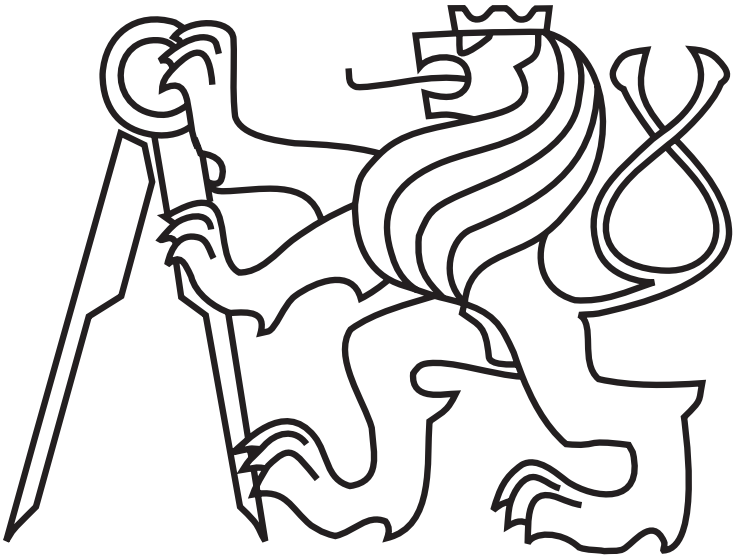
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Diplomová práce



Martin Blaha

Generování hladkých trajektorií ve 3D

Katedra kybernetiky

Vedoucí diplomové práce: **RNDr. Miroslav Kulich, Ph.D.**

Praha 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Martin B l a h a

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Generování hladkých trajektorií ve 3D

Pokyny pro vypracování:

1. Seznamte se s volně dostupnými knihovnami pro generování Voroného diagramů množiny bodů ve 3D, zejména s knihovnou Voro++ (http://math.lbl.gov/voro++/).
2. Seznamte se s metodami reprezentace hladkých křivek ve 3D.
3. Implementujte (a) Voroného diagramy pro množinu mnohostěnů a (b) operace průniku Voroného buněk s hladkými křivkami (např. B-splines).
4. Navrhněte a implementujte algoritmus pro optimalizaci počáteční trajektorie založený na efektivním výpočtu funkce vzdálenosti od překážek.
5. Funkčnost implementovaných algoritmů ověřte experimenty. Experimentální výsledky popište se zaměřením na rychlost a robustnost algoritmů a kvalitu generovaných výsledků.

Seznam odborné literatury:

[1] M. Kulich: Vyhlazování cesty pro autonomní vozítko. Diplomová práce, MFF UK Praha, 1996.
[2] C. H. Rycroft: Voro++: a three-dimensional Voronoi cell library in C+, http://math.lbl.gov/voro++/doc/voro++_overview.pdf [online]
[3] W.J. Schroeder, K. Martin, W. Lorensen:The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, Third Edition. Kitware, Inc. 2006.
[4] E.G. Gilbert, D. W. Johnson: Distance functions and their application to robot path planning in the presence of obstacles, IEEE Journal of Robotics and Automation, vol.1, no.1, pp.21-30, Mar 1985.

Vedoucí diplomové práce: RNDr. Miroslav Kulich, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Prohlášení autora práce

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

V Praze dne..... .....

Poděkování

Děkuji vedoucímu práce RNDr. Miroslavu Kulichovi, Ph.D. za cenné rady, oceňuji jeho vstřícný a konstruktivní přístup. Děkuji své rodině za podporu v průběhu celého studia.

Abstrakt

Tato práce se zabývá optimalizací počáteční trajektorie založené na efektivním výpočtu funkce vzdálenosti od překážek pro 3D prostředí. Nejprve je představen algoritmus pro tvorbu aproximací 3D Voroného diagramu, který je nutný k výpočtu funkce vzdálenosti trajektorie od překážky. Trajektorie je reprezentována B-spline křivkou. Následně je popsán algoritmus výpočtu funkce trajektorie od překážky založený na prohledávání do šířky. Při optimalizaci trajektorie jsou použita jako hodnotící kritéria vzdálenost trajektorie od překážek a délka trajektorie. Funkčnost navrženého algoritmu byla otestována na mapě tvořené sedmi kvádry, pro různé váhy funkce vzdálenosti od překážek a délky trajektorie. Návrhy na zlepšení jsou diskutovány.

Abstract

This thesis deals with the optimization of the initial trajectory. The optimization is based on calculation of the effective function of the distance between obstacles and path in 3D environments. First, an algorithm for the calculation of approximations of 3D Voronoi diagram is described. The Voronoi diagram is necessary to calculate the distance function between obstacles and a trajectory. The trajectory is represented by a B-spline curve. The algorithm for calculation of the distance function of path based on breadth-first search is described. The criteria of the distance function and the length of the trajectory are used to optimize trajectory.

Functionality of the proposed algorithm was tested on a map. The map consists of seven obstacles and the test was performed for various weights of both criteria. Suggestions for improvement are discussed.

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

1 Úvod

Mezi základní funkcionality mobilního robota patří plánování, které převádí požadavky nadřízených specifikací do popisu, kudy se má robot pohybovat. Výstupem plánovacího algoritmu může být posloupnost akcí, které musí robot vykonat, aby bylo dosaženo zadaného cíle, nebo jenom posloupnost bodů, které musí robot projet. Mezi požadavky patří přípustnost plánu, kdy se nebere ohled na jeho efektivitu, ale na proveditelnost, která muže být podmíněna například rozměry robota. Dalším požadavkem je kvalita plánu, tj. je snahou nalézt proveditelný plán, který je z nějakého hlediska optimální. Na optimální trajektorii jsou kladeny především požadavky na energetickou náročnost, délku trajektorie, kterou musí robot urazit k cíli a maximální vzdálenost trajektorie od okolních překážek.

Plánovací algoritmy lze rozdělit podle stavového prostoru na diskrétní a spojité. Toto rozdělení se odráží i v použité reprezentaci prostředí, ve kterém se robot pohybuje. Diskrétní plánovací algoritmy, například Dijkstrův algoritmus, prohledávají konečný prostor stavů, který může být například zadán tzv. rastrovou mapou, ve které jednotlivé buňky obsahují informaci o pravděpodobnosti výskytu překážky.

Při plánování trajektorie robota v prostředí reprezentovaném geometrickou mapou může být stavový prostor spojitý a nespočitatelný. Složitost geometrického modelu záleží na řešeném problému, ve 2D většinou postačí reprezentace překážek a robota pomocí polygonů. Nespočitatelný stavový prostor lze také popsat tzv. konfiguračním prostorem. Tento popis je vhodný pro plánování, pokud je transformace ze spojitého do diskrétního popisu a další použití diskrétního plánovače příliš složité. Dimenze konfiguračního prostoru je závislá na řešeném problému, například pro pohyb ve 3D prostředí kvadrokoptéry je dimenze 6. K dosažení proveditelného plánu je z konfiguračního prostoru odebrán prostor překážek a prostor konfigurací, které nesplňují rozměrové a fyzikální omezení. Výsledný plán je pak získán například algoritmem „Rapidly-exploring Random Tree“ [1]. Mnoho plánovacích algoritmů je použitelných i v jiných vědních oborech.

Problematikou optimalizace trajektorie založené na výpočtu funkce vzdálenosti od překážek se zabývá článek [2]. V tomto článku je uveden postup, který popisuje plánování trajektorie v závislosti na překážkách omezením stavového prostoru. Omezení daná překážkami jsou zohledněna vzdáleností mezi trajektorii a překážkou, s kterou hrozí srážka. V tomto článku jsou uvedeny vlastnosti funkce vzdálenosti, především její diferencovatelnost. Tato funkce obecně nemusí mít derivaci, ale pro striktně konvexní překážky ji má, což umožňuje k optimalizaci použít metody, které rychle konvergují. Autoři článku zmiňují jako nevýhodu výpočetní náročnost.

Tato nevýhoda je odstraněna v [3]. Namísto počítání funkce vzdálenosti od všech překážek je počítána funkce vzdálenosti jen od nejbližší překážky. K určení oblastí, které jsou nejblíže, je použito Voroného diagramu. Algoritmus popsaný v [3] je určen pro 2D prostředí překážek reprezentované seznamem vrcholů a hran polygonů. Plánovací algoritmus dokáže pracovat s jednobodovým i polygonálním robotem.
Cílem této práce je navrhnout na základě [3] algoritmus pro optimalizaci trajektorie ve 3D prostředí. Hlavní těžiště práce je věnováno změnám výpočtu funkce vzdálenosti pro překážky, dále návrhu nových algoritmů pro efektivní výpočet hodnotící funkce.

Popis navrženého algoritmu je v kapitole 2. Informace o použitých knihovnách a implementace je uvedena v kapitole 3. V kapitole 4 jsou zjištěny vlastnosti algoritmu provedením několika experimentů. V poslední kapitole 5 je uvedeno zhodnocení celé práce.

2 Popis algoritmu

V této kapitole je popsán navržený algoritmus generování hladké trajektorie. Algoritmus lze rozdělit na tři hlavní části:

- výpočet Voroného buněk překážek mapy
- výpočet počáteční trajektorie
- optimalizace počáteční trajektorie.

Vstupní proměnné jsou mapa překážek, startovní a cílová pozice, výstup algoritmu je optimalizovaná trajektorie popsaná B-spline křivkou.
Při optimalizaci trajektorie je využito výpočtu funkce vzdálenosti od překážek na základě vzdálenosti části trajektorie k nejbližší překážce. Tato funkce je také označovaná jako funkce bezpečnosti [3]. Prostor kolem překážek je rozdělen pomocí Voroného diagramu, který každé překážce přiřadí část prostoru. Dle našich znalostí neexistuje použitelná knihovna pro výpočet Voroného diagramu pro 3D tělesa, proto byl navržen postup výpočtu, který je popsaný v kapitole 2.2.1.

Optimalizace požaduje na začátku přípustnou trajektorii od startovního do cílového bodu. Nejdříve se vypočítá přípustná cesta pomocí již spočítaných Voroného buněk a Dijsktrova algoritmu. Následně se cesta převede na B-spline křivku.
V poslední části se B-spline křivka optimalizuje na základě požadavků na bezpečnost a délku trajektorie, viz kapitola 2.5. V následujících kapitolách jsou jednotlivé kroky podrobně vysvětleny.

2.1 Mapa prostředí

Pro vlastní algoritmus plánování cesty je nutné vhodně reprezentovat překážky v 3D prostředí robota. Při vytváření mapy jsou uvažovány překážky ve tvaru kvádru nebo krychle. Zadaná mapa překážek musí splňovat předpoklad, že se překážky vzájemně nedotýkají ani do sebe nezasahují. Tento předpoklad je důležitý pro vytváření Voroného diagramu, který je popsán v kapitole 2.2. Jednotlivé překážky jsou popsány seznamy vrcholů, hran a stěn, které je tvoří. Překážky se do mapy nemusí zadávat danými seznamy, ale stačí zadat těžiště překážky a jednotlivé délky v osách x, y, z. Takto vytvořenou překážku lze posunout, případně otočit o zadaný úhel kolem definované osy.

2.2 Voroného diagramy

Co jsou Voroného diagramy lze ukázat na následujícím příkladu. Předpokládejme množinu obchodních domů, tzv. míst, které poskytují určité služby. Naším úkolem je pro všechna místa zjistit, kde žijí zákazníci, kteří požadují služby. Dále předpokládejme, že cena služby je ve všech obchodních domech stejná, cena získané služby je rovná její ceně a ceně přepravy k obchodnímu domu, cena transportu je úměrná Eukleidovské vzdálenosti a zákazníci se snaží minimalizovat cenu získané služby. Z těchto předpokladů vyplývá, že celá plocha je rozdělena na regiony jednotlivých obchodních domů a to tak, že lidé ze stejného regionu jezdí do stejného místa. Region je složen ze všech bodů (obydlí zákazníků), které jsou blíže danému místu než k ostatním místům [4].

Formálně je Voroného diagram definovaný následujícím způsobem. Mějme množinu n bodů $S = \{p_1, \ , ..., p_n\}$ v rovině, pro každý prvek p z S je otevřený Voroného region dán

$$D\left(p, S\right) = \bigcap_{q \in S, q \neq p} D\left(p, q\right), \tag{1}$$

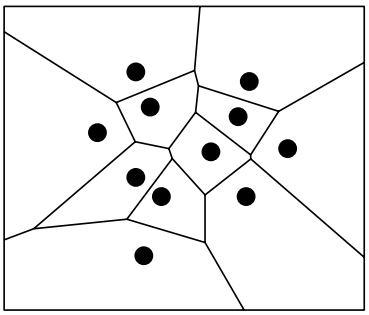
kde $p,\ q \in S,\ p \neq q$ a $D(p, q)$ je polorovina, která patří bodu p vypočtená podle

$$D\left(p, q\right) = \left\{z \in \mathbb{R}^2; |p - z| < |q - z|\right\}. \tag{2}$$

Hranice všech regionů, která se nazývá Voroného diagram množiny S, je definována jako

$$V\left(S\right) = \bigcup_{q \in S} \partial D\left(p, S\right). \tag{3}$$

Na obrázku 1 je ukázka Voroného diagramu pro množinu bodů v rovině.



Obrázek 1: Voroného diagram pro množinu bodů v rovině, převzato z [4]

Algoritmus výpočtu je popsaný například v [4]. Pro množinu, které není zadána jenom body, je nutné použít algoritmy, které dělící úsečky nahrazují obecnou dělící křivkou, přesný

popis je v [5]. Dle našich znalostí neexistuje použitelná knihovna, která dokáže vypočítat Voroného buňky pro 3D tělesa. V důsledku toho byl navržen algoritmus, který je popsán v následující části.

2.2.1 Voroného diagram pro svět robota

Při výpočtu bezpečnostní funkce je třeba počítat vzdálenost trajektorie od překážky. Je vhodné překážky rozdělit to třech kategorií, protože se vzdálenost trajektorie od vrcholu, hrany a stěny překážky počítá odlišným způsobem. Překážky v mapě se nejprve aproximují množinou bodů, pro které se vypočítají Voroného buňky. Počet Voroného buněk je v této fázi větší, než je počet částí překážek, proto jsou jednotlivé Voroného buňky sloučeny. Sloučeny jsou i rozdělené stěny, které vznikly v důsledku aproximace překážek body. V následující sekci jsou popsány jednotlivé krovky vytváření aproximace Voroného buněk pro překážky.

