VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

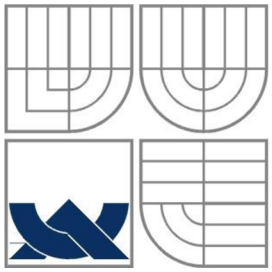
# VIZUALIZÁCIA ALGORITMOV PRE PLÁNOVANIE CESTY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

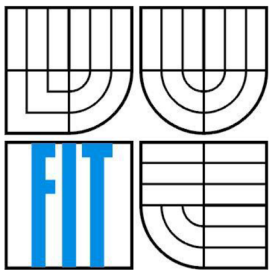
AUTOR PRÁCE
AUTHOR

FILIP MUTŇANSKÝ

BRNO 2016



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# VIZUALIZACE ALGORITMŮ PRO PLÁNOVÁNÍ CESTY

PATH PLANNING ALGORITHMS VISUALISATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

FILIP MUTŇANSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2016

# Abstrakt

Táto práca sa zaoberá algoritmami pre plánovanie cesty, obsahuje teoretický popis najznámejších prístupov pre plánovanie cesty a to: bug algoritmy, roadmapy, bunečná dekompozícia, potenciálové polia a pravdepodobnostné plánovanie. Tiež obsahuje návrh aplikácie, ktorá bude vizualizovať jednotlivé algoritmy.



# Abstract

This thesis deals with path planning algorithms,  there is description of most used approaches to path planning such as: bug algorithms, roadmaps, cell decomposition, potential fields, sampling based planning. There is also design of application, that will be visualizing certain algorithms.



# Klíčová slova

plánovanie cesty, bug algoritmy, roadmapy, bunečná dekompozícia, pravdepodobnostné plánovanie



# Keywords

path planning, bug algoritmy, roadmaps, cell decomposition, sampling based planning



# Citace

Filip Mutňanský: Vizualizácia algoritmov pre plánovanie cesty, bakalářská práce, Brno, FIT VUT v Brně, 2016

# Vizualizácia algoritmov pre plánovanie cesty

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana Ph.D.
Další informace mi poskytli...
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Filip Mutňanský
4.2.2016

## Poděkování

Rád by som poďakoval vedúcemu práce za cenné rady.

# Obsah

# 1 Úvod

Plánovanie cesty patrí medzi základné odvetvie robotiky. Plánovanie cesty je možné aplikovať v mnohých oblastiach ako napríklad v automatickej výrobe, počítačových hrách, prehľadávaní neznámeho priestoru, alebo v molekulárnej biológií.

Jeden z najstarších problémov plánovania ciest je *piano mover problem*, kde poznáme prostredie, rozloženie prekážok, rozmery klavíru a potrebujeme klavír preniesť z bodu A do bodu B bez toho aby nastala akákoľvek kolízia s prekážkou.

Existuje mnoho prístupov k plánovaniu cesty s rôznymi využitiami. Ak je potrebné nájsť cestu v neznámom prostredí, tak sú použité Bug algoritmy založené na senzoroch. Pre hľadanie cesty v známom prostredí, kde poznáme rozloženie prekážok sú použité roadmapy, potenciálové polia, bunečná dekompozícia, alebo pravdepodobnostné plánovanie.

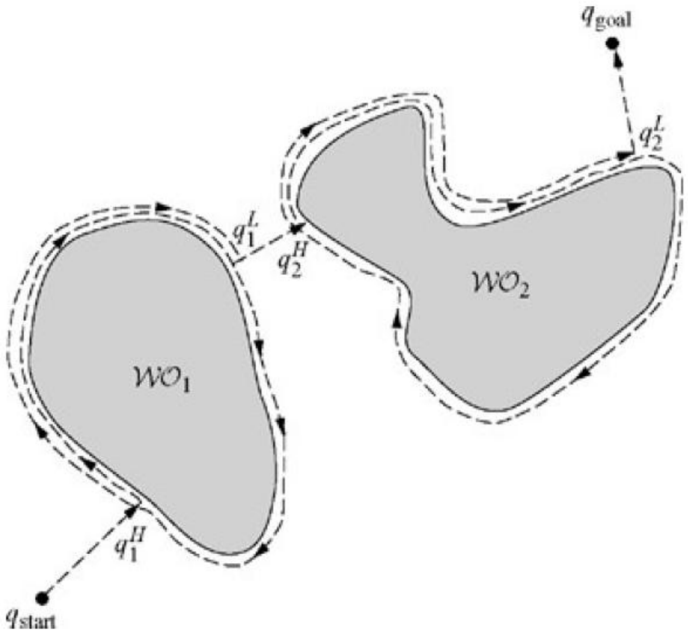
# 2 Bug algoritmy

Tento typ algoritmov patrí medzi algoritmy založené na senzoroch. Predpokladajú, že robot je bod v rovine obsahujúci senzor detekujúci prekážky. Tieto algoritmy požadujú dva typy správania: pohyb po priamke a obchádzanie prekážky. Ak má robot senzor s nulovým dosahom, je použitý algoritmus Bug1, alebo Bug2. Ak má senzor, nenulový dosah, je použitý Tangent bug algoritmus.

## 2.1 Bug 1 Algoritmus

Bug 1 predstavuje najjednoduchší prístup ku hľadaniu cieľa v neznámom priestore. Predpokladá, že robot je bod v rovine, má senzor s nulovým dosahom a pracovný priestor je ohraničený.

Štart a cieľ sú označené ako $q_{\mathtt{start}}$ a $q_{\mathtt{goal}}$. Na začiatku sa robot začne pohybovať z $q_{\mathtt{start}}$ do $q_{\mathtt{goal}}$, kým nedosiahne cieľ, alebo nenarazí na prekážku. Ak narazí na prekážku tento bod sa označí ako $q^H{}_i$. Robot následne obíde prekážku až pokým sa nevráti do bodu $q^H{}_i$. Následne robot určí na obvode prekážky bod najbližšie ku cieľu, tento bod je označený ako $q^L{}_i$ a robot do neho prejde. Ak priamka z bodu $q^L{}_i$ do bodu $q_{\mathtt{goal}}$. prechádza cez súčasnú prekážku, potom neexistuje cesta do cieľa. Následne sa vykoná cesta z bodu $q^L{}_i$ do $q_{\mathtt{goal}}$ a inkrementuje sa $\mathtt{i}$.



