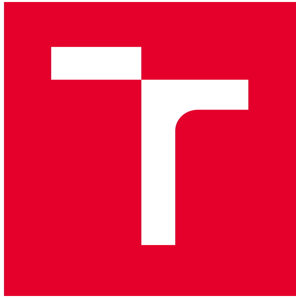


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

PLÁNOVÁNÍ TRAJEKTORIE PRŮMYSLOVÉHO MANIPULÁTORU S PŘEKÁŽKAMI V PRACOVNÍM PROSTORU

TRAJECTORY PLANNING OF INDUSTRIAL MANIPULATOR WITH OBSTACLES IN THE WORKSPACE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE **Marek Vlček**

AUTHOR

VEDOUCÍ PRÁCE **Ing. Matej Rajchl**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Marek Vlček**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Matej Rajchl**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Plánování trajektorie průmyslového manipulátoru s překážkami v pracovním prostoru

Stručná charakteristika problematiky úkolu:

Práce navazuje na předešlou práci vedenou v rámci mechatronické laboratoře, kde bylo využito algoritmu A*, který se ukázal nebýt příliš vhodným na daný úkol. Tato práce proto využije rozdílného algoritmu, který pracuje efektivně i když sub–optímálně v daném stavovém prostoru typického průmyslového manipulátoru.

Cíle bakalářské práce:

1. Udělejte rešerši algoritmů vhodných pro prohledávaní vícerozměrných stavových prostorů.

2. Vyberte nejvhodnější algoritmus pro úlohu plánování trajektorie v stavovém prostoru typického manipulátoru, jehož pracovní prostor obsahuje překážky.

3. Za pomoci programovacího prostředí MATLAB/Python implementujte daný algoritmus do přehledné a funkční knihovny.

4. Ověřte funkčnost na manipulátoru UR5e dostupném v mechatronické laboratoři.

Seznam doporučené literatury:

GREPL, R.:Kinematika a dynamika mechatronických systémů CERM, Akademické nakladatelství, 2007.

GREPL, R.:Modelování mechatronických systémů v Matlab/SimMechanics BEN - technická literatura, 2007.

MURRAY, R.:M.; SASTRY, S. S. & ZEXIANG, L.: A Mathematical Introduction to Robotic Manipulation CRC Press, Inc., 1994.

RUSSEL, S., NORVIG, P.: Artificial Intelligence. A Modern Approach. Prentice Hall 2009.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc.	doc. Ing. Jiří Hlinka, Ph.D.
ředitel ústavu	děkan fakulty

Abstrakt

V této práci jsou vysvětleny základy řízení manipulátorů a algoritmů, které slouží k jejich navigaci v prostoru s překážkami. Dále je obhájeno využití algoritmu RRT*, jehož fungování je detailně vysvětleno a popsáno.

Summary

In this thesis, the basics of manipulator control and algorithms used for their navigation in obstacle-filled space are explained. Furthermore, the utilization of the RRT* algorithm is justified and its functioning is explained and described in detail.

Klíčová slova

Navigace, RRT*, Rapidly-exploring random trees, Manipulátor, UR5e, Přímá kinematika, Inverzní kinematika, Detekce kolizí

Keywords

Navigation, RRT*, Rapidly-exploring random trees, Manipulator, UR5e, Forward Kinematics, Inverse Kinematics, Collision Detection

Bibliografická citace

Citace tištěné práce:

VLČEK, Marek. *Plánování trajektorie průmyslového manipulátoru s překážkami v pracovním prostoru*. Brno, 2023.
Dostupné také z:https://www.vut.cz/studenti/zav-prace/detail/149881. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Matej Rajchl.

Citace elektronického zdroje:

VLČEK, Marek. *Plánování trajektorie průmyslového manipulátoru s překážkami v pracovním prostoru* [online]. Brno, 2023 [cit. 2023-05-20].
Dostupné z:https://www.vut.cz/studenti/zav-prace/detail/149881. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Matej Rajchl.

Čestné prohlášení

Prohlašuji, že tato práce byla vypracována zcela samostatně, s využitím odborné literatury a jiných akademických zdrojů uvedených v seznamu na konci práce.

21. května 2023

Marek Vlček

Poděkování

Tímto děkuji Ing. Mateji Rajchlovi za rady a vedení této bakalářské práce. Dále bych chtěl poděkovat Bc. Vojtěchu Jindrovi za pomoc s pochopením problematiky týkající se algoritmizace.

Obsah

1 Úvod **9**

2 Řízení a rešerše algoritmů **11**
2.1 Řízení sériových manipulátorů . . . 11
2.2 Prohledávání stavových prostorů . . . 13
2.2.1 Search-based algoritmy . . . 13
2.2.2 Sampling-based algoritmy . . . 15

3 RRT a RRT* **18**
3.1 Princip fungování . . . 18
3.2 Vlastnosti a využití . . . 20
3.3 Rozšíření na RRT* . . . 21

4 Implementace algoritmu RRT* **23**
4.1 Přepis algoritmu a vysvětlení skriptu . . . 23
4.2 Funkce pro detekci kolizí . . . 26
4.3 Komplikace a jejich řešení . . . 29
4.3.1 Vyřešené . . . 29
4.3.2 Nevyřešené . . . 32

5 Využití v praxi **33**
5.1 Návod k použití . . . 33
5.2 Ukázka . . . 34
5.2.1 Základní pohyby . . . 35
5.2.2 Složitý pohyb . . . 37

6 Závěr **38**

Seznam zdrojů **40**

Seznam obrázků **41**

Seznam úryvků kódu **42**

Seznam tabulek **43**

Seznam příloh **44**

1. Úvod

V posledních letech se průmyslové manipulátory staly nedílnou součástí moderních výrobních procesů. Tyto stroje zásadně změnily průmyslovou výrobu tím, že automatizovaly únavné a opakující se úkoly, zvyšují efektivitu a snižují náklady. Také mohou vykonávat práce, které by pro člověka byly nebezpečné, a tím zvyšovat bezpečnost na pracovišti. V praxi se používá velká řada manipulátorů, které je možné rozdělit do skupin podle typu kloubů, tvaru pracovního prostoru a řešením celkové konstrukce.

Na základě úkolu, který je potřeba vykonat, je důležité zvolit správný tvar pracovního prostoru. Ten je určen typem kloubů, které jsou použity v konstrukci manipulátoru. Mezi základní typy patří rotační kloub, který umožňuje rotační pohyb kolem jedné osy a lineární kloub, který umožňujě přímý translační pohyb podél jedné osy. Je samozřejmostí, že translační pohyb je v kartézkých souřadnicích a rotačních naopak v úhlových souřadnicích. Existují také cylindrické klouby, které kombinují translační a rotační pohyb nebo sférické klouby, které umožňují rotaci ve všech třech osách.

Důležité však je, jakým způsobem tyto klouby využijeme v konstrukci manipulátoru. Mezi základní typy průmyslových manipulátorů patří sériové a paralelní. V případě paralelního manipulátoru je koncový efektor připojen k více robotickým pažím, pevně připojeným k podstavě, které ve spolupráci určují jeho pozici a natočení. Tento kinematický model disponuje pevností a velkou nákladní kapacitou. Je však omezen ve svém dosahu a není vhodný pro práci v prostředí s překážkami. Naopak sériový manipulátor tvoří jedna robotická paže, sestavena z jednotlivých článků. Ty jsou k sobě připojeny různými typy kloubů, a tím dávájí manipulátoru určité vlastnosti a pracovní prostor. Obecně jsou sériové manipulátory mnohem flexibilnější než paralelní, ale nezvládnou unést stejně těžké předměty.

V této práci se věnujeme manipulátoru UR5e, což je R-R-R sériový manipulátor. To znamená, že paže manipulátoru má tři části, které jsou propojeny rotační vazbou. UR5e má však celkem kloubů šest. První tři lze považovat jako tělo a slouží k pohybu manipulátoru v jeho pracovním prostoru. Zbylé tři klouby už nepohybují dlouhými rameny, ale určují orientaci koncového efektoru s třemi stupni volnosti. V praxi se využívají i jiné kombinace, jako například P-P-P, kdy jsou všechny klouby lineární. Pracovní prostor takového manipulátoru je kvádrový a nazýváme jej DELTA robot. Kromě klasického průmyslového využití je můžeme vidět například v 3D tiskárnách. Dále jsou využívané i jiné kombinace, kdy jejich užití je závislé na úkolu, který má manipulátor vykonávat.

Účinnost těchto manipulátorů však silně závisí na algoritmech, které řídí jejich pohyby. Vyhledávací algoritmy hrají klíčovou roli při efektivním řízení průmyslových manipulátorů. Tyto algoritmy jsou zodpovědné za nalezení optimální trajektorie pro pohyby manipulátoru za daných omezení a cílů. Použití efektivních vyhledávacích algoritmů může výrazně snížit čas a zdroje potřebné pro výrobní proces, což vede k produktivnějšímu a výnosnějšímu provozu. Existuje velké množství algoritmů a ne všechny z nich jsou vhodné pro každý manipulátor. Je proto vysoce důležité seznámit se s problematikou řízení těchto manipulátorů a nastudovat si, které algoritmy budou nejvhodnější k řízení právě manipulátoru UR5e.

V této práci si představíme základy problematiky řízení manipulátorů a některé z vyhledávacích algoritmů. Většina práce se však věnuje zvolení nejvhodnějšího algoritmu pro práci s manipulátorem UR5e a jeho řízení v prostoru s překážkami. Zvolený algoritmus bude důkladně rozebrán a pečlivě vysvětlen. Celá práce je doprovázena obrázky a úryvky ze skriptu pro jednoduché pochopení a následné využití kódu k řízení manipulátoru. Na konci vyhodnotíme sérii testů, kterými byla ověřena funkčnost výsledného skriptu.

2. Řízení a rešerše algoritmů

V první kapitole se věnujeme vysvětlení základní myšlenky prohledávání prostoru s překážkami. Představíme si nejpoužívanější algoritmy, které se pro tyto úkoly používají, rozdělíme si je do tříd na základě způsobu, jakým k řešení přistupují a zhodnotíme jejich výhody a nevýhody. Dále si vysvětlíme, jak jsou tyto algoritmy implementovány v praxi při řízení průmyslových manipulátorů. V neposlední řadě zvolíme nejvhodnější algoritmus pro řízení manipulátoru UR5e v prostoru s překážkami dle zadání a obhájíme jeho použití.

2.1 Řízení sériových manipulátorů

Průmyslové sériové manipulátory jsou tvořeny články propojenými v sérii, mezi kterými se nacházejí klouby umožňující pohyb, ať už rotační či lineární. Abychom přemístili koncový efektor manipulátoru z bodu A do bodu B, tak je potřeba přiřadit hodnotu jednotlivě každému z kloubů. Pohyb manipulátoru je tedy řízen vektorem obsahujícím tyto hodnoty. Ten je získán pomocí přímé a inverzní kinematiky.

