky.

Dalším předpokladem u metody RRT*FN byla nižší časová náročnost než u RRT*. Jak lze vidět v tabulkách výše, je to splněno jen v některých případech. Obecně by se dalo říci, že algoritmus je rychlejší jen v případech, kdy maximální počet uzlů ve stromě je v hodnotě od 1/5 do 1/3 z maximální povolené hodnoty vygenerovaných uzlů. Nižším počtem uzlů ve stromě je pak riskováno špatné vygenerování trajektorie. Simulace s nižším počtem než je 1/5, byly z více jak 90% neúspěšné, proto nejsou zahrnuty do tabulky výsledků. Tento problém neúspěšnosti je ovšem možné kompenzovat prodloužením maximálních vzdáleností mezi jednotlivými uzly. Naopak simulace s větším počtem povolených uzlů v paměti stromu

zapříčiňovali pouze zvýšení času simulace. Vygenerovaná trajektorie byla velmi podobná simulacím s nižším počtem povolených uzlů a proto také nebyly zahrnuty do tabulky. Zvýšení času je pravděpodobně zapříčiněné právě pravidlem pro mazání uzlů, tedy pro nalezení maximální vzdálenosti k cíly, neboť tato vzdálenost není počítána v klasickém procesu algoritmu. Jak již bylo psáno výše, tento problém by mohl být kompenzován pravidlem na míru danému prostředí.

Další možností pro urychlení RRT*FN je mazání většího počtu uzlů v jediném kroku algoritmu. Během simulace bylo toto pravidlo potvrzeno pouze v některých případech, v jiných bylo naopak na škodu a proto z důvodu velké variance naměřených hodnot tyto simulace nebyly zapsány do tabulky výsledků.

Tato fakta budou dál konzultována po simulacích v prostoru 3D.

Testování ve 3D spojitém prostoru

Simulace v této podkapitole proběhnou pro každý prostor s různými počátečními nastaveními. Tyto počáteční podmínky ve formě volených parametrů budou rozděleny pod názvy Sada 1-4, které budou korespondovat s tabulkami uvedenými pod každým testováním. Obrázky k jednotlivým testováním budou uvedeny v módu 1 a pak v módu 2 pro 1. a 3. prostor a při vyšší povolené vzdálenosti (jedná se tedy o sadu parametrů 4). Prostor 2 pak bude zobrazen jen v módu 1, neboť zahlcenost prostoru v módu 1 i 2 je téměř identická. Z důvodu chaotičnosti generovaných uzlů a podobnosti generování u algoritmů RRT* a RRT*FN budou obrazové výsledky uvedeny jen u algoritmu RRT*.

Jednotlivé sady:

Sada 1

- Mód: 1
- Maximální počet vygenerovaných uzlů: 2000
- Maximální vzdálenost mezi uzly: 2 metry
- Maximální povolená vzdálenost pro optimalizaci: 4 metry

Sada 2

- Mód: 1
- Maximální počet vygenerovaných uzlů: 2000
- Maximální vzdálenost mezi uzly: 4 metry
- Maximální povolená vzdálenost pro optimalizaci: 6 metry

Sada 3

- Mód: 2
- Maximální počet vygenerovaných uzlů: 2000
- Maximální vzdálenost mezi uzly: 2 metry
- Maximální povolená vzdálenost pro optimalizaci: 4 metry

Sada 4

- Mód: 2
- Maximální počet vygenerovaných uzlů: 2000
- Maximální vzdálenost mezi uzly: 4 metry
- Maximální povolená vzdálenost pro optimalizaci: 6 metry

V případě RRT*FN je pak zachován poměr maximálního počtu povolených uzlů ve stromě vůči maximálnímu počtu vygenerovaných uzlů jako v prostoru 2D. Tento parametr byl zvolen z důvodu snahy zjistit, zda předpoklady z prostoru 2D mohou být aplikovány na prostor 3D.

Maximální počet vygenerovaných uzlů byl nejprve uvažován s hodnotou 3000, aby byl zkoumaný prostor zaplněn podobně jako prostor 2D, ovšem z důvodu praktičnosti byl zvolen maximální počet uzlů na 2000, ale maximální vzdálenosti byly oproti prostoru 2D zvýšeny. Jelikož se jedná o přenastavení maximálních hodnot, algoritmus může i tak generovat reálně vypadající trajektorie.

Testování algoritmů proběhlo ve stejných prostředích jako testování v diskrétním prostoru, popisy těchto prostorů jsou tedy uvedeny pouze v kapitole 4.1.2. a k nim v příslušném testování.