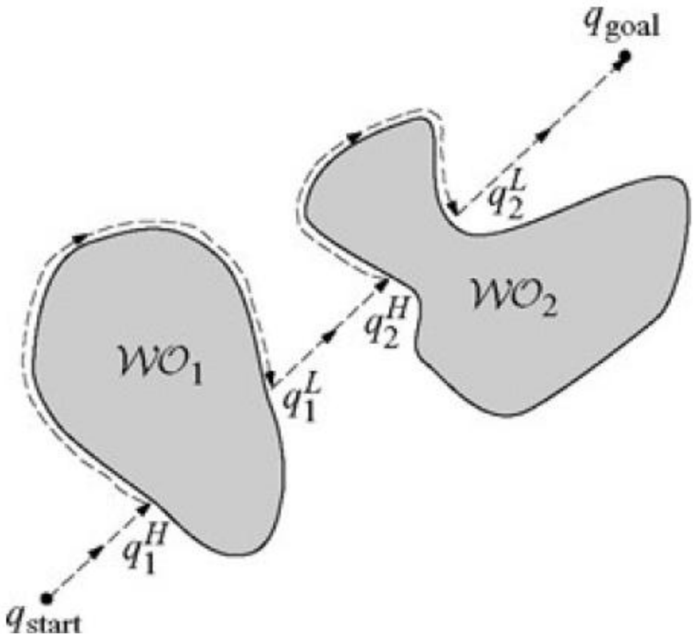

Obrázok 1: Hľadanie cesty algoritmom Bug 1

```
Input: Bod v ktorom sa robot nachádza
Output: Cesta do q_goal, alebo rozhodnutie, že cesta neexistuje
1: while True do
2:   repeat
3:     Od q^L_{i-1} sa pohni smerom ku q_goal
4:   until q_goal je dosiahnutý or je dosianutá prekážka v bode q_Hi
5:   if q_goal je dosiahnutý then
6:     Exit.
7:   end if
8:   repeat
9:     Nasleduj hranice prekážky.
10:  until q_goal je dosiahnutý or q^H_i je znova dosiahnutý
11:  Urči bod q^L_i na obvode prekážky, ktorý má najkratšiu cestu do q_goal
12:  Choď do bodu q^L_i.
13:  if robot sa hýbe smerom ku q_goal then
14:    Vyhlás, že q_goal je nedosiahnuteľný a exit.
15:  end if
16: end while
```

Algoritmus 1: Bug 1 algoritmus

## 2.2 Bug 2 algoritmus

Na rozdiel od Bug 1 obsahuje priamku medzi $q_{start}$ a $q_{goal}$. Na priesečníkoch tejto priamky s prekážkami vznikajú potenciálne body z ktorých bude robot opúšťať prekážku. Týmto sa docieli, že robot nebude musieť v niektorých prípadoch obchádzať celú prekážku. V niektorých prípadoch je Bug 1 efektívnejší, tento fakt je popísaný v nasledujúcej kapitole.




Obrázok 2: Hľadanie cesty algoritmom Bug 2

```
Input: Bod v ktorom sa robot nachádza
Output: Cesta do q_goal, alebo rozhodnutie, že cesta neexistuje
1: while True do
2:   repeat
3:     Od q^L_{i-1}, sa pohni smerom ku q_goal po priamke m.
4:   until q_goal je dosiahnutý or je dosianutá prekážka v bode
q_Hi
5:   Zatoč doľava (alebo do prava)
6:   repeat
7:     Nasleduj hranice prekážky
8:   until
9:   je dosiahnutý q_goal or
10:  je znovu dosiahnutý q^H_i or
11:  priamka m je znovu dosiahnutá v takom bode m, ktorý:
12:    m ≠ q^H_i,
13:    d(m, q_goal) < d(m, q^H_i), a
14:    Ak sa robot pohne smerom k cieľu, nenatrafí na prekážku
15:  Nech q^L_{i+1} = m
16: Inkrementuj i
17: end while
```

Algoritmus 2: Bug 2 algoritmus

### 2.2.1 Porovnanie s Bug 1

Na prvý pohľad sa môže zdať, že Bug 2 je efektívnejší ako Bug 1, pretože robot nemusí obchádzať celú prekážku, ale v niektorých prípadoch je efektívnejší Bug 1. Tento fakt je možné demonštrovať na dĺžkach jednotlivých ciest pri použití Bug 1 a Bug 2 algoritmu.

Pri použití algoritmu Bug 1 je potrebné obísť celú prekážku a následne prejsť do bodu $q^L_i$ tento bod je v najhoršom prípade vzdialený polovicu obvodu prekážky, čiže robot prejde jeden a pol obvodu prekážky.

Pri algoritme Bug 2 je $n_i$ bodov, z ktorých robot môže opustiť prekážku. Polovica týchto bodov je nepoužiteľných, pretože ak by sa robot vydal z tohto bodu ku cieľu, pohyboval by sa smerom ku prekážke a nastala by kolízia.

$$L_{Bug1} \leq d(q_{start}, q_{goal}) + 1{,}5 \sum_{i=1}^{n} p_i$$

Rovnica 1: Horná hranica dĺžky cesty pri algoritme Bug 1

$$L_{Bug2} \leq d(q_{start}, q_{goal}) + 0{,}5 \sum_{i=1}^{n} n_i p_i$$

Rovnica 2: Horná hranica dĺžky cesty pri algoritme Bug 2

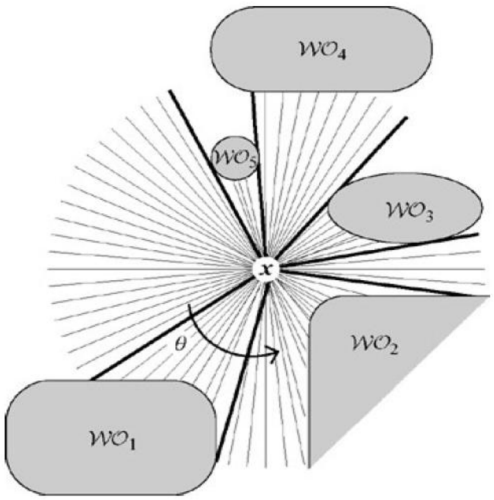
## 2.3 Tangent bug algoritmus

Tangent bug je rozšírenie Bug 2 algoritmu, ktoré určí kratšiu cestu k cieľu pomocou diaľkového senzoru s 360 stupňovou orientáciou. Tento senzor možeme modelovať ako funkciu: $\rho: \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$. Robot je bod v $x \in \mathbb{R}^2$ s lúčmi radiálne vychádzajúcich z tohto bodu. Pre každé $\theta \in S^1$ je hodnota $\rho(x, \theta)$ vzdialenosť od najbližšej prekážky na lúči z bodu $x$ pod uhlom $\theta$.