Převod mapy Vrcholy, hrany a stěny překážek ze zadané mapy se aproximují body. Přesnější aproximaci překážek lze dosáhnout zvětšením počtu aproximujících bodů. Příliš velký počet aproximujících bodů má ale za následek výrazné zpomalení běhu algoritmu. Dalším důležitým aspektem je vzdálenost mezi jednotlivými body. Tato vzdálenost je určena pomocí délky nejkratší hrany z celé zadané mapy a počtu bodů, které ji mají aproximovat. Experimentálně bylo zjištěno, že dostatečný počet bodů na nejkratší hraně překážek jsou dva. Na obrázku 2a je ukázka aproximace dvou překážek body. Barevné rozdělení označuje příslušnost bodů Voroného buněk k jednotlivým částem překážek. Postup vzorkování popisuje algoritmus 1.

Vstup: vrcholy hrany x_0 a x_1, minimální vzdálenost $minL$, minimální počet vzorků $minS$

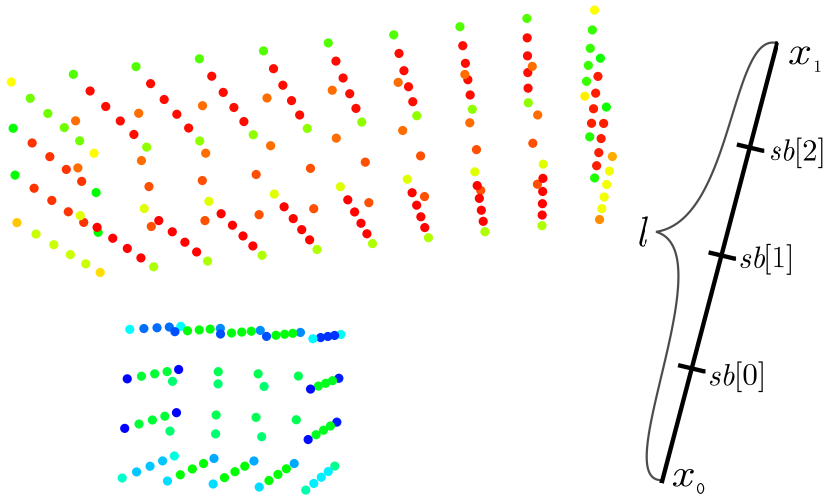
Výstup: seznam bodů aproximujících hranu překážky sb

1: Vypočti délku kroku $d{=}\frac{minL}{minS}$. Vypočti délku hrany $l = |x_0\ x_1|$.
2: Počet aproximujících bodů $np = round(\frac{l}{d} + 2)$.
3: **for all** $id \in \langle 1, np - 1)$ **do**
4: Přidávání bodů $sb = sb \cup \{x_0 + t\,(x_1 - x_0)\}$, kde $t = \frac{id}{np-1}$.
5: **end for**

Algoritmus 1: Algoritmus aproximace hrany překážky, převzato a upraveno z [6]

Počet aproximujících bodů se určí na řádku 2 a následně se vypočítají jednotlivé aproximující body hrany. Na obrázku 2b je zobrazena ukázka vzorkování hrany. Skutečná vzdálenost mezi body aproximujícími část překážky je obecně menší než zadaná, což způsobuje zaokrouhlování počtu aproximujících bodů na celé číslo, viz řádek 2.
Modifikací algoritmu 1 lze vzorkovat stěny překážek. Stěna je zadána čtyřmi body x_0 až x_3, které jsou využity pro určení délky hran $l_1 = |x_0\, x_1|$, $l_2 = |x_0\, x_3|$ a určení počtu aproximujících bodů np_1 a np_2. Jednotlivé body jsou vypočteny podle

$$x = x_0 + s(x_1 - x_0) + t(x_3 - x_0) \quad t \in \langle 1; np_1 - 1) \; s \in \langle 1; np_2 - 1). \tag{4}$$



(a) Výsledná aproximace překážek body

(b) Vzorkovaní hrany

Obrázek 2: Výsledná aproximace překážek a hrany body

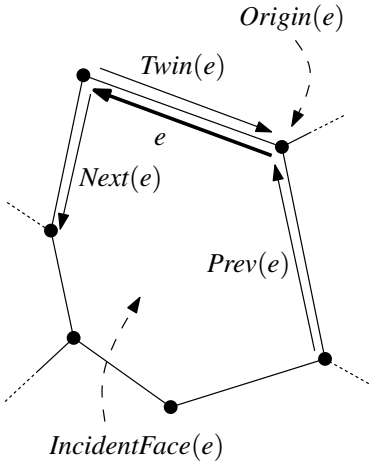
Pro body v seznamu *sb* se vypočítají 3D Voroného buňky. Přesné určení příslušnosti Voroného buňky k bodu, respektive k části překážky, je zajištěno tím, že aproximující body se vyskytují v seznamu pouze jednou. Jedinečnost položek seznamu bodů je splněna splněním podmínky na mapu překážek. Při výpočtu Voroného buněk je nutné definovat část prostoru, ve kterém se výpočet provede. Definovaný prostor výpočtu je volen podle překážek mapy. Má tvar kvádru, který obklopuje všechny překážky. Při výpočtu Voroného buněk jsou stěny, které sousedí s definovaným ohraničením indexovány záporně.

Předpokládaný počet Voroného buněk pro jednu překážku ve tvaru kvádru je osm pro vrcholy, dvanáct pro hrany a šest buněk pro stěny překážky. Počet Voroného buněk je v této fázi daleko větší než očekávaný. Následující postup je především zaměřen na redukci

počtu Voroného buněk jejich slučováním. Na obrázku 4 jsou zobrazeny Voroného buňky jednotlivých částí překážek před dalším zpracováním.

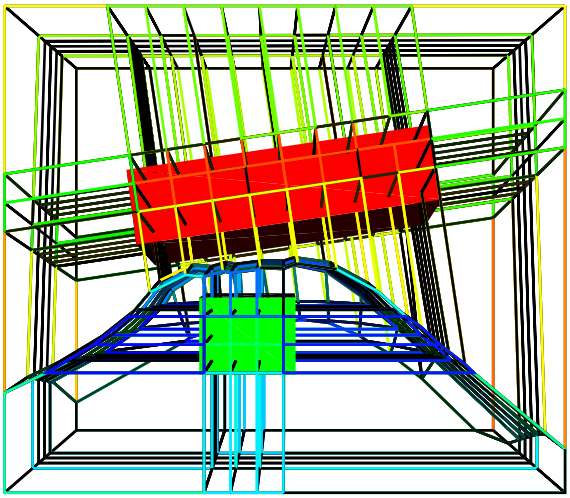
Slučování Voroného buněk Snížení počtu Voroného buněk lze dosáhnou spojováním sousedních buněk do jedné, respektive vyřazením sousedních stěn z obou buněk. Spojovat lze pouze Voroného buňky, které sousedí s bodem ze stejné překážky a ze stejných částí hrany nebo stěny překážky. Aproximaci vrcholu představuje bod samotný, proto Voroného buňky vrcholů není nutné slučovat.

V další fázi jsou všechny buňky převedeny do struktury podobné DCEL(„Doubly Connected Edge List“), která se skládá ze tří seznamů: vrcholů, hran a stěn. Převod do upravené struktury DCEL umožní efektivně procházet stěny po hranách. Struktura pro jednotlivé vrcholy obsahuje souřadnice jeho vrcholu, dále je rozšířená o id vrcholu a seznam všech hran buňky, které ve vrcholu začínají. Tato úprava umožňuje přecházení z polygonu (stěny) do přilehlého polygonu přes vrchol dané stěny.
Struktura pro stěnu obsahuje ukazatel na první hranu, která stěnu ohraničuje, id stěny, id sousední buňky, která přes danou stěnu sousedí a typ sousední buňky, tj. ke které části překážky patří.
Struktura pro hranu obsahuje „twin edge“, to je dvojice sousední hrany e „half edge“. Na obrázku 3 jsou graficky znázorněné závislosti „half edge“. „Half edge“ má ukazatel na vrchol, kde má počátek, dále ukazatel na další, předchozí a protilehlou hranu a ukazatel na stěnu, která je vlevo ve směru hrany. Dále obsahuje id hrany a logický příznak pro určení, jestli hrana míří do překážky v mapě. [4]



Obrázek 3: Závislosti hrany e ve struktuře DCEL, převzato z [4]

Na obrázku 4 je vidět, že vzorkování způsobilo rozdělení stěn Voroného buňky i tam, kde má být stěna souvislá. V následující části je řešeno spojování rozdělených stěn.



Obrázek 4: Vypočtené Voroného buňky pro jednotlivé body získané z překážek

Redukce nadbytečných stěn Redukce nadbytečných stěn spočívá v nahrazení několika sousedních stěn jednou novou stěnou. Nejdříve se určí stěny, které je možné nahrazovat. Mezi nadbytečné stěny se považují:

- Stěny, které tvoří stěnu se stejným okrajem, jsou získány ze stejné překážky a sousedí spolu alespoň jednou hranou.
- Stěny, které jsou mezi Voroného buňkami vytvořenými z jiné části stejné překážky, například stěna mezi buňkou hrany a přilehlou buňkou stěny.
- Stěny mezi dvěma hranami z jiné překážky, které leží v rovině.
- Stěny mezi dvěma Voroného buňkami, které vznikly ze stěny překážky mezi dvěma různými překážkami.

Algoritmus hledání okraje nové stěny Při hledání okraje nové stěny se na začátku určí jeden vrchol ležící ohraničení nové stěny. Postup hledání počátečního vrcholu nové stěny popisuje algoritmus 2. Cyklus na řádku 1 vytváří seznam všech vrcholů, které definují stěnu, která vytváří hranici mezi dvěma buňkami nebo buňkou a okrajem. Počáteční vrchol nové stěny musí být na jejím okraji. Počáteční vrchol je vrchol, který je nejdále od prvního

vrcholu v seznamu vrcholů.

Vstup: Voroného buňka c, sousední buňky *okraj*
Výstup: vrchol v

1: **for all** stěnu f buňky c **do**
2: **if** stěna f sousedí s buňkou *okraj* **then**
3: Přidej všechny vrcholy stěny f do pomocného seznamu vrcholu *psv*.
4: **end if**
5: **end for**
6: Odstraň duplicity ze seznamu *psv*.
7: **for all** prvek v_1 seznamu $psv \setminus \{psv\,[1]\}$ **do**
8: $d = |v_1\ psv\,[1]\,|$
9: **if** $d > maxd$ **then**
10: $maxd = d$ a $v = v_1$
11: **end if**
12: **end for**

Algoritmus 2: Algoritmus hledání prvního vrcholu nové stěny

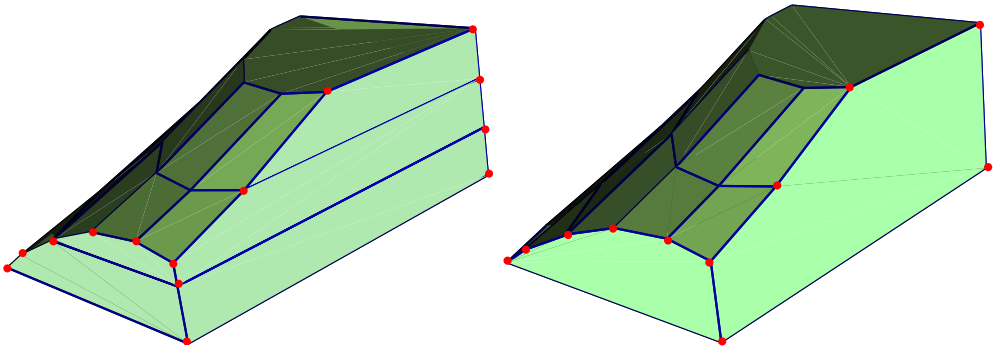
V dalším kroku se vytvoří seznam všech bodů, které leží na okraji nové stěny. Opakováním algoritmu 3 pokaždé pro naposledy nalezený vrchol v, dokud další nalezený vrchol není prvním vrcholem nové stěny, se získá seznam vrcholů, které tvoří okraj nové stěny. Na řádku 2 algoritmu 3 se testuje, jestli hrana e ze seznamu hran vrcholu v_0 je hraniční. Nalezený seznam vrcholů nové stěny většinou obsahuje hodně vrcholů, které leží na přímce. Tyto vrcholy lze odstranit, ale pouze pokud bude odstraněn vrchol i v přilehlé stěně, jak ukazuje obrázek 5. Jinak by ve struktuře pro Voroného buňky nesouhlasil počet „half edge“ a počet „twin edge“. Vrcholy nové stěny, která nelze odstranit, jsou přidány do seznamu neodstranitelných vrcholů.

Vstup: Voroného buňka c, sousední buňky *okraj* a počáteční vrchol v_0
Výstup: vrchol v

1: **for all** Hrany e vrcholu v_0 **do**
2: **if** e.face.neighbor == *okraj* AND e.twin.face.neighbor !=*okraj* **then**
3: v = e.twin.vertex;
4: **end if**
5: **end for**

Algoritmus 3: Algoritmus hledání dalších krajních vrcholů nové stěny

Algoritmus 4 popisuje postup při snížení počtu vrcholů tvořících novou stěnu, který je založen na výpočtu jednotlivých směrnic vrcholů jdoucích za sebou. Pokud je testovaný vrchol vrcholem, který nemůže být odstraněn, je přidán do filtrovaného seznamu hned. V opačném případě se vypočte směrnice hrany před a za testovaným vrcholem. Testovaný vrchol je přidán, pokud je vzdálenost mezi souřadnicemi směrnic větší než určená mez. Staré stěny se nahradí novou stěnou, odstraní se již nepoužívané hrany a vrcholy, které leží uvnitř nové stěny.