Ze získaných výsledků (které budou uvedeny dále) lze vypozorovat, že výsledná trajektorie je v každém prohledávaném prostoru nejlépe (z pohledu délky trajektorie) nalezena algoritmem RRT*FN. Předpoklad pro algoritmus RRT byl takový, že generované trajektorie budou ve

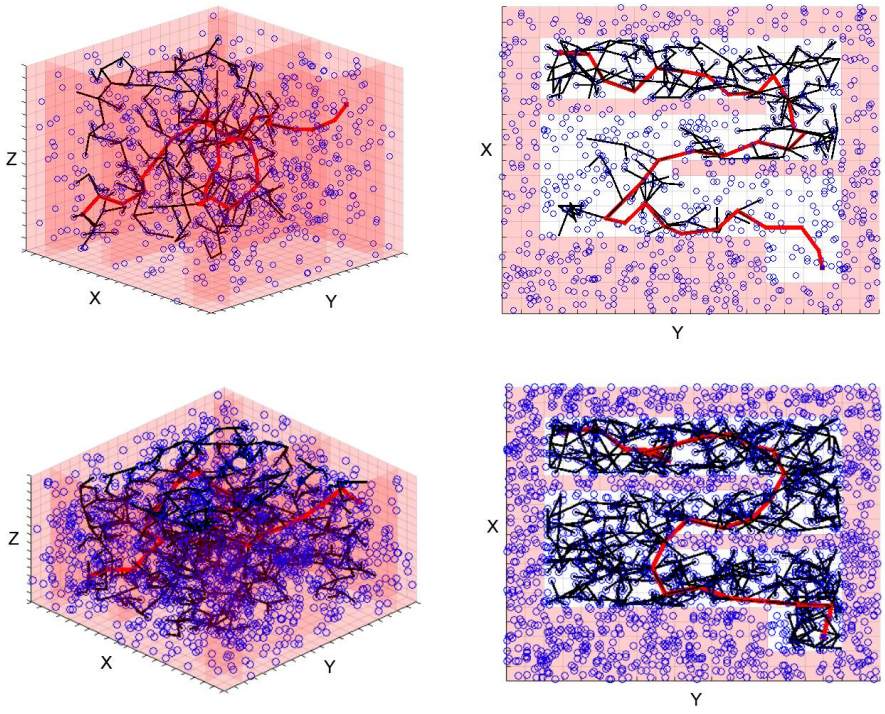
většině případů nejdelší a z tabulek níže je možné vidět, že tato predikce byla správná. Co se týče doby zpracování algoritmu, ta je vždy nižší než doba u RRT* a u RRT*FN, což bylo také očekáváno. Důvodem je samozřejmě prohledávání okolí v případě RRT* a RRT*FN. RRT tuto matematickou funkci nepoužívá a je díky tomu rychlejší. Avšak z důvodu nerealisticky vypadající trajektorie je RRT hodnocena jako neoptimální algoritmus pro plánování trajektorie UAV. Tento fakt by mohla řešit případná optimalizace trajektorie po proběhnutí algoritmu. Tuto optimalizace již částečně používají zbylé dvě metody během procesu a proto jsou předmětem zájmu této práce.

Z testů ve 2D prostoru bylo dále zjištěno, že v případě zvolení maximálního počtu uzlů v paměti v rozmezí do 1/5 do 1/3, pro algoritmus RRT*FN, je pak jeho doba procesu nižší než u algoritmu RRT*. V případě testů v prostoru 3D byl tento předpoklad splněn téměř v každém testu, dokonce pro prostor 2 byla doba běhu algoritmu téměř poloviční. To je pravděpodobně zapříčené tím, že cílový uzel byl blízko počátečnímu uzlu, ale oddělovala je překážka. Ale díky promazávání stromu u RRT*FN, kde pravidlo mazání bylo nastaveno právě, tak, že promazáván byl volný prostor, který byl daleko od cílového uzlu, veškeré soustředění generování uzlů se pak koncentrovalo právě do části konfiguračního prostoru, kde se nacházel cílový i počáteční uzel.

