

Algoritmy pro plánování cesty a jejich implementace v Pythonu

Filip Nevřala

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Filip Nevřala
Osobní číslo: A22023
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Algoritmy pro plánování cesty a jejich implementace v Pythonu
Téma práce anglicky: Path Planning Algorithms and their Implementation in Python

Zásady pro vypracování

1. Seznamte se s různými přístupy k plánování cesty pro mobilní roboty.
2. Podrobně popište plánovací algoritmy A*, Dijkstra, RRT, RRT* a PRM.
3. Vytvořte implementaci plánovacích algoritmů v jazyku Python z bodu 2 s možností vytvářet překážky a hledáním optimální cesty.
4. Vytvořte aplikaci pro vizualizaci implementovaných plánovacích algoritmů vhodnou také jako výuková pomůcka.
5. Zhodnoťte vytvořené řešení.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SIEGWART, Roland; NOURBAKHSH, REZA, Illah a SCARAMUZZA, Davide. *Introduction to Autonomous Mobile Robots*. Second edition. MIT Press, 2011. ISBN 0262015358
2. LAVALLE, Steven M. *Planning Algorithms*. Cambridge University Press, 2006. ISBN 978-0521862059
3. LUTZ, Mark. *Learning Python. 4th ed.* Beijing O'Reilly, 2009. ISBN 978-0-596-15806-4
4. KLANČAR, Gregor; ZDEŠAR, Andrej; BLAŽIČ, Sašo a ŠKRJANC, Igor. *Wheeled mobile robotics*. Elsevier, 2017. ISBN 978-0-12-804204-5

Vedoucí bakalářské práce: **Ing. Jakub Novák, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací interaktivní aplikace pro vizualizaci a srovnání vybraných algoritmů plánování cesty mobilních robotů. V teoretické části jsou popsány přístupy k reprezentaci prostředí, holonomní a neholonomní modely pohybu a jednotlivé algoritmy plánování cesty a její úpravy.

Klíčová slova: algoritmy plánování trasy, mobilní roboti, Python

ABSTRACT

This bachelor thesis deals with the design and implementation of an interactive application for the visualization and comparison of selected path planning algorithms for mobile robots. In the theoretical part, approaches to environment representation, holonomic and non-holonomic motion models, and individual path planning algorithms and their modifications are described.

Keywords: route planning algorithms, mobile robots, Python

Chtěl bych poděkovat panu Ing. Jakubovi Novákovi, Ph.D. za vedení práce a cenné rady, trpělivost a čas, který mi během práce věnoval.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 PŘÍSTUPY K PLÁNOVÁNÍ CESTY PRO MOBILNÍ ROBOTY	13
1.1 HOLONOMNÍ A NEHOLONOMNÍ POHYB.....	13
1.2 PLÁNOVÁNÍ CESTY A REAKCE.....	13
1.3 REPREZENTACE PROSTŘEDÍ	14
1.4 VYHÝBÁNÍ SE PŘEKÁŽKÁM	15
1.4.1 Vektorové histogramy	15
1.4.2 Bubble band	15
1.4.3 Curvature-velocity (CVM) a Lane-curvature metoda (LCM)	15
1.4.4 Dynamic Window Approach (DWA)	16
2 PLÁNOVACÍ ALGORITMY	17
2.1 BUG ALGORITMY	17
2.1.1 Bug0	17
2.1.2 Bug1	17
2.1.3 Bug2	18
2.2 ALGORITMY PRO TVORBU GRAFU	19
2.2.1 Graf viditelnosti	19
2.2.2 Voroného diagram.....	20
2.2.3 PRM algoritmus	22
2.3 ALGORITMY PRO HLEDÁNÍ CESTY	22
2.3.1 Breadth First Search (BFS)	22
2.3.2 Depth First Search (DFS).....	23
2.3.3 Dijkstrův algoritmus.....	24
2.3.4 A* algoritmus.....	25
2.3.5 RRT algoritmus.....	26
2.3.6 RRT* algoritmus	27
3 ÚPRAVA NALEZENÝCH CEST	28
3.1 VYHLAZENÍ CESTY	28
3.2 ÚPRAVA POMOCÍ POLYNOMICKÝCH KŘIVEK	28
3.3 ÚPRAVA POMOCÍ SPLINE KŘIVEK	29
II PRAKTICKÁ ČÁST	30
4 IMPLEMENTACE ALGORITMŮ V JAZYKU PYTHON	31
4.1 PYTHON.....	31
4.2 GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ (GUI)	31
4.2.1 PyQt6.....	31
4.2.2 Třída ScalableGraphicsView	32
4.2.3 Třída PolygonScene	32

4.2.4	Použité widgety pro GUI.....	32
4.3	POMOCNÉ FUNKCE V GUI	32
4.3.1	Uložení mapy	33
4.3.2	Načtení mapy.....	33
4.3.3	Vytvoření snímku scény	33
4.4	DETEKCE KOLIZÍ.....	34
4.5	PLÁNOVACÍ ALGORITMY.....	34
4.5.1	Grid-based algoritmy.....	35
4.5.2	Algoritmy založené na náhodném vzorkování	36
4.5.3	PRM algoritmus	38
4.6	VYHLAZENÍ CESTY	39
5	POROVNÁNÍ ALGORITMŮ	41
5.1	PRM PARAMETRY	41
5.2	RRT PARAMETRY	42
	ZÁVĚR.....	43
	SEZNAM POUŽITÉ LITERATURY	45
	SEZNAM OBRÁZKŮ	47
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	48

ÚVOD

Plánování cesty je klíčovou oblastí v robotice a autonomních systémech, umožňující efektivní navigaci v prostředí s překážkami. Zásadní roli tento aspekt hraje zejména u samořiditelných vozidel, kde je nutné bezpečně a spolehlivě procházet komplikovanými dopravními situacemi i neustále se měnícími podmínkami na silnicích. Proto je nezbytný vývoj pokročilých algoritmů pro plánování cesty k dosažení vysoké úrovně autonomie a bezpečnosti.

Servisní robotika je také důležitým směrem, kde plánování cesty umožňuje robotům efektivně plnit úkoly v různých prostředích od domácností po průmyslové provozy. Servisní roboti často pracují v dynamicky se měnícím prostředí, kde je nutno reagovat na neočekávané překážky a přizpůsobovat trasu aktuálním podmínkám. Efektivní plánování cesty tak zvyšuje jejich funkčnost a přispívá k lepšímu využití v reálných aplikacích.

Mobilní roboti, včetně autonomních vozidel i servisních robotů, jsou v současnosti předmětem intenzivního výzkumu a vývoje. Cílem praktické části této práce je vytvořit interaktivní aplikaci, která poslouží k vizualizaci a porovnávání různých plánovacích metod. Tato aplikace pomůže k lepšímu porozumění chování jednotlivých algoritmů a jejich účinnosti a současně nabídne uživatelům možnost experimentovat s různými strategiemi navigace v prostředí s překážkami.

Plánovací algoritmy se obvykle učí pomocí textových popisů a pseudokódu, což je sice přesné, ale často komplikované na pochopení jejich dynamiky. Vizualizace a interaktivní práce s algoritmem studentům umožňuje lépe pochopit jednotlivé kroky a jejich vliv na výsledky. Implementace ve výukové aplikaci spojuje grafickou vizualizaci s možností ovlivnit parametry běhu, čímž nabízí efektivní výukový nástroj, který podporuje jak intuitivní pochopení, tak hlubší rozbor chování plánovacích algoritmů.

I TEORETICKÁ ČÁST

1 PŘÍSTUPY K PLÁNOVÁNÍ CESTY PRO MOBILNÍ ROBOTY

Mobilní roboti potřebují umět efektivně naplánovat cestu do cíle s vyhýbáním se překážkám a přizpůsobováním se na proměnlivé prostředí. Plánování cesty je klíčovou schopností, která propojuje strategické rozhodování s okamžitými reakcemi na aktuální data ze senzorů. Vzhledem k různorodosti prostředí a nárokům na mobilitu existuje spousta metod, které mají různé přístupy k plánování cesty.

1.1 Holonomní a neholonomní pohyb

V mnoha aplikacích mobilních robotů shledáme, že jejich pohyb není zcela volný, avšak podléhá konkrétním kinematickým limitům. Tyto limity rozdělujeme na holonomní, které ovlivňují pouze polohové souřadnice robota, a neholonomní, které navíc omezují i rychlosti a směry jeho pohybu.

Holonomní robot může v každém okamžiku upravit svoji konfiguraci bez dalších omezení. Stačí tedy nalézt libovolnou cestu, která zůstává ve volném konfiguračním prostoru, a robot ji dokáže přesně vykonat. Prakticky to znamená, že robot dokáže změnit svou rychlost nebo směr jízdy libovolně ve všech osách, aniž by musel nejdříve zaujmout nějakou speciální polohu. Tento přístup značně zjednodušuje samotné plánování cesty. [4]

Naopak neholonomní roboti čelí dalším omezením, která se vztahují k jejich možné rychlosti a orientaci pohybu. Typickým příkladem je například automobil, robot s diferenciálním podvozkem nebo vozidlo s Ackermannovým řízením, které musí nejdříve upravit své natočení, aby mohlo jet přímo vpřed či vzad. Plánování trasy pro neholonomní roboty proto musí zohlednit jejich kinematický model, tedy omezení na otáčení, maximální poloměr zatáčky nebo poměr rychlostí kol. Cestu tak navrhujeme nikoliv jako pouhou křivku v prostoru, ale jako posloupnost příkazů, která tyto pohybové limity respektuje. [4]

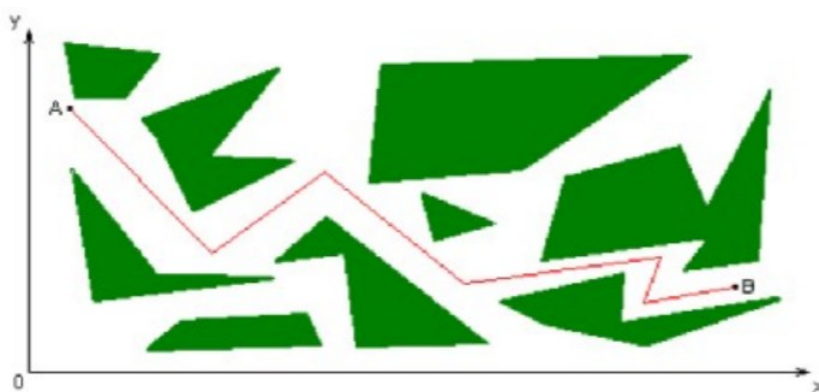
1.2 Plánování cesty a reakce

Plánování a reakce jsou často považovány za protikladné přístupy. U mobilních robotů ale tyto přístupy spolupracují, čímž vzniká silný navigační systém. Plánování je strategická část, kde robot určí optimální cestu k cíli podle dostupných informací a mapy prostředí. Reaktivita na druhé straně zahrnuje taktickou část, kde robot v reálném čase mění cestu na základě údajů ze senzorů. [1]

1.3 Reprezentace prostředí

Pro správné plánování je důležité definovat prostředí, ve kterém se robot pohybuje. Existují dva základní přístupy a to spojitý, nebo diskrétní.