Obrázek 5: Voroného buňka před a po redukci stěn

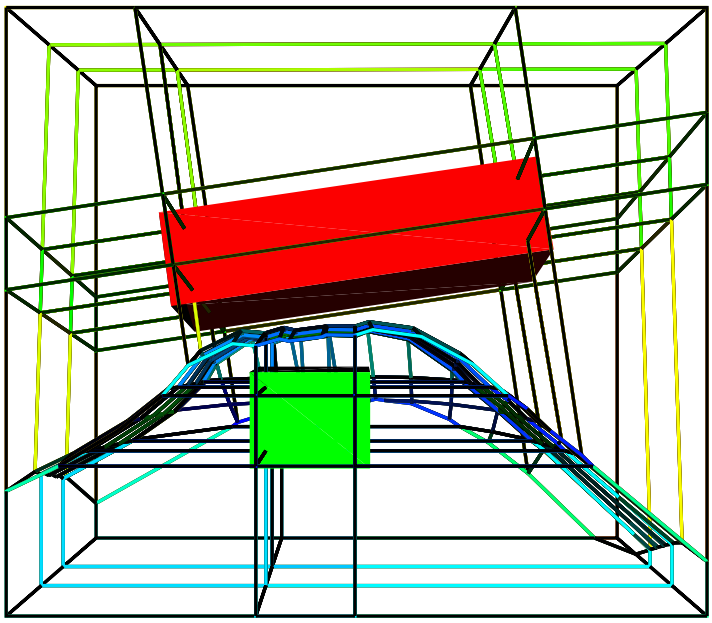
Vstup: seznam vrcholů v, seznam neodstranitelných vrcholů nv, práh p
Výstup: redukovaný seznam vrcholů rv

1: Velikost seznamu $vs = |v|$. Na začátek v přidej $v\,[vs]$. Na konec v přidej $v\,[1]$.
2: **for all** $i \in \langle 1, vs - 1\rangle$ **do**
3: **if** vrchol $v[i]$ je v nv **then**
4: Přidej nakonec rv vrchol $v[i]$
5: **else**
6: Vypočítej směrnici s_1 ($v[i-1]$ a $v[i]$).
7: Vypočítej směrnici s_2 ($v[i]$ a$v[i+1]$).
8: Normování směrnic s_1 a s_2.
9: **if** $|s_1\, s_2| > p$ **then**
10: Přidej nakonec rv vrchol $v[i]$
11: **end if**
12: **end if**
13: **end for**

Algoritmus 4: Algoritmus redukce vrcholů nové stěny

Na obrázku 6 jsou zobrazeny Voroného buňky pro redukci počtu stěn. Určitou nevýhodou

je, že okraje Voroného buněk zabíhají do překážek, což je způsobeno použitým postupem aproximace překážek množinou bodu.



Obrázek 6: Voroného buňky po redukci stěn

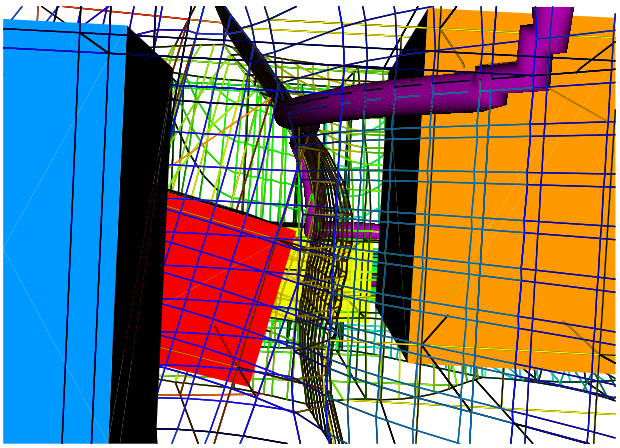
Algoritmus, který je použit na výpočet průsečíků B-spline křivky a Voroného buňky, prochází každou stěnu buňky, proto je vhodné, aby stěn buňky bylo co nejméně. Na obrázku 7 jsou v popředí vpravo zobrazeny stěny mezi Voroného buňkami, které vznikly z hrany a stěny různých překážek. Tato část buňky se skládá z malých a velkých stěn. Sloučení malých stěn se sousedními stěnami s velkým obsahem by vedlo ke snížení počtu stěn buňky.

2.3 Plánování cesty

Na seznam struktury Voroného buněk lze nahlížet jako na orientovaný graf s kladným ohodnocením hran (délka hrany). Pro nalezení nejkratší cesty ze startovního do cílového vrcholu byl použit Dijkstrův algoritmus [7]. Výsledná naplánovaná cesta je pak převedena na B-spline křivku, která je využita jako počáteční trajektorie pro optimalizaci.

2.3.1 Převod Voroného buněk pro plánovací algoritmus

Vstupem plánovacího algoritmu je seznam všech vrcholů, které definují stěny Voroného buněk částí překážek. Prvky seznamu všech vrcholů jsou popsány strukturou, která ob-



Obrázek 7: Nezpracované stěny Voroného buňky

sahuje id vrcholu v seznamu, souřadnice vrcholu a seznam výskytu daného vrcholu v jednotlivých buňkách a id tohoto vrcholu v konkrétní Voroného buňce. Identifikátor vrcholu v konkrétní buňce je kódován pomocí vztahu

$$id = ic \cdot 1000000 + iv, \tag{5}$$

kde ic je id Voroného buňky a iv je id vrcholu v rámci Voroného buňky. Vztah (5) klade omezení na počet vrcholů v jedné buňce, který nemůže být vetší jak jeden milion, protože by pak negeneroval jednoznačné id.

Postup vytváření seznamu vrcholů pro plánovač popisuje algoritmus 5. Na řádcích 1 až 6 se vytváří seznam všech vrcholů, které tvoří Voroného buňky. Na řádcích 7 až 13 se určí výskyty jednotlivých vrcholů ve všech buňkách. Na řádku 16 se vytváří položka n seznamu všech vrcholů sv, do které se přidají údaje o souřadnicích vrcholu a seznam výskytů z k. Na řádku 18 se ze seznamu sp odstaní prvky, které jsou v seznamu výskytů prvku n. Tímto postupem je zajištěno, že v seznam sv budou všechny vrcholy pouze jednou.

Naplánovaná cesta nesmí procházet překážkami, k tomuto účelu byly označeny všechny hrany, které míří z překážky nebo do překážky. Při hledání těchto hran se využilo faktu, že tyto hrany začínají, respektive končí, blízko vrcholů překážek. V seznamu všech vrcholů se najdou vrcholy, které jsou danému vrcholu překážky nejblíže. Pro tyto vrcholy se určí hrany, kterým se nastaví logický příznak.

Logický příznak kolize je také nastaven hranám, které leží v ohraničení celé mapy překážek nebo do něho směřují. Nejprve se určí stěny Voroného buněk, které vytváří hranici s ohraničením. Hranám, které začínají,respektive končí, ve vrcholech těchto stěn se opět

Vstup: seznam Voroného buněk *sc*

Výstup: seznam všech vrcholů *sv* ze seznamu Voroného buněk *sc*

```
1: for all Voroného buňku b z sc do
2:    for all Vrchol v ze seznamu vrcholů buňky b do
3:       Vytvoř položku seznamu sp jednotlivých vrcholů v.
4:       Přidej položku do seznamu sp.
5:    end for
6: end for
7: for all Položku s1 seznamu sp do
8:    for all Položku s2 seznamu sp do
9:       if |s1 s2| < mez then
10:         Přidej položce s1 do seznamu výskytu identifikátor s2.
11:      end if
12:   end for
13: end for
14: while sp není prázdný do
15:   Vezmi poslední prvek k z sp.
16:   Vytvoř nový prvek n.
17:   Do n přidej seznam výskytů z k.
18:   Odstraň z sp prvky, ve kterých se vyskytuje n.
19:   Přidej n do seznamu sv.
20: end while
```

Algoritmus 5: Algoritmus vytvoření seznamu vrcholů pro plánovač

nastaví příznak kolize. Tato úprava zajistí, že naplánovaná cesta nevede po ohraničení celé mapy.

Algoritmus 6 popisuje Dijkstrův algoritmus, který slouží k nalezení nejkratší cesty. Na začátku algoritmu 6 se vytvoří prioritní halda, která má na začátku vrchol s nejmenší vzdáleností od startu. Při inicializaci se všem vrcholům nastaví vzdálenost na nekonečno. Seznam předku *prev* slouží ke zpětnému určení nalezené cesty. Startovní vrchol má předka -1, což je využito pro ukončení zpětného hledání cesty. V každém kroku se odebere jeden vrchol v prioritní haldě, což zajišťuje konečnost Dijkstrova algoritmu. Na řádku 10 je počítána nová vzdálenost expandovaného vrcholu, která vznikne sečtením dosavadní vzdálenosti expandovaného vrcholu a vzdáleností mezi expandovaným vrcholem a jeho potomkem. Pokud je nová vzdálenost menší jak vzdálenost, kterou má potomek, tak se po-

Vstup: startovní vrchol s, cílový vrchol c, seznam vrcholů sv, seznam Voroného buněk sc
Výstup: seznam vrcholů cesty svc

1: Vytvoř prioritní haldu h seznamu sv.
2: Inicializuj prioritní fronty, seznamů vzdáleností $dist$ a předku $prev$.
3: $dist[start] = 0$; h.update(start, 0)
4: **repeat**
5: vrchol v s nejmenší vzdáleností z h
6: Odstraň v z h.
7: **if** $v! = c$ **then**
8: Expanduj vrchol v do seznamu ex.
9: **for all** Položku ex_0 seznamu ex **do**
10: vzdálenost $alt = dist[v] + |v\, ex_0|$
11: **if** $alt < dist[ex_0]$ **then**
12: $dist[ex_0] = alt$
13: $prev[ex_0] = v$
14: h.update(ex_0, alt)
15: **end if**
16: **end for**
17: **else**
18: Ukonči vyhledávání.
19: **end if**
20: **until** h není prázdné **or** $c == v$
21: Do seznamu svc přidej na začátek cílový vrchol c.
22: $u = prev[c]$
23: **while** $u! = -1$ **do**
24: Přidej u na začátek seznamu svc
25: $u = prev[u]$
26: **end while**

Algoritmus 6: Dijkstrův algoritmus

tomkovi nastaví nová vzdálenost, expandovaný vrchol jako rodič a aktualizuje se hodnota vzdálenosti potomka v prioritní haldě.

Algoritmus 7 popisuje postup expanze vrcholu, který vrací seznam vrcholů potomků. Seznam Voroného buněk slouží k filtraci přechodu, které by vedly po kolizní hraně z ex-

pandovaného vrcholu do vrcholu potomka.

Vstup: expandovaný vrchol ev, seznam vrcholů sv, seznam Voroného buněk sc
Výstup: seznam potomků sp

1: Vezmi seznam výskytu vrcholů svv vrcholu ev z sv.
2: **for all** položku ex_0 seznamu svv **do**
3: Vezmi seznam hran h vrcholu ex_0.
4: **for all** Položku h_0 seznamu h **do**
5: **if** h_0.twin nesměřuje do překážky **then**
6: Přidej h_0.twin.vrchol do pomocného seznamu psv
7: **end if**
8: **end for**
9: **end for**
10: **for all** položku psv_0 seznamu psv **do**
11: Přidej do seznamu sp vrchol z sv, který odpovídá vrcholu psv_0.
12: **end for**
13: Odstraň duplicitní vrcholy v sp

Algoritmus 7: Expanze vrcholu pro Dijkstrův algoritmus

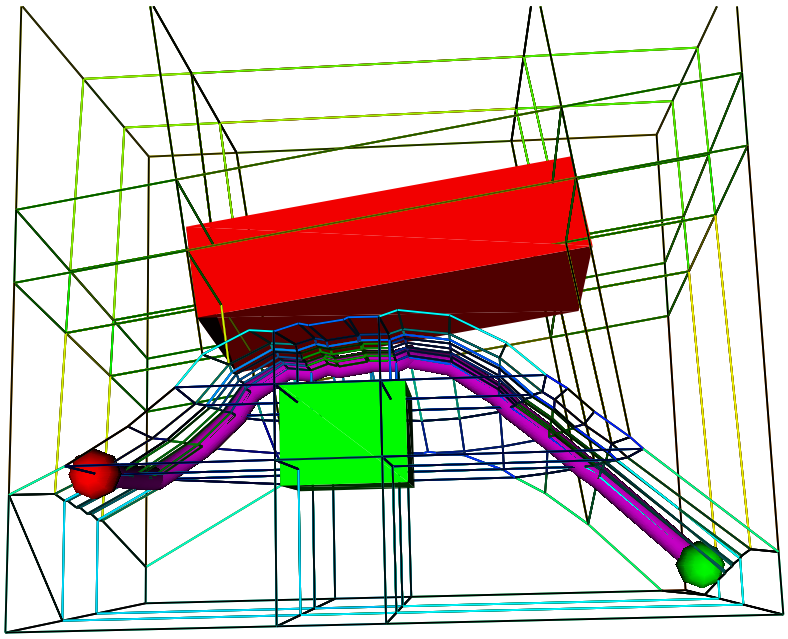
Na obrázku 8 je zobrazen výsledek plánování pro mapu se dvěma překážkami na vrcholech Voroného buněk. Zelená kulička označuje startovní vrchol a červená kulička označuje cílový vrchol.

V další fázi se převede naplánovaná cesta do B-spline křivky, která se následně podle diskutovaných požadavků optimalizuje.