Pretože reálne senzory majú obmedzený dosah, je potrebné definovať funkciu $\rho_R: \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$. Táto funkcia naberá rovnaké hodnoty ako funkcia $\rho$ keď je v dosahu senzoru a nekonečno keď je mimo dosahu senzoru R. Formálne zapísané:

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \rho(x,\theta) < R \\ \infty, & inak \end{cases}$$

Od robota sa očakáva, že vie detekovať nespojitosti v $\rho_R$. Na obrázku 3 tenké čiary znázorňujú hodnoty $\rho_R(x,\theta)$ a hrubé čiary reprezentujú nespojitosti, ktoré nastanú v dvoch prípadoch: Keď prekážka prekrýva druhú, alebo keď je dosiahnutý dosah senzoru. Koncové body týchto intervalov sa označujú písmenom *O* a využívajú sa ďalej v hľadaní cesty.



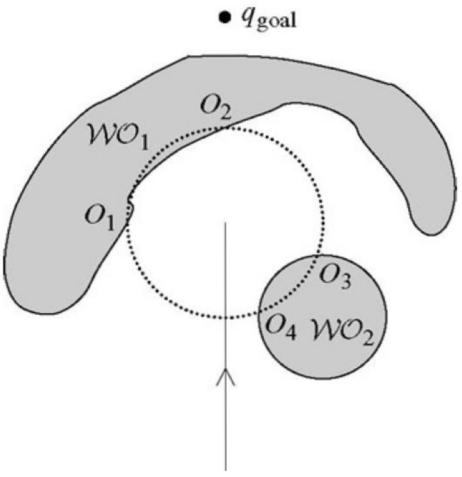

Obrázok 3: Nespojitosti v $\rho_R$

Robot najskôr vyvolá správanie typu pohyb k cieľu. Toto správanie má dve časti, najprv sa robot pohybuje priamo k cieľu pokým nezachytí prekážku vzdialenú R jednotiek priamo medzi robotom a cieľom. Na obrázku 4 to znamená, že priamka medzi $q_{goal}$ a robotom je preseknutá intervalom *WO1*.

Robot sa následne posúva smerom k bodu $O_i$ takom, že zmenšuje vzdialenosť k cieľu v našom prípade to je vzdialenosť $d(x, O_i) + d(O_i, q_{goal})$. Množina $\{O_i\}$ je neustále aktualizovaná podľa toho, ako sa robot hýbe.

Správanie typu pohyb k cieľu je realizované dovtedy kým pohyb nezmenšuje vzdialenosť k cieľu, to znamená, že našiel lokálne minimum $d(x, O_i) + d(O_i, q_{goal})$. Teraz sa robot prepne do režimu obchádzanie prekážky. V tomto režime sa robot pohybuje podobne ako v predchádzajúcom režime smerov ku $O_i$. Kým sa robot hýbe týmto smerom zároveň aj aktualizuje hodnoty $d_{reach}$ a $d_{followed}$. Hodnota $d_{followed}$ je najkratšia vzdialenosť medzi prekážkou, ktorá bola zaznamenaná a cieľom.

Hodnota $d_{reach}$ je vzdialenosť medzi cieľom a najbližším bodom na obchádzanej prekážke. Keď nastane $d_{reach} < d_{followed}$ robot končí obchádzanie prekážky.




Obrázok 4: Znázornenie intervalov spojitosti

*T* predstavuje bod kde dosah senzoru s polomerom R pretne priamku spájajúcu robota (*x*) a $q_{goal}$.

**Input:** Bod v ktorom sa robot nachádza
**Output:** Cesta do $q_{goal}$, alebo rozhodnutie, že cesta neexistuje
1: **while** True **do**
2: **repeat**
3: Pohybuj sa do bodu $n \in \{T, O_i\}$ tak, aby sa zmenšovala hodnota $d(x, n) + d(n, q_{goal})$
4: **until**
- cieľ je dosiahnutý **or**
- aktuálny smer začne zväčšovať hodnotu $d(x, q_{goal})$

5: Zvoľ smer obchádzania prekážky tak, aby pokračoval v smere posledného smeru štart - cieľ.
6: **repeat**
7: Aktualizuj $d_{reach}$, $d_{followed}$, and $\{O_i\}$.
8: Pohybuj sa smerom ku $n \in \{Oi\}$
9: **until**
- cieľ je dosiahnutý.
- robot obíde celú prekážku, čo znamená, že cieľ je nedosiahnuteľný
- $d_{reach} < d_{followed}$

10: **end while**

Algoritmus 3: Tangent bug algoritmus

# 3 Roadmapy

Na rozdiel od Bug algoritmov tento prístup vyžaduje znalosťp prostredia, to znamená polohu a tvar všetkých prekážok. Zväz jednorozmerných kriviek je roadmapa *RM* taká, ak pre každé $q_{start}$ a $q_{goal}$ v priestore $Q_{free}$ ktoré môžu byť spojené cestou majú tieto vlastnosti:

1. Dostupnosť - existuje cesta medzi $q_{start} \in Q_{free}$ do niektorých $q'_{start} \in Q_{free}$
2. Departability - existuje cesta medzi $q'_{start} \in RM$ do $q_{goal} \in Q_{free}$
3. Prepojiteľnosť - existuje cesta medzi v *RM* medzi $q'_{start}$ a $q'_{goal}$

Existuje 5 typov roadmáp: visibility maps, deformation retracts, retract-like structures, piecewise retracts a silhouettes. V tejto práci budú podrobnejšie popísané mapy viditeľnosti a deformation retracts.

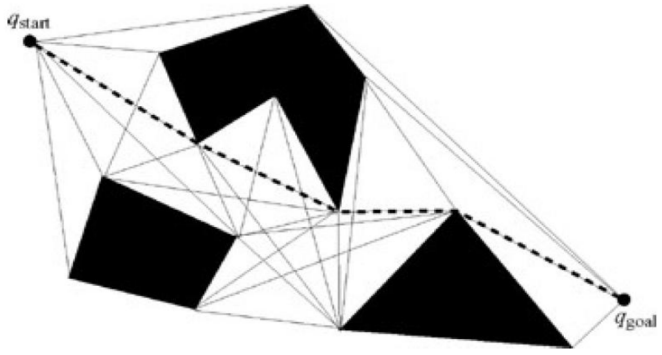
## 3.1 Mapy viditeľnosti

Základnými charakteristikami máp viditeľnosti sú to, že jej uzly zdieľajú hranu ak sa navzájom vidia a to, že všetky body v robotovom voľnom priestore sú v priamke s aspoň jedným uzlom mapy viditeľnosti. Ďalšou charakteristikou je, že má vlastnosti roadmapy dostupnosti, departability. Popíšeme si najjednoduchšiu mapu viditeľnosti graf viditeľnosti.

