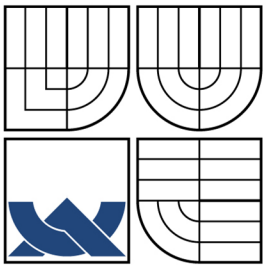
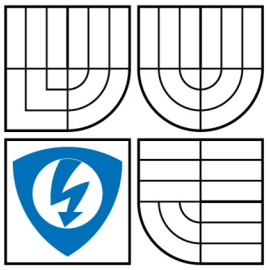


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A
KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

INDOOR ROBOT – ŘÍDÍCÍ NEURONOVÁ SÍŤ

INDOOR ROBOT – CONTROL NEURONS NET

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — PAVEL KŘEPELKA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. LUDĚK ŽALUD, PH.D.
SUPERVISOR

BRNO 2009

ABSTRAKT

V dokumentu popisuji možnosti navigace mobilních robotů. Tato problematika je řešena mnoha různými přístupy, ovšem dodnes není zcela vyřešena. Naleznete zde popis jednoduchých deterministických algoritmů, které lze použít pro jednoduché akce, jako je objíždění překážek nebo jízda v koridoru. Pro globální navigace však deterministické algoritmy selhávají. Další částí dokumentu je teorie umělých neuronových sítí (perceptron, vícevrstvé sítě, samoorganizující se sítě) a jejich použití v robotice. Vlastní navigační algoritmy byly otestovány na vytvořeném mobilním robotu nebo v simulačním softwaru popsaným v kap. 6. Návrh vlastních řídících algoritmů je založen právě na neuronových sítích (Kohonenova mapa), ať už pro navigaci do jednoho cíle nebo komplexní globální navigaci. V dokumentu je uvedeno srovnání jednotlivých přístupů k navigaci, jejich výhody a nevýhody. Cílem bylo nalézt efektivní algoritmus pro navigaci a umělá inteligence se zdá být tím správným řešením.

KLÍČOVÁ SLOVA

Mobilní robot, robotika, umělá inteligence, umělé neuronové sítě, navigace, řízení, algoritmy, plánování, deduktivní vrstva, Kohonenova síť, perceptron.

ABSTRACT

In this document, I describe possibilities of mobile robot navigation. This problems are solving many different ways, but there isn't satisfactorily result to this day. You find there describe of deterministic algorithms, this algorithms can be used for simply actions like obstacle avoiding or travel in corridor. For global navigation this algorithms fails. In next part of document is theory of artificial neural nets (perceptron, multi layer neural nets, self organization map) and using them in mobile robots. Own navigation algorithms was tested on constructed mobile robot or simulated in SW described in chapter 6. Design own control algorithms is based on neural net (Kohonen net). Designed algorithms can be used for one-point navigation or complex global navigation. In document, there is comparing of various ways to navigation, their advantages and disadvantages. Goal of this document is find effective algorithm for navigation and artificial intelligence appears to be the right solution.

KEYWORDS

Mobile robot, robotics, artificial intelligence, artificial neural network, navigation, control, algorithms, planning, deductive layer, Kohonen networks, perceptron.

B I B L I O G R A F I C K Á C I T A C E

Křepelka, Pavel. *Indoor robot – řídící neuronová síť* . Bakalářská práce FEKT – VUT v Brně, 2009, 60s., 1 příloha. Vedoucí práce: doc. Ing. Luděk Žalud, Ph.D.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Indoor robot – řídící neuronová síť jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **1. června 2009** ……
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Luďku Žaludovi, Ph.D za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **1. června 2009** ……
podpis autora

OBSAH

1. ÚVOD

Dokument má za úkol obecně nastínit problematiku navigace mobilní robotiky, obzvláště použití umělých neuronových sítí. Následně představuje robustní řešení tohoto problému pomocí SOM.

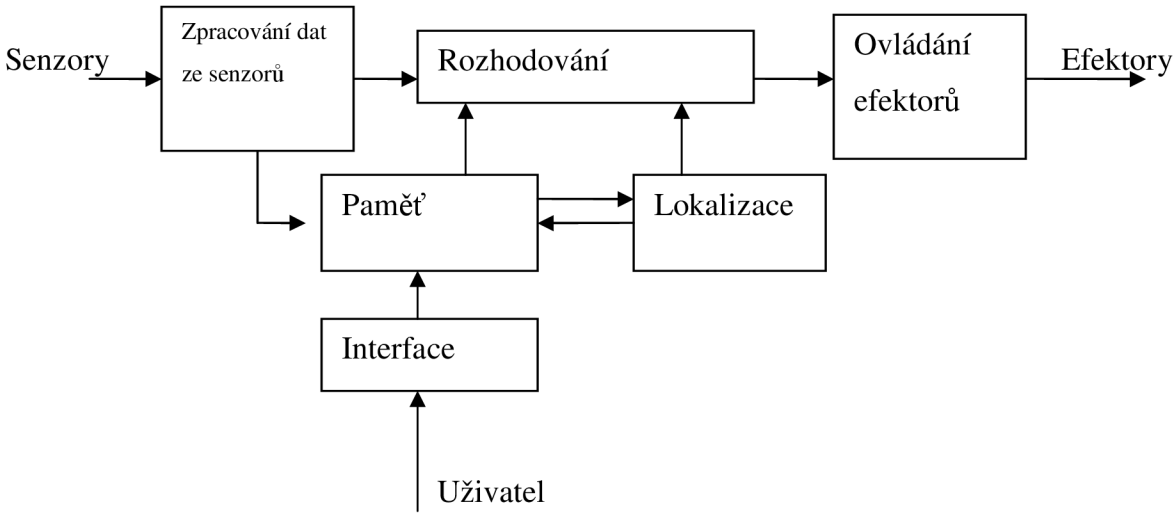
V první části dokumentu se věnuji některým jednoduchým deterministickým algoritmům použitých pro navigaci mobilních robotů. Po dosažení závěru nevhodnosti těchto algoritmů se věnuji využití umělé inteligence. V obsáhlých kapitolách 4 a 5 jsou srovnány jednotlivé typy neuronových sítí vhodných pro řízení robotů, jejich popis, praktické aplikace a stručný závěr. Pozornost je věnována hlavně percepronu, vícevrstvé neuronové síti, asociativní paměti a SOM. Pro složitější prostředí je jednoznačně vybrána SOM. Jsou představeny některé algoritmy pro navigaci do jednoho bodu. V kapitole 7 je navržen robustní algoritmus pro globální navigaci mobilního robotu v reálném čase. Je zde podrobně vysvětlen proces učení SOM. Tento proces je inicializační částí představovaného SOMAP algoritmu. V následujících kapitolách je charakterizován postup generování oblastních bodů, jakožto zobecnění výstupu SOM. Dále jsou uvedeny problémy SOMAP algoritmu a návrh jeho řešení.

2. ŘÍDICÍ SYSTÉMY ROBOTŮ

Řídicí systém autonomního robotu tvoří jeho „inteligenci“ a je zásadní částí výsledného chování robotu. Nejjednodušší systémy jsou implementovány na mikrokontrolérech (PIC,AVR…), složitější algoritmy pracují na průmyslových počítačích, PC, embedded systémech apod. Řídicí systém musí být schopen zpracovávat data z okolí interpretovaná senzory v reálném čase a obsluhovat příslušné efektory (motory).

Základní úkoly řídicích systémů autonomních mobilních robotů:

- Přečíst, vhodně interpretovat, analyzovat a popř. zrekonstruovat data ze senzorů.
- Vhodný interface pro zadání dat od uživatele (např. cíl cesty, úloha apod.).
- Nějaká forma lokalizace (většinou nutná paměť pro uchování mapy apod.).
- Rozhodování o následné konfiguraci robotu (např. směr cesty robotu, natočení ramena…).
- Ovládání efektorů (PWM, krokové motory).



Obrázek 2-1 Obecné schéma řídícího algoritmu.

Rozhodování v řídicích systémech není triviální záležitost. Je nutné akci dlouhodobě plánovat (časově náročné), ale zároveň reagovat na významné změny okolního prostředí. Obecné bokové schéma řídicího algoritmu je zachyceno na obrázku 2-1. Z tohoto důvodu má rozhodovací proces většinou tyto vrstvy:

- Reaktivní – zajišťuje bezprostřední reakci na daný podmět, pro správné vykonání akce stačí aktuální data ze senzorů (nebo pár sekund stará).
- Deduktivní – plánuje dlouhodobé akce (hledání cesty) většinou nad mapou prostředí.

Pokud systém obsahuje obě tyto vrstvy, mluvíme o tzv. kombinovaných softwarových architekturách.
Podle typů algoritmu lze řídící systémy robotů dělit na:

- Deterministické systémy - založeny na deterministických algoritmech s konstantní složitostí, stavových automatech, algoritmech na hledání cest v grafu apod.
- Systémy s umělou inteligencí – použití neuronových sítí, lineární asociátor, lze sem zařadit i pravděpodobnostní plánování (RRT).

3. DETERMINISTICKÉ ALGORITMY

3.1 BUG ALGORITMY [1]

Tyto reaktivní deterministické algoritmy popsané V. Lumenskym dokážou najít neoptimální cestu v neznámem prostředí s překážkami libovolného tvaru s konečným obvodem.

3.1.1 Bug1

Robot jede po přímce START-CÍL do chvíle kontaktu s překážkou. V místě kontaktu se označí bod H (hit). Po objetí celé překážky se vypočítá nejbližší bod k cíli a označí se L (leave). Robot dojede kratší cestou do bodu L a odtud se odpoutá a pokračuje směrem k cíli.

3.1.2 Bug2

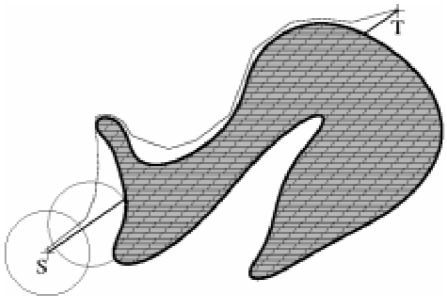
Stejné jako Bug1. Od překážky se odpoutá po protnutí virtuální přímky START-CÍL.

3.1.3 Bug1+2

Navigace pomocí Bug2, v případě selhání (průsečík je dále od cíle než H) pracuje podle Bug1.

3.1.4 VisBig

Bug2 doplněný a senzor vzdálenosti (obrázek 3-1).



Obrázek 3-1 VisBug[1].

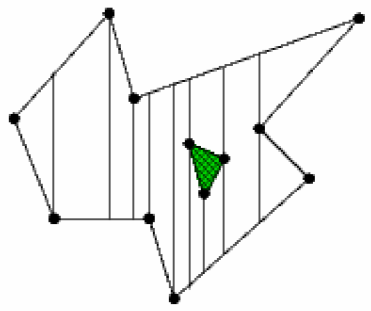
Lokalizace je většinou řešena odometrií, v paměti se uchovávají pouze souřadnice startu, cíle, H, L a pozice robotu. Výhodou je jednoduchost a dobrá funkčnost pro jednoduché prostředí. Tyto algoritmy nevyžadují téměř žádnou paměť. Ve složitém, nekonvexním či dynamickém prostředí selhávají.

3.2 EXAKTNÍ PLÁNOVÁNÍ [1]

Exaktní plánování je takové, které najde cestu vždy pokud existuje. Tyto deduktivní algoritmy pracují nad mapou prostředí. Ta je implementována ve většině případech ve formě lichoběžníků.

3.2.1 Lichoběžníková dekompozice

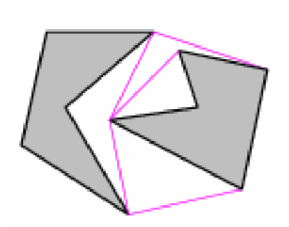
Jádrem algoritmu je dělení prostoru na jednodušší útvary (lichoběžníky – obr. 3-2), kde se nevyskytuje překážka. Pokud se robot a cíl nachází ve stejném lichoběžníku, jednoduše se vydá směrem k cíli. V opačném případě se najde pomocí hranového grafu seznam lichoběžníků které má robot překonat.



Obrázek 3-2 Lichoběžníková dekompozice[1].

3.2.2 Graf viditelnosti

Robot se pak pohybuje na úsečkách spojující vrcholy navzájem viditelných vrcholů překážek, jak je vidět na obrázku 3-3.

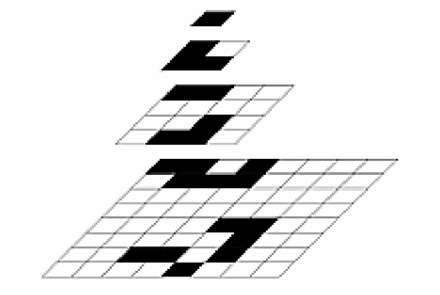


Obrázek 3-3 Graf viditelnosti [1].

Výhodou těchto algoritmů je schopnost najít optimální cestu vždy, pokud existuje. Jedná se o poměrně jednouché a paměťově ani výpočetně nenáročné algoritmy. Problém je ve vytvoření mapy. Pro vytvoření mapy prostředí je potřeba velice přesná sebelokalizace, která je v praxi téměř nemožná. Další nevýhodou je předpoklad bezrozměrného robotu pohybujícím se ve 2D světě. Nepříjemná vlastnost je tzv. GIGO efekt (garbage in – garbage out). To znamená že úspěšnost navigace značně závisí na přesnosti senzorů (při tvoření mapy nebo samotné navigaci).