Přímá kinematika se zabývá postupným dopočítáváním poloh jednotlivých ramen manipulátoru až po koncový bod. V příkladu přímé kinematiky je potřeba vědět rozměry manipulátoru a konfiguraci všech jeho kloubů. Poté se od základny, kterou předpokládáme vetknutou, vypočítá pomocí transformačních matic poloha koncového bodu manipulátoru. Každá transformační matice obsahuje rotační matici a polohový vektor, které popisují polohu koncového bodu daného ramena vůči vztažnému bodu. V práci jsou využívány rotace kolem osy x a z, které mají rotační matice ve tvaru znázorněném v rovnicích 2.1 a 2.2. Polohový vektor nám udává posun ve všech třech osách, jak můžeme vidět ve vztahu 2.3.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \tag{2.1}$$

$$R_z = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

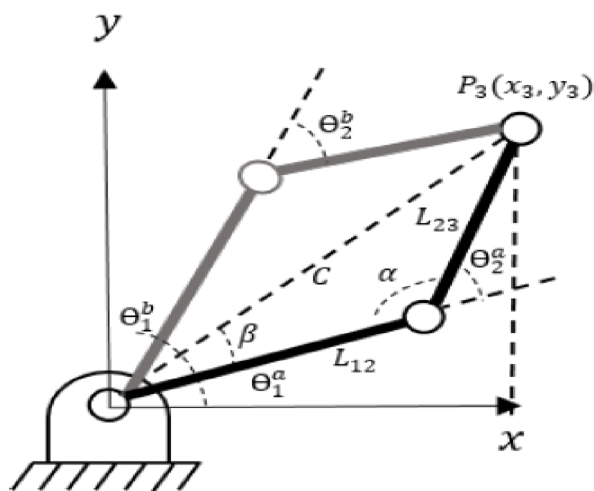
$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2.3}$$

Transformační matice poté nabývají tvaru z rovnice 2.4 a obsahují jak rotaci kolem osy z, tak posun v ose z. První transformační matice popisuje natočení a posun konce prvního článku manipulátoru. Každá další matice se však vztahuje pořád k výchozímu bodu. Je tedy potřeba přidat ještě posun předchozí matice. Toho docílíme vynásobením první matice novou, stejně jako je učiněno v rovnici 2.5. Tímto způsobem postupujeme dál a získáme matice, které popisují koncové body všech ramen manipulátoru. Ty jsou využívány k simulaci pohybu manipulátoru.

$$T10 = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$T_{20} = T_{10} \cdot T_{21} \tag{2.5}$$

Inverzní kinematika funguje opačným způsobem. V tomto případě zadáváme polohu koncového efektoru manipulátoru a řešič inverzní kinematiky vypočítá optimální natočení jednotlivých kloubů. Narozdíl od přímé kinematiky kde je vždy pouze jedno řešení, inverzní kinematika má řešení mnoho. To si můžeme demonstrovat na obrázku 2.1, kde je RR manipulátor, kterému jsou dopočítávány úhly natočení pomocí inverzní kinematiky. Protože jediné parametry jsou pracovní prostor a koncový bod manipulátoru, jsou obě tyto konfigurace možné. Při navyšování počtu ramen začne být výpočet příliš složitý a je potřeba použít numerické řešiče pro získání finálních souřadnic. Proto v této práci využíváme pouze přímou kinematiku.



Obrázek 2.1: Inverzní kinematika RR manipulátoru

V praxi se využívají tyto dva druhy výpočtů současně a řízení manipulátorů je možné jak určováním polohy jednotlivých kloubů, tak pouze určením polohy koncového efektoru. Při plánování trasy manipulátoru v prostředí s překážkami jsou tedy také použitelné dva způsoby. První způsob pracuje s myšlenkou, že se neprohledává kartézský prostor, ve kterém se nachází manipulátor, ale prostor tvořený možnými hodnotami natočení jednotlivých kloubů. Tento způsob je vhodný pro vyhýbání se překážkám a je rozebírán ve větším detailu v kapitole 4.1. Druhý způsob je hledat trasu přímo v kartézském prostoru, ve které se pohybuje manipulátor a výslednou trasu poté kopírovat koncovým efektorem manipulátoru a úhly natočení kloubů dopočítávat. Tento postup je však velice náročný z výpočetní stránky a nevhodný pro pohyb v prostředí s překážkami. Naopak se hodí pokud je potřeba, aby koncový efektor vykonával přesné pohyby ve volném prostoru.

2.2 Prohledávání stavových prostorů

Nejprve si vysvětlíme co je to stavový prostor. Jedná se o soubor všech možných stavů, které mohou pro daný systém nastat. V našem případě jsou tyto stavy souřadnice v prostoru, ve kterém se snažíme najít cestu. Algoritmů pro plánování optimální trasy existuje v dnešní době mnoho. Je však důležité zvolit ten, který svými vlastnostmi odpovídá problému, který chceme vyřešit. Některé algoritmy vynikají v nalezení perfektní trasy, ale potřebují dlouhý výpočetní čas a naopak. Jiné jsou bez konkurence v nižších dimenzích, ale nefungují optimálně (někdy vůbec) v prostorech o více dimenzích. Z toho důvodu se pro dosažení požadovaného výsledku v praxi využívají často algoritmy, které jsou kombinací více přístupů k problematice a vzájemně eliminují své nedostatky. V této práci se však kvůli jejich komplexnosti a nedostatku praxe věnujeme pouze algoritmům, které využívájí pouze jeden princip hledání cesty k cíli.

2.2.1 Search-based algoritmy

Pojem search-based algoritmus zahrnuje široké množství různých algoritmů. Nejdůležitějším faktorem rozhodujícím o tom, zda se jedná o search-based algoritmus je systematičnost. Tyto algoritmy tvoří trasu postupně, ať už od počátku či konce a přecházejí z jednoho stavu do dalšího podle předem stanovených pravidel. Jsou většinou doprovázeny výpočty, které systematicky určují nejvhodnější nebo naopak nevhodný stav. Díky tomu jsou obecně search-based algoritmy užitečné při hledání optimální cesty. Je to však na úkor rychlosti. Za předpokladu, že se prostředí ani počátek s cílem nezmění, tak cesta, kterou takové algoritmy najdou, bývá většinou stejná. Nyní si prohlédneme search-based algoritmy, které byly zvažovány pro plánování trasy.

A*

Tento algoritmus nebyl pro práci pouze zvažován, ale byl i použit v předchozí citované práci. Algoritmus A* je vhodný pro nalezení nejkratší cesty. Vychází z Djikstrova algoritmu, který prohledával prostor ve všech směrech kolem sebe, dokud nenarazil

na cíl, což jej dělalo velice pomalým. A* narozdíl od něj prohledává prostor přímým směrem k cíli, pokud je to možné. V případě, že narazí na překážku, vyhodnotí pomocí své heuristické funkce, kterým směrem je vhodné překážku obejít, a pak se snaží nadále postupovat přímo k cíli. Tento algoritmus byl navržen pro prohledávání prostoru v rovině a ukázalo se, že v komplexním 6D prostoru je jeho výpočet přesný, ale příliš pomalý.

Obrázek 2.2: Ukázka algoritmu A*

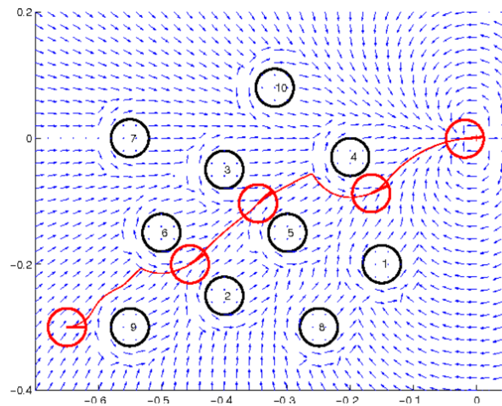
Potenciální pole

Prostor je reprezentován virtuálním polem, které tvoří vysoké (překážky) a nízké (cíl) potenciály. V tomto prostoru je trasa tvořen tím způsobem, že je simulován průchod tímto prostorem, kdy překážky robota odpuzují a cíl jej naopak přitahuje. Ten se tedy pohybuje podél gradientu těchto potenciálních polí. Výpočetní čas je spotřebován nejen při vytváření tohoto virtuálního prostředí, ale i při navigací v něm. To dělá algoritmus relativně pomalý v porovnání s jinými zvažovanými algoritmy. Navíc samotný princip vytváření takového prostředí v 6D se ukázal být příliš komplikovaný.

Obrázek 2.3: Ukázka algoritmu potenciální pole

Vektorová pole

V tomto případě je opět vytvořen virtuální prostor, ale tentokrát místo skalárních hodnot reprezentujících přitažlivé síly, je tento prostor tvořen přímo vektory, které určují směr pohybu směrem k cíli ze všech míst v prostoru. Tento algoritmus exceluje v případech, kdy je potřeba dynamicky měnit pozici cíle bez změny prostředí. Tato vlastnost by byla u manipulátoru užitečná, protože by stačilo vytvořit tuto virtuální mapu pouze při zapnutí, a poté by odpovídal relativně rychle. Je také vhodnější pro využití ve vyšších dimenzích než potenciální pole. Přesto je vytvoření takto komplexních vektorových polí složité.



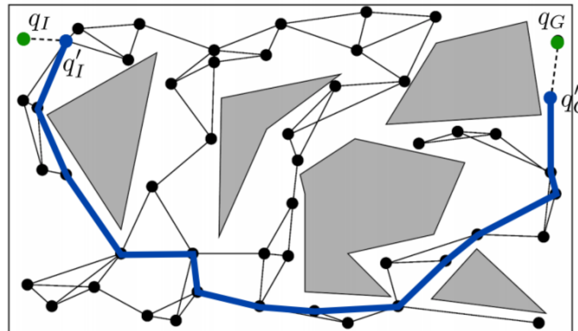
Obrázek 2.4: Ukázka algoritmu vektorová pole

2.2.2 Sampling-based algoritmy

Tato třída algoritmů pro plánování pohybu se zaměřuje na průzkum prostoru pomocí náhodného vzorkování stavů. To znamená, že narozdíl od search-based algoritmů nepoužívají žádný systematický postup, kterým by se snažili přiblížit k cíli, ale náhodně (existují však i informované verze) prohledávají prostor. Využívají různé způsoby k třídění těchto vzorků a vytváření struktur, ze kterých jsou poté schopné vytvořit cestu k cíli. Jsou obzvláště účinné ve vysokodimenzních prostorech a vynikají ve vyhýbání se překážkám rychle a efektivně. Jejich největší nevýhoda spočívá v jejich náhodném principu fungování, které způsobuje tvoření suboptimálních tras.

PRM

Celým názvem Probabilistic Roadmap pracuje s náhodně vytvořenými klíčovými body v prostoru, které následně propojí a vytvoří mapu. Poté je již schopný vytvořit libovolnou cestu pomocí spojnic mezi body této mapy. Dokud se nezmění prostředí, není potřeba vytvářet novou mapu. Díky tomu je tento algoritmus efektivní při dynamických změnách pozic manipulátoru, při neměnném prostředí. Ve vyšších dimenzích je vytvoření takové mapy výpočetně a časově náročné. Mohou se objevit i problémy s nedostatkem operační paměti.



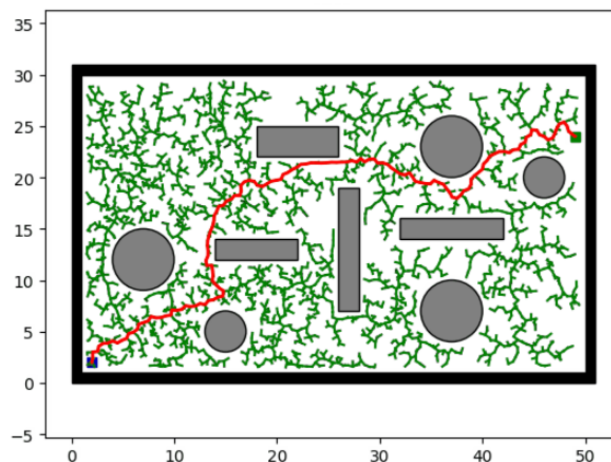
Obrázek 2.5: Ukázka algoritmu PRM

EST

Tento algoritmus se soustředí na rychlé prohledání prostoru, tím způsobem, že pomocí náhodných bodů vytváří v prostoru stromovou strukturu. Ta se rozrůstá každou iterací ze všech existujících větví zároveň. Díky tomu je velice rychlý v prohledávání stavových prostorů, ale trasy, které plánuje, nejsou ani zdaleka optimální. Kvůli velkému počtu náhodných vzorků se ve vyšších dimenzích stává detekce kolizí časově náročnou, a proto se při EST používá "líná" detekce kolizí. Ta se snaží detekovat kolize pouze když je to nutné a ne při každé expanzi stromu. Nejčastěji se využívá pro navigaci ve vysokodimenzních prostorech, který zvládá rychle prohledat. Obrázek sdílí s následujícím algoritmem z toho důvodu, že oba vytváří teoreticky stejnou stromovou strukturu, pouze jiným způsobem.

RRT

Stejně jako předchozí algoritmus RRT vyhledává cestu tím způsobem, že se snaží za pomoci náhodných bodů vytvářet stromovou strukturu. Je zaměřené více na prohledávání prostoru, kdy jednotlivé větve zvládnou prohledat větší kus prostoru s menším počtem bodů. Obdobně jako EST vyniká RRT v komplexnějších prostorech, které zvládá prohledat rychle a najít přijatelnou i když suboptimální cestu. RRT vyniká svojí přímočarostí a množstvím rozšířených verzí, které řeší různé jeho nedostatky.