### 3.1.1 Graf viditeľnosti

Graf viditeľnosti je definovaný v dvojrozmernom priestore, pričom prekážky sú definované ako polygóny. Uzly $v_i$ grafu viditeľnosti zahŕňajú štartovaciu pozíciu, cieľ a všetky vrcholy prekážok. Hrany grafu $e_{ij}$ sú úsečky spájajúce navzájom viditeľné uzly $v_i$ a $v_j$.

$$e_{ij} \neq 0 \Leftrightarrow sv_i + (1-s)v_j \in cl(Q_{free}) \quad \forall s \in [0,1]$$




Obrázok 5: Graf viditeľnosti

Takto vytvorený graf má veľa nadbytočných hrán, ktoré sú pre hľadanie cesty zbytočné. Použitím supporting a separating priamok môžeme redukovať počet hrán v grafe viditeľnosti. Supporting hrana je dotyčnica dvoch prekážok, pričom obe prekážky ležia na jednej strane tejto priamky. Separating hrana je naopak dotyčnica, na ktorej ležia prekážky na opačných stranách.

Nakoniec zostanú iba tie hrany ktoré ležia na supporting, alebo separating hranách. Pri nekonvexných prekážkach zostanú hrany, ktoré keď sa trochu predĺžia, nebudú presahovať do prekážky (vrátane hraníc prostredia).

### 3.1.2 Konštrukcia grafu viditeľnosti

Nech $V = \{V_1, \ldots, V_n\}$ je množina vrcholov prekážok v priestore vrátane začiatočného bodu a cieľa. Pre vytvorenie grafu viditeľnosti musíme pre každé $v \in V$ určiť ktoré vrcholy sú pre $v$ viditeľné. Jednou z možností je overiť, či každé dvojice $vv_i, v \neq v_i$ nepretne hranu akéhokoľvek polygónu. Pre určitú dvojicu $vv_i$ je $O(n)$ priesečníkov pre otestovanie. Ak je $O(n)$ potenciálnych dvojíc z $v$, tak je $O(n^2)$ otestovaní pre určenie, ktoré vrcholy sú viditeľné z $v$. Toto všetko musí byť vykonané pre každé $v$ z množiny $V$ preto celková zložitosť konštrukcie bude $O(n^3)$. Čo je pomerne vysoká zložitosť, preto je potrebné prísť s jednoduchším riešením.

```
1:   Pre každý vrchol v_i, vypočítaj α_i, uhol z horizontálnou
     osou a vv_i.
2:   Vytvor list vrcholov ε, obsahujúci uhly α_i usporiadaných
     vzostupne podľa vzdialenosti od v.
3:   Vytvor zoznam hrán S, obsahujúci zoradené hrany, ktoré
     boli pretnuté pomocnou priamkou z v.
4:   for all α_i do
5:     if v_i je viditeľné z v then
6:       Pridaj hranu (v, v_i ) do grafu viditeľnosti.
7:     end if
8:     if v_i je začiatkom hrany E, ktorá nie je v S then
9:       Pridaj E do S.
10:    end if
11:    if v_i je koncom hrany v S then
12:      Zmaž hranu z S.
13:    end if
14:  end for
```

Algoritmus 4: Plane sweep algoritmus

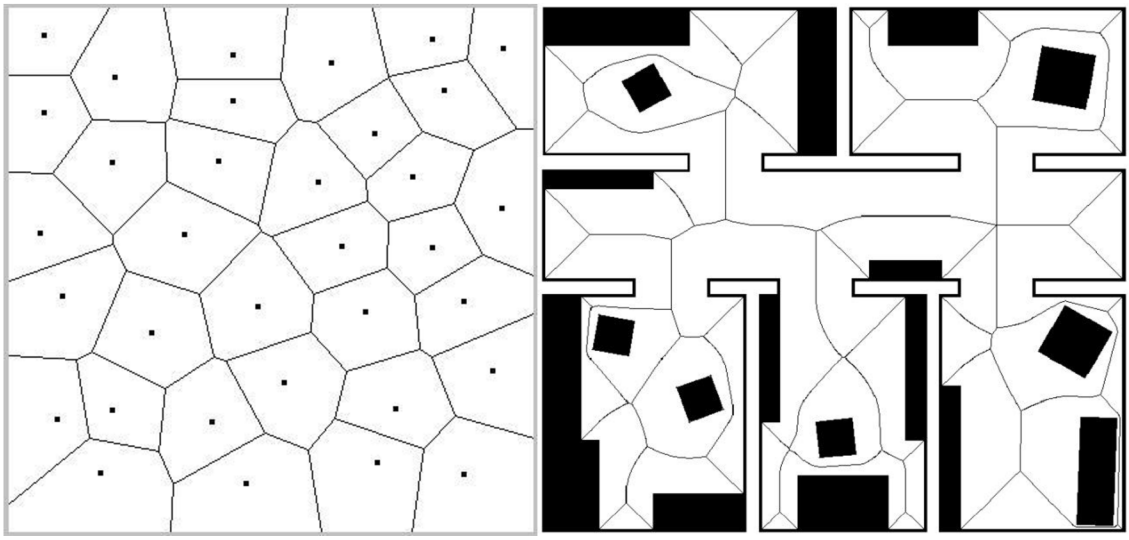
## 3.2 Voronoi diagram

Pre definovanie, čo je to Voronoi Diagram je potrebné vysvetliť pojem ekvidištancia. Bod je ekvidištantný od množiny objektov, ak vzdialenosti medzi tým bodom a každým objektom v danej množine je rovnaký.

Voronoi diagram (VD) je množina bodov nazývaných riadiace body. Voronoi oblasť je potom množina bodov ekvidištantná ku dvom riadiacim bodom, teda priestor je rozdelený do oblastí v ktorých je najbližšie iba ku jednému a to svojmu riadiacemu bodu. Pre potreby plánovania cesty je ale VD nepoužiteľný, pretože prekážky sú viacrozmerné ako riadiaci bod v jednoduchom Voronoi diagrame. Preto je potreba definovať Všeobecný Voronoi diagram (GVD) kde oblasť je množina bodov, ktoré majú najbližšie ku prekážke $O_i$.

$$F_i = \{q \in Q_{free} \mid d_i(q) \leq d_h(q) \quad \forall h \neq i\}$$

Kde $F_i$ je všeobecná Voronoi oblasť a $d_i(q)$ je vzdialenosť ku prekážke $O_i$ z $q$.



Obrázok 6: Porovnanie Voronoi diagramu a GVD

GVD splňuje všetky podmienky roadmapy a to dostupnosť departability a prepojiteľnosť. Hľadanie cesty je dosiahnuté pohybom od najbližšieho bodu na GVD, následne pohybom po diagrame ku blízkosti cieľa a potom z diagramu do cieľa.

# 4 Bunečná dekompozícia

Ďalším prístupom ku plánovaniu cesty a rozdeleniu priestoru je bunečná dekompozícia. V tomto prípade je priestor rozdelený na jednoduché regióny nazývané bunky. Dve bunky sú priľahlé, ak zdieľajú svoje hranice. Graf priľahlostí dokáže reprezentovať vzťahy priľahlosti medzi bunkami, kde uzol predstavuje bunku a hrana medzi uzlami znamená, že tieto bunky sú priľahlé.