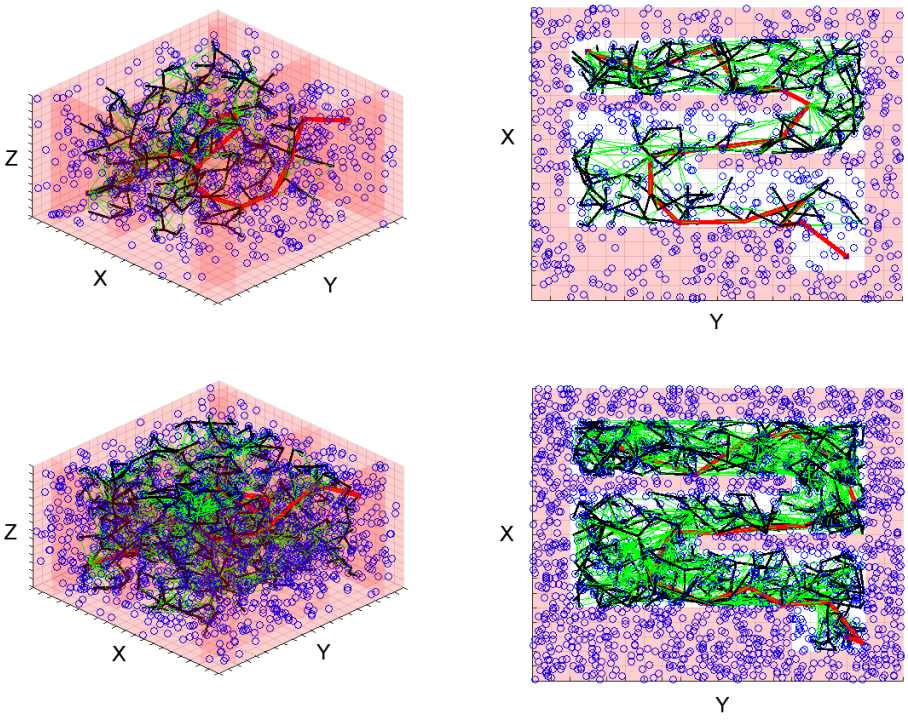
V několika případech byla doba procesu u RRT*FN, při zvolení jedné poloviny povolených uzlů v paměti, dokonce nižší než doba procesu u algoritmu RRT*. Tento fakt se objevil u testů především v módě 2 a při využití větších povolených vzdáleností.

I přes některé vyšší časové rozdíly procesů mezi algoritmy RRT* a RRT*FN, délky trajektorie byly ve většině případů velmi podobné. Ovšem při použití vhodných parametrů, tedy módu 2 a povolení vyšších vzdáleností mezi uzly či mezi uzly pro optimalizaci trasy, vychází lépe právě algoritmus RRT*FN. Proto bude tento algoritmus využit pro simulaci letu modelu UAV v další kapitole.

Prostor 1



Obr. 32 Vývoj RRT – 3D, prostor 2



Obr. 33 Vývoj RRT* - 3D, prostor 2

Sada 1	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	55.6559	52.1007	52.8687	52.7695	50.9060	50.5184
	106.83%	100%	101.48%	101.29%	97.70%	96.96%
Čas [s]	0.9092	1.4644	1.2559	1.3827	1.4756	1.5267
	62.09%	100%	85.76%	94.42%	100.76%	104.25%

Tab. 6 Porovnání RRT, RRT*, RRT*FN – prostor 2, sada 1

Sada 2	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	45.5008	38.9544	43.2325	38.9811	39.4231	38.0564
	116.81%	100%	95.01%	100.07%	101.20%	97.69%
Čas [s]	0.2645	0.4563	0.3743	0.3677	0.3675	0.4692
	57.97%	100%	82.03%	80.58%	80.54%	102.83%

Tab. 7 Porovnání RRT, RRT*, RRT*FN – prostor 2, sada 2

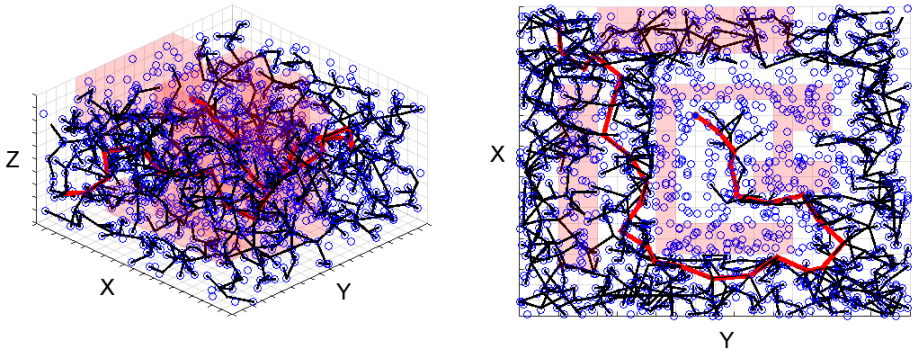
Sada 3	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	57.3988	46.1881	51.2760	52.7098	49.1245	51.4127
	124.27%	100%	110.01%	114.12%	106.36%	113.11%
Čas [s]	5.9156	9.2906	7.7363	8.8308	11.1993	10.5190
	63.69%	100%	83.27%	95.05%	120.54%	113.22%