Obrázek 2.6: Ukázka algoritmu RRT (přibližně EST)

Nejvhodnější algoritmus

Je důležité poznamenat, že všechny představené algoritmy, i mnoho dalších, jsou použitelné pro řízení manipulátorů. Většinou je však potřeba využít logiku více než jednoho algoritmu pro dosažení optimálních výsledků.

Při zvolení nejvhodnějšího algoritmu pro tuto práci byla zvažována tato kritéria:

1. Funkčnost algoritmu ve vyšších dimenzích
2. Preferování rychlosti nad optimální trasou v rozumné míře
3. Výstupy algoritmu a jeho aplikovatelnost na řízení manipulátoru
4. Rozšířenost a množství podkladů k algoritmu

Ačkoli některé search-based algoritmy, jako například vektorová pole, jsou schopné pracovat v 6D. Pro většinu jsou vyšší dimenze příliš náročné na výpočetní čas nebo je jejich rozšíření do nich příliš komplikovaný v poměru k jejich účinnosti. Proto byly zvažovány především sampling-based algoritmy, které jsou pro tyto problémy lépe uzpůsobeny (s vyjímkou PRM). Nejvhodnější algoritmus se ukázal být RRT. Důvodem jeho výběru je nejen poměr rychlosti a schopnosti vytvořit přijatelnou cestu, který sdílí s EST, ale především rozšířenost toho algoritmu ve světě robotiky. Bylo důležité přihlédnout nejen k praktickým a potenciálním možnostem těchto algoritmů, ale také tomu, jestli existuje dostatek podkladů k jejich studiu a náročnost jejich implementace. RRT je tedy sampling-based algoritmus velice vhodný pro řízení manipulátoru, který disponuje velkým množstvím podkladů, videí a nádstavbových verzí, což snížilo riziko budoucích komplikací.

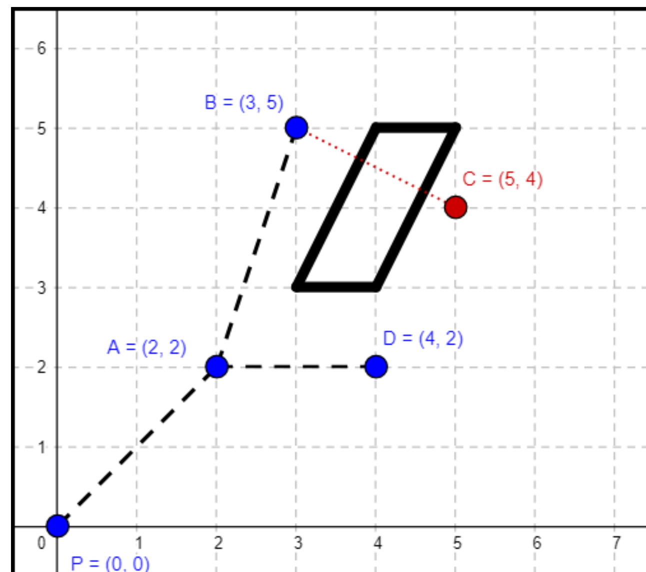
3. RRT a RRT*

RRT, celým názvem Rapidly-Exporing Random Trees je sampling-based algoritmus, který byl pro tuto práci vybrán jako nejvhodnější. Jedná se o vysoce univerzální a pro robotiku důležitý algoritmus, který poprvé v roce 1998 navrhl Steven M. LaValle a James J. Kuffner Jr.. Do dnešního dne slouží jako základní kámen nebo inspirace pro algoritmy používané k prohledávání stavových prostorů a navigaci moderních robotů.

3.1 Princip fungování

Základní myšlenkou RRT je rychle prozkoumat prostor s překážkami pomocí náhodných bodů v prostoru. Pomocí těchto bodů je vytvořen strom, který rychle ačkoli náhodně prohledává daný stavový prostor. Pokaždé, když je vytvořen nový bod, vyhodnotí se jeho vzdálenost od ostatních bodů v prohledávacím stromě. Poté je spojen s nejbližším bodem ve stromě, který se stává jeho rodičovským bodem a on sám se stává potomkem. Po dosažení limitního počtu nových bodů nebo časového limitu je vybrán nejbližší bod k cíli a zpětně je pomocí rodičovských bodů vytvořena cesta k počátku.

Pro vysvětlení použijeme prostor 10x10 centimetrů, který můžeme vidět na obrázku 3.1. Každý nový náhodný bod v tomto prostoru bude mít souřadnice v oboru celých čísel (Například bod A = [1;3]). Dále uvažujeme počátek P se souřadnicemi [0;0].



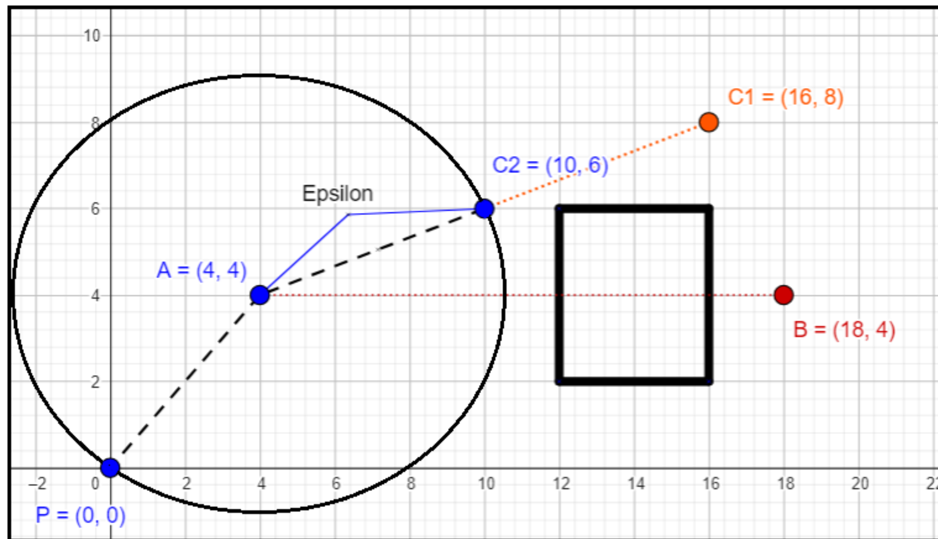
Obrázek 3.1: Základy RRT

Při první iteraci skriptu je vygenerován náhodný bod A=[2;2]. Poté je potřeba zjistit, zda spojnice mezi body P a A prochází překážkou nebo se bod A nachází přímo v překážce. Pokud ke kolizi došlo, tak je bod terminován a program přechází do další iterace. V našem případě ke kolizi nedošlo, a proto jsou body propojeny s tím, že se bod A stává potomkem bodu P. Následuje druhá iterace a vytvoření dalšího náhodného bodu B=[3,5]. Protože v prostoru už není pouze jeden bod, je potřeba změřit vzdálenost bodu B od bodu A i P. Tu změříme pomocí rovnice (3.1) pro euklidovskou vzdálenost. Pokud je to možné, dojde ke spojení s nejbližším bodem, v našem případě s bodem A. Pokud by ale spojnice s nejbližším bodem byla v kolizi s překážkou, bude tento bod odstraněn. Pro ukázku použijeme body C=[5;4] a D=[4;2]. Nejbližším sousedem bodu C je bod B, který se ale nachází za překážkou. Proto bod C nebude uvažován v našem prohledávacím stromě. Narozdíl od něj bod D může být ke stromu připojen.

$$d = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2} \tag{3.1}$$

Ve větších prostorech tato jednoduchá varianta nestačí. Tentokrát uvažujeme mnohem větší prohledávaný prostor s více překážkami. V takovém případě by algoritmus vyžadoval body na specifických místech, aby rozšiřoval prohledávací strom a velice často by se stávalo, že by se vracel zpátky, místo toho aby postupoval kupředu. Proto je potřeba rozšířit jej o funkci, která omezí vzdálenost, ve které jsou nové body stromové struktury vytvářeny.

Pokaždé, když je vytvořen náhodný bod, je mu stejně jako v předchozím příkladě přidělen nejbližší soused. Poté je mezi těmito body zkontrolována kolize. Rozdíl je v tom, že body nejsou propojeny ihned, ale nejdříve je zkontrolováno, jestli se tento náhodný bod nachází v limitní vzdálenosti epsilon (ve scriptu EPS) od tohoto sousedního bodu. Pokud ano, jsou propojeny a dojde ke kontrole kolizí. Pokud se bod nachází mimo, tak je na spojnici náhodného a sousedního bodu ve vzdálenosti epsilon vytvořen nový bod, který nahradí náš původní náhodný bod. Toto je pouze jedna z možností, jak tohoto výsledku dosáhnout. Je také možné rozšířit generování náhodných bodů tak, že se nevytváří v celém prostoru, ale pouze ve vzdálenosti epsilon kolem posledního bodu. Důvodem, proč byl zvolen první způsob je ten, že kontrola kolizí probíhá stále mezi sousedním a původním náhodným bodem. Strom se tedy rozrůstá po krocích daných pomocí EPS, ale získává určitou formu informovanosti. V případě kdy se náhodný bod vytvoří za překážkou nebo v ní, tak i když by se nový zkrácený bod mohl vytvořit a nedošlo by ke kolizi, není tak učiněno. Strom se tedy nerozrůstá zbytečně směrem, kde detekoval překážku. Je potřeba si být vědom, že při velké hustotě překážek by tento systém vedl naopak k terminaci většiny bodů. V takovém případě je lepší kontrolovat kolize mezi sousedním a nově zkráceným bodem.



Obrázek 3.2: Omezení vzdálenosti nového bodu pomocí EPS

3.2 Vlastnosti a využití

Jednou z hlavních výhod algoritmu je schopnost naplánovat trajektorii v komplexních a vysokodimenzionálních prostorech. Není pro něj totiž rozdíl mezi náhodným bodem ve 2D či 6D prostoru a jediné místo v kódu, které je navyšováním dimenzí ovlivněno, je kontrola kolizí. Ta často také neprobíhá v 6D, ale pouze ve 3D mezi simulovaným manipulátorem a překážkami. Z tohoto důvodu je nejčastěji používán ke generování bezkolizních trajektorií pro industriální roboty s vysokým počtem stupňů volnosti. Dále se využívá v navigaci autonomních vozidel ve složitém a dynamickém prostředí nebo také plánování komplexních pohybů humanoidních robotů.

Další výhodou algoritmu je možnost vyladit jeho parametry k řešení daného problému. Je možné nastavit větší vzdálenost EPS, což povede k více chaotické trase, ale prostor se bude prohledávat po větších krocích. V takovém případě není potřeba velkého množství bodů, což zkrátí výpočetní dobu. Naopak pro získání hladší a přímější trasy je možné nastavit vzdálenost EPS na menší hodnotu. Zde je potřeba navýšit počet bodů, jinak by se mohlo stát, že strom se nezvládne rozrůst prostorem a najít cestu k cíli. Právě díky možnosti kompromisu mezi výpočetním časem a optimální trasou se stal RRT algoritmus jedním z nejpopulárnějších algoritmů k prohledávání stavových prostorů.