3.2.3 Rastrové mapy, plánování na mřížce

Tento způsob plánování odstraňuje nevýhodu závislosti na přesných senzorech. Vektory u exaktních algoritmů jsou popsány souřadnicemi o určité přesnosti, které ale senzory se současnou technologií nedosahují. V případě rastrových map se prostor rozdělí na buňky o určité velikosti, každá buňka se pak označí jako prázdná/obsazená. Algoritmy se dají urychlit hierarchickým uspořádáním map s menším rozlišením, tj. vygenerování mapy s většími buňkami (pokud je aspoň jedna buňka v původní mapě, kterou nová překrývá, obsazená, nová buňka bude též označena jako obsazená). Hierarchické mapy jsou uvedeny na obrázku 3-4. Na tyto mapy lze použít mnoho algoritmů z oblasti grafové nebo maticové reprezentace (Potenciálová pole, A* …).

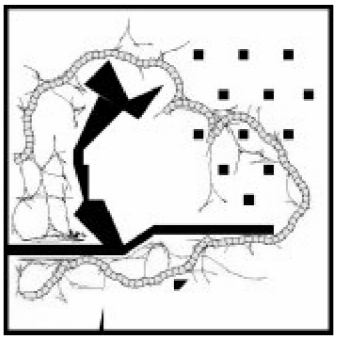


Obrázek 3-4 Hierarchické skládání map [1].

4. NĚKTERÉ NEDETERMINISTICKÉ ŘÍDICÍ ALGORITMY

4.1 PRAVDĚPODOBNOSTÍ PLÁNOVÁNÍ [1]

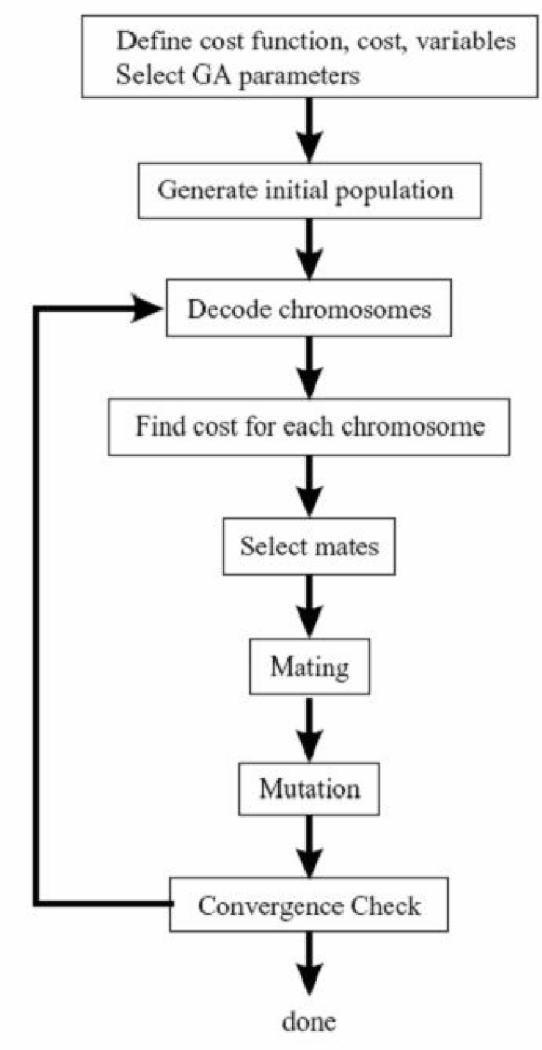
Pravděpodobnostní plánování je zajímavý algoritmus, který na základě náhodného vzorkování konfiguračního prostoru vytváří možné cesty pro robot. Algoritmus náhodně vybere souřadnici na mapě a najde k ní nejbližší bod již zkontrolované trajektorie, od tohoto bodu táhne trajektorii ke zvolenému bodu (s respektováním pravidel pohybu robotu – zatáčení po kružnicích apod.). Další souřadnice, které musí algoritmus často „vybírat“, je cíl cesty. Na vykonstruované cesty (obr. 4-1) se opět aplikují algoritmy pro hledání nejkratší cesty v grafu. Z výhodou se používá pro Ackermanovo řízení. Zde ovšem přetrvává problém vytvoření mapy prostoru.



Obrázek 4-1 Pravděpodobnostní plánování [1].

4.2 GENETICKÉ ALGORITMY [6]

Tyto stále více populární algoritmy lze efektivně použít nad velmi složitou mapou prostředí. Tímto způsobem je možné najít přijatelnou cestu za krátký čas. Další možností je využití evolučních algoritmů pro optimalizaci neuronových sítí (tzv. evolučně vyšlechtěné neuronové sítě). Toto je jedna z cest pro další vylepšení SOMAP algoritmu. Principiální schéma je uvedeno na obr. 4-2.



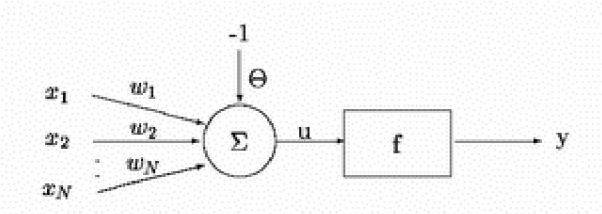
Obrázek 4-2 Princip genetických algoritmů [7].

5. UMĚLÉ NEURONOVÉ SÍTĚ

Umělé neuronové sítě jsou systémy sestávající se z výpočetních jednotek – neuronů, které jsou navzájem propojeny. Výpočet samotného neuronu spočívá ve vynásobení vstupů s příslušnými vahami – po sumarizaci se aplikuje tzv. aktivační funkce, jejíž výsledek je výstup neuronu, který může být vstupem pro další neurony nebo může být výstupem sítě. Schopnost adaptovat váhy – učit se (na základě předložených trénovacích dat – s učitelem), je základní vlastností umělých neuronových sítí. Další důležitou vlastností je nacházet závislosti v trénovacích datech a ty reprezentovat pomocí vah (učení bez učitele). Umělé neuronové sítě jsou analogií některých neuronových architektur nalezených v mozku živých organismů.

5.1 PERCEPTRON

Zobecněním modelu neuronu F.Rosenblatt navrhl tzv. perceptron (obr 5-1). Jeho vstupy mohou nabývat hodnot z množiny reálných čísel, výstup je binární. Vstupní vektor se vynásobí váhovým vektorem perceptronu, na výsledek (vnitřní potenciál neuronu) se aplikuje aktivační funkce, nejčastěji funkce signum nebo jednotkový skok. Výstup této funkce je výstupem neuronu.



Obrázek 5-1 Model perceptronu [2].

Výpočet i-tého neuronu je tedy:

$$y_i = sign(\sum_j w_{ij} x_j) \text{ . [2]} \qquad (1)$$

Učení probíhá s učitelem, tzn. síti se předloží množina vstupních dat a množina příslušných výstupních dat. Následně se upraví váhy tak, aby výsledek konvergoval k žádanému výstupu:

$$\varpi_i(t+1) = \varpi_i(t) + \eta(d(t) - y(t))x_i \text{ . [2]} \qquad (2)$$

Kde

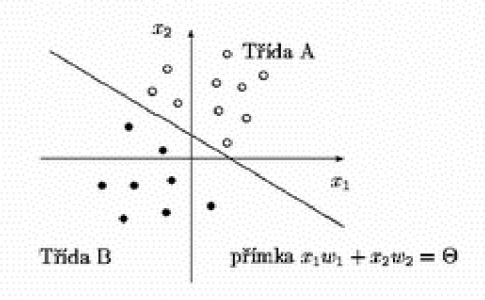
ϖ_i je modifikovaná váha neuronu

η je parametr učení – o kolik se změní váhy (vstupní parametr)

d je požadovaný výstup

x_i je příslušný vstup neuronu

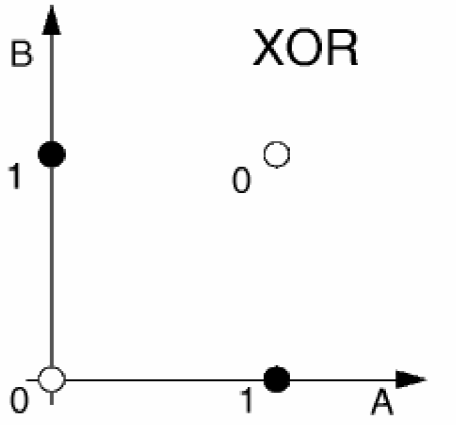
Z funkce perceptronu vyplývá, že jeden perceptron dokáže lineárně rozdělit vstupní prostor na dva podprostory, jak je vidět na obrázku 5-2.



Obrázek 5-2 *Rozdělení konfiguračního prostoru jedním perceptronem [2].*

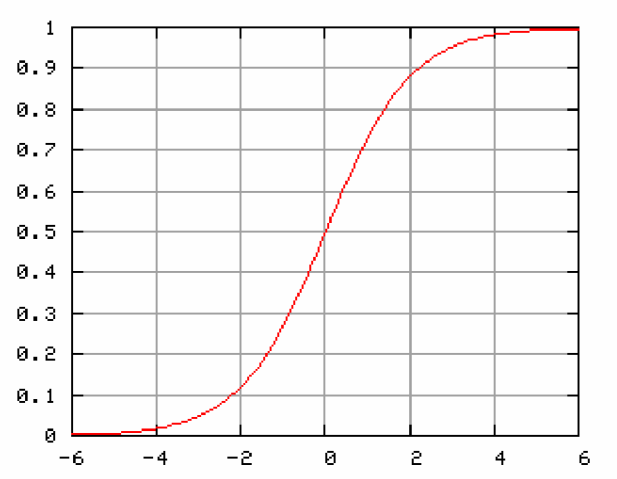
5.2 VÍCEVRSTVÁ NEURONOVÁ SÍŤ – SÍTÉ TYPU BACK-PROPAGATION

Jeden perceptron nedokáže klasifikovat vzory, které jsou lineárně neseparibilní, například funkce XOR (obrázek 5-3).

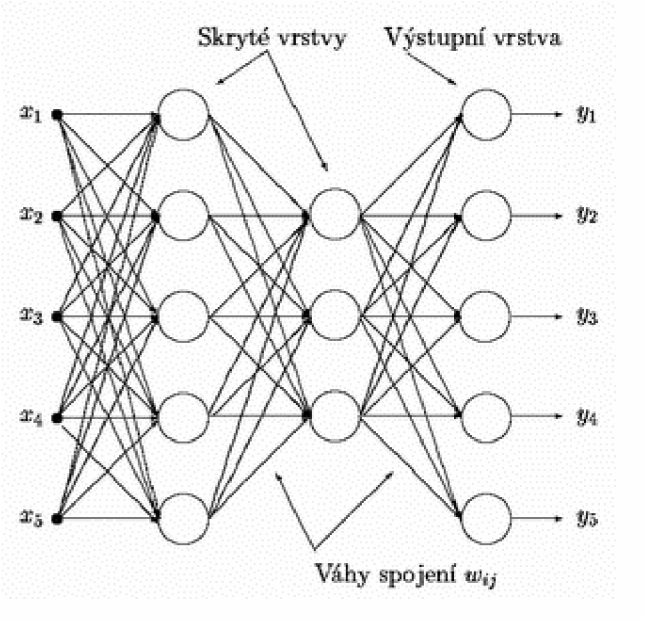


Obrázek 5-3 XOR problém[3].

Jde rovněž o síť s učitelem. Síť obsahuje několik vrstev perceptronů, každý neuron ve vrstvě je propojen se všemi neurony v sousedních vrstvách, topologii ukazuje obrázek 5-5. Jako aktivační funkce se používá sigmoida (obr. 5-4).



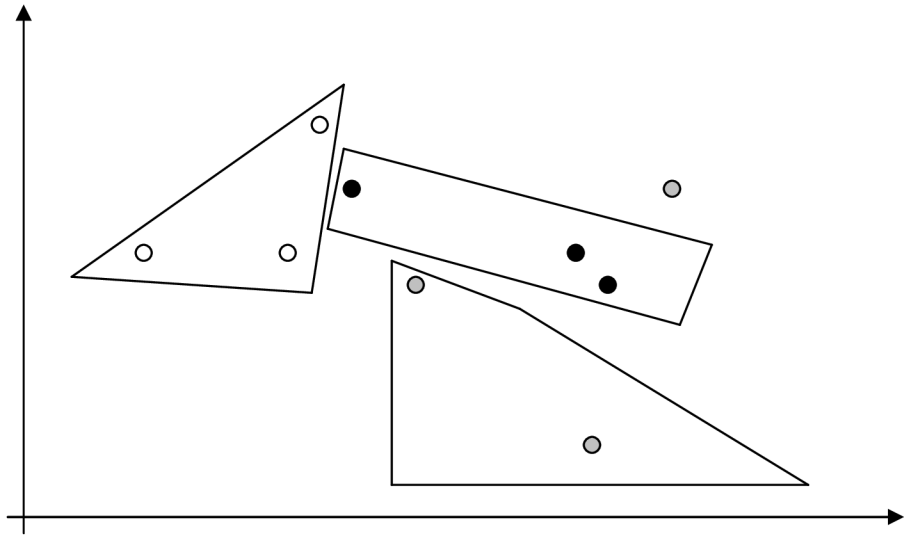
Obrázek 5-4 Sigmoida [8].



