UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Evidenčné číslo: 8b03cc49-304b-43f4-9a0d-d1c86895fe95

REINFORCEMENT LEARNING V ROBOTIKE

Diplomová práca

Študijný program :	Aplikovaná informatika
Študijný odbor:	9.2.9. aplikovaná informatika
Školiace pracovisko:	Katedra
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava 2011

Bc. Peter Jurčo

Abstrakt

Cieľom tejto práce je spraviť prehľad o súčasných metódach a algoritmoch v oblasti reinforcement learningu, paradigme strojového učenia, s dôrazom na aplikácie v robotike. V prvej časti predstavujem reinforcement elarning ako taký, uvádzam niektoré základné pojmy a algoritmy. V ďalšej časti opisujem niektoré najčastejšie problémy pri nasadení algoritmov reinforcement learningu do praxe a následne uvádzam súčasné riešenia týchto problémov v podobe základných tried reinforcement learning algoritmov. V poslednej časti aplikujem dva z reinforcement learning algoritmov na základné robotické úlohy – sledovania chodby a vyhýbanie sa prekážkam. Výsledky testov ukazujú, že robot sa vo väčšine prípadov naučil plniť požadované úlohy a kde sa to nepodarilo prinášam analýzu problémov, ktoré nastali pri učení.

Kľúčové slová: Reinforcement learning, robotika, Decision Tree algoritmus

Poďakovanie

Rád by som sa poďakoval môjmu školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za cenné rady a pripomienky pri tvorbe tejto práce a trpezlivosť, ktorú so mnou mal. Tiež by som sa rád poďakoval svojim rodičom a súrodencom za podporu, špeciálne Ondrejovi a Timovi za vypožičanie prídavnej výpočtovej sily a Márii že na mňa počkala. Ďakujem aj Bohušovi za nakopnutie do práce.

Obsah

Slovník

Reinforcement learning je dobre spracovaná a zdokumentovaná teória. Avšak drvivá väčšina materálov je v anglickom jazyku. Neexistuje slovenská publikácia, ktorá by zaviedla špecifické pojmy z teórie reinforcement learningu v slovenčine. Niektoré pojmy sú síce známe aj z iných oblastí umelej inteligencie, informatiky, či štatistiky, avšak pri preklade tých ostatných som sa musel spoliehať len na vlastné znalosti angličtiny a porozumeniu teórie za nimi. Pri prvom použití týchto pojmov väčšinou uvádzam aj ich anglický ekvivalent. Prípadný čitateľ by si však možno chcel niektoré pojmy kvôli lepšiemu porozumeniu naštudovať dôkladnejšie, preto k práci pripájam tento slovník najdôležitejších pojmov, kde uvádzam ich pôvodné (anglické) znenie, môj slovenský preklad a kapitolu, kde som ich v práci prvýkrát zaviedol.

Samotný pojem reinforcement learning som sa rozhodol neprekladať, ničmenej, jeho preklad do slovenčiny by bol *učenie posilňovaním*, ale ustálený je aj názov *učenie odmenou a trestom* či *učenie zo skúseností*.

Slovenský preklad	Anglický originál	kap.
odmena	reward	1.2
posilňovací signál	reward signal	1.1
stratégia	policy	1.1
učenie s učiteľom	supervised learning	1.2
učenie bez učiteľa	unsupervised learning	1.2
kompromis medzi objavovaním a využívaním	exploration vs. exploitation trade-off	1.3
hodnotová funkcia	value function	1.5
odhadovaná hodnota	estimated value	1.5
stavová hodnotová funkcia	state-value function	1.5
akciová hodnotová funkcia	action-value function	1.5
diskontný faktor	discount rate	1.5
aktualizačné pravidlo	update rule	1.6.1
stopa vhodnosti	eligibility trace	1.6.1
akumulačná stopa vhodnosti	accumulating eligibility trace	1.6.1
nahradzujúca stopa vhodnosti	replacing eligibility trace	1.6.1
pokles gradientu	gradient descent	2.4.1
nevychýlený odhad	unbiased estimation	2.4.1
konečno-rozdielové metódy	finite-difference methods	2.4.1
metódy vierohodnostného pomeru	likelihood ratio methods	2:4.1
prirodzený gradient	natural gradient	2.5
diferenciálový pohon	differential drive	3.3
sledovanie chodby	corridor following	3
vyhýbanie sa prekážkam	obstacle avoidance	3