Ak máme priestor rozdelený na bunky, tak plánovanie cesty prebieha v dvoch krokoch. Najprv sa určia bunky, ktoré obsahujú štartovaciu pozíciu a cieľ, následne sa prehľadáva graf za účelom nájdenia cesty štart - cieľ.

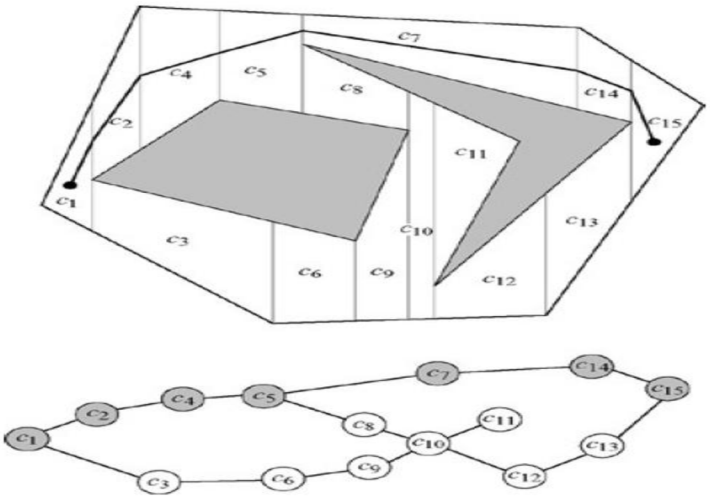
Na rozdiel od iných metód plánovania cesty poskytuje bunečná dekompozícia možnosť pokrytia celého priestoru. Pokrytie priestoru znamená prechod robota cez každý bod priestoru. Keďže žiadna bunka neobsahuje prekážky, je možné pokryť každú bunku jednoduchým prechádzaním po bunke.

Najznámejšou metódou rozdelenia priestoru je lichobežníková dekompozícia. Táto metóda vyžaduje, aby boli prekážky popísané polygónmi. Medzi všeobecnejšie metódy, ktoré si poradia aj s nepolygónovým popisom prekážok patrí Morseova dekompozícia.

## 4.1 Lichobežníková dekompozícia

Táto metóda vytvára bunky v tvare lichobežníkov, prípadne trojuholníkov. Predpokladá sa, že priestor je dvojrozmerný a prekážky, alebo hranice priestoru sú popísané polygónmi. Pre jednoduchosť predpokladajme, že každý vrchol $v_i$ má inú x súradnicu. Aby sa dané rozdelenie vytvorilo, je potrebné z každého vrcholu $v$ urobiť vertikálne predĺženie do oboch smerov. Predĺženie znamená vytvoriť priamku, ktorá začína vo $v$ a končí na prvej prekážke, alebo na hranici priestoru. Následne sa vytvorí graf priľahlostí.

Po určení, ktoré bunky obsahujú začiatočnú a koncovú pozíciu, je potrebné nájsť najkratšiu cestu zo štartu do cieľa. Výsledkom je sekvencia uzlov, čo pre naplánovanie cesty pre robota nestačí. Je potreba vytvoriť sekvenciu bodov v priestore, keďže lichobežník je konvexný, akékoľvek dva body sú spojiteľné úsečkou nekolidujúcou s prekážkou. Cesta pre robota sa vytvorí spájaním stredov vertikálnych predĺžení.




Obrázok 7: Vytvorenie buniek lichobežníkovou dekompozíciou, vytvorenie grafu a nájdenie cesty

# 5 Potenciálové polia

Metódu potenciálových polí je možné aplikovať na viac druhov priestorov, vrátane neeuklidiánskych, alebo viac rozmerných priestorov. Potenciálová funkcia je derivovateľná funkcia s reálnymi hodnotami $U: \mathbb{R}^m \rightarrow \mathbb{R}$. Na hodnotu potenciálovej funkcie môže byť nazerané ako na energiu, teda gradient potenciálu jej sily. Gradient je vektor:

$$\nabla U(q) = DU(q)^T = [\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)]^T$$

ukazuje smerom, ktorý lokálne maximálne zvyšuje *U*.

Táto metóda navádza robota akoby to bola častica pohybujúca sa v gradientovom vektorovom poli. Gradienty si môžeme predstaviť ako sily pôsobiace na pozitívne nabitý náboj predstavujúci robota. Prekážky sú nabité pozitívnym a cieľ negatívnym nábojom, čo by malo robota viesť okolo prekážok zo štartu do cieľa.

Robot ukončí pohyb keď sa dostane do bodu $q^*$ kde je gradient nulový, teda $\nabla U(q*) = 0$. Tento bod je nazývaný kritický bod *U*, môže to byť lokálne minimum, lokálne maximum, alebo sedlový bod. O aký bod sa jedná sa dá určiť z Hessovej matice. Hessova matica je matica obsahujúca druhé derivácie určitej funkcie, v našom prípade to bude funkcia *U(q)*. Ak je táto matica pozitívne definitívna, tak je kritický bod lokálne minimum, ak je negatívne definitívna, tak je kritický bod lokálne maximum.

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Obrázok 8: Hessova matica

## 5.1 Zostup gradientu

Základom tohto prístupu je pohyb zo začiatočnej konfigurácie po krokoch v smere opačnom ako hodnota gradientu. Tento pohyb vytvorí novú konfiguráciu a predchádzajúci krok je opakovaný kým nie je hodnota gradientu nulová.

V nasledujúcom algoritme je *q(i)* hodnota bodu *q* v i - tej iterácií, hodnota $\alpha$ určuje veľkosť kroku v i - tej iterácií. Výsledkom algoritmu je postupnosť bodov $\{q(0), q(1), \ldots, q(i)\}$. V reálnej implementácií sa nikdy nedosiahne hodnota gradientu v určitom bode nula, preto musíme počítať s vhodne zvolenou odchýlkou $|\nabla U(q(i))| < \varepsilon$.

```
1: q(0) = q_start
2: i = 0
3: while ∇U (q(i)) ≠ 0 do
4:    q(i + 1) = q(i) + α(i)∇U (q(i))
5:    i = i + 1
6: end while
```

Algoritmus 5: Zostup gradientu

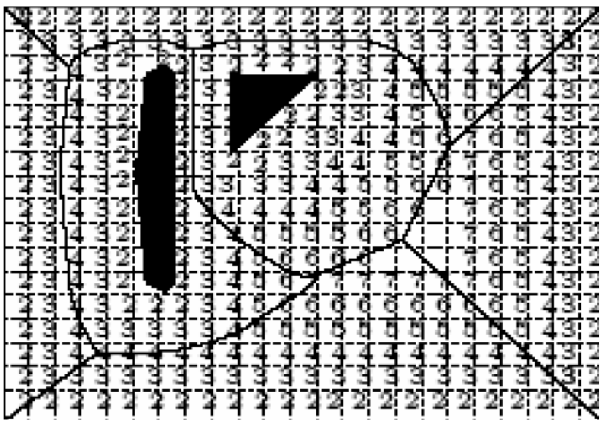
## 5.2 Plánovanie na mriežke

Pri plánovaní na mriežke je priestor reprezentovaný mriežkou, kde prvok mriežky (pixel) obsahuje nulu ak neobsahuje prekážku, a jednotku ak je čiastočne, alebo celkovo obsadený prekážkou. Ak berieme do úvahy vzťahy medzi dvoma susednými pixelmi na diagonále, tak hovoríme o osembodovej konektivite. Ak nie, tak sa jedná o štvorbodovej konektivite.