Obrázek 5-5 Topologie vícevrstvé neuronové sítě [2].

Poslední vrstva (vzhledem k toku dat) se nazývá výstupní vrstva, ostatní vrstvy neuronů se nazývají skryté vrstvy**.** Počet vrstev a počet neuronů v nich je volitelným parametrem sítě, který musí zvolit návrhář sítě (existuje i sítě s proměnným počtem neuronů). Počet neuronů by však měl být optimální, jinak hrozí nedostatečné vnímání závislostí v předkládaných datech (malý počet) nebo tzv. přeučení (malá schopnost generalizace).

Tyto sítě jsou pilířem umělých neuronových sítí. Své využití najdou i v mobilní robotice, kde jsou využívány podobným způsobem jako jednoduchý perceptron (viz. kapitola 6). Mohou ale zpracovávat mnohem složitější data a tím podávat lepší výsledky i v netriviálních prostředích. Problémem je ne zcela vyřešený učící algoritmus (back-propagation), který může selhat při pokusu naučit síť dalšímu vzoru. Počet neuronů první vrstvy určuje stupeň klasifikační nadroviny (bod-přímka-rovina…). Další vrstva počítá kombinaci výstupů první vrstvy. Klasifikační prostor se pak formuje do konvexních útvarů jak je znázorněno na obr. 5-6.



Obrázek 5-6 Rozdělení prostoru pomocí vícevrstvé neuronové sítě (2 vstupy – X,Y).

5.2.1 Vybavování

Na vstupy sítě (vstupy první vrstvy) se předloží data. Data mohou být z množiny reálných čísel. Jednotlivé neurony v první vrstvě provedou výpočet jejich výstupu podle vztahu 3.

$$y_j = sigmoida(\sum_{i=1}^{m}(x_i w_i)) \tag{3}$$

Kde

y – výstup neuronu

j – index neuronu v první vrstvě

x – vstupní data

w – váha vstupu neuronu

m – počet vstupů neuronu

Po výpočtu první vrstvy se spočítají výstupy následující vrstvy, jejichž vstupy budou výstupy předchozí vrstvy. Výstupy poslední (výstupní) vrstvy jsou výstupy sítě. Výstup sítě může být rovněž z množiny reálných čísel (binárně zakódovaný).

5.2.2 Učení

Pro učení se používá algoritmus back-propagation, což je gradientní interaktivní metoda pro minimalizaci odchylky sítě od předpokládaného výstupu, jde tedy o učení s učitelem. Po předložení k-tého trénovacího vzoru se provede vybavení výstupu. Ten se porovná s očekávaným výstupem a určí se chyba sítě.

$$E_k = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - d_{ki}\right) \text{ [4]} \tag{4}$$

nebo

$$E_k = \sum_{i=1}^{n}\left(y_i - d_{ki}\right)^2 \text{ [4]} \tag{5}$$

Kde

E_k – chyba k-tého trénovacího vzoru

n – počet neuronů ve výstupní vrstvě (dimenze výstupního vektoru)

d – předpokládaná chyba

Po vypočítání odchylek všech vstupních vzorů se určí celková chyba sítě.

$$E = \sum_{k=1}^{p} E_k \text{ [4]} \tag{6}$$

Kde p je počet trénovacích vzorů.

Učení probíhá po předložení všech vzorů. Jde tedy o akumulované učení. Při inicializaci sítě se nastaví váhy na náhodné malé hodnoty. Po předložení všech vzorů se upraví váhy podle vztahu 7.

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \text{ [4]} \tag{7}$$

Kde i,j je index neuronu a index příslušné váhy.
Změna váhy se počítá gradientní metodou jejíž odvození je nad rámec této práce.

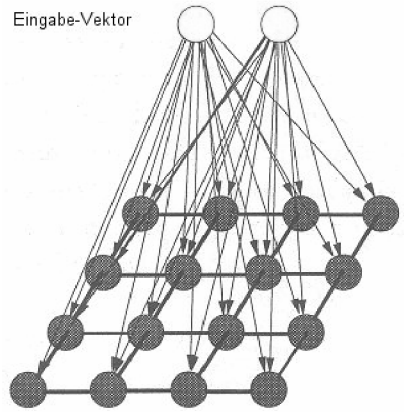
$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} \text{ [4]} \tag{8}$$

Kde η je parametr učení který udává velikost změny vah.

5.3 SAMOORGANIZUJÍCÍ SE MAPA (SOM, KOHONENOVA SÍŤ)

Architektura 2D SOM se dvěma vstupy je na obr. 5.7. Neurony jsou uspořádány v topologické jednovrstvé mřížce (nejčastěji čtvercová, dále kosočtvercová, trojúhelníková apod.). Každý neuron je reprezentován vektorem vah.

Přikládaný vstup (vektor) je porovnáván s vektorem vah jednotlivých neuronů. Neuron, jehož váhy jsou ke vstupnímu vektoru nejblíže, určíme jako výstupní (resp. jeho souřadnice, pořadové číslo apod.) Výstupem sítě je tedy identifikovaný vítězný neuron, který reprezentuje třídu vstupního vektoru. Kohonenova síť v podstatě provádí vektorovou kvantizaci (aproximace rozložení trénovací množiny), učení sítě je tedy bez učitele.



Obrázek 5-7 Topologie SOM [7].

5.3.1 Vybavování (výpočet výstupu) v SOM

Uvažujme čtvercovou síť o velikosti n*n. Dimenze vektorů (vstupních i váhových) je m. Váhu i, neuronu j označme w_{ji}, Dále existuje vstupní vektor X (i-dimenzionální).Nejprve jsou vypočítány vzdálenosti neuronů od předloženého vstupu. Metod pro výpočet vzdálenosti je mnoho, v tomto článku se omezíme na vztah uváděný Kohonenem:

$$d_j = \sum_{i=1}^{m} (x_i - w_{ji})^2 \quad \text{. [4]} \tag{9}$$

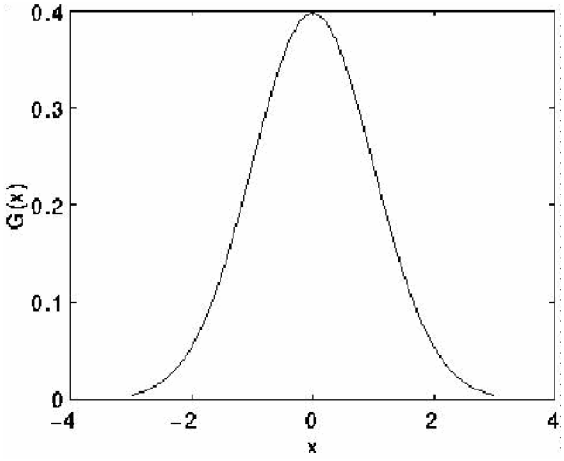
Vybere se neuron (j*) s minimální vzdáleností a označí se za vítězný (aktivní).

$$d_{j^*} = \min_j (d_j) \quad \text{[4]} \tag{10}$$

5.3.2 Učení v SOM

Učení je založeno na porovnávání vstupních vzorů a vektorů uložených v každém neuronu (váhové vektory). Je nutné zavést pojem okolí neuronu. Jeho tvar je dán topologií mřížky neuronů (v tomto případě čtvercová). Velikost okolí závisí na čase (kroku učení) *v(t)*. Adaptační funkce *h(v(t))* (7) určuje velikost změny vah příslušného neuronu v závislosti na vzdálenosti od vítězného neuronu. Nejčastějším typem adaptační funkce je tvar Gaussovy funkce (obr. 5-8), která je aproximací vlivu neuronu na své okolí v biologických neuronových sítích.

$$h(v) = h_0 e^{-\frac{v^2}{\sigma^2}} \text{ [4]} \qquad (11)$$



Obrázek 5-8 Gaussova funkce.

Je vidět, že vítězný neuron (*v = 0 v obrázku 5-8 x = 0*) podlehne největší změně, se vzdálenějšími neurony změna klesá. Samotné učení probíhá takto:

- Zvolí se parametr učení p z intervalu <0, 1>, který udává rychlost učení. Většinou se volí ze začátku učení parametr p blízko jedné a pak monotónně klesá na hodnotu blížící se k 0. Tím dosáhneme toho, že ze začátku se váhy adaptují rychleji (hrubé rozmístění neuronů) a ke konci pomalu (přesné dolaďování).

- Inicializují se váhy (náhodná malá čísla).
- Po předložení vstupu se určí vítězný neuron a jeho okolí.
- Provede se změna vah vítězného neuronu a jeho okolí podle následujícího vztahu:

$$w_{ij}(t+1) = w_{ij}(t) + p(t)h(v(t))(x - w_j(t)) \text{ . [4]} \qquad (12)$$

Změna váhy w_{ij} je dána parametrem učení (se zvyšujícím se časem se snižuje), topologickou vzdáleností od vítězného neuronu (určuje ho funkce $h(v)$, velikost okolí v s rostoucím časem též klesá) a vzdáleností jednotlivých vah vítězného neuronu od vstupu. Po aktualizaci parametru se proces (od 3. kroku) opakuje, dokud se nevyčerpá předepsaný počet kroků

5.3.3 Metoda LVQ

Nastavení vah pomocí shlukové analýzy (kterou učení SOM používá) nemusí být v některých případech optimální. Proto lze konfiguraci sítě dodatečné upravit využitím LVQ algoritmů (learning vector kvantization). Data se klasifikují podle četnosti vítězství neuronu. Cílem je najít optimální hranici mezi sousedními třídami. Metoda nalezení této hranice se liší podle verze algoritmu (LVQ1,LVQ2,LVQ3 …)

5.3.3.1LVQ1 [4]

Princip této metody spočívá v oddálení vah špatně klasifikovaného vzoru a přiblížení vah k správnému vektoru vzoru.

- Výběr třídy pro modifikaci – t.
- Předložení všech trénovacích vzorů.
- Identifikace nejčastěji vítězícího neuronu pro třídu t – j.
- Opětovné předložení všech trénovacích vzorů.

- Pro správně identifikované neurony (neuron j pro třídu t).

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_i - w_{ij}(t)) \quad [4] \tag{13}$$

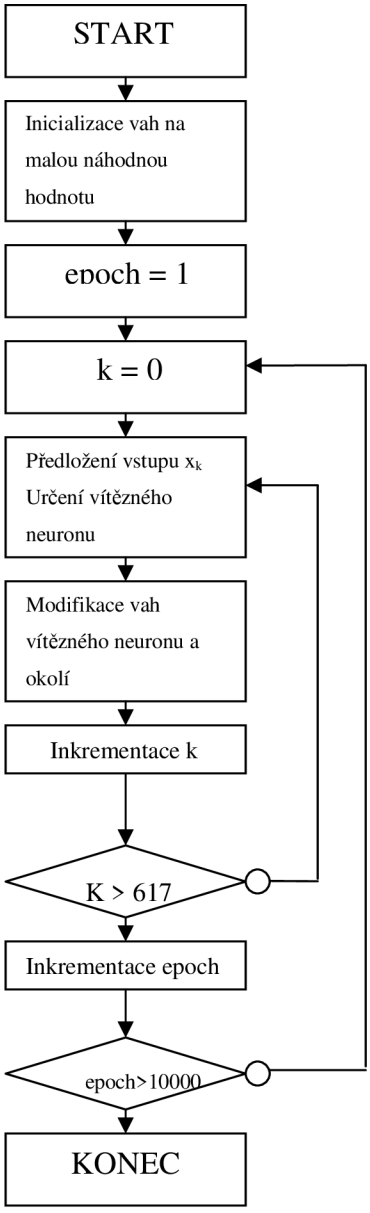
- Pro nesprávně identifikované neurony (neuron jiný než j pro třídu t).

$$w_{ij}(t+1) = w_{ij}(t) - \eta(x_i - w_{ij}(t)) \quad [4] \tag{14}$$

- Pro ostatní vektory (třída není t).

$$w_{ij}(t+1) = w_{ij}(t) \quad [4] \tag{15}$$

Kde η je parametr učení, který se volí řádově v setinách, parametr klesá monotónně s časem. Vývojový diagram učení SOM je na obrázku 5-9.



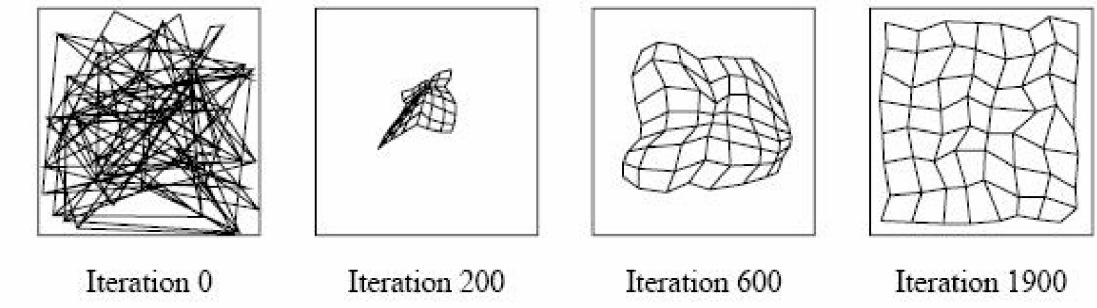
Obrázek 5-9 Vývojový diagram učení SOM (počet vzorů = 617, 10000 epoch).