Tab. 8 Porovnání RRT, RRT*, RRT*FN – prostor 2, sada 3

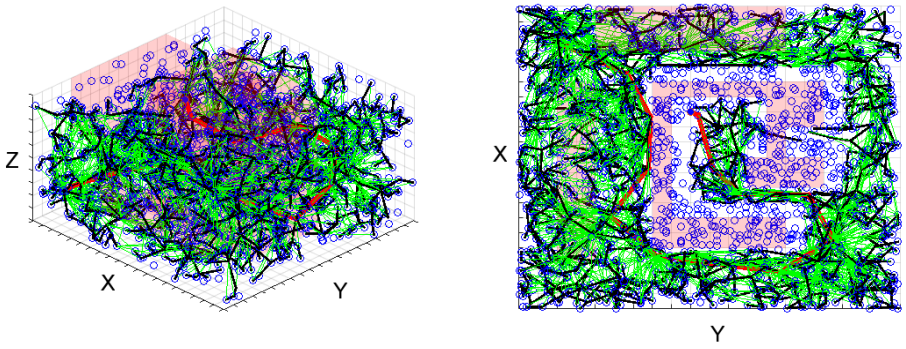
Sada 4	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	63.2382	49.9547	46.9837	48.0143	46.8475	45.2460
	126.59%	100%	94.05%	96.12%	93.78%	90.57%
Čas [s]	5.9595	11.2049	11.3855	13.4410	15.2311	15.9961
	53.19%	100%	101.37%	119.96%	135.93%	142.76%

Tab. 9 Porovnání RRT, RRT*, RRT*FN – prostor 2, sada 4

Prostor 2



Obr. 34 Vývoj RRT - 3D, prostor 3



Obr. 35 Vývoj RRT* - 3D, prostor 3

Sada 1	**RRT**	**RRT***	**RRT*FN**			
			400	500	700	1000
Trajektorie [m]	59.5080	45.8919	51.4798	49.9848	48.6896	47.2185
	129.67%	100%	112.18%	108.92%	106.10%	102.89%
Čas [s]	3.6096	12.0180	4.9627	6.7831	11.4335	12.0438
	30.03%	100%	41.29%	56.44%	95.14%	100.22%

Tab. 10 Porovnání RRT, RRT*, RRT*FN – prostor 3, sada 1

Sada 2	**RRT**	**RRT***	**RRT*FN**			
			400	500	700	1000
Trajektorie [m]	62.5122	50.4312	48.2073	46.6169	48.8110	45.6808
	123.96%	100%	95.59%	92.44%	96.79%	90.58%
Čas [s]	0.9015	3.9275	2.5689	3.8569	7.0857	12.8702
	22.95%	100%	65.41%	98.20%	180.40%	327.69%

Tab. 11 Porovnání RRT, RRT*, RRT*FN – prostor 3, sada 2

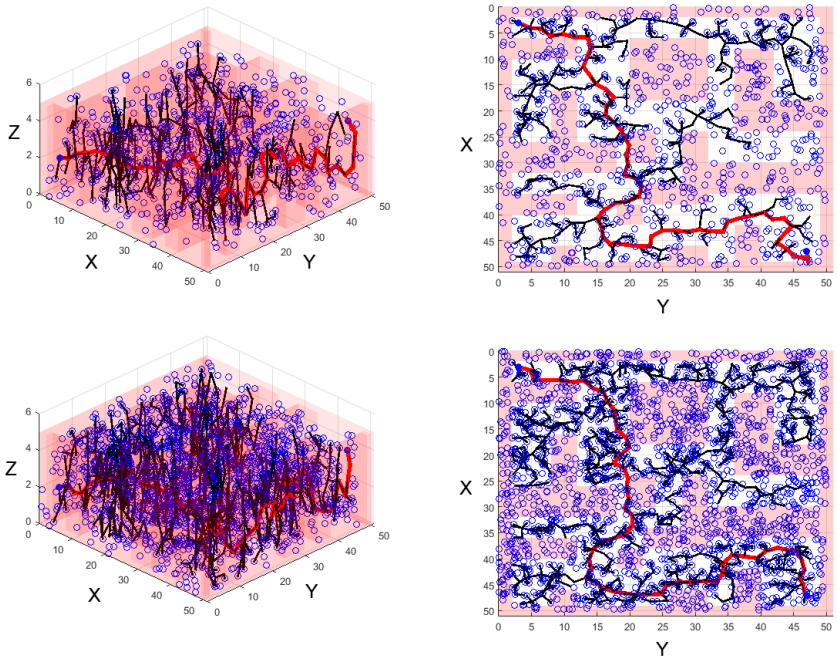
Sada 3	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	61.4200	48.9517	49.2039	46.7572	44.4831	51.2812
	125.47%	100%	100.52%	95.52%	90.87%	104.76%
Čas [s]	9.6523	16.2500	11.6952	13.0641	16.5327	19.7568
	59.40%	100%	71.97%	80.39%	101.74%	121.58%