2.4 Popis trajektorie

Výsledná trajektorie je reprezentována pomocí B-spline křivky, jejíž části tvoří Coonsovy kubiky. Coonsova kubika je křivka definovaná předpisem

$$C(t) = C_0(t)P_0 + C_1(t)P_1 + C_2(t)P_2 + C_3(t)P_3,\ t \in \langle 0, 1 \rangle, \tag{6}$$

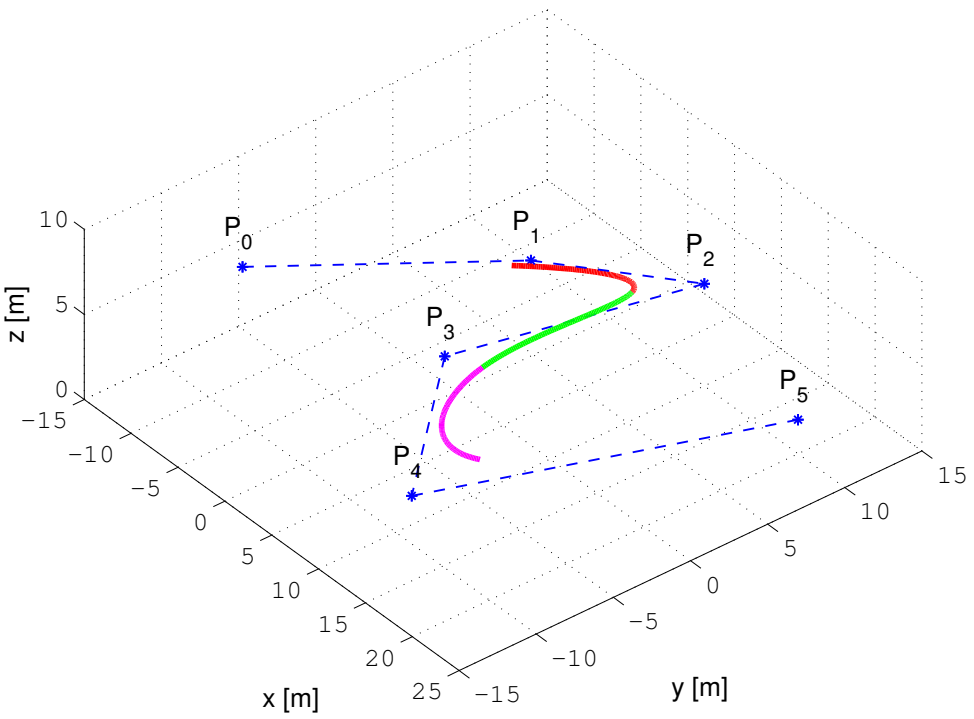


Obrázek 8: Nejkratší cesta vygenerovaná plánovačem

kde P_0, P_1, P_2 a P_3 jsou řídící body a C_0, C_1, C_2 a C_3 jsou Coonsovy polynomy

$$\begin{aligned} C_0(t) &= \frac{1}{6}(1-t)^3 \\ C_1(t) &= \frac{1}{6}(3t^3 - 6t^2 + 4) \\ C_2(t) &= \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1) \\ C_3(t) &= \frac{1}{6}t^3. \end{aligned} \tag{7}$$

Poloha počátečního bodu křivky leží v antitěžišti trojúhelníku $P_0P_1P_2$ a poloha koncového bodu v antitěžišti trojúhelníku $P_1P_2P_3$.
Coonsův kubický B-spline je zadán řídícími body P_0, P_1, ..., P_n, které tvoří $n-2$ Coonsových kubik. Příklad B-spline křivky, která je složená z několika kubik, je na obrázku 9. Důležitou vlastností B-spline křivky je, že změna souřadnic jednoho řídícího bodu změní pouze kubiky, kde se řídící bod vyskytuje. Napojení kubik a samotné kubiky mají spojitou derivaci druhého řádu. [8]



Obrázek 9: B-spline křivka složená ze tří Coonsových kubik

2.4.1 Převod naplánované cesty na B-spline křivku

Počet vrcholů cesty na obrázku 8 je 27, pokud by se použily všechny vrcholy cesty, B-spline křivka by zbytečně přesně sledovala naplánovanou cestu. Algoritmus 8 popisuje postup převodu cesty na B-spline křivku. Na řádku 1 se generují souřadnice řídícího bodu, který se musí nalézat v blízkém okolí daného vrcholu naplánované cesty. Toto okolí je definované konstantou ϵ. Malé náhodné posunutí je důležité pro hledání počáteční Voroného buňky v algoritmu pro výpočet bezpečnostní funkce. Další důvod pro posunutí je, že generovaná cesta by ležela v blízkosti rozhraní mezi dvěma Voroného buňkami, což by vedlo na k velkému počtu průsečíků B-spline křivky s Voroného buňkami. Na řádku 2 se přidává vygenerovaný řídící bod do seznamu řídících bodů, které definují B-spline křivku. Startovní a cílový řídící bod se přidává třikrát z důvodů zachování startovní a cílové pozice. Snížení počtu řídících bodů oproti počtu vrcholů naplánované cesty se děje tak, že se přidává každý N-tý náhodně vygenerovaný bod v blízkosti N-tého vrcholu cesty, viz řádek 3 a 4.

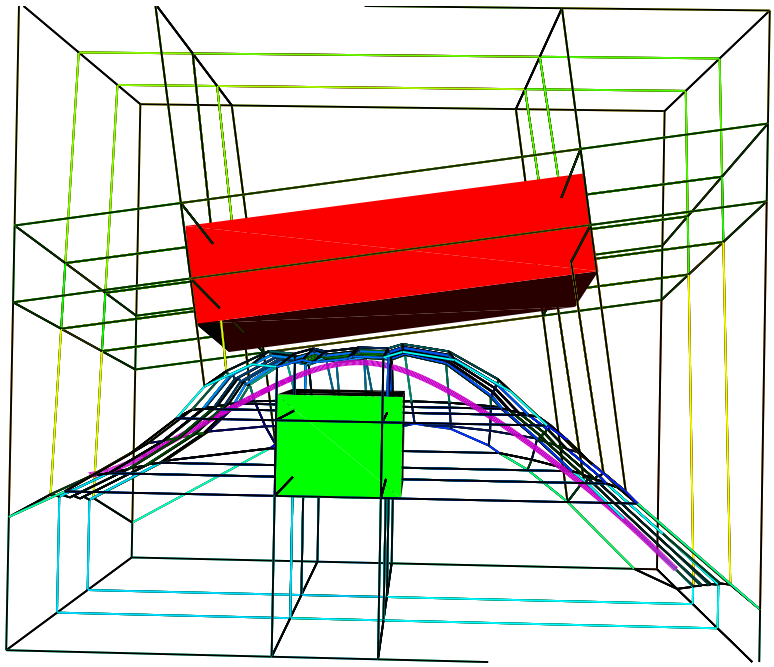
Na obrázku 10 je fialově zobrazena B-spline křivka, která je zadána deseti řídícími body.

Vstup: naplánovaná cesta nc

Výstup: řídící body B-spline křivky bs

1: Vygeneruj náhodně bod sb v epsilonovém okolí startovního vrcholu.
2: Přidej do seznamu řídících bodů bs třikrát bod sb.
3: **for all** $i \in \langle 1; |nc - 1| \rangle$ **do**
4: **if** $\mathrm{mod}\,(i,\ N) == 0$ **then**
5: Vygeneruj náhodně bod b v epsilonovém okolí $nc[i]$ vrcholu.
6: Přidej do seznamu řídících bodů bs bod b.
7: **end if**
8: **end for**
9: Vygeneruj náhodně bod cb v epsilonovém okolí cílového vrcholu.
10: Přidej do seznamu řídících bodů bs třikrát bod cb.

Algoritmus 8: Převod naplánované cesty na B-spline křivku



Obrázek 10: B-spline křivka z naplánované cesty

2.5 Optimalizace trajektorie

Dalším krokem je optimalizace vygenerované počáteční trajektorie. Na výslednou trajektorii jsou požadavky, aby vzdálenost mezi překážkou a naplánovanou trajektorií byla maximální a energetická náročnost byla minimální. Proces optimalizace je definován ve formě

$$\min_{x \in \mathbb{R}^n} f(x), \tag{8}$$

kde $x \in \mathbb{R}^n$ a $f : \mathbb{R}^n \to \mathbb{R}$. $f(x)$ je nelineární multidimenzionální hodnotící funkce. Pro hledání minima neomezené multidimenzionální hodnotící funkce je použita tzv. metoda přímého vyhledávání „direct search method“ [9]. Metoda přímého vyhledávání pouze vyžaduje spojitou hodnotící funkci. Metoda nepotřebuje derivaci ani odhad derivace hodnotící funkce a snadno se implementuje.
V následujících podkapitolách je popsán postup výpočtu hodnotící funkce na základě rovnic z [3], které jsou rozšířeny pro 3D prostředí překážek.

2.5.1 Bezpečnostní funkce

Bezpečnostní funkce zohledňuje požadavek na maximální vzdálenost trajektorie a okolních překážek, viz rovnice (9), převzato z [3].

$$F_B = \sum_{i=1}^{N} \sum_{b \in Cesta \cap B_i} \chi(d(b, B_i)) \tag{9}$$

V rovnici (9) je N počet Voroného buněk, $d(b, B_i)$ je vzdálenost bodu trajektorie od překážky ve Voroného buňce. Funkce χ hodnotí blízkost trajektorie k překážce. Funkce, podle které se počítá bezpečnost části trajektorie j, která je v buňce i, je definována

$$F_B^{ij}(\vec{P}) = \sum_{k=1}^{K} F_B^{ijk}(\vec{P}), \tag{10}$$

kde $\vec{P}$ je vektor řídících bodů B-spline křivky, K je počet souvislých částí B-spline křivky, které jsou uvnitř Voroného buňky a $F_B^{ijk}(\vec{P})$ je definováno rovnicí (11).

$$F_B^{ijk}(\vec{P}) = \int_{C_j} \chi(d(C_j(\vec{P}, t), B_i))\mathrm{d}t,\ t \in \langle \tau_1(\vec{P}, i, j, k), \tau_2(\vec{P}, i, j, k) \rangle \tag{11}$$

V rovnici (11) je $\tau_1(\vec{P}, i, j, k)$ k-tý počátek části souvislého průniku j-té části cesty s i-tou buňkou,$\tau_2(\vec{P}, i, j, k)$ k-tý konec části souvislého průniku j-té části cesty s i-tou buňkou a

$C_j(\vec{P},t)$ vyjadřuje body B-spline křivky v závislosti na řídících bodech a proměnné t. Pro další výpočty je vhodné rovnici (11) upravit do tvaru

$$F_B^{ijk}(\vec{P}) = \int_{\tau_1(\vec{P},i,j,k)}^{\tau_2(\vec{P},i,j,k)} \chi(d(C_j(\vec{P},t),B_i)) \sqrt{(\frac{\partial C_{jx}}{\partial t}(\vec{P},t))^2 + (\frac{\partial C_{jy}}{\partial t}(\vec{P},t))^2 + (\frac{\partial C_{jz}}{\partial t}(\vec{P},t))^2}\mathrm{d}t. \tag{12}$$

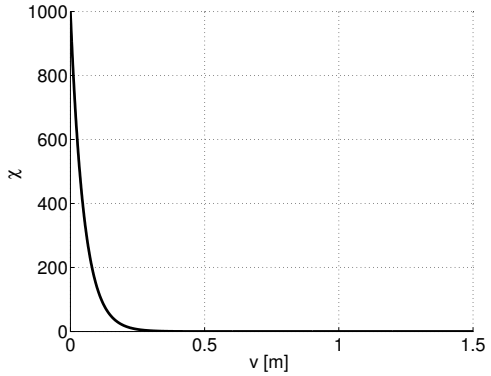
V následujících podkapitolách jsou popsány jednotlivé funkce z rovnice (12).

2.5.2 Funkce χ

Funkce χ má za úkol penalizovat blízkost trajektorie od překážek, na obrázku 11 je zobrazena tato funkce, která má funkční předpis

$$\chi(v) = 1000\mathrm{e}^{-20v}, \tag{13}$$

kde v je vzdálenost.



Obrázek 11: Průběh funkce χ

2.5.3 Funkce τ

Krajní body jednotlivých částí funkce τ jsou definovány průsečíky B-spline křivky se stěnami Voroného buněk. Průsečíky jsou získány řešením soustavy

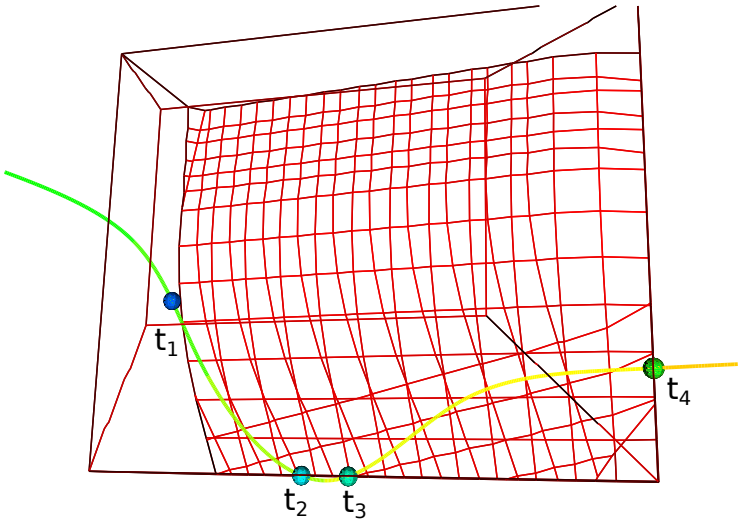
$$C_{jx}(\vec{P},t) = X + u_x s + v_x r \quad t,s,r \in \langle 0,1\rangle \tag{14}$$

$$C_{jy}(\vec{P},t) = Y + u_y s + v_y r \tag{15}$$

$$C_{jz}(\vec{P},t) = Z + u_z s + v_z r, \tag{16}$$

kde $\vec{u} = (u_x, u_y, u_z)$ a $\vec{v} = (v_x, v_y, v_z)$ jsou lineárně nezávislé směrové vektory stěny a (X, Y, Z) je bod stěny.
Dalšími úpravami získáme kubickou rovnici, která se řeší pomocí Cardanových vzorců [10]. Přípustná řešení jsou taková, že $t \in \langle 0, 1 \rangle$ a zároveň průsečík musí ležet uvnitř stěny Voroného buňky. K určení, zda průsečík leží uvnitř stěny, je použita metoda „Ray casting“, která počítá, kolikrát protne polopřímka z testovaného bodu hranice polygonu. Pokud je počet průsečíků polopřímky a hranice polygonu lichý, průsečík leží uvnitř.

Na obrázku 12 je ukázka nalezených průsečíků. Z tohoto obrázku je také vidět, že v této Voroného buňce B-spline křivka prochází třemi stěnami a funkce τ má dvě souvislé části.