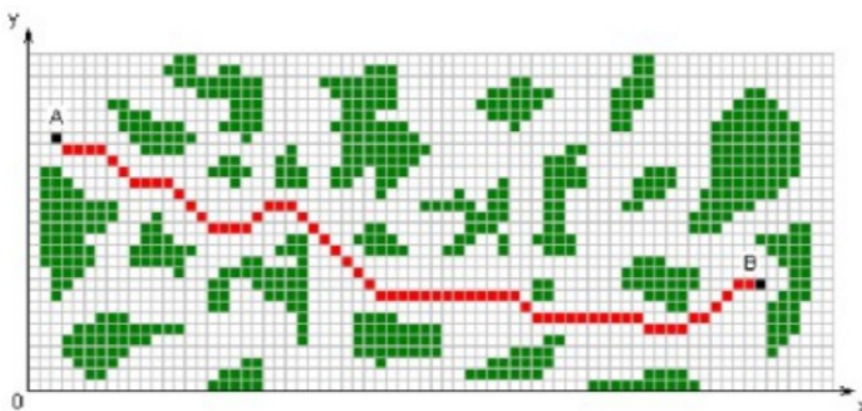
Spojité prostředí je takové, které není definováno mřížkou. Znamená to, že mezi dvěma body je v podstatě nekonečno bodů. Umožňuje tedy velmi detailní plánování cesty pomocí geometrických funkcí, což je vhodné pro analytické metody plánování, ale může být výpočetně náročné.



Obrázek 1 Spojité prostředí

(Zdroj: [5])

Diskrétní prostředí je definováno mřížkou. Je proto omezené, což zjednodušuje výpočty a implementaci algoritmů, které používají mřížku, jako je např. Dijkstrův algoritmus nebo A*. Nevýhodou je ovšem zvýšení nepřesnosti pohybu a optimální cesty v závislosti na velikosti mřížky.



Obrázek 2 Diskrétní prostředí

(Zdroj: [5])

Konfigurační prostor je množina všech přípustných stavů, ve kterých se robot může vyskytovat. Do něj pak jsou mapovány překážky a množina, kde se robot může volně pohybovat se pak nazývá volný konfigurační prostor.

1.4 Vyhýbání se překážkám

Překážky se vyskytují jako objekty o různých velikostech a tvarů. Jejich popis může být opět buď spojitý, nebo diskrétní. Pokud že překážky nejsou předem známé, nebo se dynamicky mění, robot musí upravovat cestu podle údajů ze senzorů. Pohyb robota je pak kombinace těchto dat s pozicí cíle a vzdálenosti k němu. Mezi nejjednodušší techniky patří Bug algoritmy, které využívají pouze aktuální hodnoty senzorů a přibližný směr k cíli. Pokročilejší algoritmy pak zohledňují nejen aktuální měření, ale také historii ze senzorů, kinematické modely, nebo i dynamiku pohybu. [1]

1.4.1 Vektorové histogramy

Vektorové histogramy (VFH) vytváří v okolí robota lokální mapu v podobě mřížky s celkovou pravděpodobností překážek. Ta je pak převedena do histogramu. Poté jsou hledány úseky s dostatečným volným prostorem pro pohyb robota a podle jednoduché vážené funkce je vybrán nejvhodnější směr s nejnižší cenou. VFH+ navíc zohledňuje poloměr otáčení robota pro filtrování fyzicky nerealizovatelných směrů. [1]

1.4.2 Bubble band

Na předem naplánované trase je vytvářen pás spojovaných bublin reprezentující maximální volný prostor. Mezi bublinami působí vnitřní síly, které napínají bubliny a minimalizují tak mezeru mezi středy bublin. Během jízdy se pak robot snaží držet uprostřed těchto bublin. Systém může reagovat na změny v reálném čase a bubliny se automaticky mohou zmenšovat při detekci nové překážky a robot se tak bude stále držet co nejdále od překážek. [1]

1.4.3 Curvature-velocity (CVM) a Lane-curvature metoda (LCM)

CVM a LCM pracují ve dvourozměrném prostoru parametrů rychlosti v a úhlové rychlosti ω (otáčení robota). Metody tedy vyfiltrují pouze ty kombinace, které robot zvládne fyzicky provést a zároveň nevedou ke kolizi s překážkami. CVM pak pomocí jednoduché vážené funkce zvolí nejlepší trasu. LCM pak místo jenom jednoho jízdního pruhu (lane) spočítá několik paralelně vedle sebe a vybere nejlepší kompromis mezi šířkou průjezdu a délkou trasy. [1]

1.4.4 Dynamic Window Approach (DWA)

DWA využívá dynamické okno, což je set všech možných rychlostí a rotací. Pomocí vážené funkce pak vybere nejoptimálnější s ohledem na kolize s překážkami. Global DWA doplňuje do vážené funkce hodnoty informace z předem známé globální mapy. Tím se kombinuje výhoda směřování k cíli pomocí znalosti globální mapy s reaktivní schopností vyhýbat se překážkám v reálném čase. [1]

2 PLÁNOVACÍ ALGORITMY

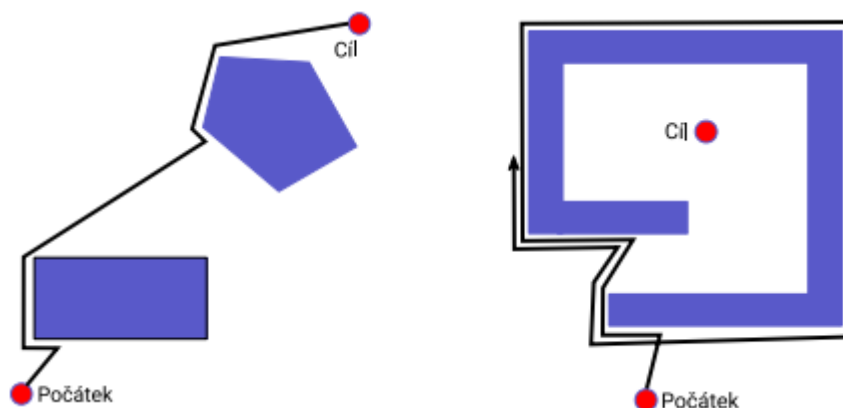
V této kapitole jsou popsány a porovnány základní plánovací algoritmy.

2.1 Bug algoritmy

Bug algoritmy jsou algoritmy, kde není předem známo prostředí, nebo se dynamicky mění. Robot pak musí upravovat svou trajektorii podle aktuálních senzorických dat. V literatuře se často rozlišují tři varianty: Bug0, Bug1 a Bug2, které se liší zejména tím, jak robot reaguje při narazení na překážku. [1]

2.1.1 Bug0

Bug0 je považován za nejjednodušší variantu, kdy se robot pohybuje přímo ke svému cíli, dokud nenarazí na překážku. Poté sleduje obrys překážky a pokusí se pokračovat přímo k cíli, jakmile to bude opět možné. Tento přístup je velmi jednoduchý, ale může vést k neefektivním cestám, nebo dokonce k zacyklení a nenalezení trasy. [1]



Obrázek 3 Ukázka Bug0 algoritmu včetně možnosti zacyklení

(zdroj: [4])

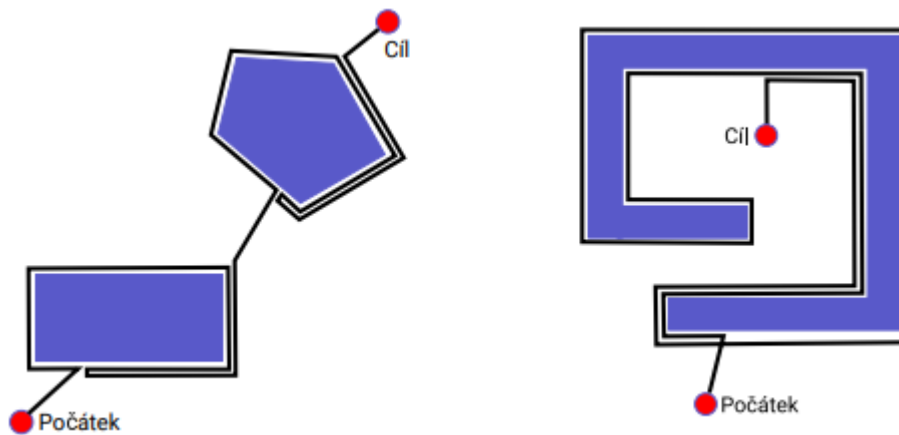
```
while (nedosažen cíl):  
    if (překážka přede mnou):  
        obejdi po obvodu překážky  
    else:  
        pohybuj se směrem k cíli
```

Pseudokód Bug0 algoritmu

2.1.2 Bug1

V algoritmu Bug1 po detekci překážky jí robot celou obejde po obvodu. Během toho sleduje vzdálenost k cíli a najde bod s nejmenší vzdáleností. Po dokončení obcházení celé překážky

se vrátí na tento nejefektivnější bod a pokračuje směrem ke svému cíli. Tento přístup zajišťuje, že robot na rozdíl od Bug0 algoritmu vždy dosáhne cíle za cenu delší cesty, protože robot musí vždy obejít celou překážku bez ohledu na možnost zkrácení cesty. [1]



Obrázek 4 Ukázka Bug1 algoritmu včetně prevence zacyklení

(zdroj: [4])

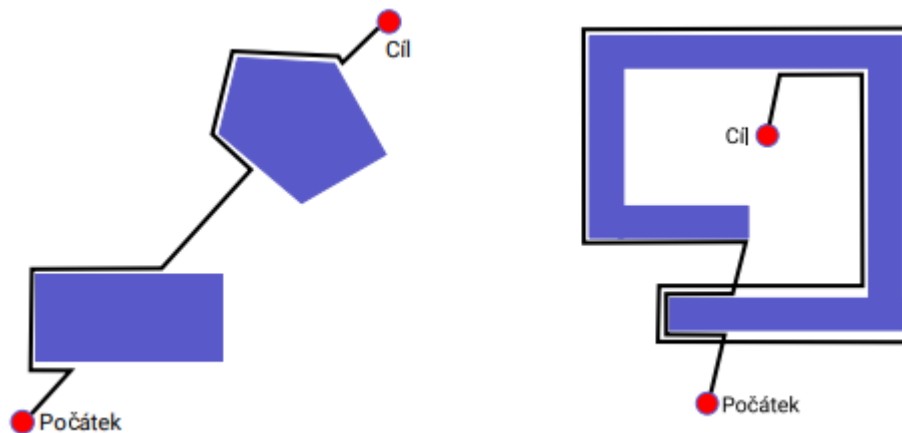
```
while (nedosažen cíl):
    if (překážka přede mnou):
        return_point = aktuální bod
        nearest_point = aktuální bod
        min_dist = vzdálenost z aktuálního bodu do cíle
        while (znovu nedosažen return_point):
            obejdi po obvodu překážky
            if (vzdálenost k cíli < min_dist):
                nearest_point = aktuální bod
                min_dist = vzdálenost z aktuálního bodu do cíle
        while (nedosažen nearest_point):
            obejdi po obvodu překážky
    else:
        pohybuj se směrem k cíli
```

Pseudokód Bug1 algoritmu

2.1.3 Bug2

Algoritmus Bug2 zefektivňuje obcházení překážek spojením rychlosti algoritmu Bug0 a zaručení nalezení trasy algoritmu Bug1. Robot po narazení na překážku sleduje její obrys, ale opustí ho, jakmile se dostane zpět na přímou linii mezi výchozím bodem a cílem. Tento přístup často vede k významnému zkrácení celkové trasy, protože robot minimalizuje čas strávený podél obrysu překážky. I přesto však v některých komplikovaných situacích může

dojít k suboptimální trase, zvláště pokud geometrie prostředí komplikuje návrat na přímou cestu. [1]



Obrázek 5 Ukázka Bug2 algoritmu včetně možné suboptimální trasy

(zdroj: [4])

```
while (nedosažen cíl):
    if (překážka přede mnou):
        distance = vzdálenost z aktuálního bodu do cíle
        while True:
            obejdi po obvodu překážky
            if (robot na přímé linii z počátku do cíle and
                vzdálenost k cíli < distance):
                break
        else:
            obejdi po obvodu překážky
    else:
        pohybuji se směrem k cíli
```

Pseudokód Bug2 algoritmu

2.2 Algoritmy pro tvorbu grafu

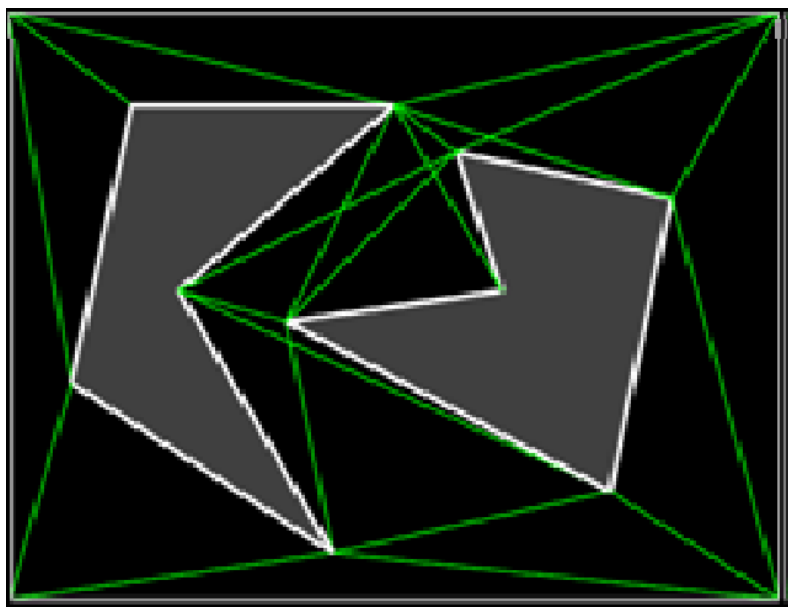
Tato skupina algoritmů je zaměřená na převod prostředí do grafové podoby. Prostor obsahuje polygonální překážky. Algoritmy poté generují uzly grafu a spojnice mezi nimi ve volném prostoru. Na takto vzniklém grafu je následně pomocí jiných algoritmů hledána nejkratší cesta.