### 5.2.1 Brushfire algoritmus

Vstupom tohto algoritmu je mriežka zložená z pixelov s hodnotami jeden, alebo nula, podľa rozloženia prekážok. Výstupom je mriežka, ktorej hodnoty odpovedajú vzdialenostiam najbližšej prekážky. Z týchto hodnôt sa dá vypočítať potenciálová funkcia, alebo gradient.

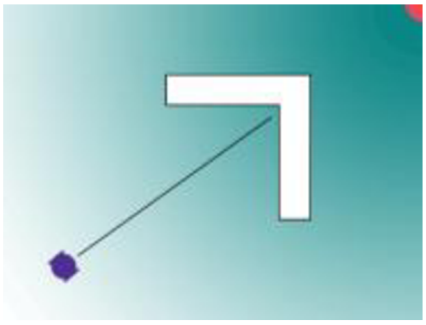
Na začiatku algoritmu sú každé nulové pixely, ktoré sú susedné k jednotkovým pixelom označené číslom tri. Následne sú nulové pixely susedné k pixelom s hodnotou dva označené číslom tri. Všeobecne sú každé nulové pixely susedné s $i$ sú označené hodnotou $i + 1$. Algoritmus končí, keď je všetkým pixelom priradená hodnota.



Obrázok 9: Mriežka vyplnená Brushfire algoritmom s použitím osembodovej konektivity

### 5.2.2 Problém lokálneho minima

Tento problém nastáva pri použití zostupného gradientu, lebo existuje možnosť lokálneho minima v potenciálovom poli. Robot sa snaží dostať do minima, ale nie je isté, že toto minimum je globálne. To znamená, že robot nenájde cieľ a uviazne v tomto bode, pretože algoritmus skončil kvôli nulovému gradientu. Tento problém nastáva pri konkávnych prekážkach, alebo pri priechode robota medzi prekážkami, ktoré sú v osovej súmernosti.



Obrázok 10: Problém lokálneho minima pri konkávnych prekážkach

### 5.2.3 Záplavové vyplňovanie

Záplavové vyplňovanie poskytuje riešenie ku problému lokálneho minima. Vstupom algoritmu je ako v prípade Brushfire algoritmu mriežka s nulovými, alebo jednotkovými pixelmi. Na začiatku je určený štartovací a cieľoví pixel. Hodnota cieľového pixelu je nastavená na dva. V prvom kroku algoritmu sú všetky nulové pixely susediace s pixelom s hodnotou dva, teda cieľom nastavené na hodnotu tri. Následne sú hodnoty všetkých nulových pixelov susediacich s pixelmi ohodnotenými číslom tri nastavené na štyri. Takto sa postupne prejde celou mriežkou a algoritmus končí keď sú všetky nulové pixely ohodnotené.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | **18** | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| 6 | 17 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 8 | 8 | 8 | 8 | 8 |
| 5 | 17 | 16 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 7 | 7 | 7 | 7 |
| 4 | 17 | 16 | 15 | 15 | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | 6 | 6 | 6 | 6 |
| 3 | 17 | 16 | 15 | 14 | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | 5 | 5 | 5 | 5 |
| 2 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 4 | 4 |
| 1 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 3 |
| 0 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | **2** |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Obrázok 11: Vyplnenie mriežky záplavovým vyplňovaním s použitím

# 6 Pravdepodobnostné plánovanie

Pravdepodobnostné plánovanie je najviac používaný prístup v praxi, je používané hlavne kvôli rýchlosti. Umožňuje plánovať cestu pre robotov s obmedzením, natočením kolies a podobne. Medzi základné pojmy tohto prístupu sú konfiguračný priestor a voľný konfiguračný priestor $Q_{free}$.

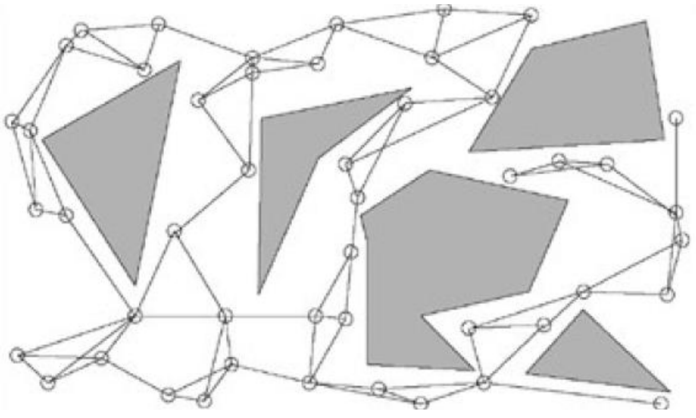
Konfiguračný priestor je n - dimenzionálny priestor, ktorý určuje pozíciu, alebo konfiguráciu robota podľa parametrov. Voľný konfiguračný priestor je priestor všetkých pozícií nekolidujúcich s prekážkou v ktorých sa môže robot nachádzať.

Princíp pravdepodobnostného plánovania sa vo všeobecnosti skladá s náhodného generovania konfigurácií a následného spájania s predošlými konfiguráciami. Následne je potrebné nájsť v grafovej štruktúre cestu zo štartu do cieľa.

Algoritmy na pravdepodobnostné plánovanie sa delia na jednodotazové a viacdotazové. Jednodotazové algoritmy umožňujú pre jeden vytvorený graf len jeden dotaz pre nájdenie cesty, patria sem algoritmy RRT a EST. Viacdotazové algoritmy umožňujú viac dotazov na jeden graf, medzi tieto algoritmy patrí PRM (probabilistic roadmaps).