Obrázek 12: Průsečíky Voroného buňky a B-spline křivky

2.5.4 Funkce vzdálenosti od vrcholu překážky

Když B-spline křivka prochází Voroného buňkou, která vznikla z vrcholu překážky, počítá se vzdálenost dvou bodů, přičemž první bod je dán B-spline křivkou a druhý bod je zadán souřadnicemi vrcholu (X, Y, Z) v překážce, viz rovnice (17).

$$d(C_j(\vec{P}, t), B_i) = \sqrt{(C_{jx}(\vec{P}, t) + X)^2 + (C_{jy}(\vec{P}, t) - Y)^2 + (C_{jz}(\vec{P}, t) - Z)^2} \qquad (17)$$

2.5.5 Funkce vzdálenosti od hrany překážky

Pokud B-spline křivka prochází Voroného buňkou, která vznikla z hrany překážky, počítá se vzdálenost bodu a přímky, která je zadána počátečním bodem $A = (A_x, A_y, A_z)$ a

směrnicí $\vec{u} = (u_x, u_y, u_z)$, přičemž bod je dán B-spline křivkou. Při výpočtu vzdálenosti se využije faktu, že normála roviny, na které leží spojnice bodů hrany a B-spline křivky, je stejná jako směrový vektor zadané hrany. Vzdálenost je vypočítána podle vztahu

$$d(C_j(\vec{P}, t), B_i) = \sqrt{r_x + r_y + r_z}, \tag{18}$$

kde r_x, r_y a r_z jsou kvadráty rozdílů jednotlivých složek bodu na hraně a na B-spline křivce, konkrétně

$$\begin{aligned} r_x &= \left(A_x - \frac{u_x \left(u_x A_X + u_y A_y + u_z A_z - u_x C_{jx}(t) - u_y C_{jy}(t) - u_z C_{jz}(t) \right)}{u_x^2 + u_y^2 + u_z^2} - C_{jx}(t) \right)^2 \\ r_y &= \left(A_y - \frac{u_y \left(u_x A_X + u_y A_y + u_z A_z - u_x C_{jx}(t) - u_y C_{jy}(t) - u_z C_{jz}(t) \right)}{u_x^2 + u_y^2 + u_z^2} - C_{jy}(t) \right)^2 \\ r_z &= \left(A_z - \frac{u_z \left(u_x A_X + u_y A_y + u_z A_z - u_x C_{jx}(t) - u_y C_{jy}(t) - u_z C_{jz}(t) \right)}{u_x^2 + u_y^2 + u_z^2} - C_{jz}(t) \right)^2 . \end{aligned} \tag{19}$$

2.5.6 Funkce vzdálenosti od stěny překážky

Pokud B-spline křivka prochází Voroného buňkou, která vznikla ze stěny překážky, počítá se vzdálenost bodu, který je dán B-spline křivkou, a stěny zadané normálovým vektorem $\vec{n}$. Vzdálenost bodu B-spline křivky a roviny lze počítat jako absolutní hodnotu kolmého průmětu vektoru bodu B-spline křivky a libovolného bodu z roviny do normály roviny

$$d(C_j(\vec{P}, t), B_i) = \frac{\left| \left(C_j(\vec{P}, t) - A \right) \cdot \vec{n} \right|}{\|\vec{n}\|}, \tag{20}$$

kde $\vec{n}$ je normálový vektor roviny a A je libovolný bod roviny [11].

2.5.7 Algoritmus výpočtu bezpečnostní funkce

Integrál v rovnici (12) je vypočítán numerickou metodou, konkrétně Rombergovou metodou pro numerický výpočet integrálu $I = \int_a^b f(x)\mathrm{d}x$ [12].
Navržený algoritmus 9 výpočtu bezpečnostní funkce je založen na principu prohledávání

do šířky.

Vstup: B-spline křivka C, seznam Voroného buněk sc

Výstup: Hodnota bezpečnostní funkce F_B

1: Vymaž seznam *open* a *close*.
2: Urči první buňku fc, kde se nachází počátek B-spline C.
3: Přidej první buňku fc na konec seznamu *open*.
4: $F_B = 0$
5: **while** *open* není prázdný **do**
6: Vezmi první buňku k z *open*.
7: Odstraň první buňku z *open*.
8: Odstraň všechny prvky seznamu průsečíků p.
9: Odstraň všechny prvky seznamu sousedních buněk *soused*.
10: **for all** stěnu f z buňky k **do**
11: **if** stěna f má průsečík s B-spline C **then**
12: Přidej průsečíky do seznamu průsečíků p.
13: Přidej sousední buňky průsečíků do pomocného seznamu sousedních buněk *soused*.
14: **end if**
15: **end for**
16: **for all** buňku cs ze seznamu *soused* **do**
17: **if** buňka cs není v *close* **and** buňka cs není v *open* **then**
18: Přidej buňku cs na konec seznamu *open*.
19: **end if**
20: **end for**
21: Vypočti hodnotu bezpečnostní funkce F_k buňky k pro průsečíky p.
22: $F_B = F_B + F_k$
23: **end while**

Algoritmus 9: Algoritmus výpočtu bezpečnostní funkce

Navržený algoritmus požaduje na řádce 2, aby první přidaná buňka do seznamu *open* měla alespoň jeden průsečík s B-spline křivkou. Pro všechny buňky se na začátku spočítá vzdálenost jejich částí překážek od počátečního bodu B-spline křivky. Buňka s nejmenší vzdáleností se přidá do seznamu *open* jako první. Tento postup lze vylepšit omezením počtu testovaných buněk, protože ze seznamu všech vrcholů Voroného buněk je možné zjistit, ve kterých buňkách se startovní vrchol trajektorie nacházel před malým náhodným posunutím.

Postup výpočtu průsečíku stěny buňky a B-spline křivky, viz řádek 11, popisuje algoritmus 10.

Vstup: B-spline křivka C, Voroného buňka vc, stěna f z vc,
Výstup: seznam průsečíků sp, seznam sousedních buněk $soused$

1: **for all** část j z C **do**
2: Vypočti koeficienty kubické rovnice r.
3: Vyřeš kubickou rovnici r a řešení dej do seznamu s.
4: **for all** řešení s_0 z s **do**
5: **if** $s_0 \in \langle 0, 1 \rangle$ **and** souřadnice vrcholu leží uvnitř stěny f **then**
6: Vytvoř průsečík a přidej ho do seznamu průsečíků sp.
7: Vezmi sousední buňku, stěny f a přidej ji do seznamu $soused$.
8: **end if**
9: **end for**
10: **end for**

Algoritmus 10: Algoritmus výpočtu průsečíků Voroného buněk a B-spline křivky

Na řádku 21 algoritmu 9 se provádí výpočet rovnice 12, seznam všech průsečíku je nejdříve doplněn na sudý počet průsečíků. Pokud seznam průsečíku obsahuje lichý počet, pak se přidá průsečík reprezentující začátek, respektive konec, B-spline křivky, jestliže začátek, respektive konec, B-spline křivky leží v právě počítané Voroného buňce.

Takto doplněný seznam se sudým počtem průsečíků je vzestupně seřazen podle j a t, díky tomu se integrační meze berou po dvojicích. Dvojice t_1 t_2 na obrázku 12 určuje jednu souvislou komponentu v buňce a dvojice t_3 t_4 další. Z obrázku 12 je také zřejmé, že souvislé komponenty nemusí končit ve stejné části B-spline křivky v jaké začaly. V tomto případě je pak pro numerickou integraci souvislá komponenta rozdělena na menší intervaly, které odpovídají částem křivky. Uvnitř překážek je nastavena vzdálenost na nulovou hodnotu, což zajišťuje přípustnost optimalizované cesty.

2.5.8 Kombinace délky a bezpečnosti

Při optimalizaci trajektorie jsou kladeny požadavky na bezpečnost a na délku hladké trajektorie. Rovnice (21) váží oba protichůdné požadavky,

$$F(\vec{P}) = \alpha F_B(\vec{P}) + (1 - \alpha) F_D(\vec{P}), \tag{21}$$

kde $\alpha \in \langle 0, 1\rangle$ je váha a $F_D(\vec{P})$ je délka B-spline křivky, která se spočítá podle

$$F_D(\vec{P}) = \sum_{j=1}^{M} \int_0^1 \sqrt{\left(\frac{\partial C_{jx}}{\partial t}(\vec{P}, t)\right)^2 + \left(\frac{\partial C_{jy}}{\partial t}(\vec{P}, t)\right)^2 + \left(\frac{\partial C_{jz}}{\partial t}(\vec{P}, t)\right)^2} \mathrm{d}t, \qquad (22)$$

kde M je počet Coonsových kubik tvořících B-spline křivku.

2.6 Zhodnocení navrženého algoritmu

Metoda přímého vyhledávání obvykle konverguje pomaleji než metody numerické optimalizace, které využívají znalost derivace hodnotící funkce. Pokud by se odvodil gradient bezpečnostní funkce podobně, jak tomu je v [3], doba běhu optimalizace trajektorie by se zkrátila.

Definované ohraničení mapy překážek pouze poskytuje ohraničení prostoru pro výpočet Voroného buněk, aby se na něj dalo nahlížet například jako na stěnu místnosti, kde jsou překážky umístěné uvnitř, musely by se jednotlivé stěny ohraničení zařadit do seznamu překážek. Navržený algoritmus předpokládá pouze jednobodového robota. V [3] je uveden postup, jak převést polygonálního robota na jednobodové vozítko. Metoda využívá k převodu několik bodů na okraji robota.

Popsaný převod naplánované cesty na B-spline křivku může vytvořit nepřípustnou cestu při použití nízkého počtu řídících bodů. Tento problém lze vyřešit použitím postupu, který namísto přidání každého N-tého bodu bude sledovat směr cesty a ten respektovat, tj. rychlé změny naplánované cesty budou popsány více body než rovné úseky.

Uvedená aproximace překážek body funguje korektně, pokud vzdálenost mezi nějakými dvěma překážkami není menší než nejmenší hrana překážky v mapě. Důsledkem toho Voroného buňky zasahují do částí překážek, kde má figurovat jiná buňka. Tento nedostatek by šlo odstranit použitím postupu, který by zohlednil velikost jednotlivých překážek a jejich vzájemnou polohu.

3 Implementace

Navržený program lze primárně rozdělit na výpočetní a vizualizační část. Implementace obou částí je provedena v programovacím jazyku C++.
Program je členěn do jednotlivých tříd, které reprezentují jednotlivé funkční bloky, jako jsou: reprezentace mapy, výpočet Voroného buněk, plánovač na vrcholech Voroného buněk, reprezentace B-spline křivky a její optimalizace. Výpočet Voroného buněk je realizován pomocí knihovny Voro++, která je popsána v kapitole 3.2. Optimalizace trajektorie je provedena knihovnou OPT++, která je popsána v kapitole 3.3.

Již během vývoje se objevila nutnost zobrazit výsledky jednotlivých kroků. Pro vizualizaci byla zvolena knihovna VTK, která je popsána v kapitole 3.1. Tato knihovna byla vybrána, protože je velmi dobře zdokumentovaná v podobě mnoha příkladů na webu a v knize [13]. Knihovna poskytuje struktury pro ukládání dat a implementaci algoritmů počítačové grafiky. Hlavní struktura, kterou se předává většina dat pro vizualizaci, je *vtkPolyData*, která obsahuje seznam bodů, vrcholů, úseček, polygonů, trojúhelníků a přídavné informace o jednotlivých seznamech.

3.1 Knihovna VTK

Knihovna VTK Visualization Toolkit je objektově orientovaná open-source knihovna pro 3D vizualizaci a zpracovaní dat, která je poskytovaná společností Kitware. Knihovna VTK je implementována v C++, dále poskytuje rozhraní pro Tcl, Python a Javu. VTK podporuje značnou škálu reprezentací dat, jako jsou množiny bodů, polygony, obrázky a jiné. Zpracování dat se provádí pomocí filtrů, které obsahují implementace algoritmů pro počítačovou grafiku. Průběh vizualizace lze rozdělit do tří kroků. První krok je načtení dat, další krok je zpracování dat a poslední krok je zobrazení dat.
Vizualizační okno je vytvořeno pomocí třídy *vtkRenderWindow*. Tomuto objektu se přiřazují další vlastnosti, nejdůležitější je *vtkRenderer* obsahující zobrazovaný objekt a *vtkRenderWindowInteractor* pro interakci vizualizačního okna s uživatelem [13].

3.2 Knihovna Voro++

Knihovna provádí výpočet Voroného buněk pro každý zadaný bod samostatně. Vytvoří kolem tohoto bodu Voroného buňku reprezentovanou jako mnohostěn, který bod obklopuje. Dále umožňuje vytvářet kontejnery, které seskupují jednotlivé body do podskupin, což je

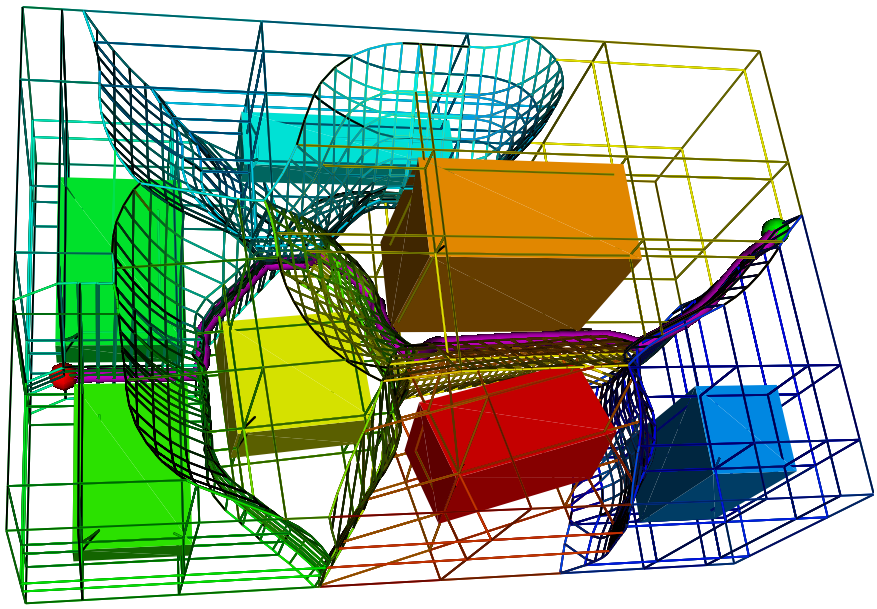
výhodné pro zvolený postup aproximace. Další výhodou je, že knihovna udržuje informaci o okolních buňkách, která je využita pro sloučení jednotlivých buněk. [14]

3.3 Knihovna OPT++

Knihovna OPT++ je objektově orientovaná knihovna, která poskytuje algoritmy pro nelineární optimalizaci. Základní myšlenka knihovny je mít optimalizační úlohu oddělenou od metody, která ji řeší. Knihovna rozlišuje optimalizační úlohy podle dostupnosti derivací hodnotící funkce od nulté až k druhé derivaci.
Metody pro hledání minimální hodnoty hodnotící funkce, které jsou v knihovně OPT++ implementované, lze rozdělit do čtyř skupin: 1) metody přímého hledání, 2) metody zobecněných gradientů, 3) metody Newtonova typu a 4) metody vnitřního bodu. Více informací lze nalézt v [15] a v online dokumentaci [16].
Objekt reprezentující problém optimalizace trajektorie nemá dostupné derivace hodnotící funkce, proto je problém reprezentován instancí třídy NLF0. Při vytváření instance třídy NLF0 nlp($ndim$, $optFunc$, $optInit$) je $ndim$ počet reálných proměnných ve vektoru x, $optFunc$ je ukazatel na funkci, která počítá hodnotu hodnotící funkce, a $optInit$ inicializuje počáteční hodnoty složek vektoru x. Takto sestavený problém se referencí předá optimalizační metodě. Optimalizaci je možné ukončit podle následujících podmínek: maximální počet iterací, maximální počet vyhodnocení hodnotící funkce a velikostí změna hodnotící funkce v následující iteraci. Z knihovny OPT++ byla pro optimalizaci použita metoda „Parallel Direct Search (PDS) Method“. Výsledné hodnoty proměnných vektoru x se získají z definovaného problému. Optimalizace byla ukončena, když změna hodnotící funkce v následující iteraci byla menší než 0,0001.