2.2.1 Graf viditelnosti

Graf viditelnosti je metoda zobrazení volného prostoru, při které se spojí všechny vrcholy polygonálního prostoru, které se navzájem vidí, včetně počáteční a cílové pozice. Tyto

úsečky představují nejkratší možné cesty mezi vrcholy, protože nejkratší cesta mezi dvěma body je vždy úsečka. Tyto úsečky jsou pak použity jako hrany grafu.

Graf viditelnosti je oblíbený pro svou jednoduchou implementaci, zejména pokud je prostor modelován jako množina polygonů. Nicméně má své nevýhody. S větším počtem překážek roste i počet vrcholů a hran, což dělá metodu méně efektivní v hustě osídlených prostředích. Dále, nalezené cesty mají tendenci vést co nejblíže k překážkám, což snižuje bezpečnost pohybu robota a zvyšuje riziko kolize. Pro řešení tohoto problému se často překážky zvětší, než ve skutečnosti jsou, nebo je výsledná cesta po naplánování ještě upravena tak, aby byla bezpečnější. [9]



Obrázek 6 Graf viditelnosti

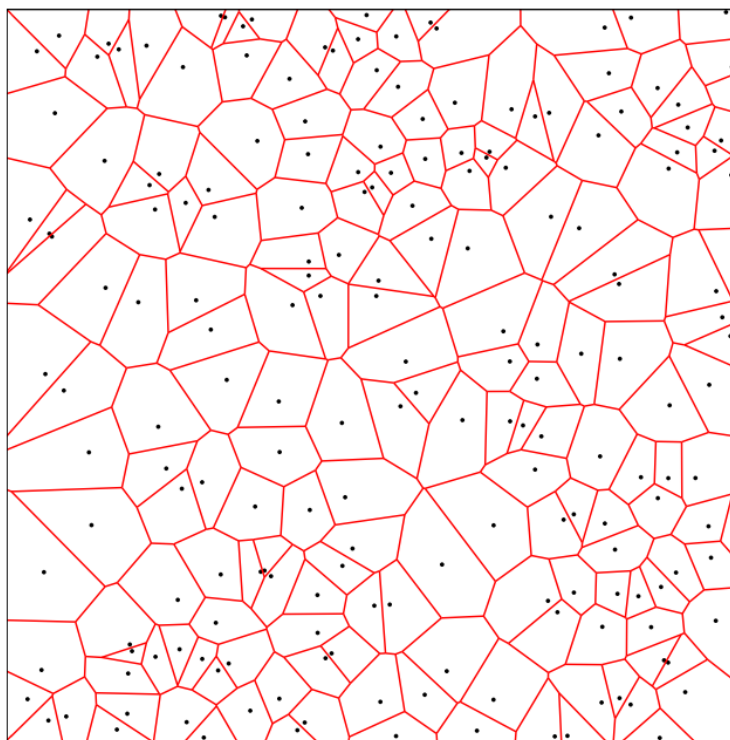
(Zdroj: [5])

2.2.2 Voroného diagram

Voroného diagram je metoda, která převádí volný prostor do sítě hran. Tyto hrany jsou definovány pomocí vzdáleností od překážek definovaných pomocí bodů. Pro každý bod ve volném prostoru se vypočítá jeho vzdálenost k nejbližší překážce a místa, kde jsou tyto vzdálenosti shodné, tvoří hrany diagramu. Hrany reprezentují oblasti s největším volným prostorem, tedy místa s maximální vzdáleností od všech překážek. [1]

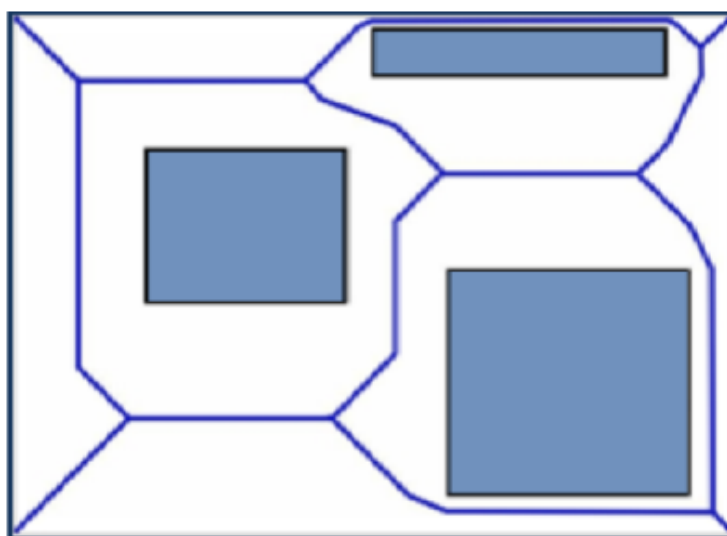
Hlavní výhodou je tato maximální vzdálenost od překážek. Díky tomu Voroného diagram poskytuje trasy, které udržují maximální odstup od překážek, a minimalizují tak riziko kolize. To je zvláště užitečné, protože při plánování trasy se robot často modeluje jako

bod s nulovou velikostí, takže tato metoda bere v potaz i velikost robota při plánování bezpečné trasy. To znamená, že i když výsledná cesta nemusí být neoptimálnější, je navržena tak, aby robot bezpečně projížděl kolem překážek, čímž se snižuje pravděpodobnost kolize.



Obrázek 7 Voroného diagram s náhodnými body

(Zdroj: [3])



Obrázek 8 Voroného diagram s překážkami

(Zdroj: [5])

2.2.3 PRM algoritmus

PRM je pravděpodobnostní algoritmus pro hledání cesty, který je obzvláště vhodný pro více dimenzní konfigurační prostor. Algoritmus funguje ve dvou krocích. V prvním kroku se sestaví graf z náhodně generovaných vrcholů v prostoru. Jako první se vytvoří definovaný počet uzlů v prostoru. Poté je každý uzel spojen se třemi nejbližšími sousedy s ohledem na kolize s překážkami. Ve druhém kroku se vyhledá nejkratší cesta ve grafu z počátku do cíle pomocí některého z algoritmů pro hledání cesty ve grafu, například Dijkstra, nebo A*. Tento přístup je nedeterministický a nezaručuje vždy nalezení cesty.

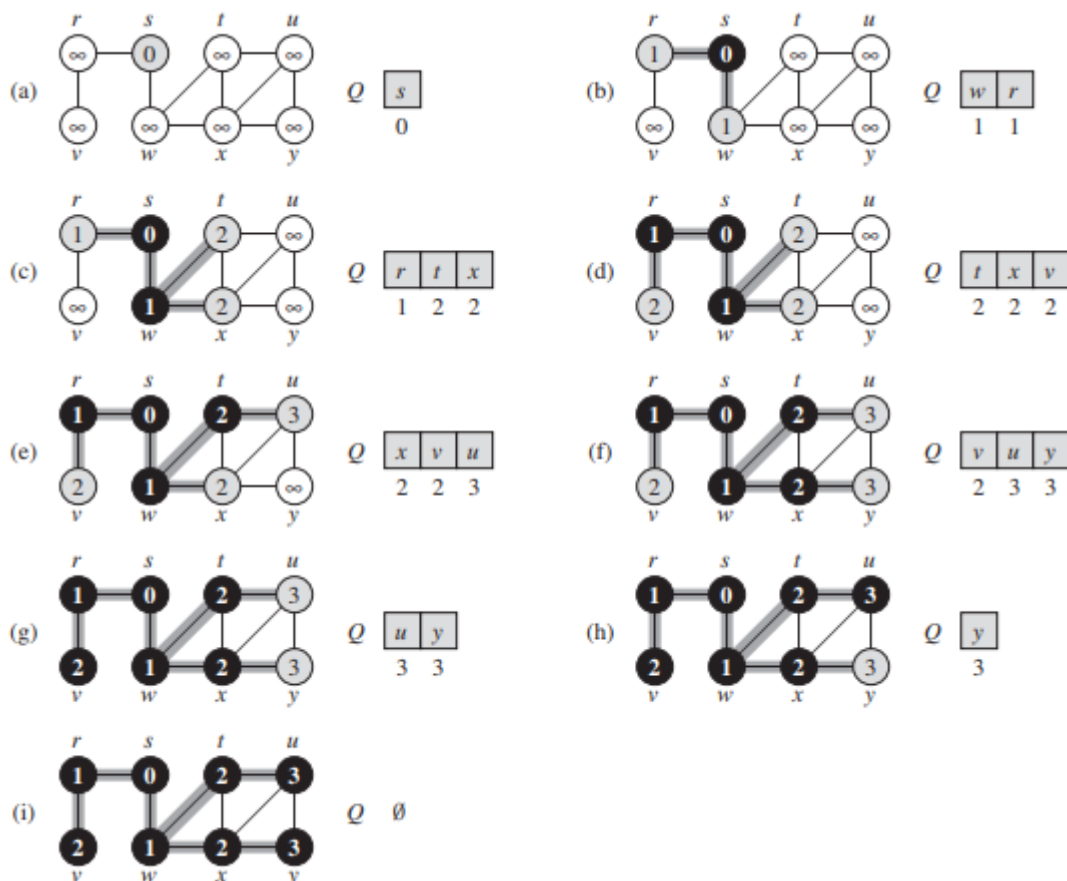
Jednou z výhod je to, že vygenerovanou roadmapu v prvním kroku lze použít pro opětovné hledání cesty, což z něj činí efektivní volbu pro statická prostředí. Metoda však nezaručuje nalezení cesty, protože závisí na náhodném vzorkování. [3][10]

2.3 Algoritmy pro hledání cesty

Tato skupina algoritmů se zabývá nalezením nejkratší cesty z počátečního bodu do cílového. Jednotlivé algoritmy používají různé strategie, od deterministických metod až po použití heuristiky či náhodnosti.