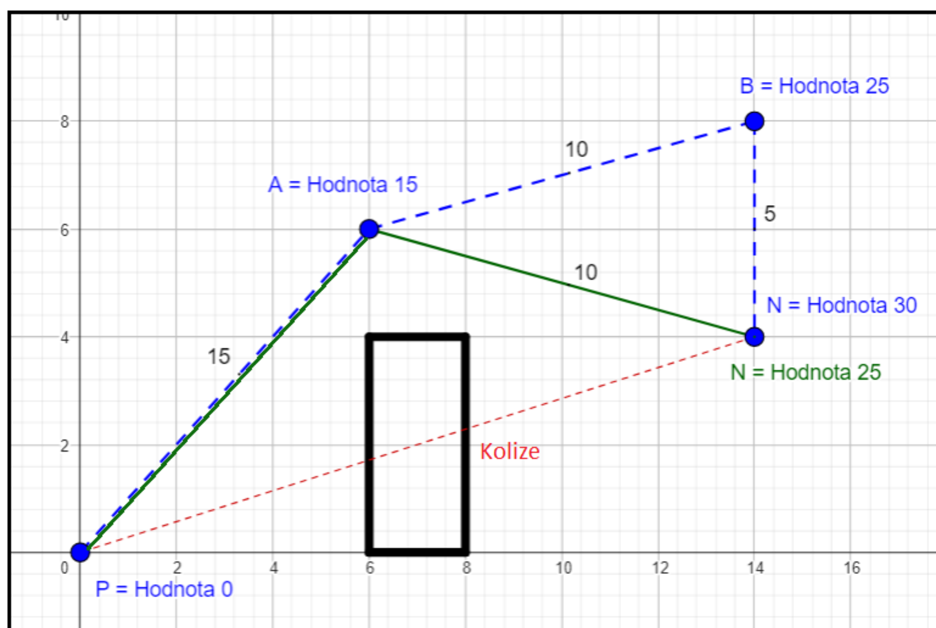
Nevhodným se tento algoritmus stává tehdy, když je požadováno, aby vytvořená trasa byla ta nejkratší. Klasické RRT není schopné takovou cestu nalézt, a to kvůli samotnému způsobu fungování algoritmu a důvodu jeho rychlosti. Není totiž zatěžován žádnou heuristickou funkcí, či jinými způsoby systematického plánování trasy. Vytvořit tedy optimální trasu pro něj není možné. Tento nedostatek je možné eliminovat použitím jednou z rozšířených verzí RRT. V našem případě se jedná o RRT*.

Pro úkoly, kde je vyžadována vyšší přesnost, je lepší použít optimalizovanou verzi RRT*. Na rozdíl od klasického RRT probíhá při každé iteraci přepojení prohledávacího stromu tak, aby ve výsledku byla vytvořena kratší a přímější cesta. Kvůli tomu je ale RRT* značně pomalejší, protože pokaždé, když je přidán nový bod, tak je celý strom znovu přehodnocen. V teorii je možné, aby RRT* nalezl optimální a nejkratší cestu pokud se počet bodů limitně blíží nekonečnu. V praxi tato situace nemůže nastat, protože čím více bodů se ve stromě nachází, tím delší je i výpočetní čas jednotlivých iterací. I přesto zvládne RRT* efektivně vyhladit a vytvořit přijatelnou trasu v krátkém čase, pokud jsou parametry správně nastaveny.

3.3 Rozšíření na RRT*

Základní princip je přiřazení hodnoty jednotlivým bodům. Díky té je možné vyhodnotit, které body ve stromě jsou výhodné k propojení, a které se naopak vyplatí vynechat. Pro vysvětlení využijeme situaci z obrázku 3.3.

V normálním případě by byla hodnota přidělena bodu ihned po jeho vytvoření, ale my musíme nejdříve zpětně určit hodnotu všech bodů ve stromě. Ta je rovna vzdálenosti jednotlivých bodů od počátku. Nejmenší hodnotu nula bude mít počátek P. Dále hodnota bodu A bude rovna euklidovské vzdálenosti bodu A od bodu P. Každý další bod, který je do stromu přidán, bude mít hodnotu rovnou součtu vzdálenosti od jeho rodičovského bodu a hodnotě tohoto rodičovského bodu. Na obrázku jsou znázorněné hodnoty všech bodů v našem prostoru.



Obrázek 3.3: Přepojování stromové struktury pomocí RRT*

V našem prostoru se nachází body P,A a B. Naším cílem je nově vytvořený bod N připojit k prohledávacímu stromu nejvhodnějším způsobem. Na obrázku 3.3 vidíme, že vynechání bodu B a spojení přímo s bodem A vytvoří přímější cestu. Aby dosáhl stejného závěru i algoritmus RRT* je potřeba zjistit, jestli by v takovém případě měl nový bod N menší celkovou hodnotu. Vezmeme tedy hodnotu bodu N, která mu byla přiřazena RRT algoritmem a porovnáme ji s hodnotou, které by nabyl, kdyby byl spojen přímo s bodem A. Na obrázku 3.3 můžeme vidět, že hodnota bodu N propojeného k bodu B je 30. Nadruhou stranu pokud by byl připojen přímo k bodu A, tak bude hodnota pouze 25. Protože je tato hodnota menší, jedná se o kratší cestu. Dále bude použita jako hodnota referenční a RRT* ji porovná s dalšími potenciálními hodnotami se zbylými body ve stromě. Tímto způsobem RRT* vždy najde nejlepší možné spojení. Při tomto přepojování je však pokaždé kontrolována kolize a jak je vidět na obrázku 3.3, tak při pokusu spojit bod N s počátkem P, což by byla nejkratší možná cesta, tak byla detekována kolize. Z toho důvodu zůstal bod N propojen s bodem A. Toto spojení probíhá ve skriptu tak, že bodu N je místo bodu B přiřazen jako rodič bod A. Vzhledem k množství bodů je toto samozřejmě nemožné dělat pro každou dvojici bodů, a proto v praxi používáme okolo nově vytvořeného bodu poloměr "R", který omezuje dosah, ve kterém se RRT* pokouší strom přepojit. Čím větší tento poloměr je, tím algoritmus vytvoří více vyhlazenou cestu, ale za cenu větší výpočetní doby.

4. Implementace algoritmu RRT*

V této kapitole se budeme věnovat tomu, jakým způsobem je RRT* algoritmus využit k řízení průmyslového manipulátoru o šesti stupních volnosti. Nejprve je potřeba si vysvětlit, jakým způsobem je algoritmus přízpůsoben na řízení takového manipulátoru. Dále se budeme věnovat samotné detekci kolizí, která jak víme z předchozí kapitoly, je tou nejdůležitější a časově nejnáročnější částí kódu. Podrobně se podíváme na specifické problémové části a rozebereme samotnou problematiku, která vzniká ve vyšších dimenzích. Vysvětlíme si, jak s celým skriptem pracovat a ověříme si jeho funkčnost v praxi.

4.1 Přepis algoritmu a vysvětlení skriptu

Nejdříve je důležité si vysvětlit, jakým způsobem je RRT* algoritmus využit v případě řízení průmyslového manipulátoru. Ten totiž neprohledává kartézský prostor, ve kterém se pohybuje manipulátor, ale místo toho zkoumá prostor úhlových souřadnic. Tento prostor má 6 dimenzí, kdy každá dimenze odpovídá jednomu stupni volnosti manipulátoru. Tento prostor má velikost ve všech osách $\langle 0;4\pi \rangle$. Každý bod v tomto prostoru je vektor o šesti souřadnicích, kdy každá jednotlivá souřadnice reprezentuje úhel natočení "theta" jednoho specifického kloubu.

Ve skriptu nazýváme všechny body slovem "Node". Ty jsou zapsané ve struktuře, kdy každý bod má své souřadnice, číselné označení svého rodičovského bodu a hodnotu. Pokaždé, když je vytvořen nový bod "randomNode", je zjištěn jeho nejbližší soused "nearNode". Abychom tohoto docílili je potřeba funkce na měření vzdálenosti "dist" (Ukázka kódu 4.1), která používá výpočet euklidovské vzdálenosti v 6D prostoru.

Ukázka kódu 4.1: Fuknce měřící euklidovskou vzdálenost dvou bodů v 6D

```
function C = dist(A,B)

C = sqrt((A(1)-B(1))^2 + (A(2)-B(2))^2 + (A(3)-B(3))^2 + ...
    (A(4)-B(4))^2 + (A(5)-B(5))^2 + (A(6)-B(6))^2);

end
```

Poté příjde na řadu přiblížení náhodného bodu pomocí funkce "steer" (Ukázka kódu 4.2). Ta funguje způsobem, jaký byl popsán v kapitole 4.1. Vstupují do ní dva měřené body, původní vzdálenost a požadovaná vzdálenost EPS. Poté, pokud je původní vzdálenost větší než epsilon, jsou pomocí analytické geometrie vypočteny nové souřadnice a vzniká nový bod, který nazveme "newNode".

Ukázka kódu 4.2: Funkce nahrazující náhodný bod novým bodem ve vzdálenosti EPS

```
function A = steer(rNode, nNode, val, eps)
   NewNode = [0 0 0 0 0 0];

   if val >= eps
       newNode(1) = nNode(1) + ((rNode(1)-nNode(1))*eps)/dist(rNode,nNode);
       newNode(2) = nNode(2) + ((rNode(2)-nNode(2))*eps)/dist(rNode,nNode);
       newNode(3) = nNode(3) + ((rNode(3)-nNode(3))*eps)/dist(rNode,nNode);
       newNode(4) = nNode(4) + ((rNode(4)-nNode(4))*eps)/dist(rNode,nNode);
       newNode(5) = nNode(5) + ((rNode(5)-nNode(5))*eps)/dist(rNode,nNode);
       newNode(6) = nNode(6) + ((rNode(6)-nNode(6))*eps)/dist(rNode,nNode);
   else
       newNode(1) = rNode(1);
       newNode(2) = rNode(2);
       newNode(3) = rNode(3);
       newNode(4) = rNode(4);
       newNode(5) = rNode(5);
       newNode(6) = rNode(6);
   end
   NewNode = [newNode(1), newNode(2), newNode(3), newNode(4), newNode(5), newNode(6)];
end
```

Dále jsou zkontrolovány kolize mezi randomNode a nearNode, je aktualizována hodnota nově vytvořeného bodu newNode a přesouváme se do RRT* sekce kódu, jehož úryvek 4.3 vidíme níže. Zde probíhá kontrola vzdálenosti nového bodu od všech ostatních ve stromě. Pokaždé když je vzdálenost některého z bodů menší než poloměr R, který je zadán uživatelem, proběhne kontrola kolizí mezi novým a tímto bodem. V případě, že je podmínka splněna a ke kolizi nedošlo, tak jsou souřadnice a hodnota tohoto bodu zapsány do struktury "nearestNode". Ta bude nakonec obsahovat všechny bezkolizní body ve vzdálenosti R.

Ukázka kódu 4.3: Zjištění nejbližších sousedních bodů v poloměru R

```
nearestNode = [];
neighborNodes = 1;

    for j = 1:1:length(nodes)
         if dist(nodes(j).position, newNode.position) <= R && ...
             jeKolize(nodes(j).position, newNode.position)

              nearestNode(neighborNodes).position = nodes(j).position;
              nearestNode(neighborNodes).value = nodes(j).value;
               neighborNodes = neighborNodes+1;
        end

     end
```

Když máme všechny nejbližší body vytříděny, můžeme přejít k samotnému přepojení stromové struktury z úryvku kódu 4.4. Nejprve si definujeme bod, ke kterému bude náš nový bod připojen jako "minNode" a minimální hodnotu "minValue". Ty se ze začátku rovnají původnímu "nearNode" a hodnotě nového bodu. Proběhne kontrola, zda by hodnota nového bodu byla menší, pokud by byl propojen s právě zkoumaným bodem, a poté se zkontrolují kolize pro jejich propojení. V případě, že jsou obě tyto podmínky splněny, je "minNode" změněna na hodnotu rovnající se zkoumanému bodu a "minValue" se bude rovnat nové menší hodnotě. Takto jsou prozkoumány všechny

"nearestNode" body a nalezeno nejkratší možné připojení nového bodu. Poslední část RRT* pouze najde pomocí for cyklu bod odpovídající pozici "minNode" a určí tento bod jako rodiče nového bodu. Tím je nový bod vytvořen, připojen k prohledávacímu stromu a zapsán do matice všech bodů ve stromě.