5.3.4 Vlastnosti SOM

Při jakékoliv topologii mřížky by mělo výsledné rozdělení neuronů v prostoru odpovídat pravděpodobnostnímu rozdělení trénovacích vzorů, jak je vidět na obrázku 5-10.

- Výhody:
  - Odolnost proti šumu ve vstupních datech.
- Nevýhody:
  - Dlouhý výpočet učení.
  - Při zachování aproximace pravděpodobnostního rozdělení, bude počet neuronů růst exponenciálně s dimenzí prostoru trénovacích dat.
  - Problém s určením vstupních parametrů které jsou:
    - Počet neuronů.
    - Topologie sítě.
    - Velikost a průběh parametru učení.
    - Velikost a průběh sousedství.
    - Počet epoch.
    - Adaptační funkce.
    - …

Problém s určováním parametrů lze odstranit použitím vhodného heuristického algoritmu. S výhodou lze použít genetické algoritmy (ovšem za cenu velmi dlouhého výpočtu – řádově dny).



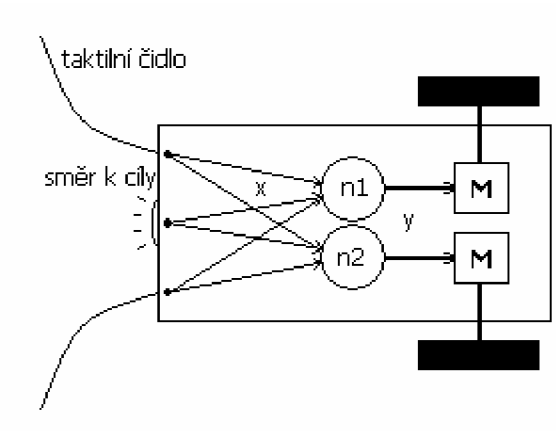
Obrázek 5-10 Postupná adaptace sítě na rovnoměrně rozdělené vstupní data.

UI lze pro řízení v reálném dynamickém prostředí s výhodou použít díky inertnosti vůči šumu (nepřesné informace ze senzorů apod.) a efektivní možnosti klasifikace. Další výhodou vycházející z podstaty UI je schopnost učit se – robot své poznatky stále upřesňuje a doplňuje (použití v dynamickém prostředí). UI je proto přímo předurčena po použití v navigaci mobilních robotů.

6. VYUŽITÍ UMĚLÝCH NEURONOVÝCH SÍTÍ V MOBILNÍ ROBOTICE

6.1 VYUŽITÍ PERCEPTRONU V MOBILNÍ ROBOTICE [5]

Předpokládejme mobilní robot s dvěma taktilními senzory vpředu, jeden na levé a jeden na pravé straně (0 – volno, 1 – kolize), jak je vidět na obr. 6-1. Dále prvek, který indikuje směr k cíli (např. IR senzor, nebo navigační algoritmus). Robot budou řídit 2 perceptrony jejichž výstupy ovládají motory (levý a pravý).



Obrázek 6-1 Robot řízený perceptrony[5].

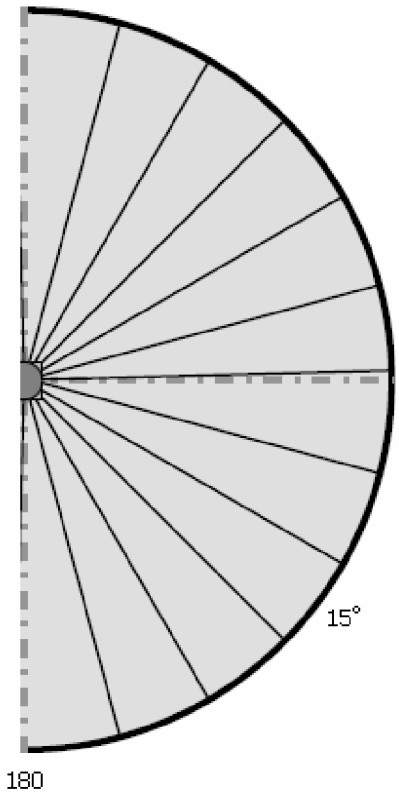
Levé čidlo	Pravé čidlo	Směr k cíli (L–levá, P–pravá)	Levý motor	Pravý motor
0	0	L	0	1
1	0	L	1	0
0	1	P	1	0

Tabulka 6-1 Možná část trénovací množiny.

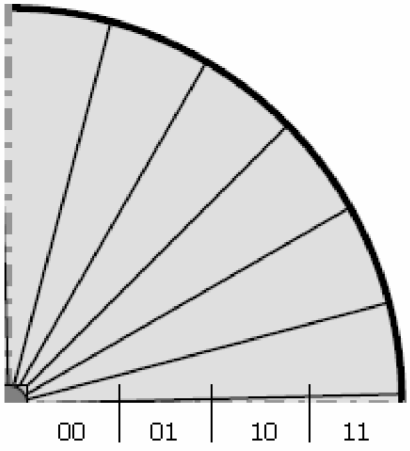
Pokud se robot povede dostatečně dlouhou dobu směrem k cíli (samozřejmě s ošetřením kolizí), sebere se dostatečně velká trénovaní množina (vstup – očekávaný výstup) a perceptrony mohou s úspěchem v jednoduchých prostředích robot řídit. Vzorek trénovací množiny je uveden v tabulce 6-1.

6.2 ROBOT ŘÍZENÝ ASOCIATIVNÍ SÍTÍ [5]

Uvažujme jednoduchý případ, kde je síť sestavena pouze z jednoho neuronu, jehož výstup řídí natočení robotu. Pro zpracování okolního prostředí je použit laserový senzor z horizontem snímání 180º, rozlišovací schopnost 15º (tedy 12 informací o vzdálenosti), jak je uvedeno na obr. 6-2. Zpětná vazba natočení robotu je realizována kompasem.



Obrázek 6-2 Laserový senzor pro řízení asociativní sítí.

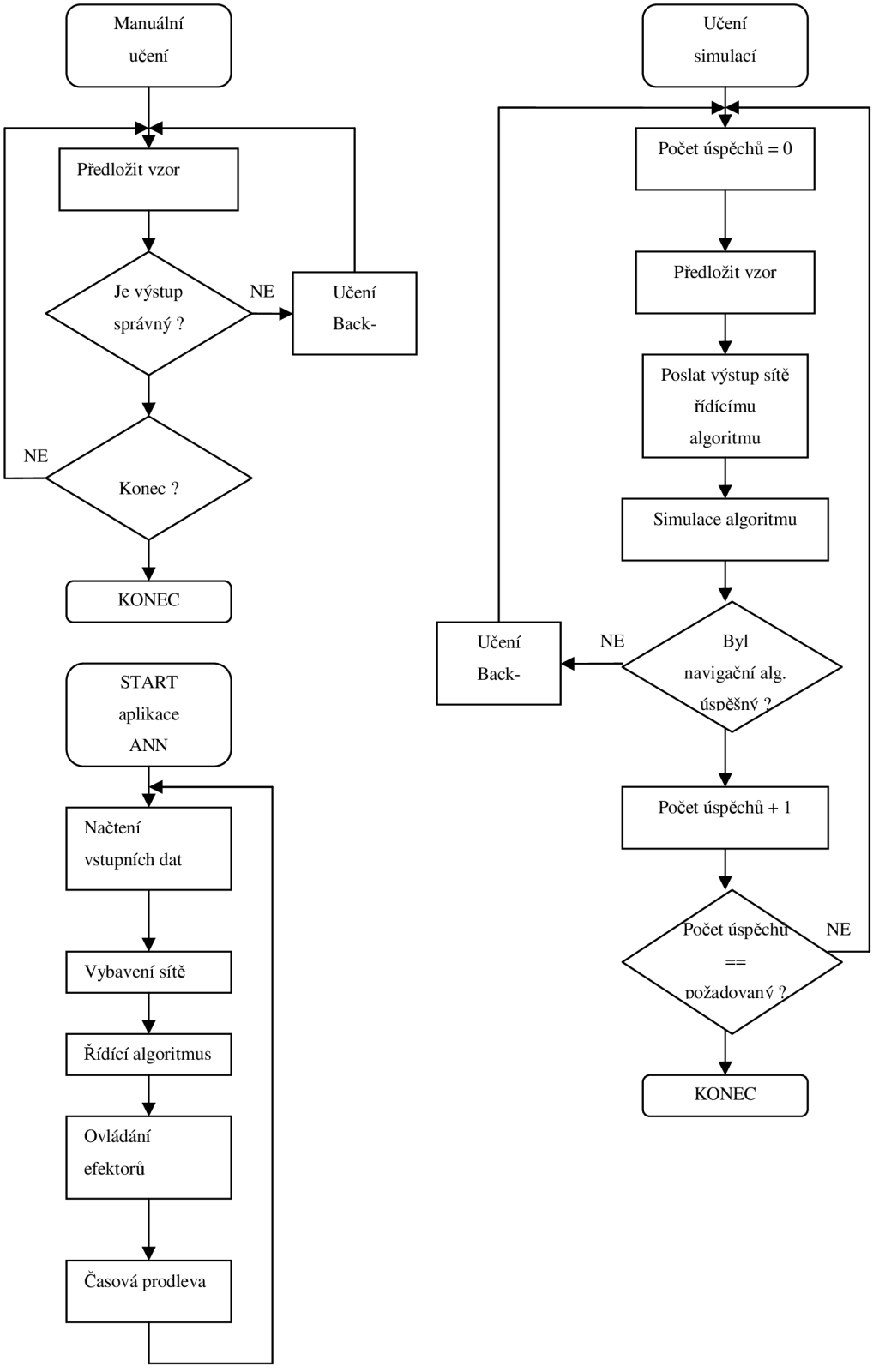


Obrázek 6-3 Upravení výstupu las. senzoru.

Výstup senzoru je vhodné rozdělit do sektorů které lze binárně zakódovat. Situaci popisuje obrázek 6-3. Tímto dosáhneme rychlejšího a efektivnějšího výpočtu. Neuron bude mít tedy 12x2 = 24 vstupů. Po „manuálním“ objíždění překážky se robot naučí (pomocí Hebbova algoritmu) překážku sledovat autonomně a reagovat i na nenaučené vzory. Výhodou je malá paměťová náročnost, vysoká rychlost výpočtu a tedy i odezvy sítě.

6.3 ŘÍZENÍ ROBOTU POMOCÍ VÍCEVRSTVÉ NEURONOVÉ SÍTĚ

Vícevrstvou neuronovou síť lze mimo jiné využít pro reprezentaci vstupní dat ze senzorů. Chápat data jako vektor čísel je pro mnohé algoritmy obtížné ne-li nemožné. Neuronová síť tyto data rozdělí a klasifikuje. Je nutné nejprve vytvořit vzorová data pro učení a to ručně nebo pomocí simulace, jak je uvedeno na vývojových diagramech na obrázku 6-4.



Obrázek 6-4 Vývojové diagramy učení a aplikace vícevrstvých neuronových sítí.

Další využití vícevrstvé neuronové sítě je pro samotný řídicí algoritmus. Neuronová síť zastává činnost paměti navigačního algoritmu (reprezentována v konfiguraci sítě), rozhodování, lokalizace a vstupní zpracování dat. Činnost tohoto adaptivního algoritmu spočívá v rozdělení dat ze senzorů + cílová pozice do tříd (klasifikace), které mají asociovaný směr optimální trajektorie.

6.3.1 Učení navigační neuronové sítě

- Operátor získá informace o okolí spojené se směrem dalšího pohybu.
- Trénovací data jsou data ze senzorů, požadovaný výstup je unikátní číslo (lze binárně zakódovat) pro každý vzor.
- Naučení sítě.

Pomocí tohoto algoritmu lze dosáhnout poměrně spolehlivé robustní navigace, založené na zobecnění informací přicházejících ze senzorů. Velká nevýhoda je značná závislost efektivity algoritmu na sběru dat a nejasné požadované výstupy při učení sítě.

6.4 POUŽITÍ SOM V NAVIGACI MOBILNÍHO ROBOTU

Samoorganizující se mapa se dá použít dvojím způsobem: pro klasifikaci pozice nebo klasifikace okolního prostředí. V prvním případě pouze zobecňuje pozici robotu. Nevýhodou je nutný značný zásah uživatele (rušení neplatných propojení apod.). Druhý způsob je založen na zobecnění okolí – tzn. algoritmus podle okolního prostředí (data se senzorů) určí, v jaké části (oblasti) prostoru je. Pro navigaci mobilního robotu lze tento algoritmus použít pouze pro hledání cesty pro jeden cíl (popř. projíždění naučené cesty). Ke každému neuronu se asociuje následný směr pohybu. Pro globální navigaci je nutno použít sofistikovanější algoritmus.

Následující algoritmy byly odzkoušeny na školním robotu, v části 6.4.1 popisuji jeho hlavní parametry a SW.

6.4.1 Školní mobilní autonomní robot

- Diferenciální řízení - 2 hnaná kola ve středu robotu, jedno pasivní kolo.
- Senzor URG-04LX (Hokyuo). (tabulka 6-2)
- Řídící počítač Acer Aspire One.