2.3.1 Breadth First Search (BFS)

Algoritmus Breadth First Search (prohledávání do šířky) systematicky prochází graf tak, že z počátečního uzlu nejdříve navštíví všechny jeho sousedy, kteří jsou vloženi do fronty, což je datová struktura typu FIFO (First In, First Out). Fronta zaručuje, že jsou uzly zpracovávány v pořadí, v jakém byly nalezeny – tedy podle rostoucí vzdálenosti od počátečního bodu. Jakmile jsou všichni sousedé zpracováni, algoritmus pokračuje prohledáváním dalších úrovní, kde jsou opět postupně vkládáni do fronty jejich sousední uzly. Metoda je jednoduchá na implementaci a efektivní při hledání nejkratší cesty v jednotně ohodnocených grafech. Nicméně, pokud se náklady na hrany liší, nemusí BFS vždy najít cestu s minimálními celkovými náklady. [1]

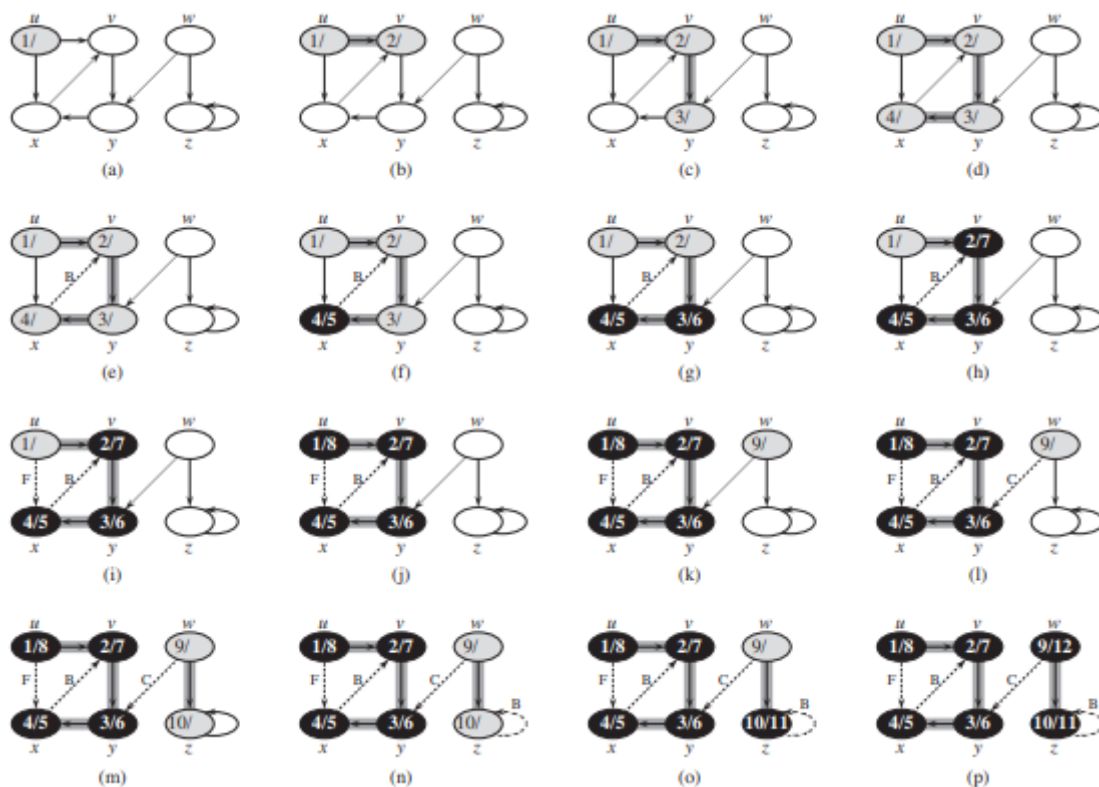


Obrázek 9 Průchod BFS algoritmu

(zdroj: [2])

2.3.2 Depth First Search (DFS)

Na rozdíl od BFS, algoritmus Depth First Search (prohledávání do hloubky) prohledává graf hluboko v jedné větvi, dokud nenarazí na uzel bez dosud nenavštívených sousedů. DFS zpravidla používá zásobník (LIFO – Last In, First Out). Tento přístup zajistí, že jako první je zpracován naposledy nalezený uzel, což umožňuje DFS co nejhlubší průzkum jedné větve, než je nutné se vrátit zpět a prozkoumat další větev. I když DFS nemusí zaručit nalezení neoptimalnější cesty, je užitečný v případech, kdy je prioritou nízká paměťová náročnost, která naroste jen na velikost nejdelší větve. DFS může narazit na redundantní prohledávání, proto je obvykle třeba implementovat kontrolu navštívených uzlů, aby se zabránilo cyklickému procházení. [1]



Obrázek 10 Průchod DFS algoritmu

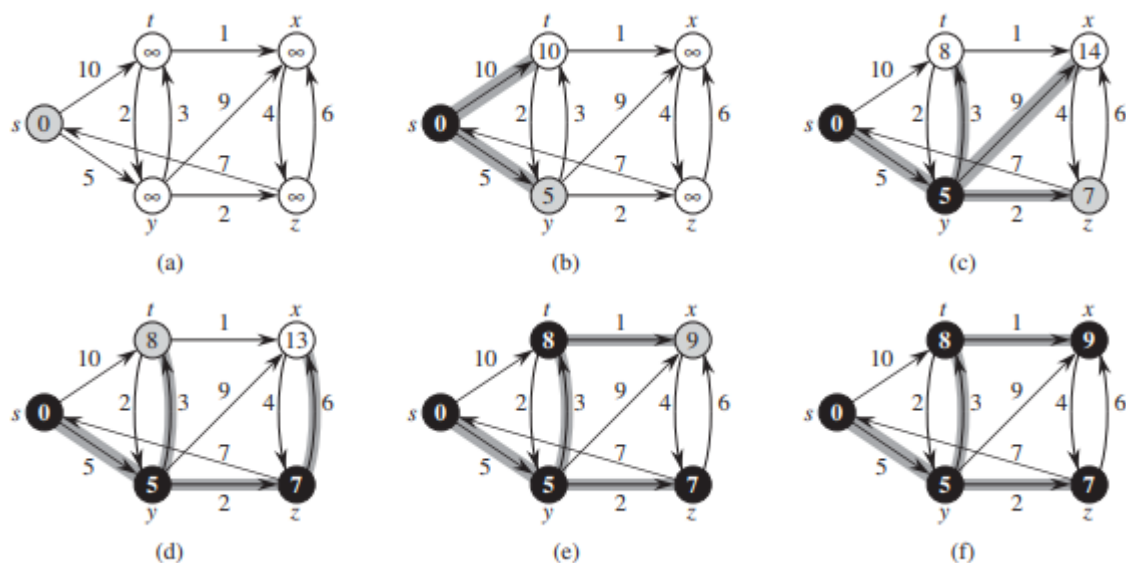
(zdroj: [2])

2.3.3 Dijkstrův algoritmus

Dijkstrův algoritmus je systematická metoda k nalezení nejkratších cest z jednoho počátečního vrcholu do všech ostatních vrcholů v ohodnocených grafech. Je speciálním případem dynamického programování. [3]

Každé hraně v ohodnoceném grafu je přiřazena pozitivní cena, která reprezentuje náklady cesty přes danou hranu. Každý vrchol pak má celkové náklady cesty, které reprezentují součet všech cen hran, kterými prochází nejkratší cesta od počátečního vrcholu.

Algoritmus v cyklu prochází všechny vrcholy a zpracovává je vždy od vrcholu s nejnižšími celkovými náklady. Toho lze dosáhnout pomocí prioritní fronty. Po zpracování vrcholu jsou uloženy jeho nejnižší celkové náklady z počátečního vrcholu. Po dokončení algoritmu získáme nejkratší cestu od počátečního vrcholu do všech ostatních vrcholů grafu.



Obrázek 11 Průchod Dijkstrova algoritmu

(zdroj: [2])

```

for each node:
    cost[n] = nekonečno
cost[start] = 0
vytvoř prioritní frontu Q a vlož do ní start s prioritou 0
while (fronta není prázdná):
    získej node s nejnižší cenou
    if (current node je cílový bod):
        získej finální cesta
    for each (sousedí current node):
        new_cost = cost[current] + edge_cost(current, neighbor)
        if new_cost < cost[neighbor]:
            cost[neighbor] = new_cost

```

Pseudokód Dijkstrova algoritmu

2.3.4 A* algoritmus

A* algoritmus je rozšířením Dijkstrova algoritmu, který zlepšuje efektivitu nalezení nejkratší cesty v ohodnocených grafech pomocí heuristického odhadu zbývajících nákladů cesty k cíli.

Algoritmus funguje podobně jako Dijkstrův, ale místo seřazení prioritní fronty pouze podle nejnižších celkových nákladů používá součet těchto skutečných nákladů cesty a heuristického odhadu vzdálenosti k cíli. Heuristika ovšem nikdy nesmí nadhodnocovat

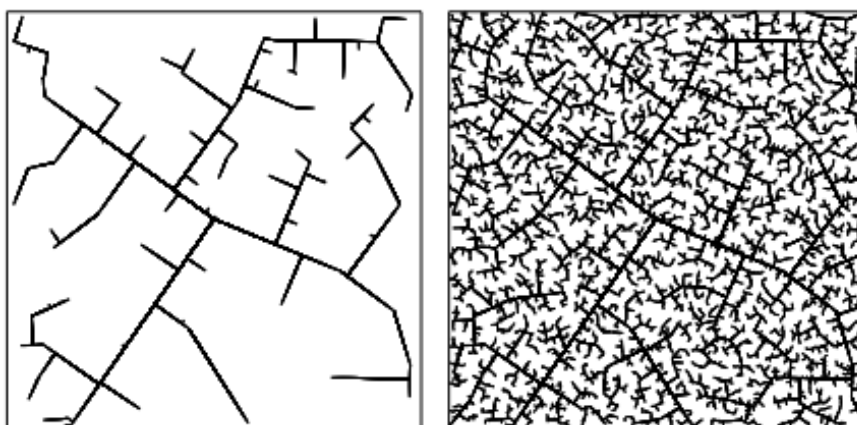
skutečnou cenu cesty. Obvykle se pro heuristiku používá Euklidovská vzdálenost mezi bodem a cílem.[3]

2.3.5 RRT algoritmus

RRT algoritmus je efektivní metoda pro hledání cesty v prostředí s překážkami bez nutnosti nastavování a ladění parametrů. Je to nedeterministický algoritmus.

Základní funkcí je inkrementální konstrukce vyhledávacího stromu. V prostoru je generován náhodný bod. K němu se nalezne nejbližší již existující uzel a z něho se vytvoří vektor směřující směrem k tomuto novému bodu. Strom se tak procentuálně přiblíží k novému bodu. Tímto způsobem pak strom postupně efektivně vyplňuje celý prostor. [3]

Vedle základního RRT existuje hned několik vylepšených verzí, které se snaží o rychlejší konvergenci či lepší kvalitu výsledné cesty. RRT-GoalBias při generování náhodného vzorku s určitou pravděpodobností záměrně volí rovnou jako náhodný bod cíl, čímž směřuje strom rychleji k cíli. RRT-GoalZoom potom k tomu přidává adaptivní přibližování, kde po úspěšném rozšíření ke kořeni stromu u cíle je omezena oblast náhodného vzorkování kolem cesty, čímž se upřesňují mezilehlé úseky. Bidirectional RRT pak současně roste dva stromy. Jeden roste od startu a druhý od cíle. Cílem je oba stromy spojit, což často zkrátí dobu hledání i délku výsledné cesty oproti jednosměrnému RRT. [5]



Obrázek 12 Průběh iterací RRT algoritmu

(zdroj: [3])

```
while (nenalezena cesta):  
    vyber náhodný bod  
    najdi nejbližší existující bod  
    přiblíž se k náhodnému bodu o krok  
    if (cesta k novému bodu je bez kolize):
```

přidej uzel do stromu a vytvoř k němu cestu

Pseudokód RRT algoritmu

2.3.6 RRT* algoritmus

RRT* algoritmus je rozšířením základního RRT algoritmu, který se snaží optimalizovat cestu.