Ukázka kódu 4.4: Přepojování stromové struktury na trasy s nejmenší hodnotou

```
minNode = nearNode;
minValue = newNode.value;

    for k = 1:1:length(nearestNode)
         if nearestNode(k).value + ...
            dist(nearestNode(k).position , newNode.position) < minValue && ...
            jeKolize(nearestNode(k).position , newNode.position)

                minNode = nearestNode(k);
                minValue = nearestNode(k).value + ...
                           dist(nearestNode(k).position , newNode.position);
         end
    end

    for j = 1:1:length(nodes)
         if nodes(j).position == minNode.position
                newNode.parent = j;
         end
    end

nodes =  [nodes newNode];
```

Nakonec je potřeba zjistit, který z bodů je nejblíže k cíli. V ukázce kódu 4.5 máme vektor "nodes" obsahující všechny body stromové struktury. Pomocí cyklu "for" zjistíme vzdálenost všech bodů stromové struktury od cíle, a poté vybereme ten nejbližší. Tento bod se stane rodičovským bodem cíle, který je tímto připojen ke stromu. Poté jsou pomocí cyklu "while" zpětně dohledány souřadnice všech bodů od cíle po počátek a uloženy v matici "CoordM". Tyto body jsou konfigurace manipulátoru. Ty jsou poté přizpůsobeny knihovně poskytnuté v mechatronické laboratoři, která řídí manipulátor.

Ukázka kódu 4.5: Zpětné dohledávání nejkratší trasy od cíle ke startu

```
nodes
D = [];

    for j = 1:1:length(nodes)
        tempDistance = dist(nodes(j).position , goal.position);
        D = [D tempDistance];
    end

[val , idx] = min(D);
goal.parent = idx;
End = goal;
nodes = [nodes goal];
CoordM = [];

    while End.parent ~= 0
        next = End.parent;
        CoordM= [CoordM; End.position];
        End = nodes(next);
    end
```

4.2 Funkce pro detekci kolizí

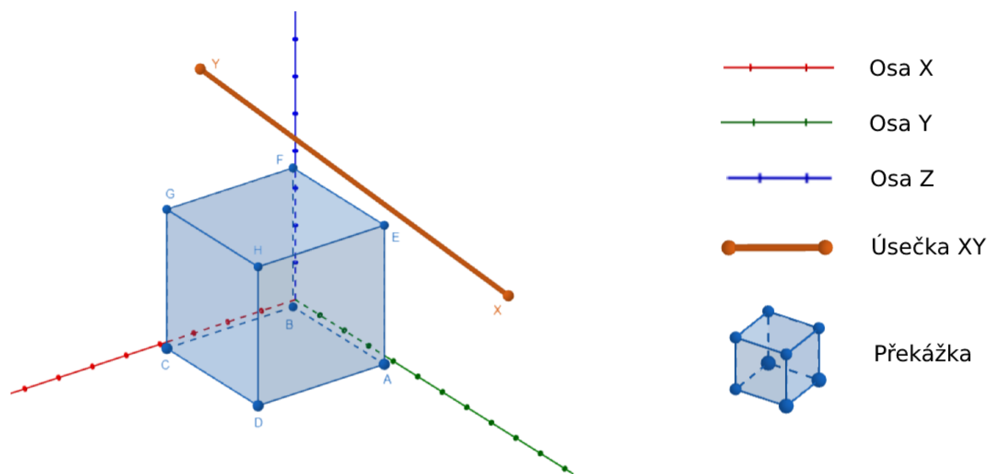
Rychlost RRT* je kromě samotného těla algoritmu značně ovlivněna i zpracováním detekce kolízí. Kolize jsou kontrolovány někdy i v řádů tisíců během jediné iterace. Je proto důležité, aby byla tato funkce co nejefektivnější. V našem případě nekontrolujeme kolize spojnice mezi dvěma body a překážkou v 6D prostoru, ale využijeme toho, že každý bod reprezentuje určitou konfiguraci manipulátoru. Kontrolujeme tedy kolize manipulátoru s kvádrovými aproximacemi překážek v 3D prostoru, ve kterém se pohybuje. Na to použijeme dvě funkce, které dohromady tvoří detekci kolizí.

První funkce se nazývá "jeKolize". Vstupují do ní dva body, mezi kterými je potřeba zkontrolovat kolize. Spojnice těchto bodů je reprezentována několika dalšími body, které jsou vytvořeny pomocí funkce "linspace". Všechny tyto body jsou poté rozděleny na jednotlivé souřadnice a postupně jsou zpracovávány funkcí "transmat". Tato funkce v sobě obsahuje transformační matice, které můžeme vidět v souboru rovnic 4.1, pomocí kterých se vypočtou koncové body jednotlivých ramen manipulátoru. Tato funkce obsahuje také rozměry manipulátoru UR5e.

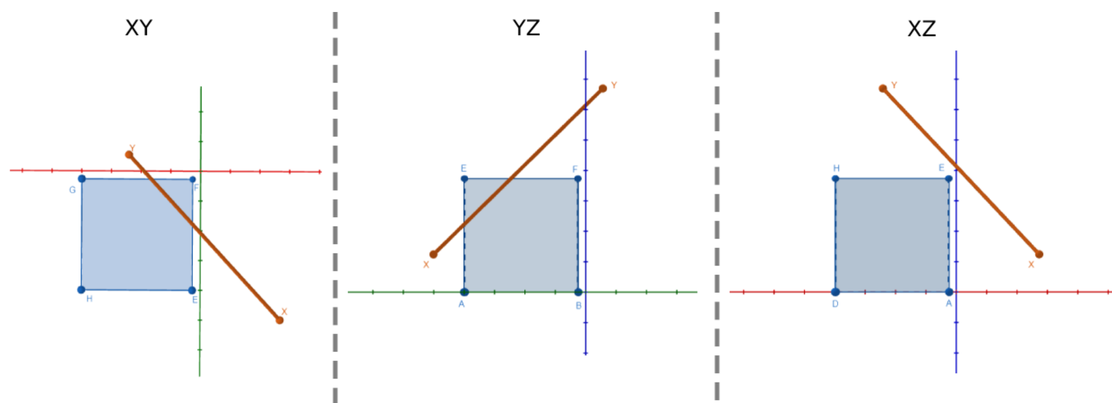
$$
T_{10} = \begin{bmatrix} cos_1 & -sin_1 & 0 & 0 \\ -sin_1 & cos_1 & 0 & 0 \\ 0 & 0 & 1 & a_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{21} = \begin{bmatrix} 1 & 0 & 0 & -a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{32} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos_2 & -sin_2 & a_3 * cos_2 \\ 0 & sin_2 & cos_2 & a_3 * sin_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$
$$
T_{43} = \begin{bmatrix} 1_0 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{54} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos_3 & -sin_3 & a_4 * cos_3 \\ 0 & sin_3 & cos_3 & a_4 * sin_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{65} = \begin{bmatrix} 1 & 0 & 0 & -a_5 \\ 0 & cos_4 & -sin_4 & 0 \\ 0 & sin_4 & cos_4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$
$$
T_{76} = \begin{bmatrix} cos_5 & -sin_5 & 0 & 0 \\ sin_5 & cos_5 & 0 & 0 \\ 0 & 0 & 1 & a_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{87} = \begin{bmatrix} 1 & 0 & 0 & -a_6 \\ 0 & cos_6 & -sin_6 & 0 \\ 0 & sin_6 & cos_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\quad
T_{98} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos_6 & -sin_6 & 0 \\ 0 & sin_6 & cos_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\tag{4.1}
$$

Spojením těchto bodů získáme úsečky v 3D prostoru, které jsou reprezentace reálného manipulátoru. Tyto úsečky jsou nyní zpracovávány funkcí "interCheck", ve které dochází ke kontrole kolize jednotlivých ramen manipulátoru s překážkou. Pokud žádné z ramen překážkou neprochází, výstup funkce je 1 a cyklus pokračuje do další iterace. Tímto způsobem jsou zkontrolovány postupně všechny body. Pokud je detekována kolize, výstup funkce je 0 a dojde k přerušení "for" cyklu.

Nyní se podrobněji podíváme na princip, kterým je kolize úseček a krychle detekována. Ten je založen na myšlence tří pohledů. Ten funguje tím způsobem, že situaci promítneme do rovin xy, yz a xz. Řešenou situaci máme na obrázku 4.1. Místo detekce průniku úsečky s krychlí, budeme řešit průnik tří úseček s třemi čtverci, které vznikly rozložením této situace na jednotlivé pohledy, jak je možné vidět na obrázku 4.2. Ke kolizi dojde pouze v případě, že došlo k průniku ve všech třech pohledech, což se v našem případě nestalo.



Obrázek 4.1: Pohled na překážku s úsečkou ve 3D a legenda



Obrázek 4.2: Metoda rozložení pohledů použitá k detekci kolizí

Zjednodušili jsme si tedy situaci z 3D na 2D, ale pořád je potřeba vyřešit průsečík úsečky se čtvercem. Toho docílíme tím, že si tento čtverec rozdělíme na čtyři strany AB, BC, CD a AD, čímž si příklad zjednodušíme na hledání průniku dvou úseček. Ten zjistíme pomocí funkce "lineintr",kterou můžeme vidět v úryvku 4.6 níže. Ta pomocí analytické geometrie zvládne vypočíst souřadnice průniku dvou přímek. Ten existuje vždy, pokud nejsou tyto dvě přímky rovnoběžné. Tím, že jsou definované vždy dvěma body, je možné dopočítat jejich parametr, který bude v případě průniku, nacházejícím se mezi těmito dvěma body v rozmezí $\langle 0;1 \rangle$. Díky tomu můžeme říci, že dvě úsečky se protínají, pokud parametry obou přímek, na kterých leží jsou v rozmezí $\langle 0;1 \rangle$.

Ukázka kódu 4.6: Detekce průsečíku dvou úseček pomocí analytické geometrie

```
function intersection = lineintr(A,B,C,D)
    tTop =    (D(1)-C(1))*(A(2)-C(2))-(D(2)-C(2))*(A(1)-C(1));
    uTop =    (C(2)-A(2))*(A(1)-B(1))-(C(1)-A(1))*(A(2)-B(2));
    bottom = (D(2)-C(2))*(B(1)-A(1))-(D(1)-C(1))*(B(2)-A(2));
    t = tTop/bottom;
    u = uTop/bottom;

    if (t >= 0) && (t <= 1) && (u >= 0) && (u <= 1)
        intersection = 1;
    else
        intersection = 0;
    end
end
```

Pokud nedojde k protnutí ani s jednou z nich, nedošlo ani k protnutí se samotným čtvercem. Aby platilo, že úsečka prochází krychlí, můsí dojít k průniku úsečky a čtverce ve všech třech pohledech. Jedinou vyjímkou je případ, kdy ve dvou pohledech dochází k průniku a ve třetím se oba konce úsečky nachází uvnitř vzniklého čtverce.

Ukázka kódu 4.7: Detekce průsečíků všech hran překážky s úsečkou ve všech pohledech

```
intr1  = lineintr(obstacle(1,1:2),obstacle(2,1:2),pointx(1:2),pointy(1:2));
intr2  = lineintr(obstacle(2,1:2),obstacle(3,1:2),pointx(1:2),pointy(1:2));
intr3  = lineintr(obstacle(3,1:2),obstacle(4,1:2),pointx(1:2),pointy(1:2));
intr4  = lineintr(obstacle(1,1:2),obstacle(4,1:2),pointx(1:2),pointy(1:2));
intr5  = lineintr(obstacle(1,[1,3]),obstacle(2,[1,3]),pointx([1,3]),pointy([1,3]));
intr6  = lineintr(obstacle(2,[1,3]),obstacle(6,[1,3]),pointx([1,3]),pointy([1,3]));
intr7  = lineintr(obstacle(5,[1,3]),obstacle(6,[1,3]),pointx([1,3]),pointy([1,3]));
intr8  = lineintr(obstacle(1,[1,3]),obstacle(5,[1,3]),pointx([1,3]),pointy([1,3]));
intr9  = lineintr(obstacle(1,2:3),obstacle(5,2:3),pointx(2:3),pointy(2:3));
intr10 = lineintr(obstacle(5,2:3),obstacle(8,2:3),pointx(2:3),pointy(2:3));
intr11 = lineintr(obstacle(8,2:3),obstacle(4,2:3),pointx(2:3),pointy(2:3));
intr12 = lineintr(obstacle(1,2:3),obstacle(4,2:3),pointx(2:3),pointy(2:3));
```

Ve funkci "noCollision" jsou načteny všechny překážky ve formě 3D matice, ve které jsou souřadnice vrcholů všech překážek. Z této matice, stejně jako z bodů, které do této funkce vstupují, jsou systematicky vybírány souřadnice, které jsou vkládány do "lineintr" funkce, což můžeme vidět v úryvku ze skriptu 4.7. Tímto vzniknou proměnné intr1-12, které obsahují všechny informace potřebné pro vyhodnocení kolize. Pro každý z pohledů xy, xz a zy je zde podmínka, která kontroluje, zda v daném pohledu došlo

k průniku včetně výjímkového případu, kdy se oba body nachází uvnitř čtverce. V příkladu kódu 4.8 vidíme pouze poslední ze tří "if" podmínek kontrolující pohled zy a ukončení funkce "interCheck".