Tab. 12 Porovnání RRT, RRT*, RRT*FN – prostor 3, sada 3

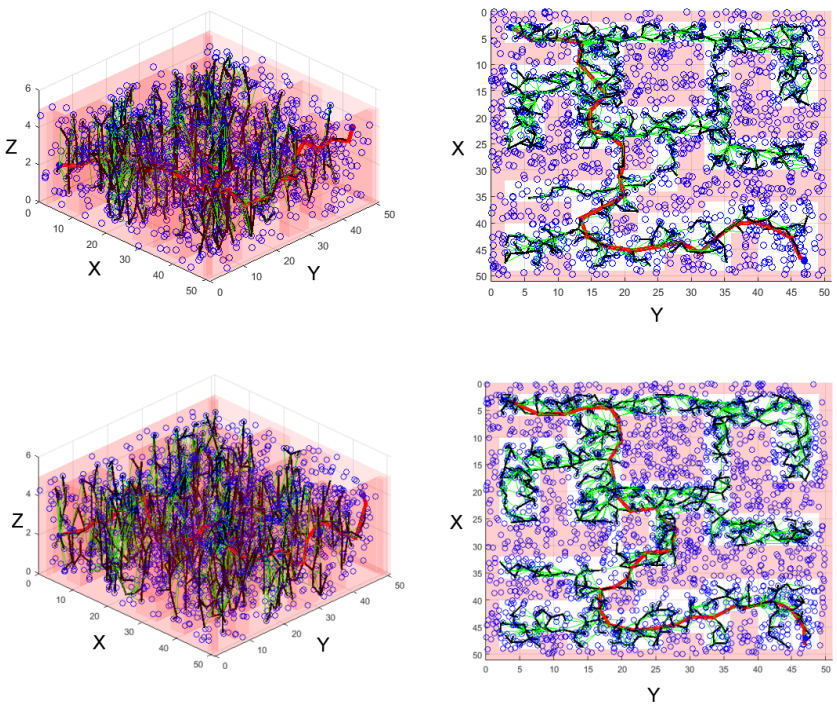
Sada 4	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	60.3361	45.0949	46.6022	46.8474	42.3405	46.1419
	133.80%	100%	103.34%	103.89%	93.89%	102.32%
Čas [s]	10.1621	38.6884	21.4849	19.3696	24.9164	29.3715
	26.27%	100%	55.53%	50.07%	64.40%	75.92%

Tab. 13 Porovnání RRT, RRT*, RRT*FN – prostor 3, sada 4

Prostor 3



Obr. 36 Vývoj RRT - 3D, prostor 4



Obr. 37 Vývoj RRT* - 3D, prostor 4

Sada 1	**RRT**	**RRT***	**RRT*FN**			
			400	500	700	1000
Trajektorie [m]	118.6797	105.3060	107.3563	110.8363	109.3914	110.1904
	112.70%	100%	90.46%	105.26%	103.88%	104.64%
Čas [s]	4.2010	7.7727	5.9929	6.5799	5.9473	4.9213
	54.05%	100%	77.10%	84.65%	76.52%	63.32%

Tab. 14 Porovnání RRT, RRT*, RRT*FN – prostor 4, sada 1

Sada 2	**RRT**	**RRT***	**RRT*FN**			
			400	500	700	1000
Trajektorie [m]	115.9104	103.9626	102.6060	104.9756	105.5307	108.3151
	114.92%	100%	98.70%	100.97%	101.51%	104,19%
Čas [s]	0.8259	1.8121	2.2536	1.6624	1.8859	1.8622
	45.58%	100%	124.37%	91.74%	104.73%	102.76%

Tab. 15 Porovnání RRT, RRT*, RRT*FN – prostor 4, sada 2

Sada 3	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	117.0083	98.6030	106.7099	105.6800	111.3333	107.8640
	118.66%	100%	108.22%	107.18%	112.91%	109.39%
Čas [s]	6.7008	10.6041	7.8659	8.2446	10.9909	10.5702
	63.19%	100%	74.18%	77.75%	103.65%	99.68%

Tab. 16 Porovnání RRT, RRT*, RRT*FN – prostor 4, sada 3

Sada 4	RRT	RRT*	RRT*FN			
			400	500	700	1000
Trajektorie [m]	118.8788	106.7668	99.7307	107.4164	105.3917	103.6720
	111.34%	100%	93.41%	100.61%	98.71%	97.10%
Čas [s]	6.2319	11.7566	9.1322	10.0937	11.8910	11.5870
	53.00%	100%	77.68%	85.86%	101.14%	95.56%