Základní princip RRT* je stejný jako u RRT. Každému uzlu je však přiřazena jeho cena. Při zvolení náhodného bodu je cesta k němu přiblížena od již existujícího uzlu s nejnižší cenou místo nejbližšího uzlu. Dalším rozdílem je průběžné přepojování spojení mezi uzly pro optimalizaci cesty. V definované vzdálenosti od nového uzlu se naleznou všechny již existující uzly a zkontroluje se, zda by nebylo efektivnější jejich přepojení na nově vytvořený uzel. Díky těmto úpravám se strom postupně konverguje k optimálnímu řešení. [3]

```
while (nenalezena cesta):  
    vyber náhodný bod  
    najdi nejbližší uzel ve stromu  
    přiblíž se k náhodnému bodu o krok  
    if (cesta k novému bodu je bez kolize):  
        najdi sousedy v okolí nového bodu  
        vyber nejlepšího rodiče s nejnižší cenou  
        přidej nový bod do stromu s tímto rodičem  
        přepoj sousedy, pokud je cesta přes nový bod výhodnější  
    if (nový bod je blízko cíle):  
        přidej cílový bod a spoj s novým bodem  
        cesta_nalezena = ano
```

Pseudokód RRT* algoritmu

3 ÚPRAVA NALEZENÝCH CEST

Plánovací algoritmy často najdou cestu, která splňuje minimální požadavky, například nejkratší cestu. Nalezená cesta ale často nevyhovuje praktickým požadavkům reálného prostředí, kde je třeba zajistit plynulý a bezpečný pohyb robota. Proto je nutné aplikovat metody úpravy (vyhlazení) nalezené trasy, aby se zbavilo ostrých zatáček, zbytečných odboček a neplynulých úseků, které by mohly způsobovat nadměrné zatížení mechanických částí nebo zhoršení stability robota. V této kapitole se zaměříme na tři základní postupy úpravy cest: obecné vyhlazení, úpravu pomocí polynomických křivek a metody založené na splinech. Každý z těchto přístupů má své výhody a nevýhody, a jejich vhodnost závisí na specifických požadavcích dané aplikace.

3.1 Vyhlazení cesty

Výsledná cesta často není optimální, většinou kvůli náhodným prvkům plánovacích algoritmů. Cesta má většinou zbytečné body, které je možno odstranit pro efektivnější cestu. Pro odstranění těchto uzlů lze místo použití složitějších metod, které mohou být výpočetně náročné, použít jednoduchý algoritmus, který rychle eliminuje většinu nadbytečných bodů, a tak přispívá ke zlepšení praktické využitelnosti výsledné cesty. [6]

Algoritmus pracuje na jednoduchém principu, přidání počátečního bodu a následného procházení posloupnosti dalších bodů. Takto se najde nejdelší přímá úsečka, která neprochází žádnou překážkou. Tento nalezený uzel je přidán do výsledné cesty a všechny uzly mezi ním a počátečním mohou být přeskočeny. Proces je opakován použitím nalezeného uzlu jako nového počátečního, dokud není dosaženo cílového bodu. Získaná cesta je výrazně zjednodušená a zkrácená, ale pořád není ideální pro pohyb robota, protože pořád obsahuje na některých místech ostré zatáčky, které robot nemusí být schopen projet bez kolize.

3.2 Úprava pomocí polynomických křivek

Dalším přístupem k úpravě nalezených tras je využití polynomických křivek, jejichž cílem je aproximovat původní cestu hladkou a plynulou křivkou. Jednou z nejčastějších metod v této oblasti je interpolace pomocí Bézierovy křivky, pojmenované po francouzském inženýru Pierre Bézierovi. Tato metoda využívá sadu řídicích bodů, které definují tvar výsledné křivky. Na rozdíl od přímé lineární interpolace nemusí výsledná křivka nutně

procházet všemi původními body, ale její tvar odráží geometrické uspořádání těchto bodů a umožňuje dosáhnout hladkého přechodu mezi nimi. [11]

Hlavním cílem použití Bézierových křivek je vytvořit trajektorii, která eliminuje ostré zatačky a náhlé změny směru, a tím přispívá ke zvýšení bezpečnosti a komfortu pohybu robota. Využitím polynomických funkcí třetího stupně lze zajistit spojitost křivky a jejích derivací až do druhého řádu, což je zásadní pro plynulý a přirozený pohyb. Výsledná křivka je pouze aproximací původní cesty, v některých případech může dojít k odchylkám, kdy křivka nedodrží přesný průchod původními body. Pokud by se taková aproximovaná křivka dostala do kolize s překážkou, lze implementovat úpravu, která zvýší váhu daného řídicího bodu, čímž se křivka posune zpět blíže k bezpečné trase.

Celkově metoda pomocí Bézierových křivek poskytuje flexibilní nástroj pro úpravu a optimalizaci plánovaných tras. Umožňuje dosáhnout vysoce hladkých a plynulých trajektorií, které lze bezpečně využít v reálných robotických aplikacích, přičemž kompromis mezi plynulostí a optimálností cesty je dosažen pečlivým nastavením stupně polynomu a případnou korekcí řídicích bodů.

3.3 Úprava pomocí spline křivek

Metoda založená na spline křivkách představuje pokročilý přístup k vyhlazení nalezených tras, kdy se diskrétní body původní cesty aproximují spojitou křivkou. Klíčovým nástrojem jsou kubické spliny, definovány polynomy třetího stupně garantující hladkost nejen samotné křivky, ale i jejích derivací až do druhého řádu. Tímto způsobem vzniká trajektorie, která postupně prochází všemi klíčovými body původní cesty, ale zároveň eliminuje ostré zatačky a nepřirozené změny směru. Díky této metodě je dosaženo plynulého zakřivení trasy, což výrazně zlepšuje bezpečnost a pohyb robota.

Hlavní výhodou použití spline interpolace je schopnost vytvořit hladkou a přesnou trajektorii, která poskytuje dostatečnou vzdálenost od překážek. Při plánování trasy se robot často modeluje jako bod s nulovou velikostí, což znamená, že výsledná křivka musí brát v potaz i reálné omezení robota, například minimální poloměr otáčení podvozku. Spline metoda dovoluje flexibilní úpravu tvaru křivky posunem uzlů, což usnadňuje optimalizaci trasy podle specifických mechanických požadavků. Přestože implementace kubických spline křivek může být náročnější z hlediska výpočtů, zejména kvůli řešení soustavy rovnic zajišťujících spojitost derivací, poskytuje výsledek, který minimalizuje riziko kolize a zajišťuje hladký průběh pohybu, což je zásadní pro aplikace v autonomní robotice.

II PRAKTICKÁ ČÁST

4 IMPLEMENTACE ALGORITMŮ V JAZYKU PYTHON

Tato kapitola pojednává o praktické implementaci plánovacích algoritmů v jazyku Python a popisuje vývojové prostředí, použité knihovny a nástroje, které umožňují vytváření interaktivních a vizualizovaných aplikací vhodné pro výuku.

4.1 Python

Python je moderní, vysoce úroňový, interpretovaný programovací jazyk vyvinutý Guido van Rossumem. Vyznačuje se jednoduchou a čitelnou syntaxí, která odpovídá pseudokódu, a plně podporuje objektově orientované paradigma. Díky tomu je Python ideální pro rychlé prototypování a vývoj rozsáhlých systémů. [7]

Jeho popularitu podpořila bohatá standardní knihovna a rozsáhlá komunita uživatelů, což zajišťuje širokou dostupnost zdrojů a modulů pro různé účely od webového vývoje a datové analýzy až po vědecké výpočty a robotiku. Python je také multiplatformní, a snadno se integruje s dalšími jazyky, jako jsou C nebo C++, čímž se ještě více rozšiřuje jeho využitelnost. Celkově je Python licencován jako open source software pod licencí Python Software Foundation License (PSFL), což umožňuje volné využití a distribuci zdrojového kódu [7]. Jeho správu knihoven výrazně usnadňuje nástroj pip, který automatizuje instalaci a aktualizaci rozsáhlé nabídky dostupných balíčků.

4.2 Grafické uživatelské rozhraní (GUI)

GUI je klíčovou součástí implementace, která umožňuje interaktivní zadávání a vizualizaci dat potřebných pro plánování cesty. Pro tvorbu GUI je použita knihovna PyQt6, která poskytuje robustní nástroje pro vytváření moderních a multiplatformních aplikací. Díky pečlivě navrženému rozhraní lze snadno konfigurovat startovní a cílové body, vytvářet překážky a nastavovat parametry jednotlivých plánovacích algoritmů.

4.2.1 PyQt6

PyQt6 je python knihovna pro framework Qt verze 6, vyvinutá společností Riverbank Computing. Qt je rozsáhlá sada multiplatformních C++ knihoven, které poskytují vývoj uživatelských rozhraní. PyQt6 umožňuje využívat tyto funkce přímo v Pythonu, což usnadňuje vývoj aplikací s bohatým GUI. PyQt je licensováno pod GNU General Public License (GPL), což znamená, že aplikace vyvíjené s touto knihovnou musí být v souladu

s podmínkami GPL, pokud není zakoupena komerční licence. Tento licenční model podporuje open source vývoj a sdílení zdrojového kódu [8].

4.2.2 Třída *ScalableGraphicsView*

Třída *ScalableGraphicsView* dědí ze třídy *QGraphicsView*. Její hlavní funkcí je automaticky přizpůsobovat scénu aktuální velikosti okna. Při každé změně rozměrů okna se spouští metoda *fitInView* v rámci události *resizeEvent*, která zajišťuje, že obsah scény je vždy optimálně zobrazen bez ohledu na aktuální velikost okna. Tento přístup umožní dynamické škálování a zachování konzistentního vzhledu celé aplikace.

4.2.3 Třída *PolygonScene*

Třída *PolygonScene* je podtřídou *QGraphicsScene*, která umožňuje interaktivní vytváření a správu prvků v grafické scéně. Uživatel zde může přidat start, cíl a tvořit polygonální překážky pomocí levého tlačítka myši. Při zadávání překážek se jednotlivé body ukládají do seznamu a jsou vizuálně označovány malými elipsami. Po opětovném kliknutí na první bod s tolerancí se polygon uzavře a převede na *QGraphicsPolygonItem*, který je následně vykreslen do scény s definovanými barvami a čarami.

4.2.4 Použité widgety pro GUI

GUI využívá spoustu vestavěných widgetů, což zajišťuje intuitivní ovládání a interaktivitu celé aplikace. Pro výběr režimu (např. nastavení startu, cíle nebo tvorbu překážek) jsou použity *QRadioButton* a jejich seskupení pomocí *QButtonGroup*, což umožňuje snadné přepínání mezi jednotlivými režimy. Pro zadávání parametrů algoritmů jsou využity *QLineEdit* a *QSlider* widgety umístěné v rámci *QFormLayout*. Dále je implementován *QStackedWidget*, který umožňuje dynamicky měnit panel s parametry podle vybraného algoritmu. Pro regulaci rychlosti simulace je použit další *QSlider*, který zobrazuje aktuální hodnotu pomocí přidruženého *QLabel*. Společně s hlavním prostorem pro vizualizaci, implementovaným pomocí tříd *PolygonScene* a *ScalableGraphicsView*, vytváří tento soubor widgetů modulární a uživatelsky přívětivé rozhraní, které usnadňuje interaktivní zadávání dat a sledování průběhu plánovacích algoritmů v reálném čase.

4.3 Pomocné funkce v GUI

Aplikace kromě hlavní funkcionality plánování cesty nabízí také několik pomocných funkcí. Patří sem tlačítka pro možnost uložení aktuální mapy, její opětovné načtení, a také uložení

snímku polygonálního prostoru. I když přímo neovlivňují samotný chod algoritmů, přispívají k větší flexibilitě a uživatelské přívětivosti prostředí. Funkce také pomáhají k lepší vizualizaci algoritmů za stejných uložených podmínek po restartu aplikace a ukládání obrázků.