4 Experimenty

V této kapitole jsou uvedeny výsledky experimentů, které byly provedeny ke zjištění vlastností navrženého algoritmu pro různé hodnoty parametru α z rovnice 21. Experimenty jsou provedeny na mapě překážek, která je zobrazena na obrázku 13. Startovní pozice je označena zelenou koulí a cílová pozice je označena červenou koulí. Na tomto obrázku 13 jsou také zobrazeny hrany Voroného buněk. Rozměry kvádru, který obklopuje mapu, jsou ve směru osy x 6,45 m, ve směru osy y 9,6 m a ve směru osy z 4,1 m.



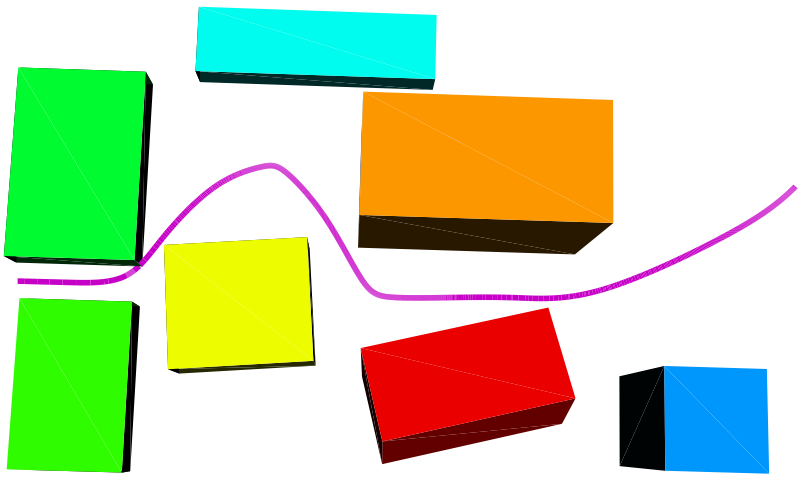
Obrázek 13: Mapa překážek s naplánovanou cestou

Naplánovaná cesta pomocí Dijkstrova algoritmu je převedena na B-spline křivku, která je zobrazena na obrázku 14. Počáteční bezpečnost této trajektorie je 125,2 a délka je 10,0 m. Tyto dvě hodnoty lze považovat jako výchozí pro porovnání optimalizované trajektorie. Dále je na obrázku 14 vidět, že B-spline křivka prochází velmi blízko rohu zelené překážky, což je nevýhodou popsaného algoritmu převodu naplánované cesty na B-spline křivku, viz kapitola 2.4.1. Pro mapu na obrázku 13 bylo provedeno šest simulací, při kterých se navzájem porovnaly výsledné trajektorie pro různé hodnoty parametru α. Výsledné hodnoty bezpečnostní funkce a délky trajektorie jsou uvedeny v levé části tabulky 1. V pravé části tabulky 1 jsou uvedeny výsledky experimentů pro mapu se dvěma překážkami.

Na obrázcích 16, 17 a 18 jsou zaznamenané hodnoty bezpečnostní funkce, délky cesty a výsledné hodnotící funkce pro dané parametry α. Zaznamenané hodnoty jsou vždy nejmenší podle hodnotící funkce v průběhu optimalizace. Proces optimalizace byl ukončen, když změna hodnotící funkce nebyla v následující iteraci větší jak 0,0001.

Na obrázku 15a je zobrazena trajektorie pro hodnotu parametru α =1,0 , což znamená, že na hodnotící funkci má vliv pouze bezpečnostní funkce. Výsledná trajektorie má pak největší vzdálenost od překážek v mapě. Z průběhu grafu 16a je patrné, že délka cesty se rychle dostane na hodnotu 15,8 m a hodnota funkce bezpečnosti rychle klesne na hodnotu 3,4.

Na obrázku 15b je zobrazena trajektorie, při které jsou obě kriteriální funkce vážené stejně α =0,5. Z průběhu grafu 16b je vidět, že nejdříve strmě klesá hodnota bezpečnostní funkce, a když se hodnoty bezpečnostní funkce F_B a délky trajektorie F_D přibližně rovnají, začne se postupně zmenšovat i délka trajektorie. Předchozí tvrzení dokládají obrázky 19. Na obrázku 19c je délka trajektorie nejdelší, na dalších obrazcích se zmenšuje. U experimentu 1 a 2 převládá vliv bezpečnosti nad délkou cesty.



Obrázek 14: Počáteční trajektorie v zadané mapě

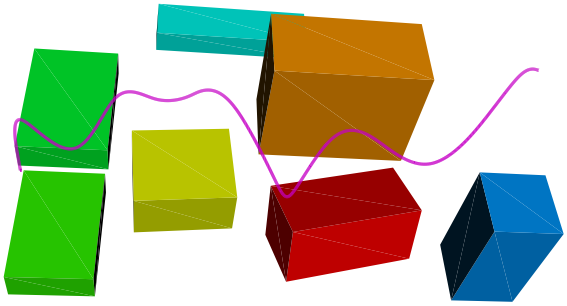
Na obrázku 15c je zobrazena trajektorie pro α =0,05 . Výsledná hodnota bezpečnostní funkce je o něco vetší než v předchozích experimentech. Vliv délky cesty se projevuje,

Tabulka 1: Tabulka výsledků provedených experimentů

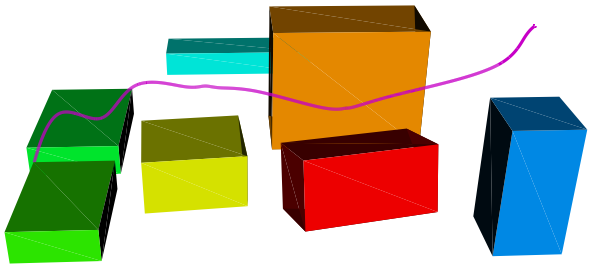
experiment	α	F_B	F_D[m]	počet iterací	experiment	α	F_B	F_D[m]	počet iterací
1	1,0	3,4	15,8	105	7	1,0	0,002	4,09	31
2	0,5	5,2	10,0	112	8	0,1	0,319	2,64	29
3	0,05	7,9	9,1	45	9	0,01	2,700	2,56	25
4	0,01	10,5	8,9	25					
5	0,002	113,5	8,6	43					
6	0,001	195,0	8,6	27					

ale funkce χ udržuje cestu dostatečně daleko od překážek. Podobně lze popsat i cestu zobrazenou na obrázku 17b. Při porovnání postupu optimalizace 17a a 17b lze pozorovat, že průběh hodnotící funkce v 17a vykazuje na začátku prudší pokles, naproti tomu v grafu 17b převládá vliv délky cesty.

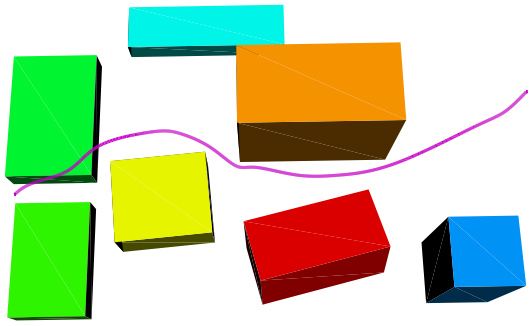
Na obrázku 15e a 15f jsou výsledky optimalizace pro α =0,002, resp. pro α =0,001. Hodnota bezpečnostní funkce je výrazně vyšší než pro předchozí hodnoty parametru α. Průběh optimalizace pro experiment číslo 5 je zachycen na obrázku 20. Optimalizovaná trajektorie se postupně přibližuje překážkám a její délka se zkracuje. Na obrázku 15f je vidět, že se výsledná trajektorie pohybuje velmi blízko hrany jedné překážky, z tohoto důvodu ji lze považovat za nepřípustnou. Navíc rozdíl délek trajektorií z experimentu 5 a 6 je nepatrný. Průběhy grafů 18a a 18b jsou silně ovlivněny délkou cesty. Pro α =0 je vliv bezpečnostní funkce úplně potlačen, což má za následek vygenerování nepřípustné trajektorie, proto lze hodnotu parametru α pro danou mapu omezit na $\alpha \in \langle 1; 0,002 \rangle$.