Úvod

Reinforcement learning je veľmi vzrušujúca a pomerne mladá časť umelej inteligencie. Jej korene síce siahajú ešte k Pavlovovému podmienenému reflexu, v informatike sa naplno začala rozvíjať až posledných 20 rokov, po tom, ako sa dostavili prvé priekopnícke úspechy. Nadšenie neutíchlo ani keď sa ukázali niektoré slabiny funkčnej aproximácie, považovanej za riešenie problémov pre použitie reinforcement learningu v praxi. Naopak, podnietilo to vedeckých pracovníkov na hľadanie lepších metód a algoritmov. Dnes je táto paradigma strojového učenia stále v plienkach, o to vzrušujúcejšie sú jej nové úspechy. Som rád, že sa touto prácou môžem aspoň trochu podieľať na týchto nových objavoch.

1. Reinforcement Learning

Učenie zo skúsenosti je asi najprirodzenejší spôsob učenia. Dieťa, ktoré sa učí chodiť nemá žiadneho učiteľa, ktorý by mu vysvetľoval pohyby potrebné na plynulú chôdzu. Zato pocíti bolesť pri páde na zem a pocit šťastia, keď sa mu podarí prísť k jeho obľúbenej hračke. Opakovaným skúšaním sa posilňujú akcie, ktoré viedli k získaniu odmeny a oslabujú akcie vedúce k zápornému signálu.

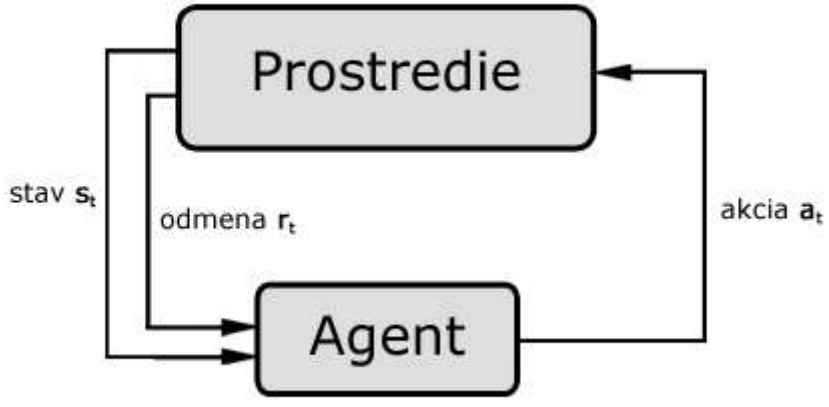
Počas nášho života je interakcia s prostredím nepochybne hlavným zdrojom vedomostí o nás a o prostredí okolo nás. Či už sa učíme riadiť auto, alebo viesť konverzáciu s priateľom, pozorne sledujeme ako prostredie reaguje na to, čo robíme a naším správaním sa snažíme ovplyvniť to, čo sa stane.

Reinforcement learning simuluje práve tento spôsob učenia. Na základe interakcií s prostredím agent posilňuje akcie v smere odmeny.

1.1 Model Reinforcement Learningu

Reinforcement learning nedefinujeme charakterizovaním učiacich metód, ale skôr charakterizovaním učiaceho problému. Každú metódu, ktorá je vhodná na riešenie tohto problému, budeme považovať za metódu reinforcement learningu.

Základný model reinforcement learningu je zobrazený na obrázku 1.1. Agent v čase t vníma stav prostredia s_t, na základe čoho sa rozhodne vykonať akciu a_t, čím zmení stav prostredia a hodnota tohto prechodu je agentovi komunikovaná cez posilňovací signál r_t.



Obrázok 1.1: Základný model reinforcement learningu

Naším cieľom je naučiť agenta optimálnu *stratégiu* (policy) $\pi : S \rightarrow A$, ktorá by pre každý stav vybrala akciu, vedúcu k zvýšeniu celkovej sumy posilňovacích signálov. Táto stratégia môže byť deterministická $a = \pi(s)$, alebo stochastická $a \sim \pi(a|s_t)$. Jej učenie prebieha systémom pokus a omyl, pričom agenta môžu riadiť rôzne algoritmy, ktoré popisujeme v ďalších kapitolách.