4.3.1 Uložení mapy

Funkce *save_map* ukládá aktuální konfiguraci mapy do souboru ve formátu JSON. Uložení je start, cíl a body všech polygonálních překážek. Funkce využívá dialogové okno *QFileDialog*, které uživateli umožňuje vybrat umístění a název souboru pro uložení. Před samotným uložením je ověřeno, zda je nastaven start i cíl. V případě že chybí, je uživatel upozorněn skrze varovné okno *QMessageBox.warning*. Po úspěšném uložení je zobrazeno informační okno *QMessageBox.information*, které potvrzuje úspěšné uložení mapy. V případě chyby při ukládání je uživatel informován skrze kritické okno *QMessageBox.critical*.

Pro manipulaci s JSON daty je použit standardní Python modul *json*, který umožňuje snadnou serializaci a deserializaci datových struktur. Struktura uloženého JSON souboru obsahuje tři klíče: *start*, *goal* a *obstacles*. Hodnoty *start* a *goal* jsou dvojice čísel reprezentující souřadnice počátečního a cílového bodu. Klíč *obstacles* obsahuje seznam překážek, kde každá překážka je reprezentována jako seznam dvojic čísel, které představují souřadnice vrcholů polygonu. Tato struktura umožňuje snadné načtení a zpracování mapy při opětovném spuštění aplikace nebo pro další analýzu.

4.3.2 Načtení mapy

Funkce *load_map* načte dříve uloženou mapu ve formátu JSON. Po spuštění této funkce se uživateli zobrazí dialogové okno pro výběr souboru. Po úspěšném načtení souboru jsou ze scény smazány všechny existující prvky, jako jsou start, cíl a překážky, aby se zabránilo jejich duplikaci. Následně jsou z načtených dat vytvořeny nový start, cíl a překážky. Každá překážka je vytvořena z polygonu definovaného seznamem bodů. Tyto prvky jsou následně přidány zpět do scény, čímž je obnovena původní konfigurace mapy.

4.3.3 Vytvoření snímku scény

Funkce *save_scene_screenshot* umožňuje uživateli uložit aktuální pohled na scénu jako obrázek ve formátu PNG nebo JPEG. Po spuštění se otevře dialogové okno pro výběr umístění a názvu souboru. Následně se vytvoří objekt *QImage* odpovídající velikosti scény,

který se vyplní bílou barvou. Pomocí *QPainter* se obsah scény vykreslí do tohoto obrázku. Po ukončení kreslení se obrázek uloží na zvolené místo.

4.4 Detekce kolizí

Detekce kolizí je založena na kontrole, zda se dvě úsečky protínají, a to pomocí funkcí, které využívají geometrický princip orientace bodů. Zásadní funkcí je *ccw* (counter-clockwise) metoda, která určuje, zda jsou tři zadané body uspořádány proti směru hodinových ručiček.

Metoda *ccw* funguje tak, že vytvoří dva vektory – například vektor AB, který je rozdílem bodů B a A, a vektor AC, který je rozdílem bodů C a A. Poté se vypočítá vektorový součin těchto vektorů, což lze vyjádřit jako hodnotu $(C.y - A.y) \times (B.x - A.x) - (B.y - A.y) \times (C.x - A.x)$. Pokud je tato hodnota kladná, znamená to, že bod C leží nalevo od směru, který udává vektor AB, a tedy body A, B a C jsou uspořádány proti směru hodinových ručiček. Pokud je tato hodnota záporná, jsou body A, B a C uspořádány po směru hodinových ručiček.

Funkce *line_intersects* používá výsledky z funkce *ccw* pro určení, zda se dvě úsečky, definované svými krajními body, vzájemně protínají. Princip této funkce spočívá v tom, že pokud jsou koncové body jedné úsečky umístěny na opačných stranách druhé úsečky, a naopak, dojde k průsečíku. Tato metoda efektivně detekuje kolize, což je klíčové pro plánování cesty, protože umožňuje ověřit, že navrhovaná cesta neprotíná žádné překážky v prostředí.

4.5 Plánovací algoritmy

Všechny plánovací algoritmy jsou implementovány pomocí abstraktního rozhraní *IPathfindingAlgorithm*, které definuje základní metody potřebné pro komunikaci s GUI.

Metoda *step* je volána třídou *QTimer*, která spouští tuto metodu v pravidelných intervalech nastavených pomocí posuvníku. To umožňuje plynulou simulaci průběhu algoritmu a aktualizace vizualizace v GUI.

Metoda *reset* smaže z GUI všechny vizualizace algoritmu, které byly vytvořeny během běhu algoritmu, jako jsou čáry, body a další vizuální objekty. Tento postup vrací stav algoritmu do výchozí podoby, což umožňuje opakované spuštění simulace bez zbytkových vizualizací předchozích výpočtů.

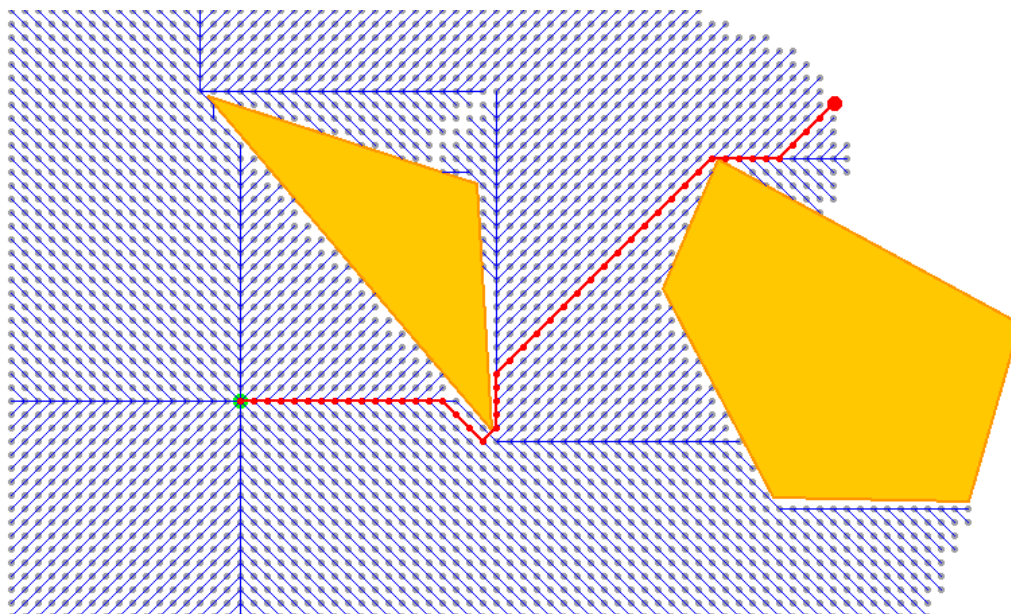
4.5.1 Grid-based algoritmy

Algoritmy využívající mřížku, jako jsou Dijkstrův algoritmus a A*, jsou založeny na třídě *GridPriorityQueueAlgorithm*. Tato třída poskytuje základní metody pro jednoduchou práci.

Metoda *get_neighbors* získá všechny sousední uzly na mřížce. Parametrem jsou souřadnice bodu s datovým typem *tuple*. Kontrolu kolizí provádí metoda *is_collision_free*, která prochází postupně všechny hrany polygonálních překážek a zjišťuje kolizi mezi nimi a novou cestou pomocí *line_intersects* funkce. Parametry jsou dva body určující novou cestu s datovým typem *QPointF*. Rekonstrukce cesty v metodě *reconstruct_path* je poté provedeno zpětným průchodem od cíle ke startu pomocí *came_from* slovníku(dictionary), který uchovává pro každý nový bod odkaz na předchozí bod blíž ke startu. Parametrem funkce je cílový bod datového typu *tuple*.

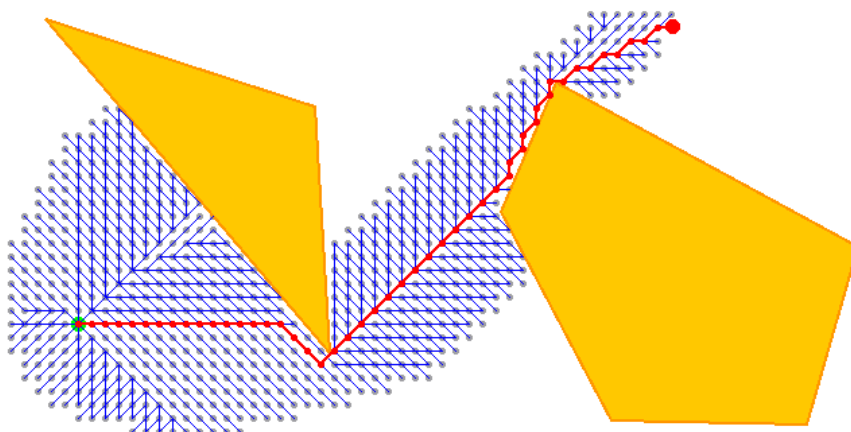
Jednotlivé algoritmy jsou pak definované v jednotlivých třídách, které přepisují funkci skóre *score_function*, kde Dijkstrův algoritmus používá pouze celkové náklady *g_score*, zatímco A* využívá součet nákladů s heuristikou *f_score*.

Nastavitelnými parametry pro tyto algoritmy je *grid_size*, který udává velikost imaginární mřížky v prostoru, po které se algoritmy pohybují.



Obrázek 13 Vizualizace Dijkstrova algoritmu

(zdroj: vlastní)



Obrázek 14 Vizualizace A* algoritmu

(zdroj: vlastní)

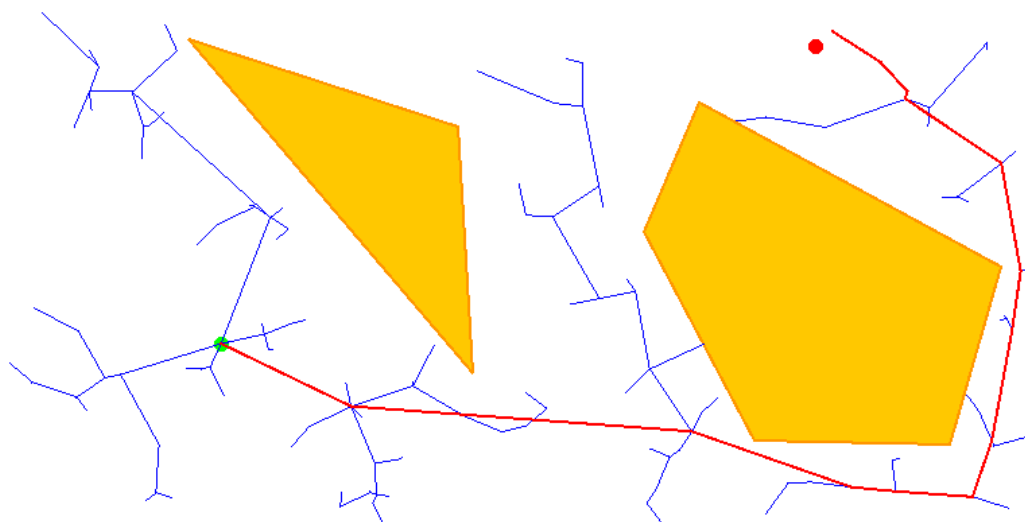
Algoritmy využívající mřížku používají pro vizualizaci šedý bod pro každý navštívený uzel v mřížce. Během hledání cesty je nejnovější cesta zobrazena fialově. Ostatní prozkoumané cesty jsou znázorněny modře a výsledná cesta pak červeně.

4.5.2 Algoritmy založené na náhodném vzorkování

Algoritmy RRT a RRT* dědí ze třídy *RandomTreeAlgorithm*. K dispozici jsou základní metody společné pro oba algoritmy.