## 6.1 PRM (probabilistic roadmaps)

PRM algoritmus najprv vytvorí náhodným generovaním konfigurácií roadmapu. Roadmapa je v tomto prípade neorientovaný graf $G = (V, E)$. $V$ je množina konfigurácií nad $Q_{free}$. $E$ obsahuje hrany medzi konfiguráciami, také, že neprechádzajú cez žiadnu prekážku. Na začiatku algoritmu je graf $G = (V, E)$ prázdny. Následne je náhodne vybraná konfigurácia z $Q$, ak je táto konfigurácia bez kolízií, tak je pridaná do roadmapy. Tento krok je opakovaný až kým sa nepridá $n$ uzlov do grafu. Pre každý uzol $q \in V$ je vybraná množina $N_q$ z $k$ najbližších susedov ku $q$ podľa metriky *dist*. Metrika *dist* je funkcia $Q \times Q \to \mathbb{R}^+ \cup \{0\}$. Plánovač sa potom pokúsi spojiť $q$ s každým uzlom $q' \in N_q$, ak je nájdená cesta medzi $q$ a $q'$, tak je hrana $(q, q')$ pridaná do roadmapy. Následne je nájdená najkratšia cesta medzi $q_{start}$ a $q_{goal}$ napríklad Djikstrovym algoritmom, alebo A*.



Obrázok 12: Graf PRM

**Input:**
$n$ : počet uzlov v roadmape
$k$ : počet najbližších susedných uzlov
**Output:**
roadmapa $G = (V, E)$
1: $V \leftarrow 0$
2: $E \leftarrow 0$
3: **while** $|V| < n$ **do**
4: **repeat**
5: $q \leftarrow$ náhodná konfigurácia v Q
6: **until** $q$ je bez kolízií
7: $V \leftarrow V \cup \{q\}$
8: **end while**
9: **for all** $q \in V$ **do**
10: $N_q \leftarrow k$ najbližších susedov z $q$ určených z $V$ podľa funkcie $dist$
11: **for all** $q' \in N_q$ **do**
12: **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ NIL **then**
13: $E \leftarrow E \cup \{(q, q')\}$
14: **end if**
15: **end for**
16:**end for**

Algoritmus 6: Algoritmus vytvorenia roadmapy v PRM

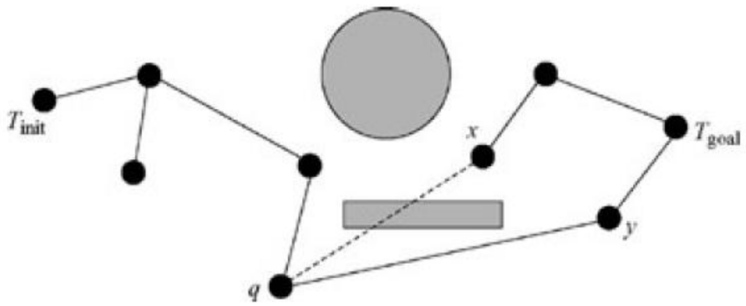
**Input:**
$q_{init}$: začiatočná konfigurácia
$q_{goal}$: cieľová konfigurácia
$k$: počet najbližších susedných uzlov
*roadmapa* $G = (V, E)$
**Output:**
Cesta z $q_{init}$ to $q_{goal}$, alebo uznanie, že cesta neexistuje
1: $N_{qinit} \leftarrow k$ najbližších susedných uzlov $q_{init}$ z $V$ podľa *dist*
2: $N_{qgoal} \leftarrow k$ najbližších susedných uzlov $q_{goal}$ z $V$ podľa *dist*
3: $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
4: nech $q'$ je najbližší susedný uzol $q_{init}$ v $N_{qinit}$
5: **repeat**
6: **if** $\Delta(q_{init}, q') \neq$ NIL **then**
7: $E \leftarrow (q_{init}, q') \cup E$
8: **else**
9: nech $q'$ je ďalší najbližší susedný uzol $q_{init}$ v $N_{qinit}$
10: **end if**
11:**until** bola nájdená cesta, alebo je množina $N_{qinit}$ prázdna
12: nech $q'$ je ďalší najbližší susedný uzol $q_{init}$ v $N_{qgoal}$
13:**repeat**
14: **if** $\Delta(q_{goal}, q') \neq$ NIL **then**
15: $E \leftarrow (q_{goal}, q') \cup E$
16: **else**
17: nech $q'$ je ďalší najbližší susedný uzol $q_{goal}$ v $N_{qgoal}$
18: **end if**

```
19:until bola nájdená cesta, alebo je množina N_qgoal prázdna
20:P ← najkratšia cesta(q_init, q_goal, G)
21:if P je neprázdne then
22:  return P
23:  else
24:    return failure
25:end if
```

Algoritmus 7: Algoritmus hľadania cesty v PRM

# 6.2 EST (Expansive-Spaces Trees)

EST je jednodotazový plánovač na rozdiel PRM, ktorý je viacdotazový. Cieľom jednodotazových plánovačov je odpovedať na dotaz čo najrýchlejšie. Graf konfigurácií sa nazýva strom. V prípade EST existujú dva stromy $T_{init}$ s koreňom $q_{init}$ a $T_{goal}$ s koreňom $q_{goal}$. Plánovač vyberie konfiguráciu $q$ z ktorej má strom $T$ rásť a navzorkuje náhodnú konfiguráciu $q_{rand}$ v okolí $q$. Následne sa plánovač pokúsi spojiť $q$ a $q_{rand}$, ak úspešne, tak je $q_{rand}$ pridané medzi vrcholy $T$ a $(q, q_{rand})$ je pridaný medzi hrany $T$. Tieto kroky sú opakované kým nie je do grafu pridaných $n$ konfigurácií. Na rozdiel od PRM, kde sú pridané všetky navzorkované konfigurácie, EST pridáva len tie ktoré sú spojiteľné s konfiguráciou od ktorej sa rozširuje. Spojenie stromov $T_{init}$ a $T_{goal}$ je docielené rozšírením prehľadávania priestoru jedného stromu do priestoru prehľadávaného druhým stromom. Na obrázku 13 sa plánovač snaží spojiť konfiguráciu $q$, ktorá patrí do stromu $T_{init}$, s najbližšími konfiguráciami $x$ a $y$ patriacimi ku stromu $T_{goal}$. Plánovač zlyhá pri konfigurácií $x$, ale uspeje v prípade $x$.