Reinforcement learning súvisí s teóriou dynamického programovania a optimálneho riadenia a svoje korene má v Pavlovových psychologických štúdiach o podmienenom reflexe.

1.2 RL ako paradigma strojového učenia

Poznáme tri základné paradigmy strojového učenia – učenie s učiteľom (supervised learning), učenie bez učiteľa (unsupervised learning) a učenie odmenou a trestom (reinforcement learning). Hlavnou rozlišovacou črtou týchto troch paradigiem je spôsob ako získavajú spätnú väzbu. Pri učení s učiteľom vyžadujeme pre každý vstup aj cieľový výstup, ktorý slúži ako spätná väzba k výstupu algoritmu. Pri učení bez učiteľa nie je k dispozícii žiadna spätná väzba, algoritmus sa len snaží nájsť štruktúry vo vstupných dátach. Reinforcement learning používa ako spätnú väzbu tzv. odmenu, numerickú hodnotu, ktorá hovorí nakoľko bola vykonaná akcia v danom stave vhodná. Od učenia s učiteľom sa teda líši v tom, že nemá k dispozícii trénovacie páry vstup/výstup. Odmena, ktorú dostane agent ako spätnú väzbu nehovorí, ktorá akcia by bola najvhodnejšia, ale aká vhodná bola vykonaná akcia. Tento druh spätnej väzby nazývame aj ohodnocovacia spätná väzba [Sutton98].

1.3 Problém objavovania a využívania

Reinforcement learning ako nová paradigma učenia so sebou prináša nové výzvy a problémy, ktoré sa v iných druhoch učenia nevyskytovali. Jedným takýmto problémom, s ktorým sa musí vysporiadať každý, kto sa pokúsi implementovať reinforcement algoritmy na praktické úlohy, je *kompromis medzi objavovaním a využívaním* (exploration vs. exploitation trade-off).

Aby agent dosiahol cieľ, musí v jednotlivých stavov vyberať akcie, ktoré sú podľa jeho znalostí najlepšie *(greedy akcie)* – *využíva* tak naučené informácie. Niekedy však aj nie práve najlepšia akcia môže viesť k lepšiemu celkovému ohodnoteniu. Agent tak musí v procese učenia *objavovať* stavy, do ktorých sa nedostane vykonaním najlepšej akcie. Otázkou ale je, ako sa pri výbere akcie rozhodnúť pre využívanie alebo objavovanie. Tento problém bol intenzívne študovaný po desaťročia mnohými matematikmi a existuje mnoho riešení tohto problému [Kaelbing96]. Pre naše účely bude stačiť, keď uvedieme niekoľko základných.

Jeden z najjednoduchších spôsobov ako sa vysporiadať s problémom využívania a objavovania sú tzv. *ε-greedy* metódy. Agent v nich vyberá *nongreedy* akcie s pravdepodobnosťou ε a *greedy* akcie s pravdepodobnosťou 1-ε ($0 < \varepsilon < 1$). Ak máme statické prostredie, parameter ε môžeme v priebehu učenia postupne znižovať, aby sme spočiatku podporili objavovanie nových stavov a neskôr zvýhodnili naučené akcie.

Môžeme tiež využiť fakt, že počiatočná veľkosť Q-funkcie nemá vplyv na konvergenciu a môže sa nastavovať ľubovoľne. Nastavíme ju teda na hodnotu väčšiu ako najväčšia Q-hodnota môže byť a na výber akcií budeme používať greedy stratégiu. Keďže po vykonaní akcie a v stave s sa hodnota $Q(s,a)$ zmenší, ďalší krát bude v danom stave vybratá ešte nevyskúšaná hodnota. Pre výber akcie sa teda vypočítaná Q-hodnota berie do úvahy až po tom, ako boli vyskúšané všetky možné akcie.

1.4 Markovov rozhodovací proces

Ak úloha reinforcement learningu má Markovskú vlastnosť, t.j. prechodová pravdepodobnosť závisí len od súčasného stavu a zvolenej akcie, hovoríme o nej ako o Markovovom rozhodovacom procese (MDP). MDP je definovaný ako štvorica (S, A, R, T), kde

- S je množina stavov,
- A je množina akcií,

- R je odmeňovacia funkcia $R: S \times A \rightarrow \mathbb{R}$
- T je prechodová funkcia $T: S \times A \rightarrow \Pi(S)$, ktorá vracia pravdepodobnostnú distribúciu cez všetky možné nasledujúce stavy.