Metoda *get_random_point* vrátí náhodný bod *QPointF* ve scéně pomocí *random* modulu. Funkce *nearest_node* pak prochází všechny již existující uzly stromu a vrátí nejbližší bodu k náhodnému bodu, který je parametrem funkce. Pro kontrolu kolizí je opět využita funkce *is_collision_free*.

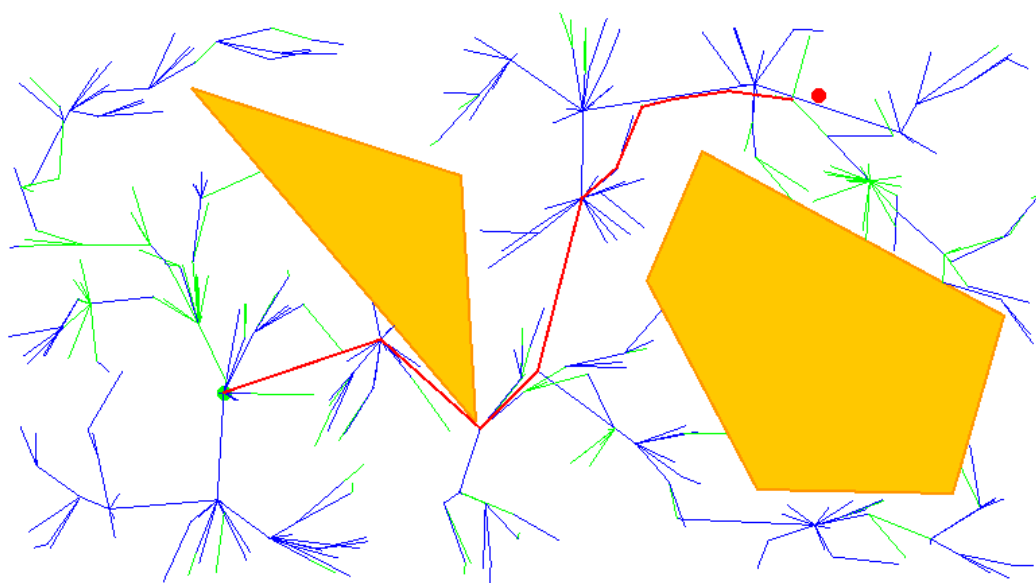
RRT má nastavitelný parametr *step_size*, který určuje o kolik % se přiblíží z existujícího bodu k novému náhodnému bodu. RRT * má ještě *search_radius*, který určuje, v jaké vzdálenosti od bodu se zkoumají již existující body k optimalizaci přepojení stromu.



Obrázek 15 Vizualizace RRT algoritmu

(zdroj: vlastní)

RRT algoritmus během hledání cesty znázorňuje fialovým křížkem náhodně vybraný bod, fialovým bodem zvolený bod, který se blíží k náhodnému bodu a nejnovější cestu fialově. Výsledné navštívené cesty jsou pak znázorněny modře a finální nalezená cesta pak červeně.



Obrázek 16 Vizualizace RRT* algoritmu

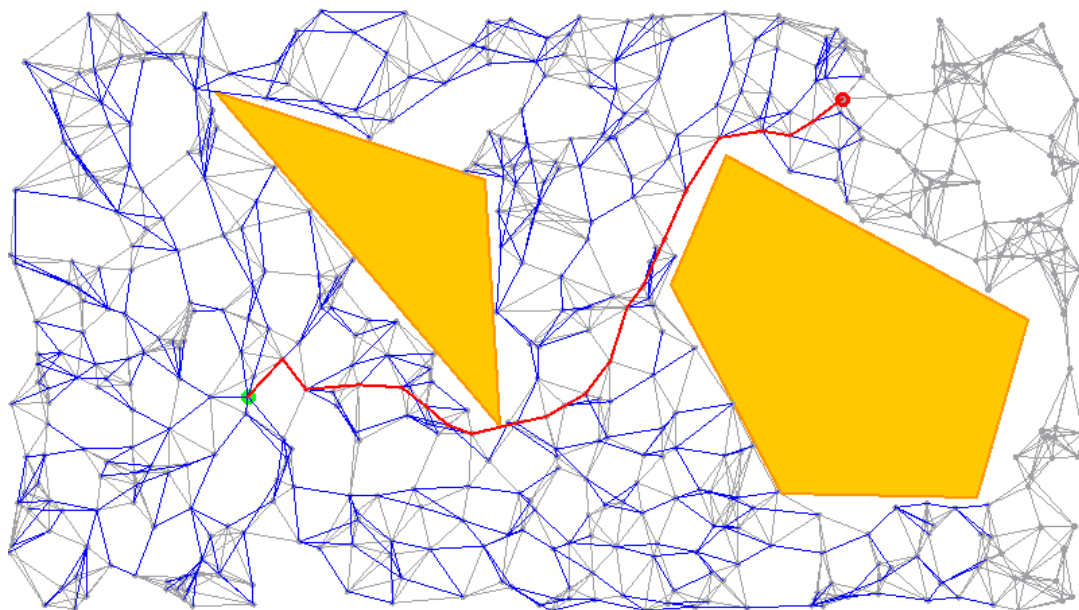
(zdroj: vlastní)

RRT* algoritmus má stejnou vizualizaci jako RRT, ale navíc znázorňuje zeleně přepojené optimalizované cesty v průběhu hledání cesty.

4.5.3 PRM algoritmus

PRM algoritmus funguje ve dvou fázích. V první fázi se generují náhodné body v poli pomocí metody *add_sample*. Ta na začátku vytvoří první dva body na startu a na cíli, aby zajistila propojení těchto bodů. Poté pomocí funkce *get_random_point* získá náhodný bod ve scéně a zkontroluje, zda se nenachází na již existujícím bodu s tolerancí nebo v překážce a případně ho opakovaně generuje do doby, než je validní. Po vygenerování všech bodů se vytvoří graf v metodě *build_roadmap*, kde se každý bod spojí s pěti nejbližšími body s ohledem na překážky. Ve druhé fázi se pak aplikuje A* algoritmus na vytvořený graf v první fázi.

Parametrem PRM algoritmu je *n_samples*, který určuje počet náhodných bodů pro vytvoření grafu v první fázi.



Obrázek 17 Vizualizace PRM algoritmu

(zdroj: vlastní)

V první fázi PRM algoritmus zobrazí šedě náhodně vygenerované body a spoje mezi nimi, čímž vykreslí celou vygenerovanou mřížku. V druhé fázi se aplikuje stejné znázornění jako u A* algoritmu, to jsou fialové nejnovější cesty, modré prozkoumané cesty a finální nalezená cesta zobrazená červeně.

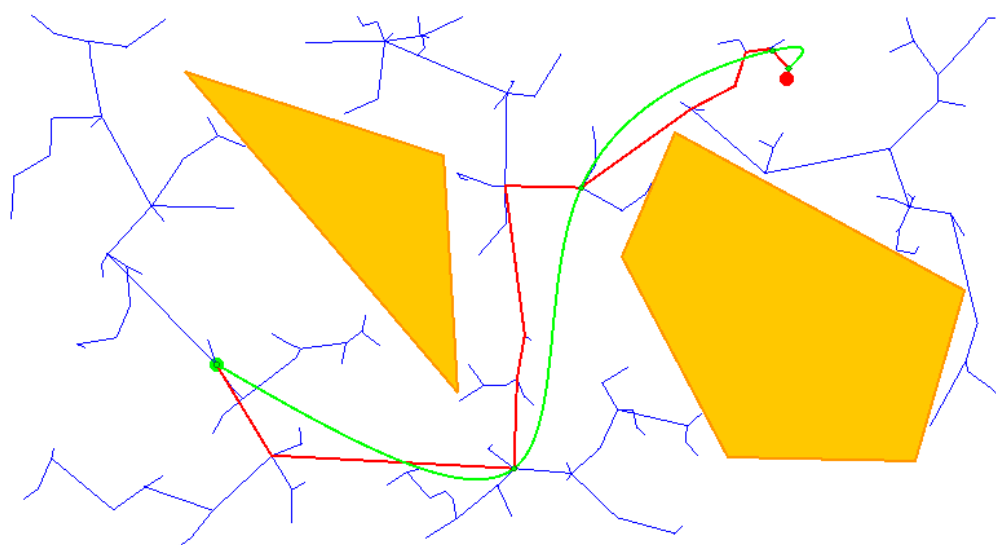
4.6 Vyhlazení cesty

GUI obsahuje možnost vyhlazení výsledné trasy pomocí funkce *smooth_path*. Funkce vyhlazuje cestu ve třech hlavních krocích.

Prvním krokem je odstranění nadbytečných bodů cesty. Na začátku vezmeme startovní bod a zkusíme vést přímou trasu k cílovému bodu. Pokud tato cesta prochází překážkou, vezmeme další bod od konce cesty, čímž efektivně nalezneme nejdelší možnou rovnou přímkou mezi dvěma nejvzdálenějšími body. Jakmile nalezneme nejdelší možnou přímku, zvolíme její konec jako nový startovní bod a proces opakujeme tak dlouho, dokud nedosáhneme cílového bodu. Tímto jednoduchým algoritmem tak snížíme celkový počet bodů pro jednodušší následné zpracování.

Druhým krokem je aplikace kubického splinu s detekcí kolizí. Jako první na naší zjednodušené cestě z prvního kroku vytvoříme kubický spline pomocí knihovny *scipy*, která nám nabízí funkci *CubicSpline*, která nám zajistí interpolaci dat na kubický polygon. Parametry této funkce jsou hodnoty osy x (indexy hodnot) a hodnoty osy y (souřadnice bodů). Poté je spline vzorkován, aby ho bylo možné lépe vykreslit a testovat kolizi s překážkami. Ta se provádí ve funkci *segment_collision*. Parametrem je list bodů části křivky. Ta se převede na geometrii pomocí třídy *LineString* a zjistí se kolize pro všechny ostatní polygony také převedené na geometrii pomocí třídy *Polygon*. Pokud je nalezena kolize, rozdělíme část přímky na polovinu a přepočítáme spline. Tento postup je opakován, dokud je přímka bez kolizí, nebo dosáhnuto maximálního počtu dělení na segment. V takovém případě se použije přímka z prvního kroku místo spline.

Třetím krokem je vykreslení výsledné vyhlazené trasy. Každá křivka je vytvořena pomocí třídy *QPainterPath*. Části, u kterých byl překročen limit dělení jsou pak zobrazeny jako jednoduchá přímka. Výsledná vyhlazená cesta je pak vykreslena zeleně včetně jejích kontrolních bodů.