Tab. 17 Porovnání RRT, RRT*, RRT*FN – prostor 4, sada 4

4.2. Simulace letu model UAV pomocí získané trajektorie

Pro otestování vygenerovaných trajektorií a průletu této trajektorie pomocí modelu UAV, bylo užito toolboxu *Robotics toolbox for matlab*[8] pro vývojové prostředí MATLAB a SIMULINK. Simulace proběhly pro nejsložitější prostředí, tedy pro prostředí 3, a použité trajektorie byly vygenerovány algoritmy Theta* pro diskrétní prostor a algoritmem RRT*FN pro spojitý prostor.

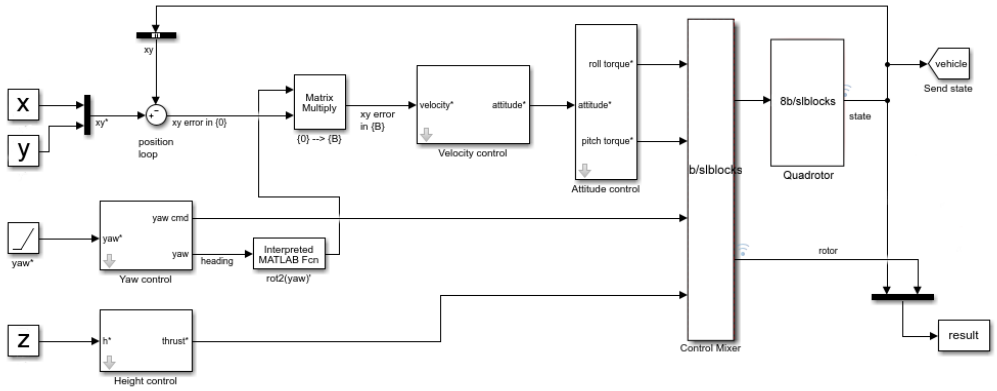
Theta* byla vybrána z důvodu reálně vypadajícího vzhledu trajektorie, která by neměla být složitá pro UAV. RRT*FN algoritmus byl vybrán z důvodu doporučení v předchozí kapitole 4.1. s optimálním nastavením parametrů pro prostředí 3. Nastavenými parametry byly:

- Maximální vzdálenost mezi uzly 4 m
- Maximální vzdálenost pro optimalizaci trajektorie 6 m
- Počet vygenerovaných uzlů 2000
- Maximální povolený počet uzlů ve stromě 400
- Mód 2

[8] (volně stažitelný toolbox z http://petercorke.com/wordpress/toolboxes/robotics-toolbox)

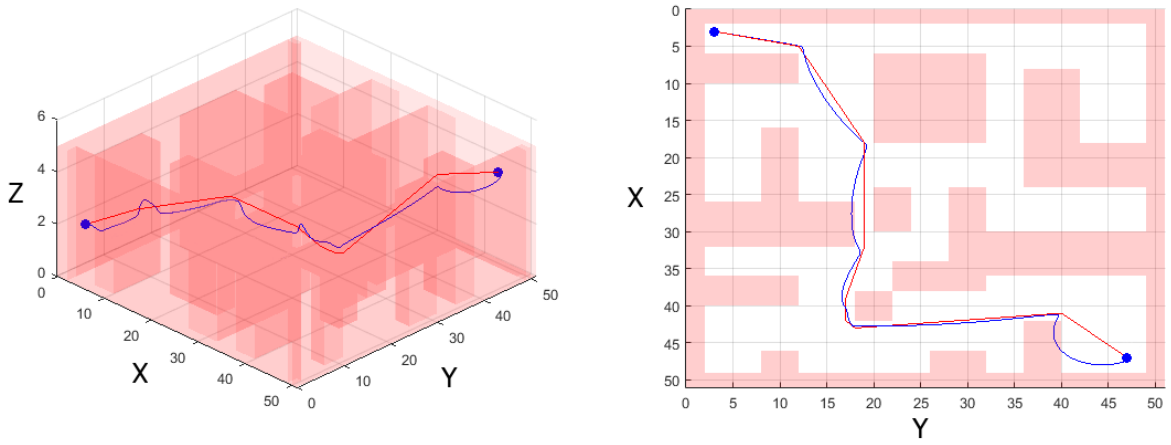
Simulace letu probíhala dosazením jednotlivých vrcholů, respektive uzlů pro spojitý konfigurační prostor, trajektorie na vstup jako souřadnice letu x, y, z. Počáteční pozice byla nastavena přímo na startovní vrchol, respektive uzel, kterým byl bod o souřadnicích [3, 3, 2] a trajektorie byla opět generována do cílového bodu [47, 47, 4]. Změna dalšího *wayipontu* trajektorie byla prováděna v případě splnění normy rozdílu poloh mezi aktuální pozicí modelu UAV a požadovaným *waypointem*, která byla stanovena na 1/5 metru.