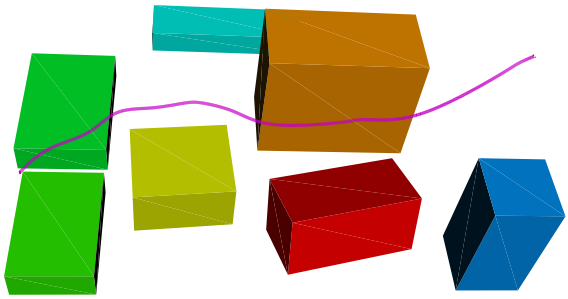
(a) Trajektorie pro α=1,0



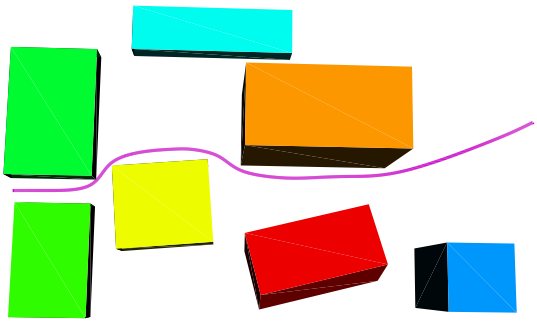
(b) Trajektorie pro α=0,5



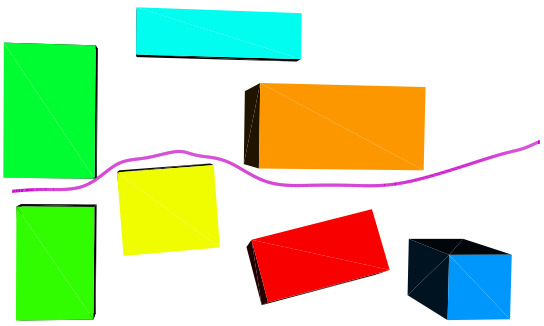
(c) Trajektorie pro α=0,05



(d) Trajektorie pro α=0,01

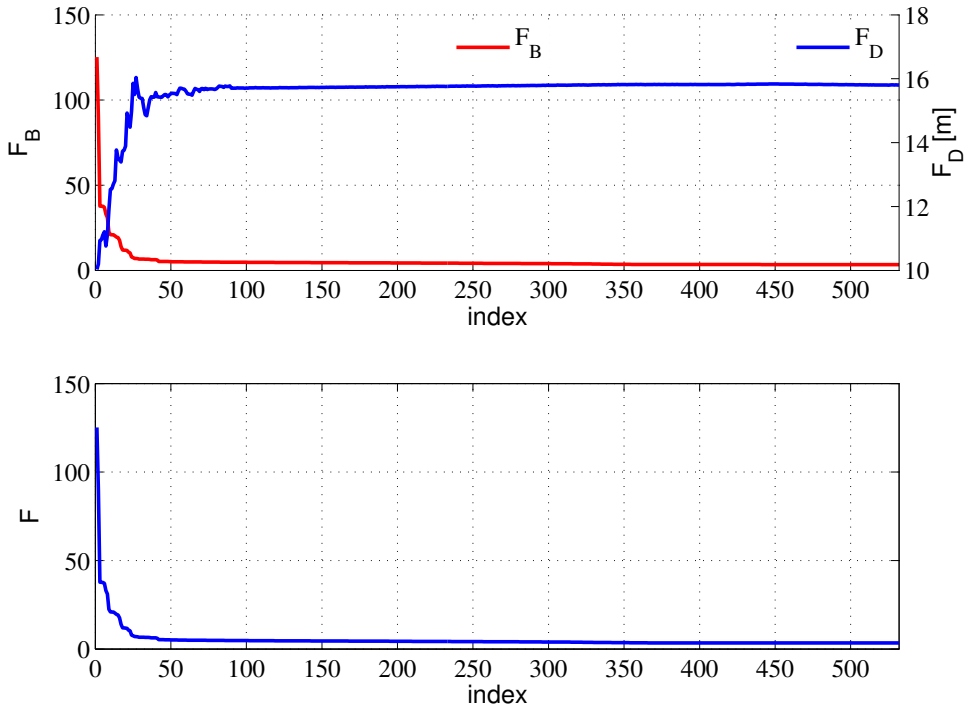


(e) Trajektorie pro α=0,002

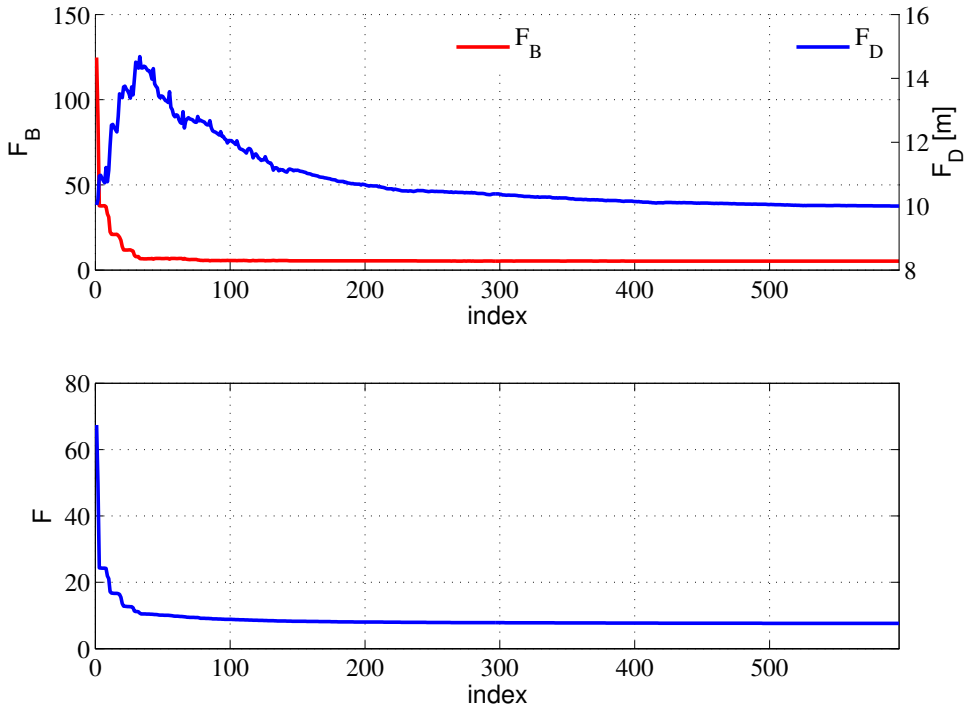


(f) Trajektorie pro α=0,001

Obrázek 15: Výsledné trajektorie po optimalizaci

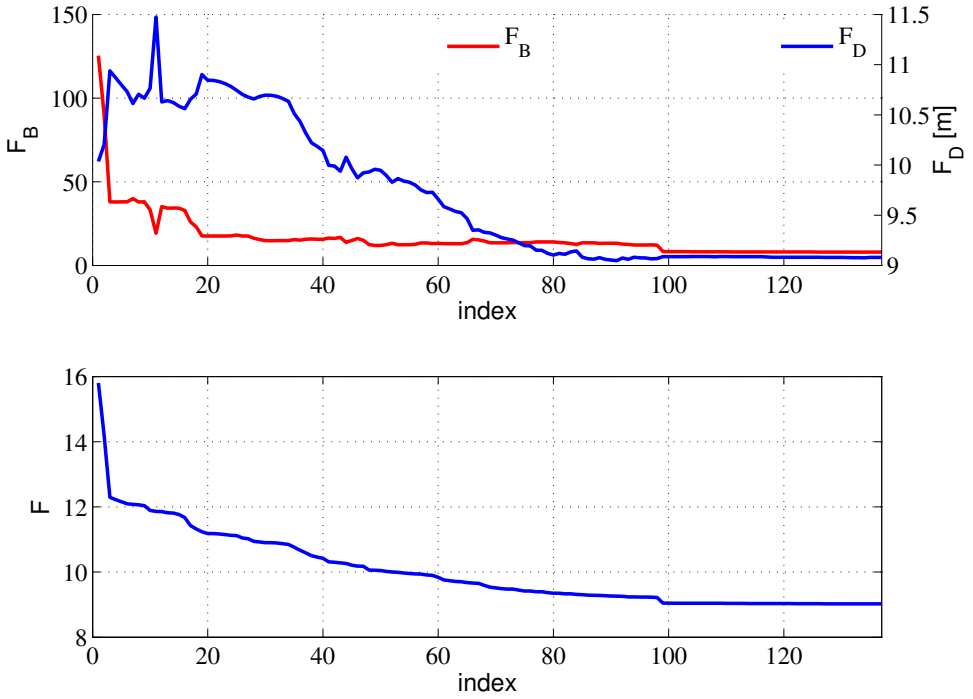


(a) Průběh optimalizace pro α=1,0

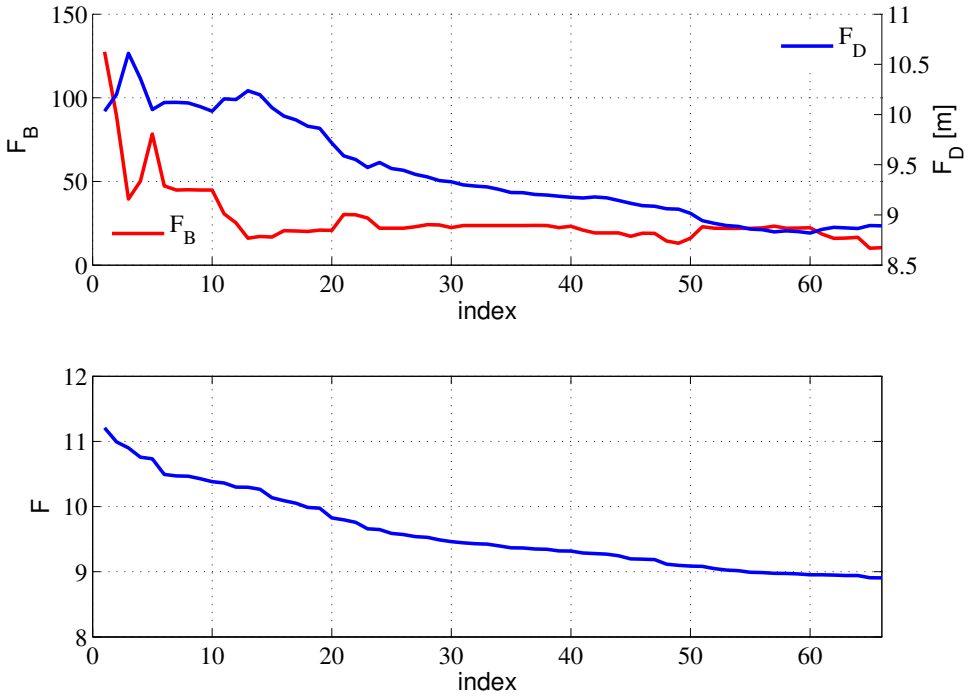


(b) Průběh optimalizace pro α=0,5

Obrázek 16: Průběh bezpečnostní funkce, délky trajektorie a hodnotící funkce 1

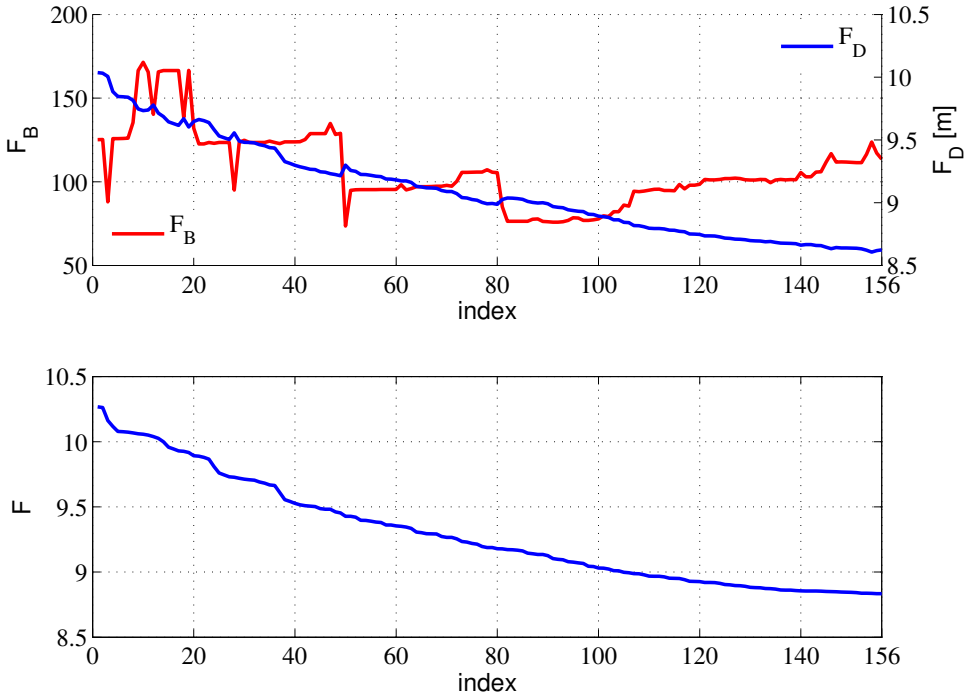


(a) Průběh optimalizace pro α=0,05

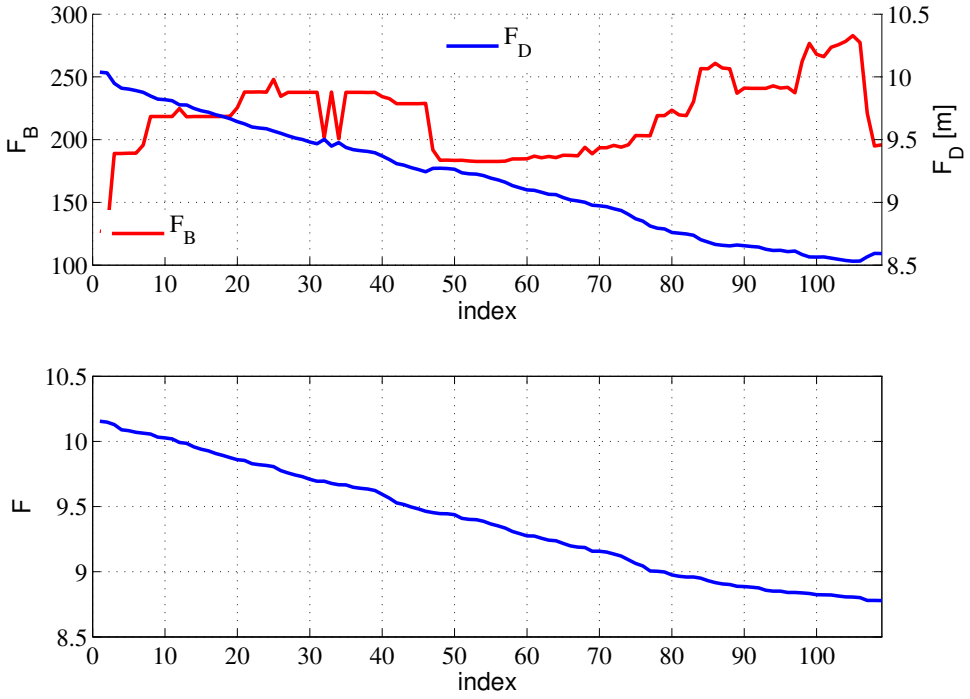


(b) Průběh optimalizace pro α=0,01

Obrázek 17: Průběh bezpečnostní funkce, délky trajektorie a hodnotící funkce 2

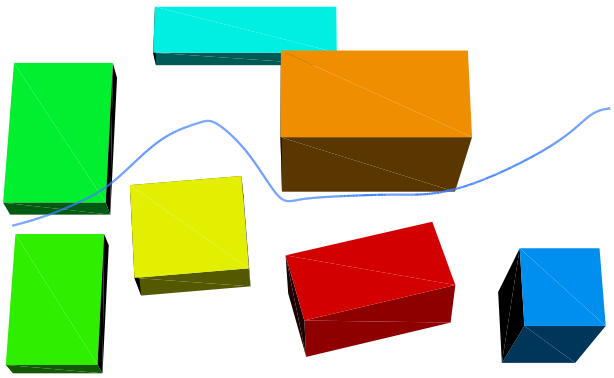


(a) Průběh optimalizace pro α=0,002

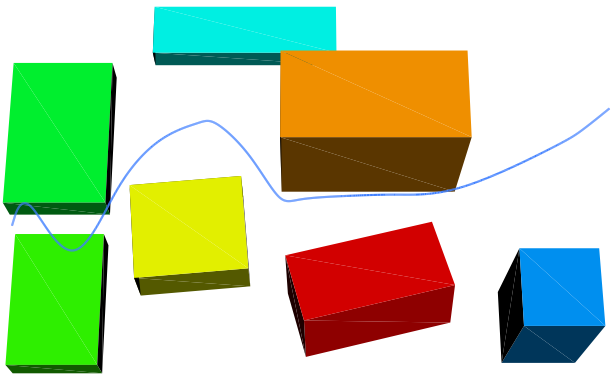


(b) Průběh optimalizace pro α=0,001

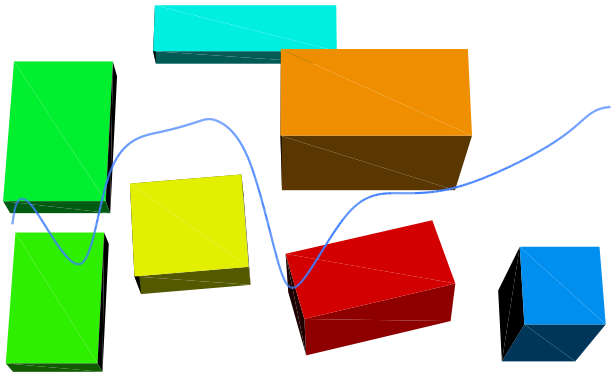
Obrázek 18: Průběh bezpečnostní funkce, délky trajektorie a hodnotící funkce 3