Detection distance	20mm ~ 4000mm*
Accuracy	Distance 20 ~ 1000mm: ±10mm* Distance 1000 ~ 4000mm: ±1% of measurement*
Resolution	1 mm
Scan Angle	240°
Angular Resolution	0.36°
Scan Time	100msec/scan

Tabulka 6-2 Specifikace HOKUYO URG-04X [10]

Laserový senzor vzdálenosti URG-04LX od společnosti HOKUYO je vynikajícím prostředkem pro sběr dat z okolí. Má však skenovaní rozsah 240º a z podstaty laserového měření vyplývají jisté nepříjemnosti (ztrácení laserového paprsku, dlouhá doba čtení – rotující zrcátko).

6.4.1.1 Softwarové vybavení robotu

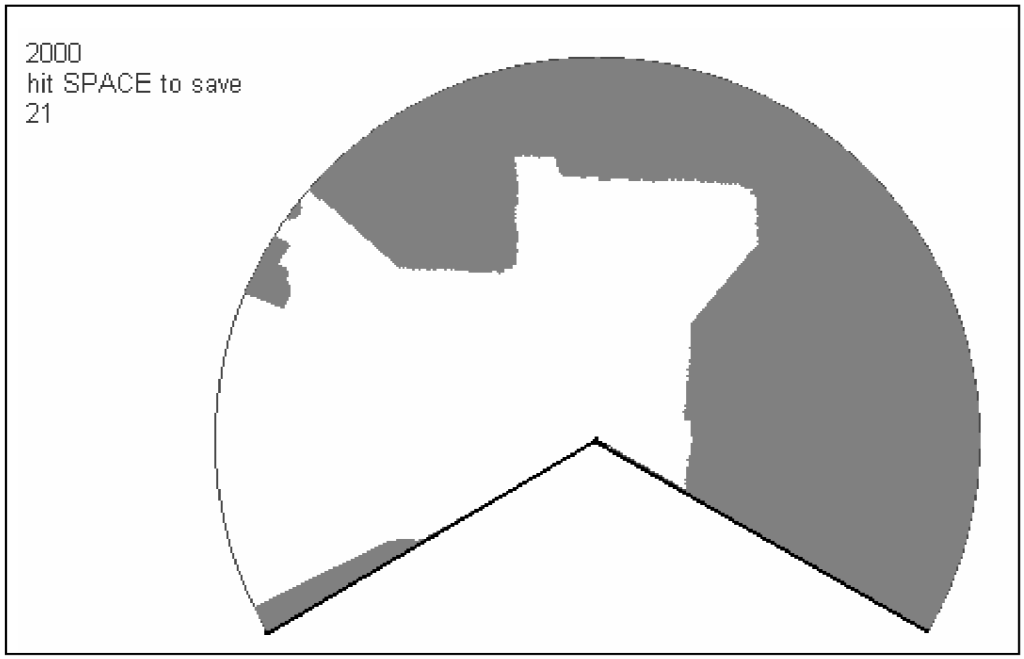
Pro implementace robotu byla naprogramována aplikace Robot Control v prostředí C#.NET (Microsoft VS 2008). Jednotlivá okna a třídy MDI aplikace jsou popsány níže. Celkový pohled na aplikaci Robot Control je na obrázku 6-7.

ScanConnect

Zabezpečuje spojení a komunikaci ze senzorem URG-04X. Čte a dekóduje příchozí data.

VizScan

Vizualizace příchozích dat ze senzoru, ukládání dat („naučná jízda“). Vizualizace je znázorněna na obrázku 6-5.



Obrázek 6-5 VizScan.

ControlConnect

Zabezpečuje spojení a komunikaci s procesorem, který řídí motory.

ManualControl

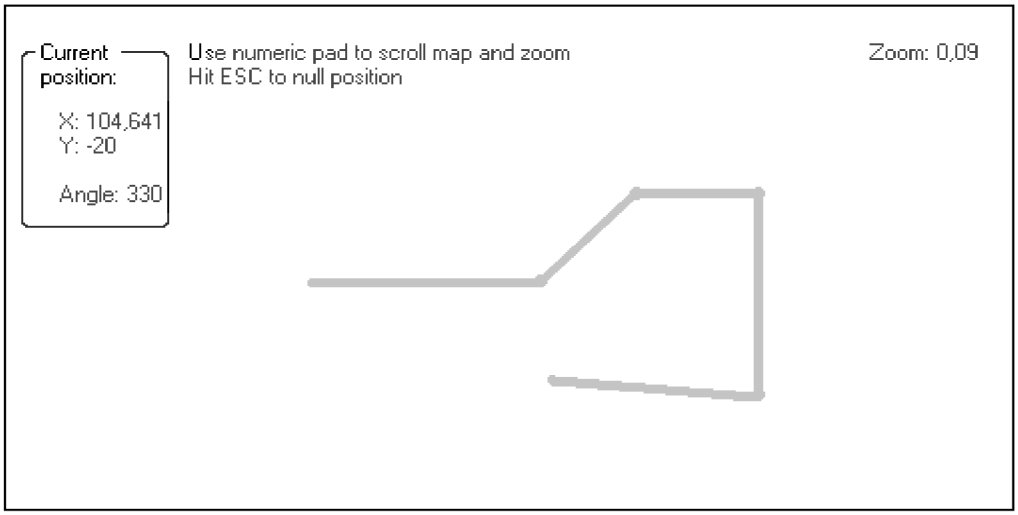
Slouží pro manuální ovládání robotu (naučná jízda).

SOM

Naučení SOM ze vzorů uložených ve VizScan, LVQ1 algoritmus, zobrazeni průběhu a výsledků učení, dále pro vizualizaci odezvy sítě.

Odometry

Počítá a zobrazuje odometrii robotu znázorněnou na obrázku 6-6.



Obrázek 6-6 Odometry.

Threading

Slouží pro správu vláken. Z podstaty aplikace je velice vhodné vlákna využívat.

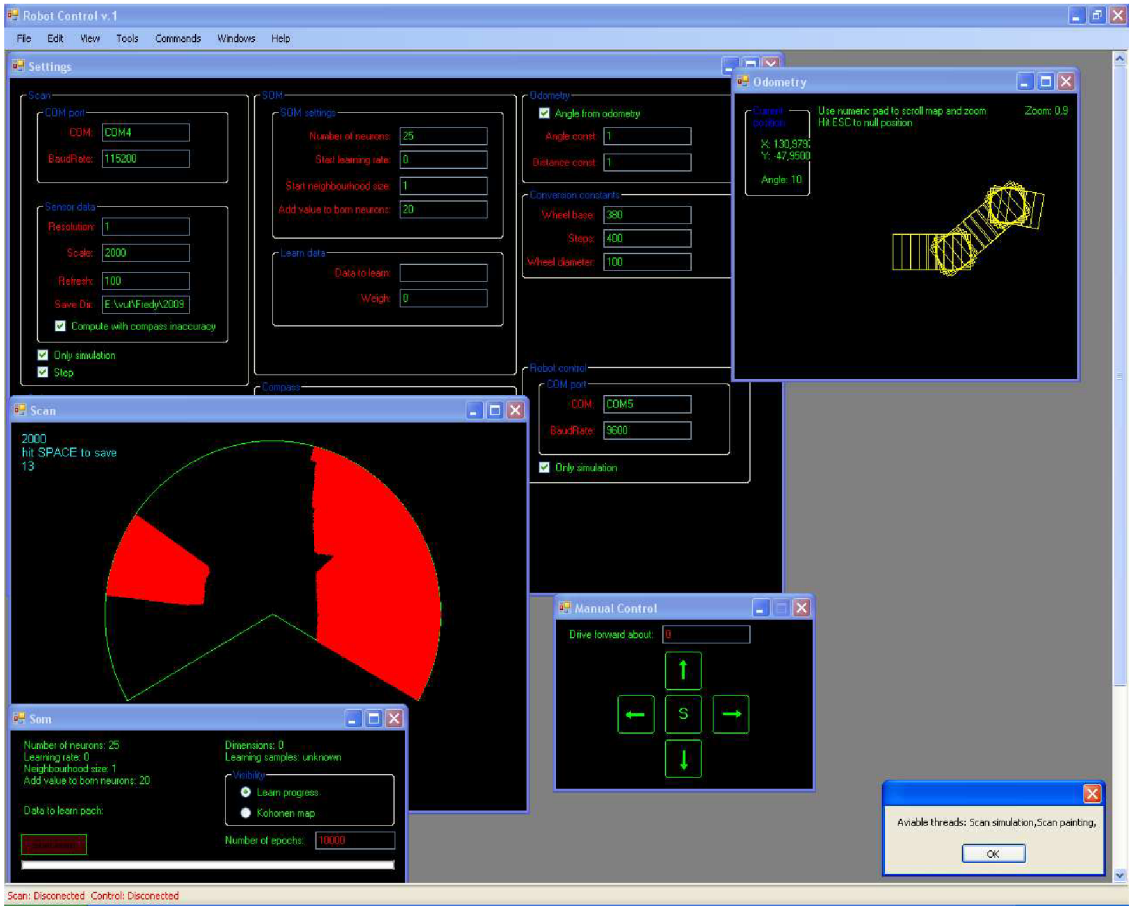
Základní vlákna v Robot Control.

- Scan connection.
- Control connection.
- Scan Reading.
- SOM learning.
- Navigation.
- …

Atribut ErrorIndex

Aktuální směr (z kompasu nebo odometrie) nemusí být přesný. Spočítaný atribut ErrorIndex vyjadřuje nejistotu měření vzhledem k natočení – udává o kolik vzorků může být obraz špatně natočený (1 vzorek ≈ 0.36 stupňů). Reprezentace senzorových dat je tedy z obou stran oříznuta právě o hodnotu ErrorIndex. V průběhu kompetice

se obraz natáčí na obě strany a vybere se ten nejvhodnější. Takto je ovlivněno zpracování dat (kompetice) ve všech algoritmech!



Obrázek 6-7 Celkový pohled na Robot Control v1.

Další SW, který byl při tvoření této práce použit pro simulace a testování SOMAP algoritmu, je SOMAPsim. Tento program umí nasimulovat prostředí, vzít z něho vzorky, naučit SOM a simulovat pohyb robotu řízeným SOMAP algoritmem.

6.4.2 Jednosměrná navigace navSOM1

Tento jednoduchý algoritmus přiřadí ke každému neuronu směry k žádanému cíli (neuron může asociovat více oblastí). Poté je seřadí podle pořadí aktivity

v naučné jízdě. Kompetice v SOM je ovlivněna přičtením váhy *w'* (zadaný parametr) vynásobenou vzdáleností neuronu od aktuálního indexu cesty.

Vytváření cesty se provádí následovně:

1. Provede se „naučná jízda" – uchovají se data ze senzorů a aktuální úhel natočení robotu.
2. Naučí se SOM (pouze z dat ze senzorů).
3. Znovu se síti předloží všechna data z „naučné jízdy" v pořadí v jakém byly uloženy.
4. Identifikuje si vítězný neuron.
5. K neuronu se přidá směr v jakém byla data uložena a pořadí vzorku.

Nechť index *r* (na začátku = 1) určuje index v sestavené cestě *route[](array)*, kde se uchovávají požadované směry.
Samotná navigace funguje takto:

1. SOM se předloží data ze senzoru. Modifikovaná kompetice má však tvar dle rovnice 17.

$$d_j = \sum_{i=1}^{m} (x_i - w_{ji})^2 + w' \min(|r + 1 - e_j|) \qquad (17)$$

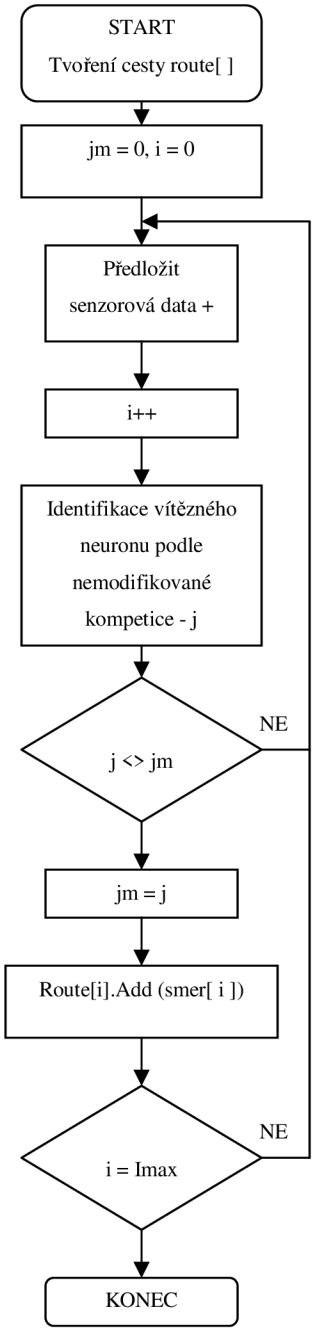
   Kde e_j je index cesty asociovaný s neuronem j. Zde je důležité si uvědomit, že neuron může mít asociovaných více směrů a indexů cesty, proto se vybírá minimální rozdíl indexů cesty.

2. $r = e_{jwin}$.
3. Je-li r roven délce cesty, zastav – konec algoritmu (cíl dosažen).
4. Nastav úhel route[r + 1] – tzn. index dalšího směru.
5. Zpět na bod 1.