Model použitý pro simulaci letu:

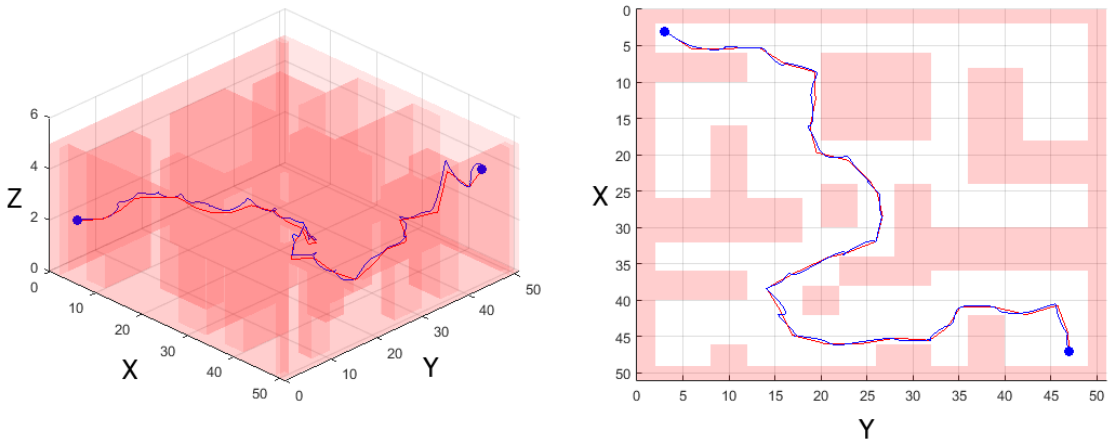


Obr. 38 Schéma modelu UAV v programu SIMULINK

Jak je možné vidět na obrázcích 37 a 38, vygenerovaná trajektorie a trajektorie uskutečněná letem UAV se příliš neshodují. To je způsobeno tím, že konfigurační prostor a algoritmy pro generování trajektorie nebyly uzpůsobené kinematice, dynamice a objemu modelu UAV. Ani výsledná trajektorie nebyla po vygenerování nijak optimalizována. V případě této simulace byl užíván jen obecný hmotný bod. Problematika této optimalizace trajektorie a konfiguračního prostoru za účelem přesnějšího průletu modelu UAV se započítáním kinematiky, dynamiky a objemu modelu, bude tématem další práce.



Obr. 39 Simulace letu modelu UAV v diskrétním konfiguračním prostoru



Obr. 40 Simulace letu modelu UAV ve spojitém konfiguračním prostoru

Kapitola 5

Závěr

V této závěrečné práci byl nejprve představen UAV, konkrétně pak kvadrátorová helikoptéra. Vysvětleny byly principy kinematiky, dynamiky a komunikace mezi jednotlivými bloky či regulátory systému UAV.

Následně byly definovány algoritmy pro vyhledávání trajektorie v prostoru. Uvedeny byly příklady pro 2D prostor i pro 3D prostor. U všech zmíněných algoritmů byly představeny klady a zápory jim příslušné včetně pseudo kódů pro nástin algoritmu.

V praktické části byly vybrány algoritmy A*, Basic Theta* a Lazy Theta* pro diskrétní prostor a RRT, RRT*a RRT*FN pro spojitý prostor. Tyto algoritmy byly implementovány ve vývojovém prostředí MATLAB. Následně proběhly série testů pro různě komplikovaná prostředí a dle výsledků byly vyvozeny hodnocení jednotlivých algoritmů. Tato hodnocení jsou popsána v kapitole 4.

V poslední části této práce byly vygenerované trajektorie aplikovány pro let modelu kvadrátorové helikoptéry. Vybrané trajektorie byly vygenerovány algoritmy Basic Theta* a RRT*FN. Z důvodu nezakomponování kinematiky, dynamiky a objemu UAV do konfiguračního prostoru, je výsledná provedená trajektorie letu UAV v nesouladu s plánovanou trajektorií a let zasahuje do překážek v prostoru. Potřebná optimalizace a zakomponování těchto faktorů do konfiguračního prostoru budou předmětem další práce.