Ukázka kódu 4.8: Podmínka kontrolující kolizi v pohledu xz

```
if (intr9 == 1 || intr10 == 1 || intr11 == 1 || intr12 == 1) || ...
   (pointx(2)<obstacle(4,2) && pointx(2) > obstacle(5,2)     && ...
    pointx(3) > obstacle(4,3) && pointx(3) < obstacle(5,3)   && ...
    pointy(2)<obstacle(4,2) && pointy(2) > obstacle(5,2)     && ...
    pointy(3) > obstacle(4,3) && pointy(3) < obstacle(5,3))

    zy = 1;
else
    collision = 1;
    continue
end

if xy == 1 && xz == 1 && zy == 1
    collision = 0;
    break
end
```

Pokud je zjištěno, že k průniku v jakémkoli pohledu nedošlo, tak je výstup funkce 1 a ihned se přechází ke kontrole další překážky, protože kolize s touto překážkou již nemůže logicky nastat. Naopak pokud je zjištěna kolize s jednou z překážek, je výstup funkce 0 a kontrola kolize s dalšími překážkami se už neprovádí, protože by byla zbytečná. Málokdy se tedy stane, že je potřeba kontrolovat všechny možnosti, což velice zrychluje samotný algoritmus.

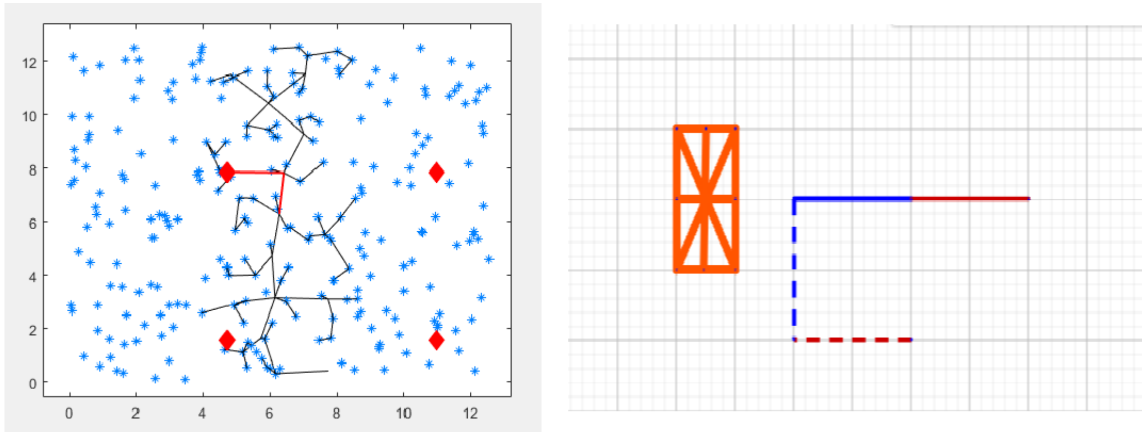
4.3 Komplikace a jejich řešení

Ačkoli by bylo vítané, kdyby se v průběhu programování nevyskytly žádné překážky, ve většině případů tomu tak není. Proto si v této podkapitole přiblížíme specifické části skriptu, které vyžadují zvýšenou pozornost a jsou nezbytné pro jeho fungování.

4.3.1 Vyřešené

Ten nejdůležitější problém, který se vyskytl při psaní skriptu vychází z toho, že natočení 4π, 2π a 0 jsou ta samá pozice. Pro každé rameno existují tím pádem vždy dvě souřadnice popisující jednu polohu. S narůstajícím počtem ramen, v našem případě dimenzí, roste i množina konfigurací popisující jednu polohu. My se pohybujeme v šesti dimenzích, a proto je celkové množství konfigurací 2^6 nebo-li 64. Pro lepší pochopení použijeme manipulátor, který má pouze 2 ramena ve 2D. V takovém případě je počet potenciální cílů 4. Náš algoritmus začne ve středu tohoto prostoru, což je v souřadnicích $[2\pi;2\pi]$. Pokud požadujeme, aby se manipulátor přesunul do pozice znázorněné na obrázku 4.3, existují čtyři kombinace souřadnic popisující tento stav, a to $[\frac{3}{2}\pi;\frac{1}{2}\pi]$, $[\frac{7}{2}\pi;\frac{5}{2}\pi]$,$[\frac{3}{2}\pi;\frac{5}{2}\pi]$ a $[\frac{7}{2}\pi;\frac{5}{2}\pi]$. Vizualizaci najdeme na obrázku 4.3 na další stránce. Hodnoty menší než 2π znamenají, že dané rameno se točí po směru hodinových ručiček a hodnoty nad 2π naopak proti směru. Můžeme vidět, že dva cíle jsou v kolizní zóně,

a to případy, ve kterých by se modré rameno točilo proti směru hodinových ručiček. V takovém případě by došlo ke kolizi. Souřadnice x musí být tedy $\frac{3}{2}\pi$. Červené rameno se může protočit oběma směry bez kolize, ale v případě, že by se točilo po směru hodinových ručiček muselo by se otočit o $\frac{3}{2}\pi$, zatím co proti směru se otočí pouze o $\frac{1}{2}\pi$. Proto je vybrána druhá možnost a souřadnice zvoleného cíle jsou $[\frac{3}{2}\pi;\frac{5}{2}\pi]$. Jak můžeme vidět, tak vzdálenost cíle od počátku je přímo úměrná společné úhlové vzdálenosti natočení všech ramen. Skript se vždy snaží preferovat nejkratší možnou cestu, ale pokud by nejbližší cíl byl znepřístupněn překážkou, tak je využit jiný cíl.



Obrázek 4.3: Vztah více cílů a směru rotace manipulátoru

V našem případě je tato situace mnohem komplikovanější kvůli exponenciálnímu nárůstu možných cílů. Je tedy potřeba upravit finální část skriptu, která propojuje strom s cílem. Pro lepší pochopení je k popisu přidána ukázka tohoto kódu 4.9. Nejdříve musíme najít všechny potenciální cesty vedoucí k požadované konfiguraci. Na to využijeme dva cykly for, které zkontrolují spojení každého bodu prohledávacího stromu a každého cíle. Pro každý cíl je vždy zjištěn právě jeden bod v okruhu Endlim, který má nejmenší hodnotu a zároveň není v kolizi. Toho je docíleno stejným způsobem vysvětleným v kapitole 5.1. Tentokrát je ale tato vzdálenost změřena a zapsána do matice "distGoal" a pořadí bodu, který byl vybrán je zapsáno do matice "finalparent". Tato kontrola tedy proběhne pro všechny možné cíle a do matice "distGoal" jsou přidávány hodnoty vzdáleností nebo v případě kdy s cílem neexistuje možné spojení tak NaN. To je z toho důvodu, aby bylo zachováno pořadí hodnot v matici, které vždy odpovídá pořadí cílů. Tímto se vytvoří soubor hodnot vzdáleností od startu do cíle a odpovídající seznam rodičovských bodů. Pomocí funkce min vybereme nejnižší hodnotu, zjistíme, který z cílů se k této hodnotě váže a přiřadíme mu rodičovský bod. Nyní už jen zpětně dohledáme cestu a zapíšeme jí do matice "CoordM", podle které je řízen manipulátor.

Ukázka kódu 4.9: Nalezení vhodného a bezkolizního cíle s nejkratší vzdáleností

```
G =[];
finalparent =[];

for k=1:1:length(goal.position)
    D = [];
    for j = 1:1:length(nodes)
        if dist(nodes(j).position,goal.position(k,:)) < Endlim && ...
           jeKolize(nodes(j).position,goal.position(k,:))

            tempDistance = dist(nodes(j).position, goal.position(k,:));
            D = [D tempDistance];
        else
            D = [D NaN];
        end
    end

    if isnan(min(D)) == 1
        skip=1;
    else
        skip=0;
    end

    [val, idx] = min(D);
    goal.parent = idx;
    v=0;
    tempgoal=goal;

    while tempgoal.parent ~= 0
        next = tempgoal.parent;
        v = v + dist(goal.position(k,:),nodes(next).position);
        tempgoal = nodes(next);
        if skip == 1
            v=NaN;
        end
    end

    distGoal = [distGoal v];
    finalparent = [finalparent goal.parent];
end
```

Další překážkou bylo propojení manipulátoru se skriptem. Nejprve bylo potřeba upravit transformační matice tak, aby odpovídaly těm, které jsou použity výrobcem. Dále manipulátor UR5e pracuje s natočením kloubů [-2π;2π]. Je tedy potřeba při komunikaci s manipulátorem úhly natočení přepočítávat. V případě startovní konfigurace přičítáme 2π a při výstupu cílové konfigurace naopak od všech hodnot odečteme 2π. V případě první a páté souřadnice je také nutno vynásobit mínus jednou a celou matici transponovat.

Přechod na reálný manipulátor odhalil i další nedostatek, který v digitálním prostředí nebyl podstatný, a to kolize manipulátoru sama se sebou. Ta nastává na dvou místech, v podstavě a u zápěstí. V případě podstavy bylo využito toho, že manipulátor se UR5e reálně nemůže narazit sám do sebe jinak, než cílovým efektorem do své podstavy. Pro tu však nejsou kontrolovány kolize, protože je nepohyblivá a zbytečně by byla zpomalována detekce kolizí. Proto byla do podstavy umístěna malá překážka, která nesmí být odstraněna.

4.3.2 Nevyřešené

Největší nedostatek skriptu je ten, že manipulátor je řízen v šesti dimenzích. Nevidí rozdíl mezi hlavními třemi rameny a klouby, které pohybují koncovým efektorem. Řeší všechny tyto úhly naráz a i když funguje přesně tak, jak byl navrhnut, tak výsledky jsou ve výjmečných případech neuspokojivé. Je to způsobeno tím, že v prostoru, ve kterém RRT* hledá cestu, hraje roli všech šest souřadnic, což znamená, že nejkratší vzdálenost není vždy optimální. Dochází k zbytečně velkým pohybům hlavních dvou ramen a někdy dokonce k vracení se zpět. To je samozřejmě očekávatelné vzhledem k volbě vyhledávacího algoritmu, ale ne v tak velké míře.

Tento problém je však možné vyřešit, ale znamenalo by to úplné přepsání skriptu. K tomuto nápadu vedlo až porozumění problematice řízení manipulátorů, které bylo získáno až prací na BP. Hlavní změnou by bylo, že nejprve proběhne vyhledávání trasy pro 3 hlavní ramena ve 3D. Poté co se hlavní část manipulátoru přesune do požadované pozice, proběhne RRT* ještě jednou, tentokrát pro 3 ramena, která nastavují pozici koncového efektoru. Časově by se tento postup příliš nelišil od původní verze. Došlo by však k mnohem přesnějšímu pohybu manipulátoru jako celku a především by bylo možné vizualizovat celý průběh. Díky tomu by byla umožněna mnohem lepší optimalizace založená na viditelných a pojmutelných podkladech. Zároveň by se zpříjemnil i proces debuggingu.

Tato změna by pomohla i při problému s kolizí zápěstí manipulátoru s vlastním ramenem zmíněném v kapitole 4.3.1. V tomto případě nebyl vyřešen. Algoritmus preferuje nejkratší cestu, která v případech, kdy je potřeba otočit základnu manipulátoru, zahrnuje často protočení zápěstí skrz vlastní rameno. Ačkoli algoritmus cestu v takových případech nalezne relativně snadno, tak manipulátor ji většinou nemůže vykonat. Zahrnutí těchto kolizí bylo téměř hotovo, ale nebylo zcela kompatibilní se způsobem, jakým jsou aktuálně kontrolovány kolize a na jeho správnou implementaci už nezbyl čas. Jednalo se o kontrolu úhlu natočení kloubů v kolizní funkci a pomocí podmínek vytřídit případy kdy jsou klouby 4 a 5 v pozici, která by vedla ke kolizi.