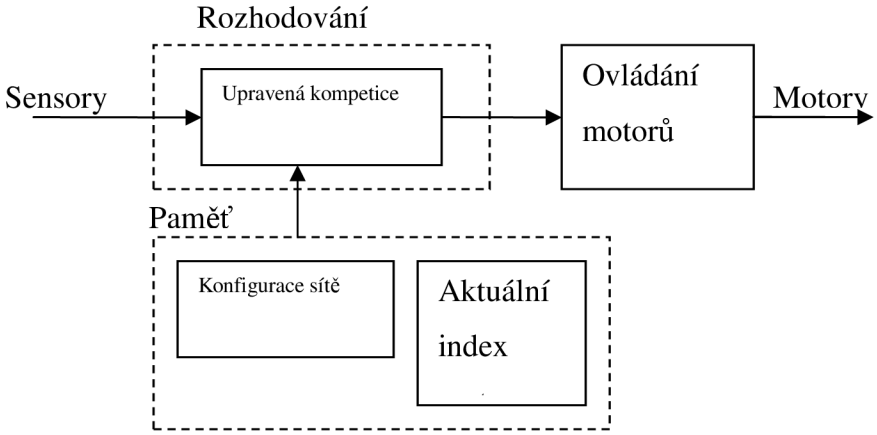
Velikostí w´ lze měnit jak bude řízení vázáno na „naučnou jízdu". Při malém w' se robot orientuje převážně podle okolí (tedy ne podle „naučné jízdy"). Učení navSOM1 je uvedeno na obr. 6-8. Struktura trajektorie je na obr. 6-10 a celková trajektorie na obr. 6-9.

Výhodou toho poměrně jednoduchého algoritmu je právě jeho snadná implementace a vysoká rychlost. Sestavení cesty není časově náročné, takže největší časová náročnost zůstává na učení SOM. I při nesprávném počtu neuronů se algoritmus celkem spolehlivě přizpůsobí (neurony klasifikují velký prostor, více prostorů, apod.). Další možností vylepšení algoritmu je LVQ. Tedy určit v jakých oblastech má robot např. měnit směr (rohy apod.). I za cenu dlouhého výpočtu LVQ učení však nepatrné vylepšení navigace zřejmě nestojí.

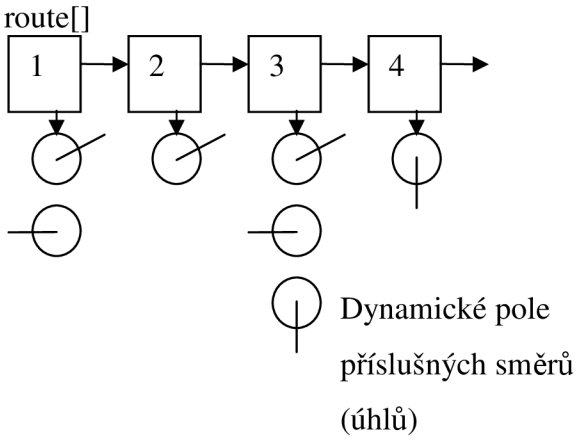
Nevýhodou je selhání algoritmu při změně prostředí. Pokud se robot více odchýlí od trajektorie „naučné jízdy“, navigační algoritmus již není schopen efektivní navigace. Většinou dojde „k zamrznutí“ na jednom neuronu. Dalším zvýhodněním následujícího neuronu (větší *w'*, indikace zamrznuti….) by již algoritmus ztratil možnost navigace podle prostředí více, než je únosná mez. Při kombinaci navSOM1 s dalším deterministickým algoritmem (objíždění překážek) optimální navigace opět selhává z důvodu neúnosného odchýlení od původní cesty.



Obrázek 6-8 Vývojový diagram učení v navSOM1.



Obrázek 6-9 Blokové schéma navSOM1 navigace.



Obrázek 6-10 Uchování cesty v navSOM1 algoritmu.

6.4.3 Jednosměrná navigace navSOM2

NavSOM2 je modifikace algoritmu navSOM1. NavSOM2 odstraňuje nevýhodu nesprávné navigace při odchýlení od naučené trajektorie. Cesta (route[ ]) je nahrazena body rozprostřenými v prostoru. Kompetice je v tomto případě ovlivněna euklidovskou vzdáleností od aktuálního bodu, jednotlivým neuronům se

nepřiřazují indexy cesty ale souřadnice. Algoritmus má dva vstupní parametry, a to kritickou vzdálenost *c* (maximální vzdálenost mezi body) a koeficient přiblížení *a* (viz. níže).

Vytváření cesty se provádí následovně:

1. Provede se „naučná jízda“ – uchovají se data ze senzorů, aktuální úhel natočení robotu a pozice uložení dat podle orometrie.
2. Naučí se SOM (pouze z dat ze senzorů).
3. Znovu se síti předloží všechny data z „naučné jízdy“ (již nezáleží na pořadí).
4. Identifikuje si vítězný neuron.
5. Identifikuje se nejbližší bod oblasti (Place point – PP) přiřazený k vítěznému neuronu.
6. Upravení/vytvoření PP:
    a) Žádný PP asociovaný s vítězným neuronem neexistuje – vytvoří se nový PP (souřadnice a směr se zkopírují z předkládaného vzorku).
    b) Vzdálenost mezi souřadnicemi vzorku a souřadnicemi nejbližšího PP je větší než *c* – vytvoří se nový PP (souřadnice a směr se zkopírují z předkládaného vzorku).
    c) Vzdálenost mezi souřadnicemi vzorku a souřadnicemi nejbližšího PP je menší než *c* a úhel PP a překládaného vzorku se rovná – nejbližší PP změní své souřadnice podle vzorce 18.

$$\begin{aligned} \{d_x, d_y\} &= \{vzor_x - p_x, vzor_y - p_y\} \\ \{p_x(t+1), p_y(t+1)\} &= \{p_x(t) + ad_x, p_y(t) + ad_y\} \end{aligned} \tag{18}$$

Kde

vzor jsou souřadnice předkládaného vzoru

p jsou souřadnice vybraného PP

a je koeficient přiblížení (parametr algoritmu)

d) Vzdálenost mezi souřadnicemi vzorku a souřadnicemi nejbližšího PP je menší než *c* a úhel PP a překládaného vzorku se nerovná – vytvoří se nový PP (souřadnice a směr se zkopírují z předkládaného vzorku).

7. Manuální označení koncových(cílových) PP – např $angle_i = -1$.

V navSOM2 se rovněž zavádí parametr *w'*, kterým se násobí euklidovská vzdálenost mezi PP a tímto se ovlivňuje kompetice.

Samotná navigace funguje takto:

1. SOM se předloží data ze senzoru. Modifikovaná kompetice má tvar:

$$d_j = \sqrt{\sum_{i=1}^{m} (x_i - w_{ji})^2} + w' \sqrt{(p_{xr} - p_{xj})^2 + (p_{yr} - p_{yj})^2} \,. \qquad (19)$$

   Kde *r* je index aktuálního PP.

   Vztahem 19 se vypočítají vzdálenosti (souřadnic i vah) ke **všem PP** (ne ke všem neuronům jako u algoritmu navSOM1).

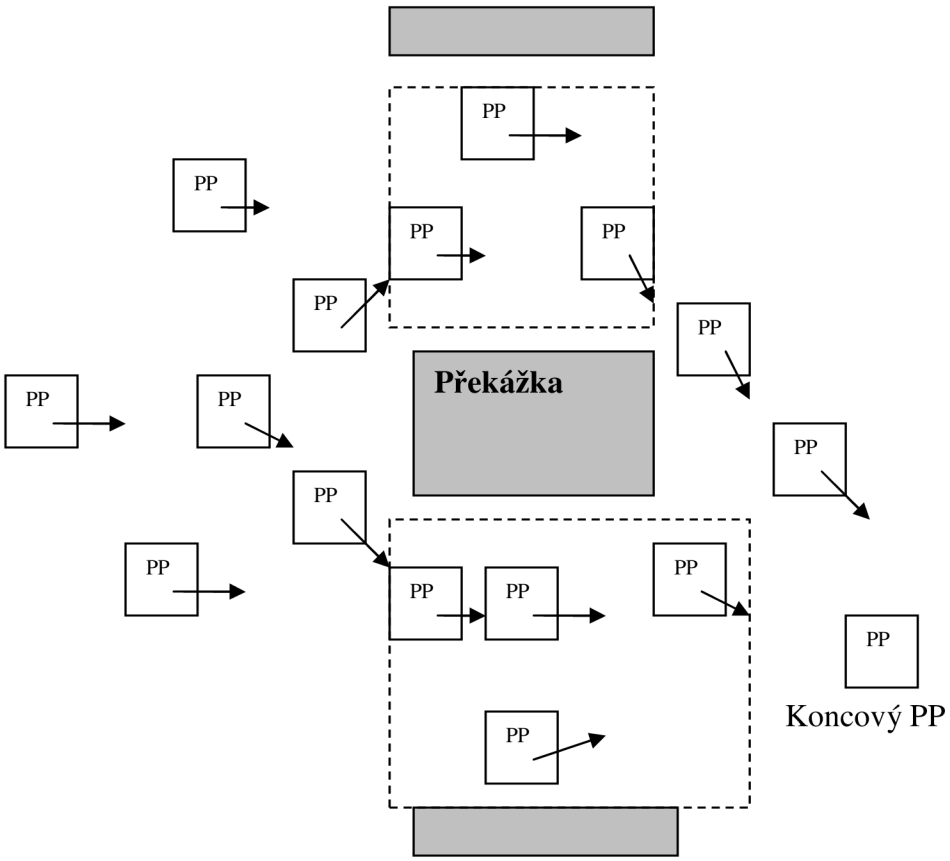
2. $r = j_{win}$.
3. Je-li PP_r označen jako cílový – zastav, cíl dosažen.
4. Nastav úhel který je uložen v PP_r .
5. Zpět na bod 1.

Parametrem w´ lze opět měnit citlivost algoritmu na okolí vs. naučené trajektorie. U navSOM2 se volí w´ velice malé (0.1), tzn. že se navigace orientuje převážně podle okolí. Pokud pokryjeme „naučnou jízdou“ dostatečně velkou oblast, navigace bude spolehlivě fungovat i při větší změně okolí.

Tento algoritmus, již poměrně složitý na implementaci, odstraňuje velkou nevýhodu navigace navSOM1. Selhání navigace při odchýlení od naučené trajektorie je odstraněno rozprostřením bodů (kde je určen směr) ve větším prostoru. Na druhou stranu přibyly dva parametry algoritmu, které je nutné ručně nastavit.

V kombinaci s vhodným algoritmem pro objíždění překážek, detekce kolize apod. lze navSOM2 využít s velmi dobrými výsledky!

Obrázek 6-11 ukazuje rozprostření PP v prostoru, čárkovaně jsou označeny PP které mohou být asociovány se stejným neuronem.

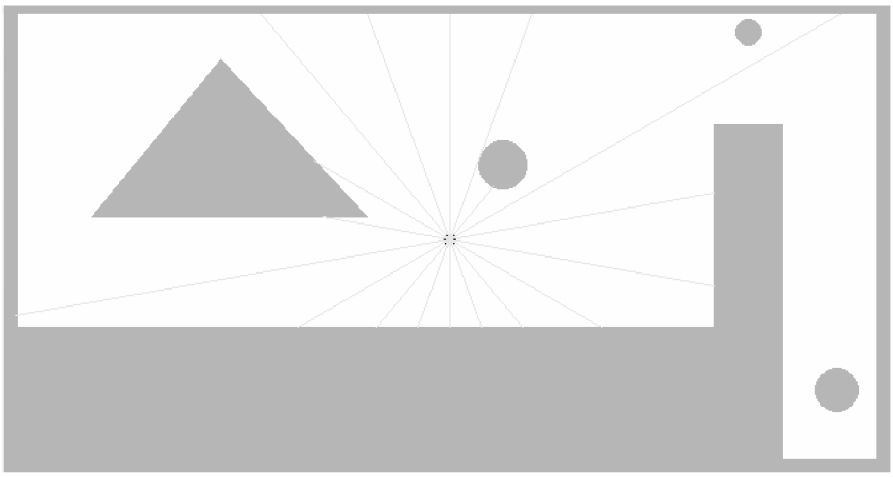


Obrázek 6-11 Rozložení PP v prostoru.

7. ŘÍDÍCÍ ALGORITMUS (SOMAP)

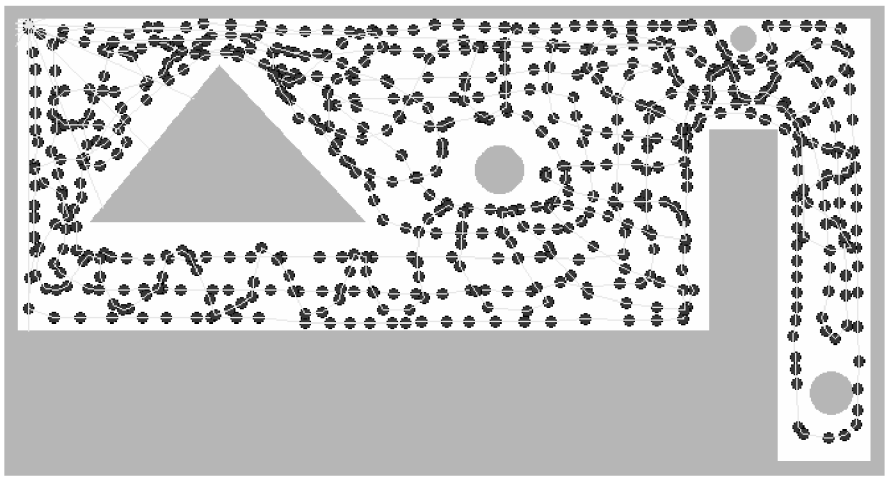
7.1 APLIKACE SOM

V první verzi se bude uvažovat robot jako bezrozměrný bod. Detektor překážek snímá v 360º v rozestupech 72º. Situaci popisuje obrázek 7-1.