Obrázek 18 Vyhlazení cesty RRT algoritmu

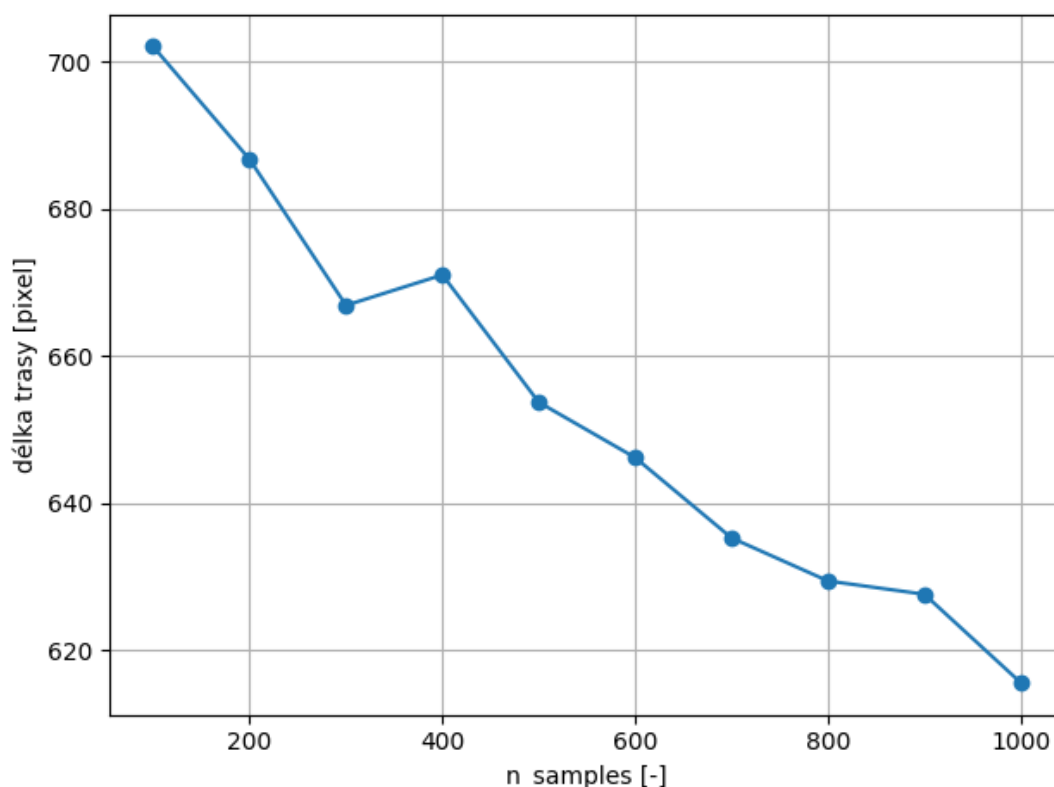
(Zdroj: vlastní)

5 POROVNÁNÍ ALGORITMŮ

V této kapitole jsou porovnány vybrané plánovací algoritmy implementované v praktické části práce. Cílem je ilustrovat jejich rozdíly nejenom z hlediska teoretických vlastností, ale především praktických důsledků při aplikaci na konkrétní úlohy plánování cesty. Tradičně se algoritmy posuzují podle časové a paměťové složitosti (time and space complexity), avšak v kontextu mobilní robotiky jsou často důležitější jiné aspekty, například délka nalezené trasy, míra optimálnosti, nebo schopnost vyhnout se překážkám s ohledem na mobilitu robota. U nedeterministických algoritmů pak můžeme provést měření několikrát za sebou a vypočítat pak průměrné hodnoty ze všech měření.

5.1 PRM parametry

Jedním z klíčových parametrů u algoritmu PRM je počet náhodně generovaných bodů ve volném prostoru. S rostoucím počtem vzorků narůstá hustota grafu a tím i pravděpodobnost nalezení kratší a hladší cesty.

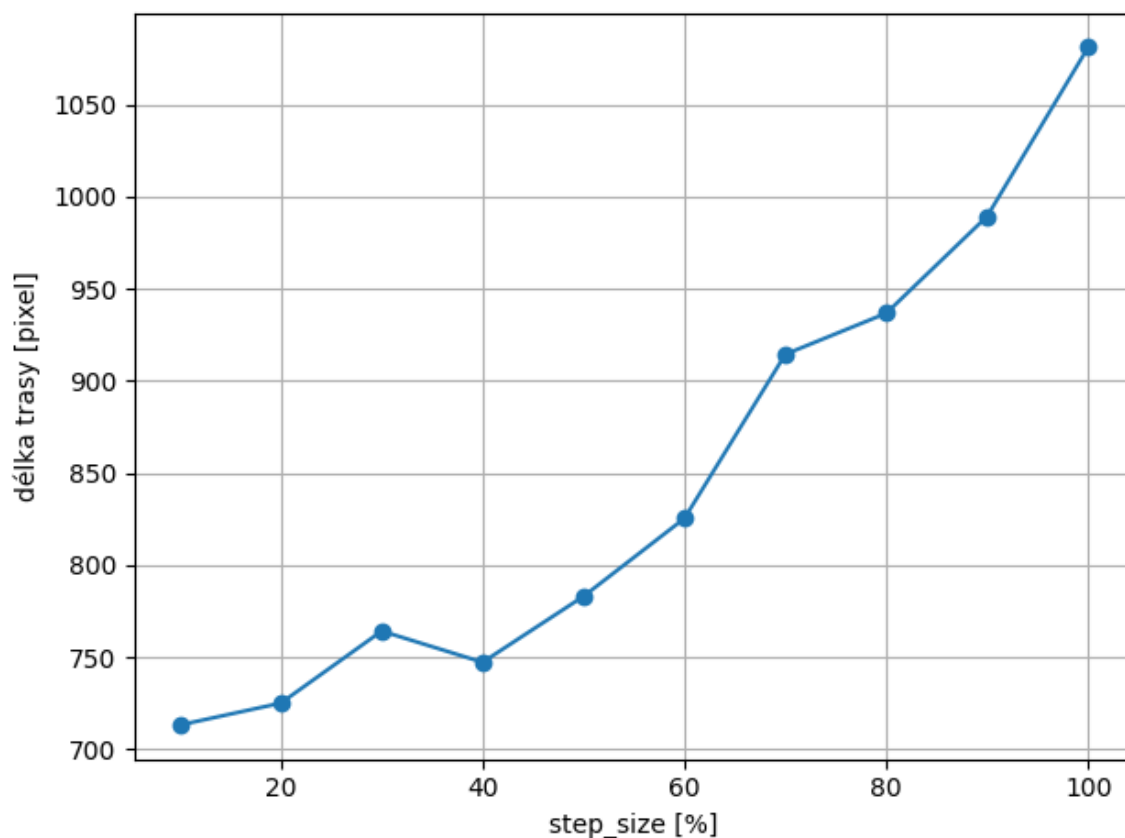


Obrázek 19 Závislost $n_samples$ na průměrné délce trasy.

(zdroj: vlastní)

5.2 RRT parametry

Parametr *step_size* v algoritmu RRT ovlivňuje, jak moc algoritmus postupuje k náhodně vybraným bodům. Malý krok vede k přesnějšímu, ale pomalejšímu růstu stromu. Velký krok může zrychlit hledání, ale způsobit méně přesné nebo méně optimalizované trasy.



Obrázek 20 Závislost *step_size* na průměrné délce trasy.

(zdroj: vlastní)

ZÁVĚR

Teoretická část práce představila základní principy a metody plánování cesty pro mobilní roboty, včetně různých přístupů, od jednodušších metod, jako jsou Bug algoritmy, až po pokročilejší přístupy, jako je Voroného diagram, BFS, DFS, Dijkstrův algoritmus, A* a metody založené na náhodném vzorkování RRT, RRT* a PRM. Byly popsány teoretické základy těchto metod, jejich výhody, nevýhody a vhodnost pro různé typy prostředí.

Praktická část pak demonstrovala implementaci vybraných plánovacích algoritmů v jazyce Python s využitím knihovny PyQt6 pro tvorbu interaktivního grafického uživatelského rozhraní. V této části byly detailně popsány použité třídy, datové struktury a metody, jako například třídy *ScalableGraphicsView* a *PolygonScene*, které umožňují interaktivní zadávání dat a vizualizaci výsledků plánování.

Veškeré implementované metody v praktické části byly testovány pouze na modelu holonomního robota, tedy bez jakýchkoliv kinematických omezení směru pohybu. Pro neholonomní roboty, jako jsou vozidla s diferenciálním či Ackermanovým řízením, by bylo nutné upravit jak reprezentaci stavu (např. orientaci), tak i samotné implementace plánovacích algoritmů tak, aby zohledňovaly omezení na směr a poloměr zatačení. Rozšíření práce o podporu neholonomních robotů by proto představovalo logický postup do budoucna, který by významně zvýšil praktické využití výukové aplikace pro reálné robotické platformy.

SEZNAM POUŽITÉ LITERATURY

- [1] SIEGWART, Roland; NOURBAKHSH, REZA, Illah a SCARAMUZZA, Davide. Introduction to Autonomous Mobile Robots. Second edition. MIT Press, 2011. ISBN 0262015358
- [2] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to Algorithms. Third edition. MIT Press, 2009. ISBN 978-0262033848
- [3] LAVALLE, Steven M. Planning Algorithms. Cambridge University Press, 2006. ISBN 978-052186205
- [4] VLADAŘ, Lukáš, ZÁPADOČESKÁ UNIVERZITA V PLZNI; FAKULTA APLIKOVANÝCH VĚD; KATEDRA KYBERNETIKY. Plánování pohybu kolových mobilních robotů. Bakalářská práce. Plzeň, 2021.
- [5] ŠINDELÁŘ, Jiří. Plánování cesty neholonomního mobilního robotu. Diplomová práce. Vysoké učení technické v Brně; Fakulta strojního inženýrství; Ústav automatizace a informatiky, 2010.
- [6] KWANGJIN YANG, Sukkarieh S.: 3D Smooth path planning for a UAV in cluttered natural environmentsIn: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008. p. 794-800.
- [7] LUTZ, Mark. Learning Python. 4th ed. Beijing O'Reilly, 2009. ISBN 978-0-596-15806-4
- [8] THE QT COMPANY. Qt for Python Documentation [online]. [cit. 2025-04-05]. Dostupné z: <https://doc.qt.io/qtforpython-6/>
- [9] CHOSET, Howie; LYNCH, Kevin; HUTCHINSON, Seth; KANTOR, George; BURGARD, Wolfram; KAVRAKI, Lydia; THRUN, Sebastian. Principles of Robot Motion: Theory, Algorithms, and Implementation. Cambridge, MA; London: MIT Press, 2005. ISBN 0262033275.
- [10] KAVRAKI, Lydia E.; ŠVESTKA, Petr; LATOMBE, Jean-Claude; OVERMARS, Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, Vol. 12, No. 4, 1996, pp. 566–580. doi: 10.1109/70.508439. ISSN 1042-296X.

- [11] PATRIKALAKIS, Nikos; MAEKAWA, Toshiaki a CHO, Sang-Kyun. *Shape Interrogation for Computer Aided Design and Manufacturing* [online]. Massachusetts Institute of Technology. Dostupné z: <https://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node8.html>

SEZNAM OBRÁZKŮ

Obrázek 1 Spojité prostředí	14
Obrázek 2 Diskrétní prostředí.....	14
Obrázek 3 Ukázka Bug0 algoritmu včetně možnosti zacyklení	17
Obrázek 4 Ukázka Bug1 algoritmu včetně prevence zacyklení	18
Obrázek 5 Ukázka Bug2 algoritmu včetně možné suboptimální trasy.....	19
Obrázek 6 Graf viditelnosti.....	20
Obrázek 7 Voroného diagram s náhodnými body	21
Obrázek 8 Voroného diagram s překážkami.....	21
Obrázek 9 Průchod BFS algoritmu	23
Obrázek 10 Průchod DFS algoritmu.....	24
Obrázek 11 Průchod Dijkstrova algoritmu	25
Obrázek 12 Průběh iterací RRT algoritmu	26
Obrázek 13 Vizualizace Dijkstrova algoritmu.....	35
Obrázek 14 Vizualizace A* algoritmu.....	36
Obrázek 15 Vizualizace RRT algoritmu.....	37
Obrázek 16 Vizualizace RRT* algoritmu.....	37
Obrázek 17 Vizualizace PRM algoritmu	38
Obrázek 18 Vyhlazení cesty RRT algoritmu	40
Obrázek 19 Závislost n_samples na průměrné délce trasy.	41
Obrázek 20 Závislost step_size na průměrné délce trasy.	42

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

BFS Breadth first search

CVM Curvature-velocity method

DFS Depth first search

DWA Dynamic window approach

GUI Graphical user interface

LCM Lane-curvature method

PRM Probabilistic roadmap

RRT Rapidly exploring random tree

VFH Vector field histogram

SEZNAM PŘÍLOH

Příloha P I: Zdrojový kód aplikace

PŘÍLOHA P I: ZDROJOVÝ KÓD APLIKACE

Zip soubor obsahuje zdrojový kód aplikace. Pro spuštění slouží main.py.