Poslední funkce, o jejíž implementace se nezdařila bylo zadávání cíle ve formě polohy koncového efektoru a jeho orientace. Jde o vcelku jednoduchý problém, protože jde pouze o základní výpočty inverzní kinematiky pro manipulátor s 6-ti stupni volnosti. Tento problém bylo tedy možné vyřešit použitím jednoho z mnoha skriptů pro inverzní kinematiku z GitHubu, které by potřebovaly pouze upravit, ale kvůli pokusu o hlubší pochopení a vlastní napsání této části kódu, nebyla tato funkce včas dokončena.

5. Využití v praxi

Bylo docíleno, aby bylo možné pomocí tohoto skriptu řídit manipulátor v prostoru s překážkami dle zadání. Manipulátor se pohybuje vždy z konfigurace A do konfigurace B. Pokud tedy je požadováno, aby byl koncový efektor manipulátoru v určité poloze, je potřeba, aby uživatel znal úhlové souřadnice všech kloubů, které zadá jako vektor. Po zadání a spuštění skriptu nejdříve dochází ke generování stromové struktury, kterou doprovází ukazatel "loading". Poté však ještě probíhá zvolení správného cíle, zmíněné v 5.3.1. Tato část skriptu nemá žádný ukazatel, přestože trvá obdobně dlouho, aby se zbytečně nezpomalovala. Poté se manipulátor bez kolize přesune do požadované pozice, program je ukončen a je potřeba zadat další požadovaný cíl.

5.1 Návod k použití

Pro spuštění programu je potřeba otevřít skript s názvem "RRTSTAR". Na řádku 6 se nachází vektor "endpos", kde je zadávána požadovaná cílová poloha manipulátoru. Pod tímto vektorem se nachází maximální počet bodů, které algoritmus vytvoří s názvem "NodeMax". Toto číslo je možné měnit podle potřeb uživatele. Výpočetní doba se bude lišit na základě komplexnosti prostředí, ve kterém se manipulátor nachází, ale v případě 4 překážek není doporučeno více než 5000. Pro zrychlení je možné počet zmenšit, ale v takovém případě může být cesta zbytečně nepřesná a vzniká riziko, že algoritmus nezvládne najít cestu. Dalším parametrem je EPS, které jsme si vysvětlili v kapitole 3.1. Doporučuje se ponechat hodnotu EPS $= 2\pi$, ale pokud je potřeba najít malé skuliny mezi překážkami, tak je možné hodnotu snížit. S tím je však doporučeno zvýšit počet maximálních bodů, protože prohledávací stromová struktura se bude rozrůstat pomalu. Poslední krok nastavení algoritmu je poloměr jeho RRT* části. Tato hodnota by vždy měla být přibližně dvojnásobně větší než vzdálenost EPS, aby bylo dosaženo smysluplného vyhlazení trasy. Tento parametr vysoce ovlivňuje výpočetní dobu skriptu, a proto by se neměl zadávat příliš vysoký. Pro možnost spuštění bez manipulátoru UR5e je nutné zakomentovat/odkomentovat určité části "RRTSTAR" skriptu podle README dokumentu.

Ukázka kódu 5.1: Zadání cílové lokace a nastavení parametrů

```
%Nastaveni cilove pozice
endpos = [j1 j2 j3 j4 j5 j6]

%Nastaveni parametru prohledavani
NodesMax = 1500;
EPS = 2*pi;
R = 4*pi
```

Poté je potřeba nastavit překážky. Na ty si otevřeme funkci "osb". Překážky jsou definované bodem A a rozměry. V obou případech se jedná o matici, kdy se jedna překážka zapisuje vždy na jeden řádek. Pokud tedy chceme přidat překážky je potřeba zadat souřadnice bodu A jako další řádek matice. Poté je potřeba připsat řádek i do matice "size", kde hodnoty odpovídají rozměru překážky, a to délce v ose x, šířce v ose y a výšce v ose z.

Ukázka kódu 5.2: Ukázka přidání další překážky

```
% 3 prekazky
sizes = [140 140 69;
          55  80 70;
          30  30 40];

  A_S = [ 0   0    0;
         85  50   95;
         40  20  100];
```

⟹

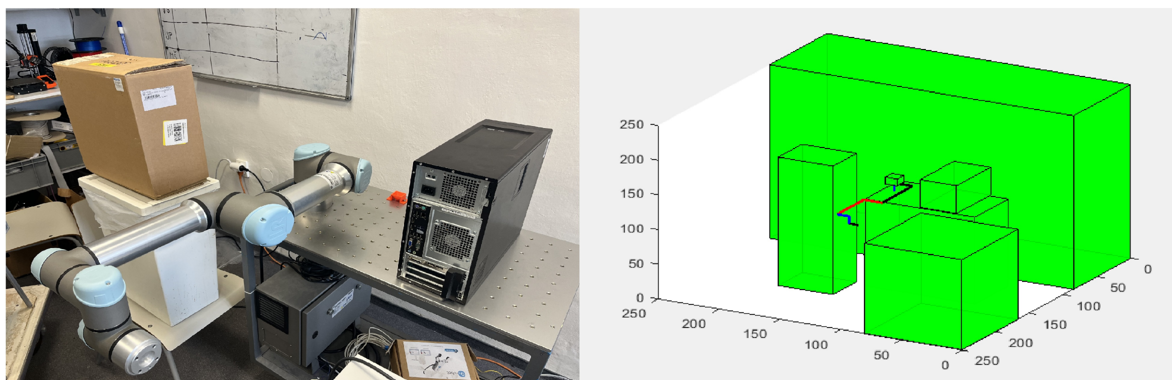
```
% 4 prekazky
sizes = [140 140 69;
          55  80 70;
          30  30 40;
          20  20 20];

  A_S = [ 0   0    0;
         85  50   95;
         40  20  100;
         20  20   20];
```

5.2 Ukázka

V této části si ověříme funčnost celého skriptu v praxi na manipulátoru UR5e. Tyto testy byly provedeny v mechlabu, kde překážky byly vytvořeny z okolních předmětů, a poté převedeny do digitální formy zvětšené zhruba o 10% . Překážky je vždy potřeba uvažovat větší než reálně jsou, protože skript pracuje pouze s úsečkami reprezentující střed ramen. Ve skutečnosti by pak mohlo dojít ke kolizi.

V prostoru s překážkami, které můžeme vidět na obrázku 5.1 byl manipulátor nastaven do neutrální polohy, kdy všechny jeho klouby měly natočení 0 radiánů. V první zkoušce byly zvoleny tři polohy, které dostal manipulátor za úkol projít. Metodou pokus-omyl bylo určeno nastavení parametrů pro získání nejlepší cesty v poměru k času. V druhé zkoušce byl proveden pohyb, který je pro algoritmus složitý a otestoval jeho limity. Videozáznam těchto zkoušek je k nalezení v příloze.



Obrázek 5.1: Ukázka prostoru

5.2.1 Základní pohyby

V této části zkoušky se manipulátor pohyboval mezi třemi body. Jejich souřadnice a nastavení parametrů můžeme vidět v úryvku níže.

Ukázka kódu 5.3: Cílové pozice a nastavení parametrů pro testování

```
% Cilove pozice zakladnich pohybu                 % Cilove pozice sloziteho pohybu
Pozice1_1 = [0 0 0 0 0 0];                         Pozice2_1 = [90 45 315 0 90 45];
Pozice1_2 = [90 45 315 0 90 45];                   Pozice2_2 = [270 45 315 0 0 90];
Pozice1_3 = [90 130 60 0 0 90];

% Zakladni nastaveni    % Zvysene vyhlazovani     % Nastaveni sloziteho pohybu
NodesMax = 750;         NodesMax = 1500;          NodesMax = 1500;
EPS = 2*pi;             EPS = 2*pi;               EPS = 2*pi;
R = 4*pi;               R = 8*pi;                 R = 4*pi;
```

První pohyb byl opakován třikrát, ve všech případech se základním nastavením parametrů. Poté byl proveden ještě jednou, ale tentokrát byly parametry změněny na speciální nastavení pro zvýšené vyhlazování z úryvku 5.3. V průběhu pohybů byl měřen výpočetní čas a počet bodů, které byly úspěšně připojeny ke stromové struktuře. Výsledky najdeme v tabulce 5.1:

Tabulka 5.1: Základní pohyb - První část

	Test 1	**Test 2**	**Test 3**	**Test 4**
Čas [s]	13,83	15,42	14,68	54,52
Počet bodů [-]	66 4	82 5	74	97

Můžeme si všimnout, že počet bodů v prohledávacím stromě je přímo úměrný výpočetní době. To je z toho důvodu, že v případě, že je bod bezkolizní, tak pro něj musí proběhnout většina kontrol, což protahuje výpočetní dobu. Co se však nemění je plynulost trasy manipulátoru. Ta byla ve všech třech případech téměř identická, což bylo očekávatelné. V případě testu 4, bylo sice vytvořeno dvakrát více náhodných bodů, ale ve stromové struktuře přibyla pouze polovina. To je způsobeno tím, že je prostor okupován velkým množstvím překážek a body mají větší pravděpodobnost být v kolizi než použitelné. Kvůli zvětšenému RRT* poloměru nabyl také čas, a to o víc jak trojnásobek. Trasa byla v tomto případě viditelně přímější, což znamená, že zvýšení poloměru "R" má smysl, pokud chce uživatel co nejpřesnější pohyb. Nutné však dodat, že jak jsme si vysvětlili v kapitole 3.2, RRT* není schopen najít nejkratší cestu, ale pouze se jí přiblížit.

Do pozice 2 se manipulátor přesouval z pozice 1. Opět byl tento pohyb proveden třikrát se základním nastavením a jednou se speciálním. Docíleno bylo těchto výsledků:

Tabulka 5.2: Základní pohyb - Druhá část

	Test 1	**Test 2**	**Test 3**	**Test 4**
Čas [s]	26,23	24,57	28,12	72,73
Počet bodů [-]	48	42	53	81

Při tomto pohybu můžeme vidět, že je zde stejná přímá úměrnost mezi body ve stromě a výpočetním časem jako v prvním testu, a to pouze s jediným rozdílem. Tím byl jejich poměr. Tentokrát bylo připojeno ke stromové struktuře méně bodů, přestože byl čas delší. To je pravděpodobně způsobeno, že v trase je manipulátor obklopen více překážkami. Ve všech třech případech byl proveden naprosto identický pohyb. Důvodem je, že zvolený cíl je v 6D prostoru, kde manipulátor hledá cestu menší než 4π. To je hodnota poloměru, ve kterém se algoritmus snaží spojovat cíl s body stromu. Ve všech případech byl tedy výsledek stejný a to včetně testu se speciálním nastavením. V tomto případě byl výpočetní čas více než 1 minutu, ačkoli výsledný pohyb byl stejný jako při předchozích třech měřeních. Zde můžeme vidět, že ne vždy je zvýšení poloměru "R" správná volba, protože dojde ke zvýšení času bez jakýchkoli výsledků. Pro ověření tvrzení byl tento test proveden dodatečně ještě jednou s "NodeMax = 0", což znamená, že nebyl vytvořen jediný bod. Výsledek i v tomto případě byl stejný.