Obrázok 13: Spojenie dvoch stromov v EST

```
Input:
q0: konfigurácia, ktorá je koreňom stromu
n : počet pokusov o rozšírenie stromu
Output:
strom T = (V, E) s koreňom q0 a má ≤ n konfigurácií
1: V ← {q0}
2: E ← ∅
3: for i = 1 to n do
4:    q ← náhodná konfigurácia z T s pravdepodobnostou πT
5:    extend EST (T, q)
6: end for
7: return T
```

Algoritmus 8: Algoritmus Build EST

```
Input:
T = (V, E): EST strom
q: konfigurácia z ktorej sa má T rozširovať
Output:
nová konfigurácia qnew v okolí q, alebo NIL v prípade zlyhania
1: qnew ← náhodná konfigurácia bez kolízií z okolia
2: if Δ(q, qnew) then
3: V ← V U {qnew}
4: E ← E U {(q, qnew)}
5: return qnew
6: end if
7: return NIL
```

Algoritmus 9: Algoritmus Extend EST

## 6.3 RRT

Nech $T$ je jeden zo stromov $T_{init}$ s koreňom $q_{init}$, alebo $T_{goal}$ s koreňom $q_{goal}$. Každý strom je inkrementálne rozširovaný. V každej iterácií je navzorkovaná konfigurácia $q_{rand}$ . Potom je nájdená najbližšia konfigurácia $q_{near}$ ku $q_{rand}$ v $T$. Ak sa posunieme *step_size* jednotiek od $q_{near}$ po pomyselnej priamke medzi $q_{near}$ a $q_{rand}$ tak dostaneme miesto novej konfigurácie $q_{new}$. Hodnota *step_size* musí byť vhodne určená. Najvhodnejšie je túto hodnotu určovať dynamicky podľa vzdialenosti medzi $q_{new}$ a $q_{near}$.

Existuje aj iná alternatíva s použitím funkcie *connect*, táto funkcia volá funkciu *extend* kým $q_{new}$ nedosiahne $q_{rand}$. Vytvára sa veľký počet uzlov.

```
Input:
q0: konfigurácia, ktorá je koreňom stromu
n: počet pokusov o rozšírenie stromu
Output:
strom T = (V, E) s koreňom q0 a má ≤ n konfigurácií
1: V ← {q0}
2: E ← 0
3: for i = 1 to n do
```

```
4:    q_rand ← náhodná konfigurácia bez kolízií
5:    extend RRT (T, q_rand)
6: end for
7: return T
```

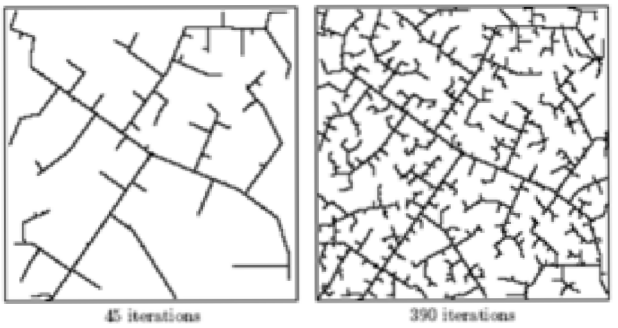
Algoritmus 10: Algoritmus build RRT

```
Input:
T = (V, E): RRT strom
q: konfiguracia ku ktorej T rastie
Output:
nová konfigurácia q_new ku q, alebo NIL v prípade zlyhania
1: q_near ← najbližší sused q v T
2: q_new ← posun q_near o step_size na priamke medzi q_near a q
3: if q_new je bez kolízií then
4: V ← V U {q_new}
5: E ← E U {(q_near, q_new)}
6: return q_new
7: end if
8: return NIL
```

Algoritmus 11: Algoritmus extend RRT

```
Input:
T = (V, E): an RRT
q: konfigurácia, ku ktorej strom T rastie
Output:
connected ak q je spojené s T; inak failure
1: repeat
2: q_new ← extend RRT (T, q)
3: until (q_new=q or q_new= NIL)
4: if q_new = q then
5: return connected
6: else
7: return failure
8: end if
```

Algoritmus 12: Algoritmus connect RRT




Obrázok 14: Porovnanie RRT stromov pri rozdielnych počtoch iterácií

# 7 Návrh aplikácie

## 7.1 Editovanie

Editor bude umožňovať užívateľovi vytvoriť priestor v ktorom sa bude robot pohybovať. Prekážky budú zadávané ako polygóny. Trieda *Editor* poskytuje metódy pre pridávanie a editovanie prekážok, menenie štartovacej polohy robota, menenie polohy cieľu. Na zachytenie vstupov od užívateľa a vykresľovanie bude použitá trieda *canvas*.

## 7.2 Vizualizácia

Súčasťou vizualizácie je vykreslenie robota v takej polohe v akej sa nachádza, prípadne vykresliť trasu akou robot prešiel. Medzi vizualizáciu patrí aj vyznačenie v akej časti algoritmu sa práve simulácia nachádza. Vizualizáciu bude mať na starosti trieda *Visualisation*.

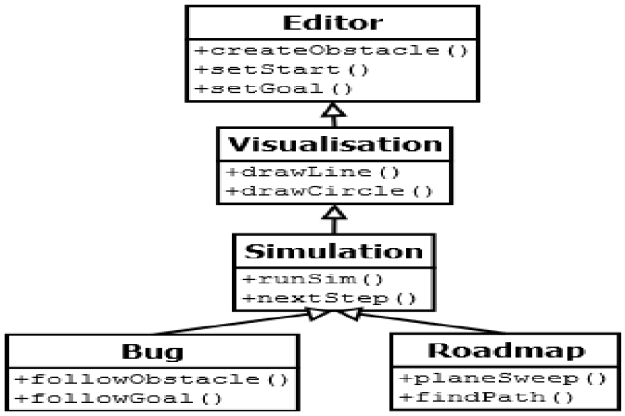
## 7.3 Simulácia

Simuláciu bude riešiť trieda *Simulation*. Táto trieda je jadro celej simulácie, umožňuje spustiť celú simuláciu, alebo sa posunúť o ďalší krok v algoritme pomocou metód *runSim(), nextStep()*. Pozastavenie v určitej fázy je docielené pomocou metódy *wait()* a spustenie *notify()*.

## 7.4 Roadmapy a bug algoritmy

Typy plánovania cesty, ktoré budú implementované sú roadmapy a bug algoritmy. Pri roadmapách som zvolil ako roadmapu graf viditeľnosti a na jeho tvorbu bude použitý algoritmus z kapitoly 3 *Plane sweep algoritmus*. Vytvorený graf viditeľnosti bude vizualizovaný pomocou metód z triedy *Visualisation*.

Pri Bug algoritmoch bude potrebné vizualizovať prejdenú cestu a priebežné hitpoints a leavepoints. V Tangent bug algoritme bude znázornený dosah senzoru. Pohyb robota bude implementovaný ako posun robota o niekoľko jednotiek a priebežné volanie metód pre vykreslenie.




Obrázok 15: Hierarchia tried

# 8 Záver

V tejto práci boli predstavené najpoužívanejšie metódy plánovania cesty v robotike. Tiež tu bol predstavený návrh webovej aplikácie, ktorá bude implementovaná vrátane textového popisu jednotlivých algoritmov. Pri implementácií bude oddelená vizualizácia od výpočtu, čo umožní jednoduchšie pridávať ďalšie algoritmy.

# Literatúra

[1] Choset, H.: *Principles of Robot Motion: Theory, Algorithms and Implementations*, 2005, ISBN 0-262-03327-5