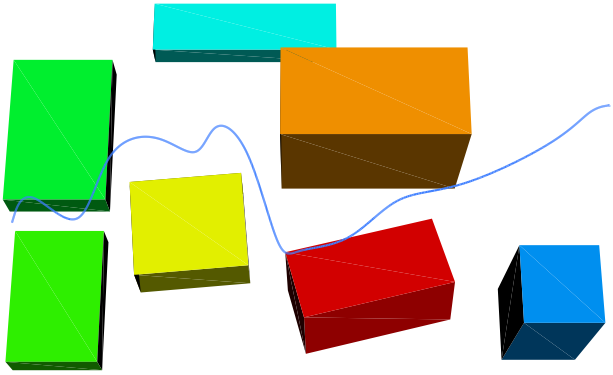
(a) Trajektorie v kroku 2



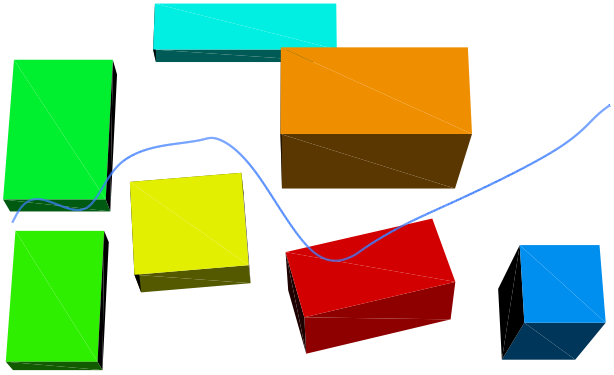
(b) Trajektorie v kroku 12



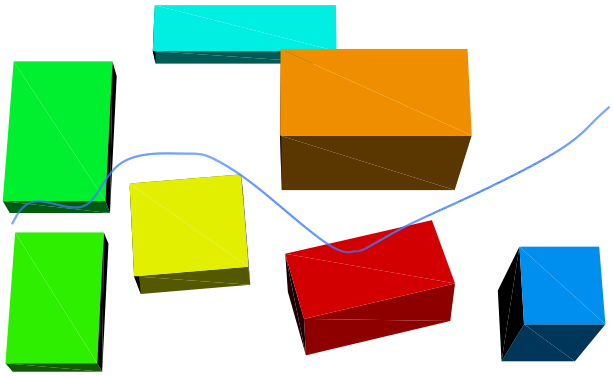
(c) Trajektorie v kroku 33



(d) Trajektorie v kroku 66

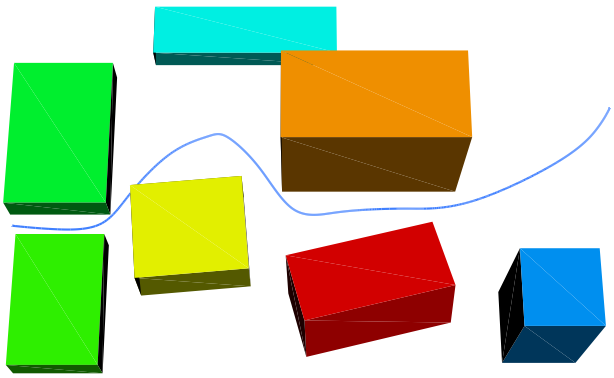


(e) Trajektorie v kroku 136

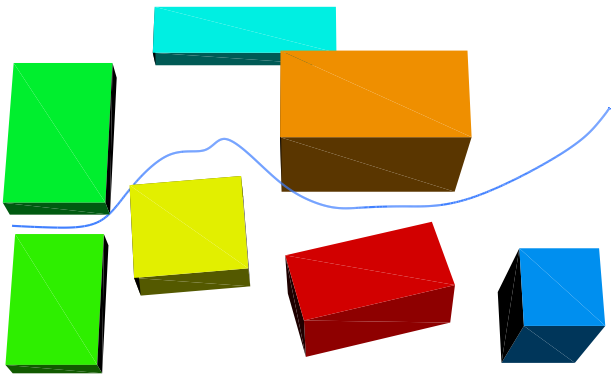


(f) Trajektorie v kroku 595

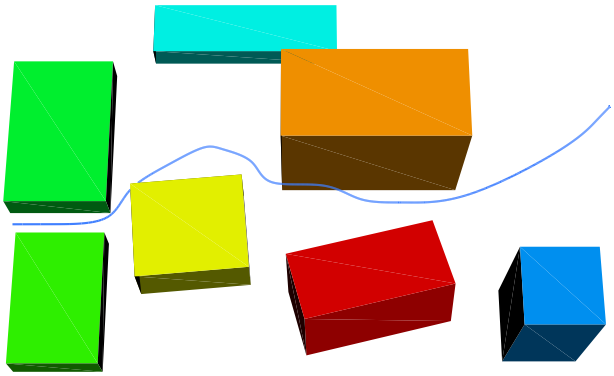
Obrázek 19: Průběh experimentu číslo 2



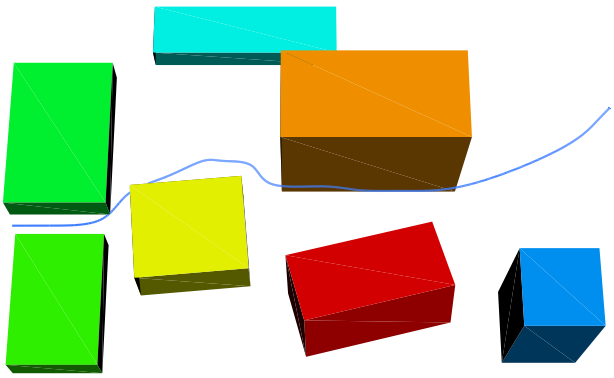
(a) Trajektorie v kroku 3



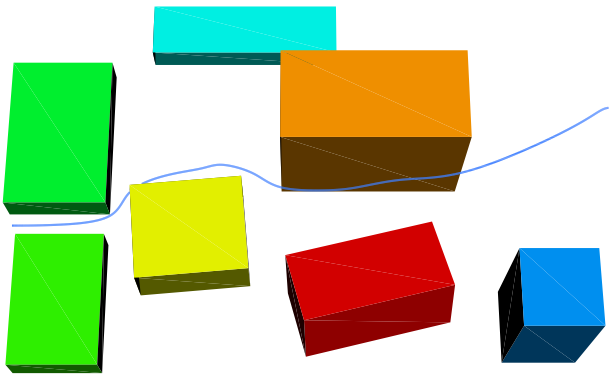
(b) Trajektorie v kroku 18



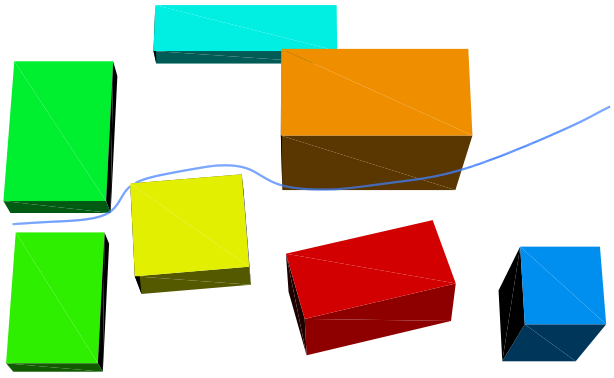
(c) Trajektorie v kroku 50



(d) Trajektorie v kroku 85



(e) Trajektorie v kroku 120

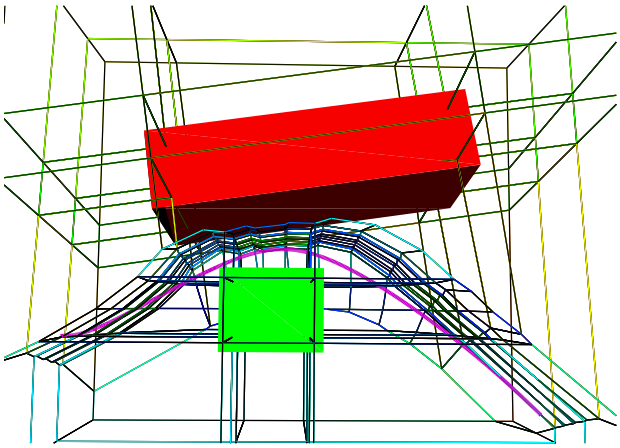


(f) Trajektorie v kroku 155

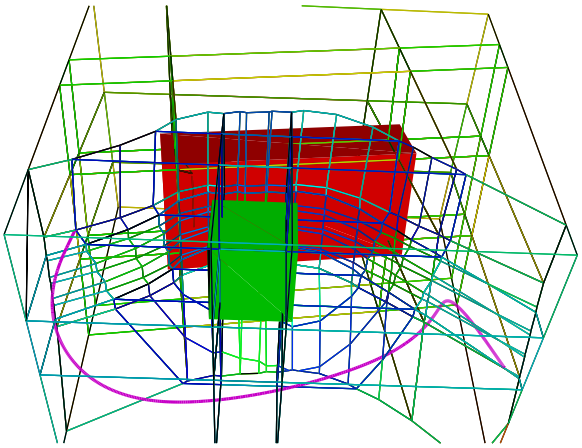
Obrázek 20: Průběh experimentu číslo 5

4.1 Objetí překážky

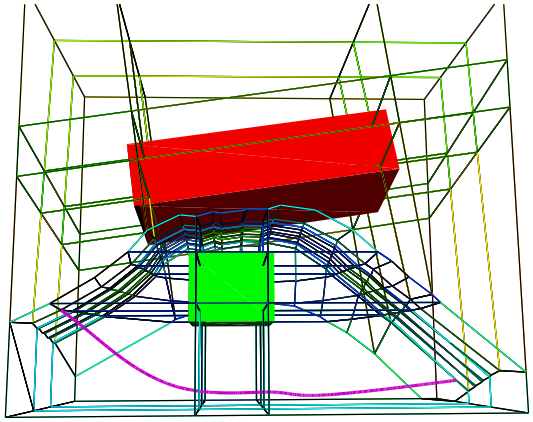
Další experiment ukazuje, jak algoritmus vygeneruje trajektorii, která objede překážku z jiné strany, než vede počáteční trajektorie. Na obrázku 21 jsou uvedeny výsledky optimalizace pro mapu tvořenou dvěma překážkami. Rozměry kvádru, který je obklopuje, jsou 2,3 m, 2,65 m a 1,87 m. Na obrázku 21a je zobrazena počáteční trajektorie, která vede mezi dvěma překážkami. Bezpečnost počáteční trajektorie je 139,9 a délka 3,0 metry. Hodnoty bezpečnostní funkce a délky trajektorie jsou uvedené v pravé části tabulky 1. Na obrázku 21b je zobrazena trajektorie pro α=1,0, výsledná B-spline křivka se drží co nejvíce od překážek, na dalších obrázcích 21c a 21d se při optimalizaci zohledňuje i délka trajektorie. Výsledné trajektorie vedou jenom kolem zelené překážky.



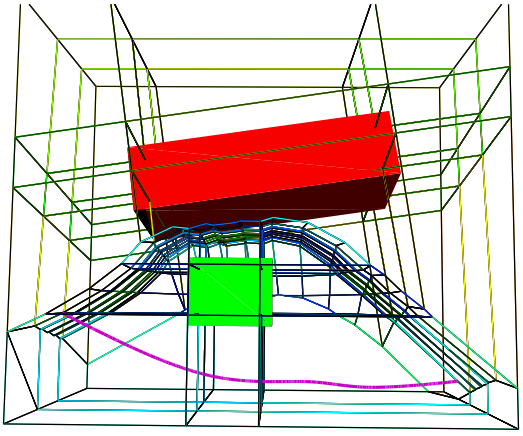
(a) Počáteční trajektorie



(b) Trajektorie pro α=1,0



(c) Trajektorie pro α=0,1



(d) Trajektorie pro α=0,01

Obrázek 21: Objetí překážky

5 Závěr

Byl navržen algoritmus pro optimalizaci počáteční trajektorie založený na efektivním výpočtu funkce vzdálenosti od překážek pro 3D prostředí. Zadaný problém byl rozdělen na menší problémy, které byly postupně řešeny. První problém spočíval v nalezení 3D Voroného diagramu pro prostředí překážek. Dle našich znalostí neexistuje použitelná knihovna pro výpočet Voroného diagramu pro 3D tělesa, proto byl navržen postup výpočtu, jehož hlavní myšlenka spočívá v aproximaci tvaru překážek body, pro které je výpočet Voroného diagramu jednoduší.

Návrh počáteční trajektorie a její optimalizace lze označit jako další problém, který bylo nutné vyřešit. Hrany a vrcholy jednotlivých Voroného buněk lze považovat za graf, proto byl použit k určení nejkratší cesty Dijkstrův algoritmus. Nalezená nejkratší cesta je převedena na trajektorii, která je popsána řídícími body B-spline křivky. Následně se trajektorie optimalizuje na základě funkce hodnotící vzdálenost B-spline křivky a překážek. Dalším optimalizačním kritériem je délka výsledné trajektorie.

Provedenými experimenty na mapě se sedmi překážkami ve tvaru kvádrů byla otestována funkčnost navrženého algoritmu. Výsledky experimentu jsou uspokojivé. Při experimentech se ukázalo, že navržený algoritmus má slabé místo, konkrétně při výpočtu průsečíku B-spline křivky a stěny Voroného buňky se vlivem zaokrouhlovací chyby může stát, že průsečík ležící uvnitř stěny blízko hrany je nesprávně určen mimo stěnu. Tento nedostatek má vliv na správnou hodnotu funkce bezpečnosti, což ovlivní výsledek optimalizace.

Na rychlost optimalizace má vliv použitá metoda, dodefinování gradientu hodnotící funkce by umožnilo použít metodu, která konverguje rychleji než současně používaná, například metodu zobecněných gradientů.

Reference

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[2] Inc. Kitware. Dokumentace VTK [online]. `http://www.vtk.org/doc/release/6.0/html/`. Přístup: 5/2/2014.

[3] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[4] E.G. Gilbert and D.W. Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *Robotics and Automation, IEEE Journal of*, 1(1):21–30, Mar 1985.

[5] Miroslav Kulich. Vyhlazování cesty pro autonomní vozítko. Master's thesis, Univerzita Karlova v Praze, Matematicko-fyzikalní fakulta, 1996.

[6] Rolf Klein, Elmar Langetepe, and Zahra Nilforoushan. Abstract voronoi diagrams revisited. *Computational Geometry*, 42(9):885 – 902, 2009.

[7] Josef Kolář. *Teoretická informatika*, volume 2. Česká informatická společnost, Praha, 2004.

[8] Ivana Linkeová. Základy počítačového modelovaní křivek a ploch. Vydavatelství ČVUT, Praha, 2008.

[9] Robert Michael Lewis, Virginia Torczon, and Michael W. Trosset. Direct search methods: then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191 – 207, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[11] Petr Olšák. Úvod do algebry, zejména lineární. Vydavatelství ČVUT, Praha, 2007.

[12] Růžena Černá, Miroslav Machalický, Jiří Vogel, and Čeněk Zlatník. *Základy numerické matematiky a programovaní*, volume 1. SNTL/ALFA, Praha, 1987.

[13] William J. Schroeder, Ken Martin, and W.E. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, Third Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002.

[14] Chris H. Rycroft. Voro++: A three-dimensional voronoi cell library in c++. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 19(4), 2009.

[15] J. C. Meza, R. A. Oliva, P. D. Hough, and P. J. Williams. OPT++: An object-oriented toolkit for nonlinear optimization. *ACM Trans. Math. Softw.*, 33(2), June 2007.

[16] Sandia Corporation. Dokumentace OPT++ [online]. `https://software.sandia.gov/opt++/index.html`. Přístup: 15/04/2014.

6 Příloha

6.1 Obsah přiloženého CD

V tabulce 2 jsou uvedena jména všech kořenových adresářů přiloženého CD s popisem obsahu.

Tabulka 2: Obsah CD

Jméno adresáře	**Popis obsahu**
dp	diplomová práce ve formátu pdf
ex	výsledky experimentů
lib	použité knihovny
source	zdrojové kódy programu