Literatura

[1] ELBANHAWI, Mohamed; SIMIC, Milan. Sampling-based robot motion planning: A review. *Ieee access*, 2014, 2: 56-77 [cit. 16.5.2019].

[2] GONZÁLEZ, David, et al. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 2015, 17.4: 1135-1145 [cit. 16.5.2019].

[3] YANG, Liang, et al. Survey of robot 3D path planning algorithms. *Journal of Control Science and Engineering*, 2016, 2016: 5 [cit. 8.2.2019].

[4] BOUČEK, Zdeněk. *Návrh řízení kvadrotorové helikoptéry*. Plzeň, 2015. Diplomová práce [cit. 12.3.2019].

[5] LUUKKONEN, Teppo. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 2011, 22 [cit. 27.5.2019].

[6] QUAN, Quan. *Introduction to multicopter design and control*. Singapore: Springer Singapore, 2017 [cit. 2.5.2019].

[7] WÁGNER, Petr. *Metoda plánování trajektorie robota v reálném čase*. Olomouc, 2014. Doktorská disertační práce. VŠB-TU Ostrava [cit. 12.5.2019].

[8] ČERNÝ, Jakub. *Základní grafové algoritmy*. České vysoké učení technické, 2013 [cit. 23.5.2019].

[9] NASH, Alex; KOENIG, Sven; TOVEY, Craig. Lazy Theta*: Any-angle path planning and path length analysis in 3D. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010 [cit. 16.10.2018].

[10] DIJKSTRA, Edsger W. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959, 1.1: 269-271 [cit. 27.9.2018].

[11] CORMEN, Thomas H., et al. *Introduction to algorithms*. MIT press, 2009, ISBN 0-262-03293-7. Sekce 24.3: Dijkstra's algorithm, pp.595–601 [cit. 8.11.2018].

[12] BELLMAN, Richard. On a routing problem. *Quarterly of applied mathematics*, 1958, 16.1: 87-90 [cit. 12.10.2018].

[13] BANNISTER, Michael J.; EPPSTEIN, David. Randomized speedup of the Bellman–Ford algorithm. In: *2012 Proceedings of the Ninth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. Society for Industrial and Applied Mathematics, 2012. p. 41-47 [cit. 12.10.2018].

[14] BOROUJENI, Zahra, et al. Flexible unit A-star trajectory planning for autonomous vehicles on structured road maps. In: *2017 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2017. p. 7-12 [cit. 2.12.2018].

[15] CORMEN, Thomas H., et al. *Introduction to algorithms*. MIT press, 2009 [cit. 22.8.2018].

[16] KOUBAA, Anis, et al. Introduction to Mobile Robot Path Planning. In: *Robot Path Planning and Cooperation*. Springer, Cham, 2018. p. 3-12 [cit. 23.8.2018].

[17] GRAJCIAR, Matej. Any-angle path-planing algorithms. 2012 [cit. 2.4.2019].

[18] DANIEL, Kenny, et al. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 2010, 39: 533-579 [cit. 16.5.2018].

[19] ROTH, Scott D. Ray casting for modeling solids. *Computer graphics and image processing*, 1982, 18.2: 109-144 [cit. 1.5.2019].

[20] FLANAGAN, Colin. *Bresenham Line-Drawing Algorithm*[online]. [cit. 8.3.2019]. Dostupné z: www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html.

[21] DEDU, Eugen. *Bresenham-Based Supercover Line Algorithm: Bresenham Modified Algorithm - ALL the Points* [online]. [cit. 8.3.2019]. Dostupné z: eugen.dedu.free.fr/projects/bresenham

[22] MARTINEZ ALANDES, CARLOS. A comparison among different sampling-based planning techniques. 2015 [cit. 5.3.2019].

[23] CHENG, Peng; SHEN, Zuojun; LA VALLE, S. RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of control sciences*, 2001, 11.3/4: 167-194 [cit. 2.2.2019].

[24] NOREEN, Iram; KHAN, Amna; HABIB, Zulfiqar. Optimal path planning using RRT* based approaches: a survey and future directions. *Int. J. Adv. Comput. Sci. Appl*, 2016, 7.11: 97-107 [cit. 24.3.2019].

[25] KARAMAN, Sertac, et al. Anytime motion planning using the RRT. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011. p. 1478-1483 [cit. 12.11.2018].

[26] QURESHI, Ahmed Hussain; AYAZ, Yasar. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 2015, 68: 1-11 [cit. 5.8.2018].

[27] SPANOGIANOPOULOS, Sotirios; SIRLANTZIS, Konstantinos. Non-holonomic path planning of car-like robot using RRT* FN. In: *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE, 2015. p. 53-57 [cit. 16.5.2018].