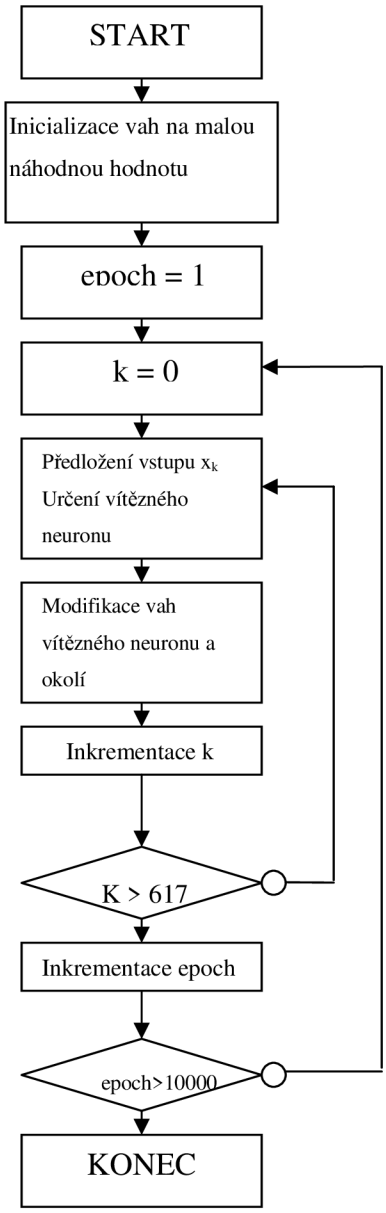
Obrázek 7-1 Snímání okolí.

V první fázi je nutné vytvořit trénovací vzory pro naučení SOM. Je žádoucí kontrolovat, popř. korigovat pozici robotu pro další zpracování (samotná SOM nemá o pozici žádnou informaci). Učení se provádí při tzv. učící jízdě (obr. 7-2).

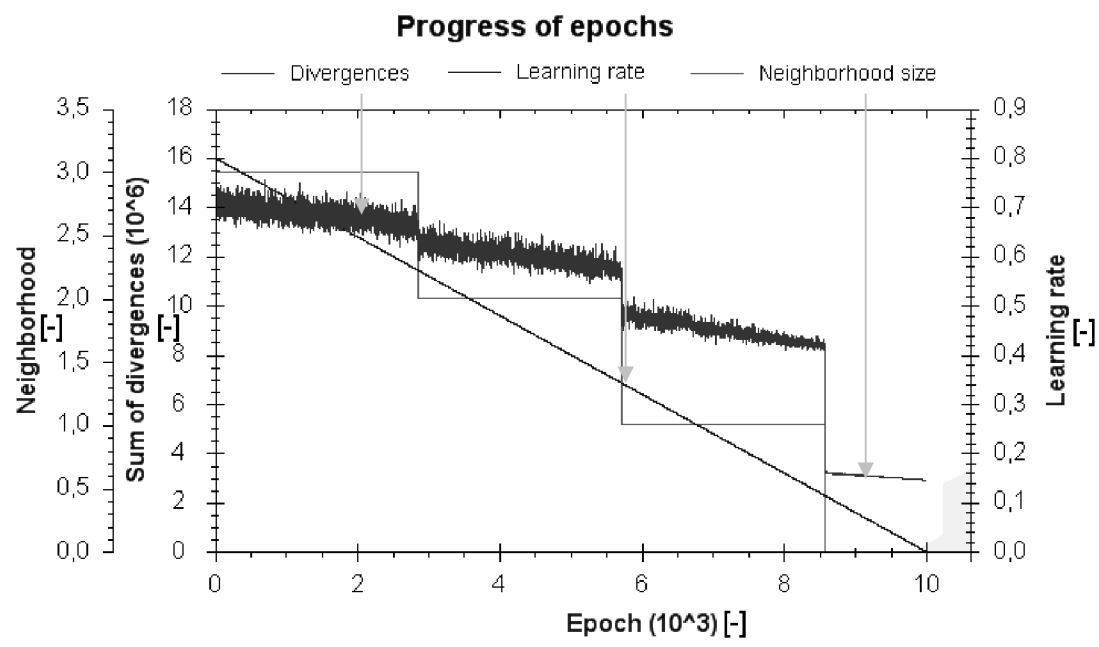


Obrázek 7-2 Učící jízda (tmavé body znázorňují místa ukládání trénovacích vzorů).

Sběr dat pro „učící jízdu“ je důležitou fází algoritmu. Tímto způsobem se řídící algoritmus dozví, jak se může v prostoru pohybovat. Nyní je k dispozici 617 trénovacích vektorů. V dalším kroku se vytvoří SOM o velikosti 7x7 (49 neuronů, algoritmus bude tedy prostor rozdělovat do 49 oblastí). Parametr učení bude monotónně klesat z hodnoty 0,8. Učení bude probíhat v 10000 epochách (10 000x se předloží všechny vstupy). Situaci popisuje obrázek 7-3. Průběh učení je na obr. 7-4.

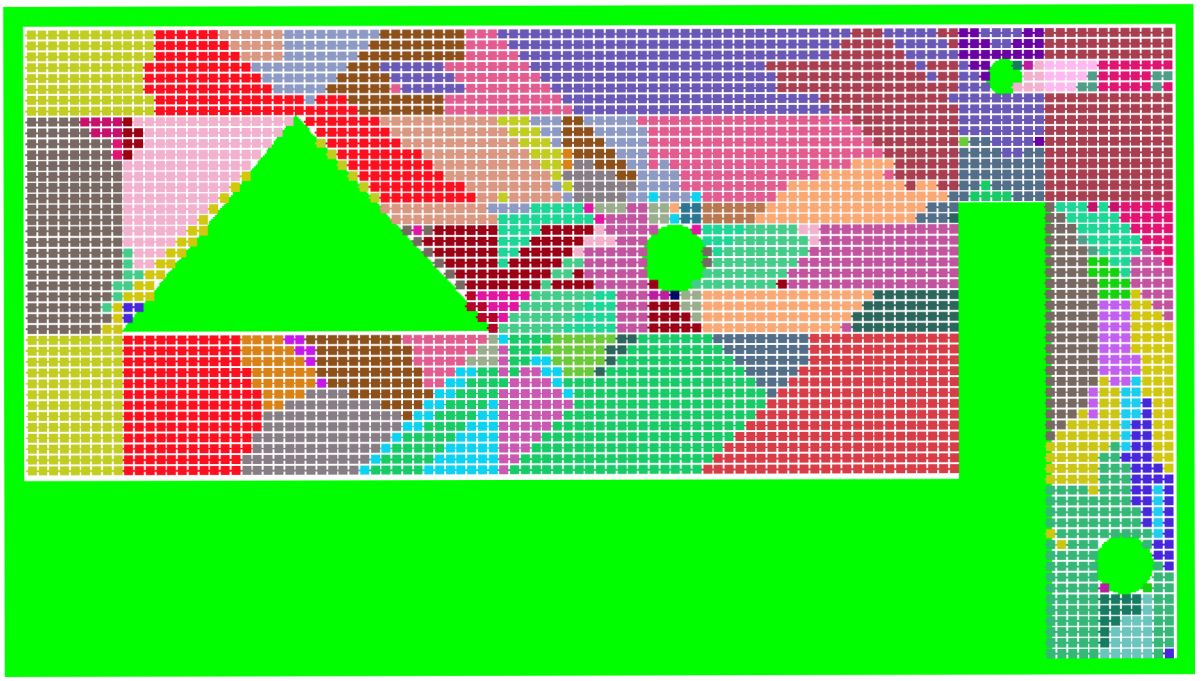


Obrázek 7-3 Vývojový diagram učení SOM (v konkrétním případě).

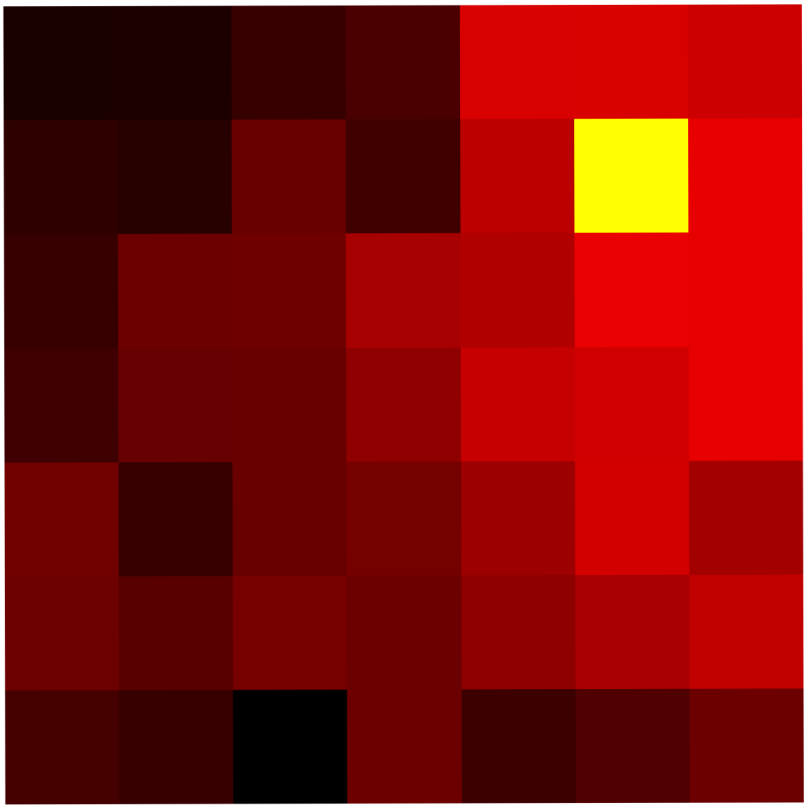


Obrázek 7-4 Vývoj veličin v průběhu učení.

Na obrázku 7-5 je vidět rozdělení prostoru do jednotlivých oblastí. Jelikož vstupní data obsahují informace jen od senzorů, může být stejná oblast na více místech. Např. žlutá oblast v rozích vlevo. Tento nežádoucí jev musí být odstraněn, např. uvažováním informace o přibližné poloze.



Obrázek 7-5 Rozdělení prostoru do jednotlivých oblastí pomocí SOM.



Obrázek 7-6 Odezva sítě.

Na obrázku 7-6 je vidět odezva sítě na vstupní vektor. Stupeň šedi vyjadřuje vzdálenost vah neuronu od vstupu. Vítězný neuron je označen žlutě.

7.2 GENEROVÁNÍ OBLASTNÍCH BODŮ (AP)

Aby algoritmus mohl globálně navigovat robot, je nutné další zobecnění podoblastí. Každé oblasti (též reprezentována každým neuronem) je přiřazen jeden oblastní bod. Pokud se jedna oblast vyskytuje na více místech konfiguračního prostou (prostředí robotu), tzn. okolní horizont je stejný na více místech, je každé podoblasti přiřazen zvláštní oblastní bod (dále AP – area point). AP jsou generovány z „naučné jízdy“ tímto způsobem:

- Vstupní vektor se předloží SOM.
- Identifikuje se výstupní neuron.
- Pokud není žádný AP k neuronu přiřazen, vytvoří se AP se stejnými souřadnicemi jako má vstupní vektor.
- Pokud je k neuronu přiřazen AP, který je vzdálen více než poloměr největší oblasti (vstupní parametr algoritmu – critical distance) vytvoří se další, tímto se zajistí shoda vstupů (stejné okolí) v jiném podprostoru.
- K vytvořenému oblastnímu bodu se přidá hrana z předchozího AP (propojení). Je jasné že AP lze propojit, protože robot při „naučné jízdě“ tudy opravdu projel.
- Pokud už byl vytvořen AP, upraví se jeho pozice o dT (vstupní parametr algoritmu – adaptive rate) směrem k pozici právě aktivního vstupu.

Po vyčerpání všech vstupních stavů z „naučné jízdy“ vzniká síť AP, nad kterou lze aplikovat algoritmy pro hledání nejkratší cesty od všech bodů (vrcholů grafu) k cíli.