Třetí pohyb byl z druhé pozice zpět do výchozí. Výsledky jsou dle očekávání obdobné jako v předchozích případech. Tentokrát bylo zvýšené vyhlazování opět zbytečné, protože produkovalo téměř stejnou trasu, ale s několikanásobným výpočetním časem. Výsledky třetího testu můžeme vidět v tabulce 3:

Tabulka 5.3: Základní pohyb - Třetí část

	Test 1	**Test 2**	**Test 3**	**Test 4**
Čas [s]	17,21	22,12	19,81	61,18
Počet bodů [-]	47 4	56	51	89

Z tohoto testování můžeme usoudit, že algoritmus funguje podle očekávání. Ve všech dvanácti případech byla nalezena trasa bez kolize a při použití stejných parametrů bylo dosaženo konzistentních výsledků. Tyto parametry byly však získány metodou pokus-omyl, kdy bylo potřeba vyzkoušet různé kombinace množství bodů "NodeMax", poloměru "R" pro RRT* a vzdálenosti "EPS". Pokud bylo sníženo "EPS", bylo potřeba zvýšit počet bodů, protože jinak nebyl prohledán celý prostor. Navíc se zvětšil počet bodů ve stromové struktuře, což zpomalovalo RRT* část algoritmu. Proto bylo potřeba snížit poloměr "R". Bylo potřeba určit hodnotu těchto parametrů, aby byl efektivně prohledán celý prostor a vytvořená trasa byla přijatelná, za relativně krátký výpočetní čas. Tento proces není ideální, a proto by bylo vhodné, aby skript zvládl tyto parametry určit automaticky na základě prostředí, požadovaného výpočetního času a složitosti pohybu.

5.2.2 Složitý pohyb

Pro ověření funkčnosti byl proveden pro algoritmus náročný pohyb, kdy bylo potřeba otočit celou základnou. Tento pohyb byl pro manipulátor komplikovaný z důvodu kolize zápěstí s ramenem manipulátoru popsaném v kapitole 4.3.2. V dlouhých a komplikovaných pohybech, kdy je po manipulátoru požadováno se zcela otočit, je nejkratší trasa, kterou algoritmus zvolí většinou v kolizi. Výsledek proto nebyl příliž závislý na parametrech, ale na náhodě, jestli se zápěstí pokusí protočit. Nebylo tedy možné objektivně hodnotit efektivnost skriptu v těchto případech, protože pohyby nebylo možné spolehlivě zopakovat. Je však důležité poznamenat, že v počítačové simulaci, kde ke kolizi nedocházelo byly všechny trasy použitelné a jejich přímost vysoce závisela na poloměru "R" od RRT* části algoritmu. V přílohách je ukázka jednoho z povedených pohybů, kdy k protočení nedošlo. Nastavení parametrů a konečné body tohoto pohybu jsou v úryvku 5.3 na začátku této podkapitoly. V tabulce 4 níže je možné vidět výsledky, kterých bylo v tomto případě docíleno.

Tabulka 5.4: Složitý pohyb

	čas [s]	**Počet bodů [-]**
Složitý pohyb	104	89,16

6. Závěr

Cílem této práce bylo napsat skript, díky kterému bude možné navádět průmyslový manipulátor, v našem případě UR5e, v prostoru s překážkami. Nejprve však bylo potřeba zvolit nejvhodnější algoritmus na plánování trasy v takovém prostoru, což znamenalo provedení rešerše na samotné řízení řízení manipulátorů (kapitola 2.1) a různé druhy plánovacích algoritmů (kapitola 2.2). Po nastudování problematiky byl zvolen algoritmus RRT, především kvůli jeho schopnosti pracovat v prostorech s vyšším počtem dimenzí, rychlosti, ale také objemného počtu podkladů, literatury a videí, díky kterým bylo možné jej zcela pochopit a implementovat bez zbytečných komplikací.

Vysvětlili jsme si, že typ algoritmů, mezi které RRT patří, se nazývá sampling-based. To znamená, že pracuje s náhodnými stavy, které poté "sampluje", nebo-li vybírá a pamatuje si pouze ty, které nejsou v kolizi. Více o tom jak tyto algoritmy fungují bylo řešeno v kapitole 2.2. Dále jsme si v kapitole 3.1 vysvětlili základní princip samotného RRT algoritmu. Zjednodušeně jsou vytvářeny náhodné body, které jsou vždy připojeny k jejich nejbližšímu sousedovi, pokud mezi nimi není překážka. Tímto způsobem je tvořena stromová struktura, která prohledává daný prostor a vyhýbá se překážkám. V práci jsme však nakonec nepoužili klasické RRT, ale jeho pokročilou verzi RRT*, která umí tuto stromovou strukturu "přepojovat" pokud je to možné bez kolize a tvořit přímější a méně náhodné trasy.

Následovalo použití tohoto algoritmu k řízení manipulátoru UR5e, čemuž se věnuje celá kapitola 4. Nejprve jsme si vysvětlili, že algoritmus nehledá cestu v kartézském prostoru, ve kterém se manipulátor pohybuje, ale v prostoru úhlových souřadnic. Tento prostor je 6D a jeho souřadnice odpovídají úhlům natočení jednotlivých kloubů. Každý bod tedy odpovídá určité konfiguraci manipulátoru a pro ty je poté kontrolována kolize. Detailně jsme si popsali funkci celého skriptu. Především byl kladen důraz na vysvětlení funkce pro detekci kolizí v kapitole 4.2, vzhledem k její důležitosti a vlastnímu zpracování. Nakonec jsme si v kapitole 4.3 prošli specifické komplikace a překážky, jak vyřešené, tak nevyřešené.

Na konci práce jsme si popsali jak pracovat se skriptem a otestovali jsme si jeho funkčnost v praxi. Kapitola 5.1 obsahuje "návod", který rychle uživateli vysvětlí základy nastavování parametrů a spouštění. Nejdůležitější část je však otestování funkčnosti, které proběhlo formou opakovaných testů, jejichž výsledek najdeme v kapitole 5.2. Tyto testy proběhly v mechatronické laboratoři, kde dostal manipulátor UR5e za úkol projít různé cílové polohy v prostoru s překážkami. Videozáznam z těchto zkoušek je v přílohách.

Zkouška skriptu byla rozdělena na dvě části, série méně náročných pohybů a komplikovanější pohyb, který měl za úkol otestovat algoritmus i v krajních případech. V první části byly výsledky testů s optimálním nastavením konzistentní. Jejich čas byl v intervalu 14-28 sekund a ve stromové struktuře bylo průměrně 65 bodů. V případě nastavení většího vyhlazování byl výpočetní čas zvýšen v průměru trojnásobně. Pouze v jednou případě bylo však vyhlazení výrazně viditelné a naopak v jednom případě bylo naprosto zbytečné. V druhé části se projevil problém s detekcemi kolizí manipulátoru sama se sebou. Ačkoli skript tyto komplikované problémy vyřešil opět v průměru za 20 sekund, tak manipulátor tyto pohyby nebyl schopen provést. Několikrát byla však vytvořena trasa, kde tato kolize nenastala a manipulátor se úspěšně přesunul do požadované polohy. Videozáznam najdeme v příloze.

Zvolený algoritmus RRT* se ukázal být vhodným k řízení manipulátoru v prostoru s překážkami. Jeho přepis a přizpůsobení k řízení manipulátoru bylo téměř úspěšné a skript funguje tak, jak bylo očekáváno. Všechny klasické pohyby zvládne algoritmus vyřešit velice rychle s přijatelnou trasou. Problém nastává v dlouhých pohybech, kdy se musí manipulátor například celý otočit. Nezvládnuté kolize manipulátoru sama se sebou, které nebyly vyřešeny včas, se v takovém případě projeví. V kapitole 4.3.2 je však možné řešení tohoto problému. Dále bylo docíleno závěru, že myšlenka řídit všechny klouby manipulátoru naráz v 6D není optimální. Vhodnější způsob by byl řídit nejprve tři hlavní ramena manipulátoru, a poté až zbylé tři k orientaci koncového efektoru. Tento způsob by omezil "překmity" hlavních ramen způsobené hledáním trasy v 6D prostoru. Nakonec je potřeba vylepšit způsob zápisu finální pozice z úhlů natočení jednotlivých kloubů na místo a orientaci koncového efektoru. Tyto úhly by byly dopočítány pomocí inverzní kinematiky, což by ulehčilo práci pro uživatele. Jak je v programování zvykem, tak se vždy najde prostor k zlepšení a optimalizaci. Celkově je však skript funkční a v případě, že se na něm bude i nadále pracovat, mohl by v budoucnu najít reálné využití.

Seznam zdrojů

[1] GREPL, R.: Kinematika a dynamika mechatronických systémů, CERM, Akademické nakladatelství, 2007.

[2] ZHOU, HUIMING. PathPlanning. GitHub repository, 2020.
Dostupné z: https://github.com/zhm-real/PathPlanning.

[3] SAI VEMPRALA (2023). 2D/3D RRT* algorithm. MATLAB Central File Exchange. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/60993-2d-3d-rrt-algorithm

[4] Forward and Reverse Kinematics for 3R Planar Manipulator. HIVEBLOG. [online]. [cit. 24.5.2023]. Dostupné z: https://hive.blog/hive-196387/@juecoree/forward-and-reverse-kinematics-for-3r-planar-manipulator

[5] JINDRA, Vojtěch. Plánovaní optimální trajektorie Rn manipulátoru v prostoru s překážkami. Brno, 2022.
Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/137009.

[6] PANAGOU, D. (2014). Motion planning and collision avoidance using navigation vector fields. 2014 IEEE International Conference on Robotics and Automation (ICRA), 2513-2518. [online]. [Cit. 25.5.2023]. Dostupné z: https://www.semanticscholar.org/paper/Motion-planning-and-collision-avoidance-using-Panagou/bdf98f07cf408ea295f840a0d9ae379cf99826b0#cited-papers

[7] Dynamic Target Tracking and Obstacle Avoidance using a Drone - Scientific Figure on ResearchGate. [Online]. ResearchGate. [Cit. 24.5.2023]. Dostupné z: https://www.researchgate.net/figure/An-example-of-a-traditional-potential-field-which-can-be-used-for-navigating-toward-a_fig2_294645105

[8] POPOVIĆ, MARIJA. 5-Sampling-based Methods, [online]. [cit. 24.5.2023]. Dostupné z: https://mpopovic.io/uploads/dmar2021/L5-sampling-based-methods.pdf

[9] ENGINEERNICK. (2021, January 30). Line Segment Intersection [Video]. Youtube. [cit. 25.5.2023]. Dostupné z: https://www.youtube.com/watch?v=5FkOO1Wwb8w

[10] Kobot UR5e — Haberkorn. Váš partner pro stavbu strojů — Haberkorn [online]. Haberkorn s.r.o. [cit. 24.05.2023]. Dostupné z: https://www.haberkorn.cz/kobot-ur5e/

Seznam obrázků

2.1 Inverzní kinematika RR manipulátoru 12
2.2 Ukázka algoritmu A* 14
2.3 Ukázka algoritmu potenciální pole 14
2.4 Ukázka algoritmu vektorová pole 15
2.5 Ukázka algoritmu PRM 16
2.6 Ukázka algoritmu RRT (přibližně EST) 17

3.1 Základy RRT 18
3.2 Omezení vzdálenosti nového bodu pomocí EPS 20
3.3 Přepojování stromové struktury pomocí RRT* 21

4.1 Pohled na překážku s úsečkou ve 3D a legenda 27
4.2 Metoda rozložení pohledů použitá k detekci kolizí 27
4.3 Vztah více cílů a směru rotace manipulátoru 30

5.1 Ukázka prostoru 34

Seznam úryvků kódu

4.1 Fuknce měřící euklidovskou vzdálenost dvou bodů v 6D 23
4.2 Funkce nahrazující náhodný bod novým bodem ve vzdálenosti EPS . . 24
4.3 Zjištění nejbližších sousedních bodů v poloměru R 24
4.4 Přepojování stromové struktury na trasy s nejmenší hodnotou 25
4.5 Zpětné dohledávání nejkratší trasy od cíle ke startu 25
4.6 Detekce průsečíku dvou úseček pomocí analytické geometrie 28
4.7 Detekce průsečíků všech hran překážky s úsečkou ve všech pohledech . 28
4.8 Podmínka kontrolující kolizi v pohledu xz 29
4.9 Nalezení vhodného a bezkolizního cíle s nejkratší vzdáleností 31
5.1 Zadání cílové lokace a nastavení parametrů 33
5.2 Ukázka přidání další překážky 34
5.3 Cílové pozice a nastavení parametrů pro testování 35

Seznam tabulek

5.1 Základní pohyb - První část 35
5.2 Základní pohyb - Druhá část 36
5.3 Základní pohyb - Třetí část 36
5.4 Složitý pohyb 37

Seznam příloh

[1] **RRTSTAR.zip**
Vypracovaný kód pro navigaci manipulátoru a Videozáznamy testovaných pohybů