7.2.1 Problém určení vstupních parametrů AP sítě

Vstupní parametry pro vytvoření AP sítě:

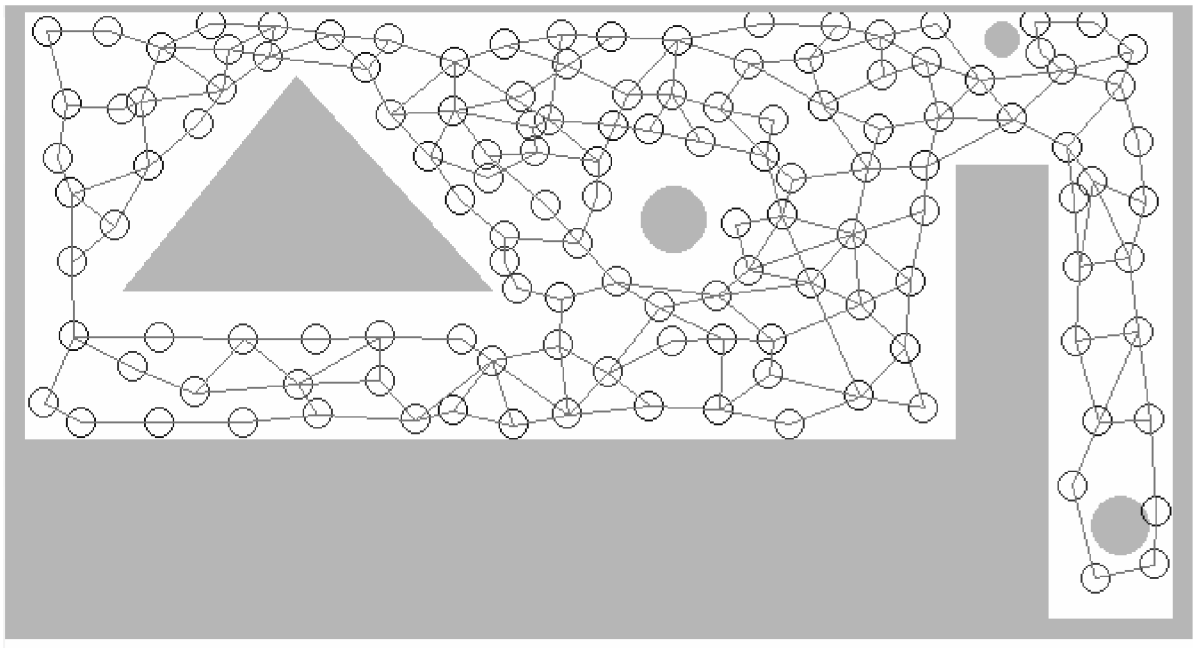
- Kritická vzdálenost - udává max. vzdálenost mezi dvěma AP, po překročení této vzdálenosti při generování AP sítě se vytvoří nový.
- Adaptivní parametr – (0 < dT < 1) udává o kolik se posune existující AP k nově vytvořenému.

  $Pn = Ps + dT(Ps - Pn)$ (20)

  Pn,Ps je nová/stará pozice.

Určení kritické vzdálenosti

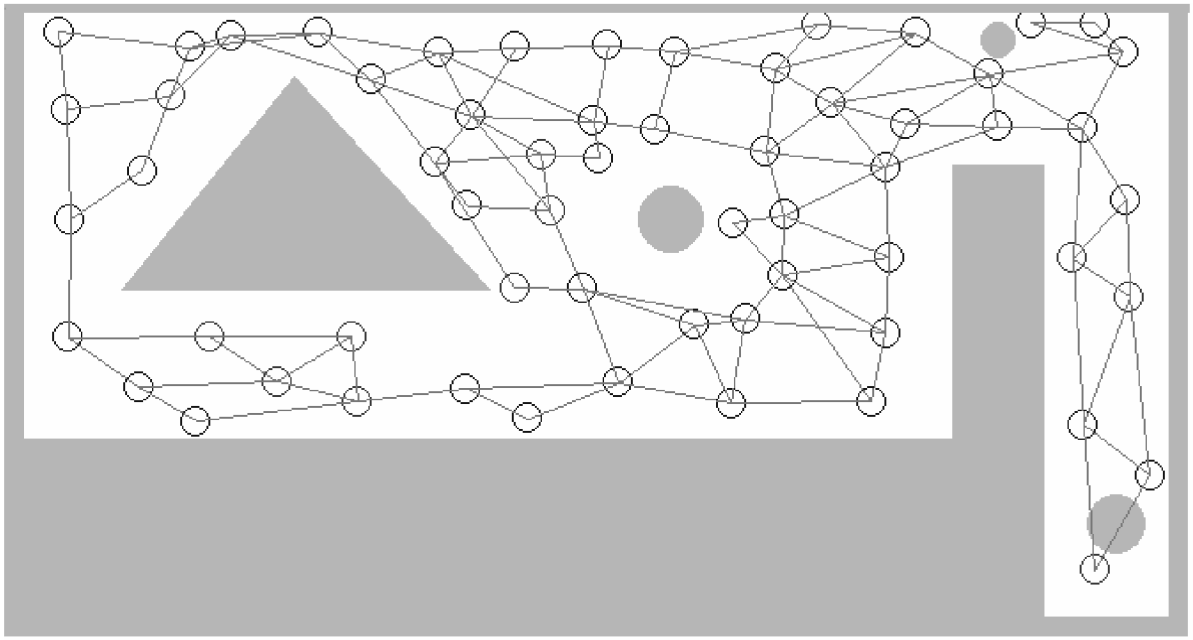
Tato část návrhu je zásadní pro celkovou funkčnost algoritmu. Pokud se zvolí malá kritická vzdálenost, vygeneruje se hodně AP (3x více než je počet neuronů – obr. 7-7) a algoritmus tím ztrácí vlastnost zobecnění souřadnic robotu, ale výsledná trajektorie vede optimálněji.



Obrázek 7-7 Malá kritická vzdál – hustá AP síť.

Při zvyšování kritické vzdálenosti řídne AP síť (dochází k zobecňování pozice). Nejlepšího výsledku by se dosáhlo při kritické vzdálenosti rovné nekonečno,

ale tímto by se neeliminovala možnost, že jeden vítězný neuron může pokrývat více oblastí na zcela jiných místech (stejné okolí na vzdálených oblastí). Optimální síť je uvedeno na obrázku 7-8.



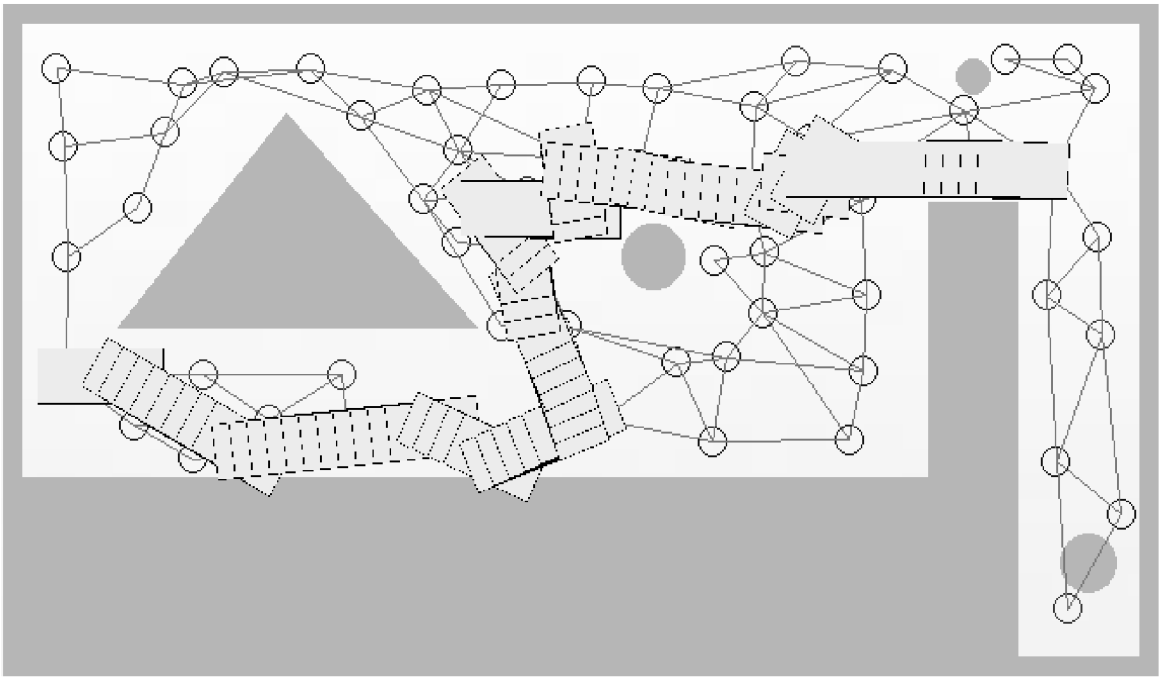
Obrázek 7-8 Optimální AP síť.

7.3 APLIKACE SOMAP ALGORITMU

Po nastavení vstupních parametrů a inicializaci (učení SOM a tvorba AP sítě) algoritmu lze přistoupit k samotnému navigování pomocí SOMAP algoritmu. To probíhá v těchto krocích:

1. Zadání cíle (index příslušného AP).
2. Výpočet nejkratší cesty ze všech vrcholů v grafu pomocí Dijkstrova algoritmu.
3. Předložení vstupů pro SOM (data ze senzoru vzdálenosti).
4. Identifikace vítězného neuronu a k němu přiřazenému AP (pomocí virtuálních souřadnic – souřadnice AP).
5. Pokud je přechod mezi AP, tak úprava souřadnic obou APOD.
6. Výpočet dalšího AP na cestě.
7. Následování souřadnic toho AP (předání řízení nižší vrstvě – Bug algoritmu).
8. Je index aktuálního AP roven cíli? Pokud ne, skoč na bod 2.

Příklad trajektorie robotu je znázorněn na obrázku 7-9.



Obrázek 7-9 Trajektorie robotu

8. ZÁVĚR

V první části jsou porovnávány vybrané deterministické algoritmy. Pro reálnou globální navigaci jsou však tyto algoritmy nepoužitelné. Zůstává tu problém sebelokalizace a nepřesnosti senzorů. Jako další logický postup se jeví použití neuronových sítí a umělé inteligence jako takové.

Perceptron a hlavně asociativní síť se zdá být ideální prostředek pro navigaci v jednoduchých prostředích, nebo k naučení základních vzorců chování, jako je objíždění překážek, sledování čáry, průjezd v koridoru apod.

Díky vlastnosti SOM – zařazovat vstupní data do tříd (shluková analýza), se osvědčila právě SOM jako ideální pro navigaci. Samotnou SOM lze použít pro navigaci do jednoho cíle, algoritmy navSOM1 a navSOM2 jsou ideálními prostředky pro tento typ řízení. Pro globální navigaci z jednoho libovolného bodu do jiného libovolného bodu jsem navrhl robustnější SOMAP algoritmus, který přináší velmi dobré výsledky. Zejména nezávislost na znalosti polohy robotu, tzn. orientace jen pomocí okolního prostředí. Při inicializaci je nutné nastavovat mnoho vstupních veličin, které zásadně ovlivňují činnost algoritmu. Některá řešení jsou nastíněna v části 7.2.1, jiná se musí určit experimentálně.

Je jisté, že SOMAP má v sobě velký potenciál. Po doplnění některých funkcí (dynamické prostředí apod.) lze použít v reálném světě.

Závěrem uvádím porovnání algoritmů v následující tabulce 8-1.

Algoritmus	**Složitost implementace**	**Jednoduché řízení (reaktivní vrstva)**	**Globální navigace (deduktivní vrstva)**	**Rychlost**	**Nastavení parametrů**
BUG	Jednoduchá	Spolehlivé	NE	Vysoká	Žádné
Exaktní plánování	Střední. Při velké mapě vysoká paměťová náročnost.	Po doplnění reaktivních algoritmů	GIGO efekt	Při velké mapě nízká	Žádné
Rastrové mapy	Střední	Po doplnění reaktivních algoritmů	Velmi dobré výsledky	Vysoká	Žádné
navSOM1	Střední	NE	V trajektorii známé cesty – spolehlivá navigace	Střední	Jen nastavení SOM
navSOM2	Vysoká	Po doplnění reaktivních algoritmů	Velmi spolehlivá	Střední	Vysoké požadavky
SOMAP	Velmi vysoká	Po doplnění reaktivních algoritmů velmi spolehlivá	Velmi spolehlivá	Nízká	Velmi vysoké požadavky

Tabulka 8-1 Srovnání navigačních algoritmů

9. SEZNAM LITERATURY

[1] http://www.robotika.cz

[2] http://www.avari.cz

[3] http://neuron.felk.cvut.cz

[4] V.Mařík,O.Šťepánková,J.Lažanský a kol – Umělá inteligence (4),ACADEMIA – 2003

[5] Petr Novák - Mobilní roboty - pohony, senzory, řízení 1.díl, BEN – 2005

[6] Jaroslav Hynek – Genetické algoritmy a gen. programování, GRADA – 2008

[7] http://www.codeproject.com

[8] http://cs.wikipedia.org

[9] Bakalářská práce, Michal Novotný FSI - VUT v Brně 2006

[10] http://www.hokuyo-aut.jp/

Zkratky

- SOM – Self-organization map, samoorganizující se síť, typ umělé neuronové sítě.
- SOMAP - Self-organization map – area points – řídicí algoritmus.
- AP – Area point – struktura v SOMAP algoritmu.
- navSOM1/2 – Navigation by Self-organization map – navigační algoritmy do jednoho bodu.
- PP – Place point – struktura v navSOM2 algoritmu.
- 2D – Dvě dimenze.
- LVQ – Learning vecrot kvantization – učení SOM kvantováním učícího vektoru.
- UI – Umělá inteligence.

Seznam příloh

Příloha 1: CD které obsahuje:

- SW pro řízení robotu – RobotControl v.1(viz kap. 6.4.1.1).
- Simulační SW pro simulaci a vizualizaci SOMAP algoritmu.
- Zdrojové kódy pro Microsoft VS 2008.