Zjednodušene povedané, Markovov rozhodovací proces je séria stavov, v ktorých sa agent musí rozhodnúť, akú akciu spraviť na základe stavu, v ktorom sa nachádza.

Ak je navyše množina stavov a množina akcií konečná, hovoríme o konečnom Markovovom rozhodovacom procese. Konečné MDP majú pre teóriu reinforcement learningu veľký význam. Väčšina RL algoritmov predpokladá, že prostredie je Markovské a konečné. Po odstránení týchto podmienok nie je zaručená konvergencia k optimálnej stratégii, hoci existujú príklady, v ktorých tieto algoritmy fungujú dobre aj po odstránení Markovskej podmienky. [Singh94]

1.5 Hodnotová funkcia

Takmer všetky reinforcement learning algoritmy sú postavené na hľadaní *hodnotovej funkcie* (value function), ktorá hovorí, ako je pre agenta dobré byť v danom stave (alebo ako je dobré spraviť danú akciu v danom stave). To, že „ako je dobré“ sa meria pomocou veľkosti odmeny, ktorú dostaneme, ak začíname v danom stave. Keďže v danom stave ešte nevieme, akú odmenu dostaneme, hovoríme o *odhadovanej hodnote*, ktorú „odhadujeme“ na základe predchádzajúcich skúseností. Veľkosť budúcej odmeny očividne závisí aj od toho, aké akcie vykonáme, preto sa hodnotová funkcia definuje vzhľadom na nejakú stratégiu π.

Poznáme dva druhy hodnotovej funkcie – stavovú a akciovú. *Stavová hodnotová funkcia* sa dá vyjadriť ako

$$V^{\pi}(s) = E[R_t \mid s_t = s] \qquad (1.1)$$

čiže ako očakávaná hodnota celkovej odmeny, ktorú dosiahneme, ak v čase t začneme v stave s a budeme pokračovať podľa stratégie π. *Akciová hodnotová funkcia* pridáva parameter akcie:

$$Q^{\pi}(s,a) = E[R_t \mid s_t = s, a_t = a] \qquad (1.2)$$

Vyjadruje teda hodnotu očakávanej odmeny, ktorú dostaneme sledovaním stratégie π, ak začneme v stave s vykonaním akcie a.

Veľkosť celkovej odmeny je funkcia postupnosti jednotlivých okamžitých odmien v stavoch cez ktoré sme prešli. V najjednoduchšom prípade to je suma týchto odmien:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T \qquad (1.3)$$

kde T je posledný časový krok. Tento prístup je možné použiť v *epizodických úlohách*, teda v úlohách, kde vieme rozdeliť interakciu agenta s prostredím na nejaké podpostupnosti – *epizódy*. Každá epizóda začína v počiatočnom stave a končí v terminálnom stave. Mnohé úlohy, ale nie sú epizodické a požadujeme neprerušovaný beh agenta. V tomto prípade môže byť $T = \infty$ a teda aj hodnota R_t potencionálne nekonečná. Pri týchto tzv. *spojitých úlohách* musíme zaviesť koncept *diskontovania*, aby sme zabránili neobmedzenému rastu celkovej odmeny. Zavedieme parameter γ, $0 \leq \gamma \leq 1$, ktorý nazývame aj *diskontný faktor*. Celkovú hodnotu odmeny budeme potom počítať:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad (1.4)$$

Ak $\gamma < 1$, táto nekonečná suma má konečnú hodnotu pokiaľ postupnosť odmien $\{r_s\}$ je ohraničená [Sutton98]. V hodnote celkovej odmeny teda majú väčšiu váhu „bližšie“ odmeny. Ak $\gamma = 0$, agent berie do úvahy iba okamžitú odmenu. Naopak, čím je γ bližšie k 1, tým viac sa do úvahy berú budúce odmeny.

Optimálnu hodnotovú funkciu si môžeme vyjadriť ako funkciu, ktorá pre všetky stavy s (a akcie a) vracia maximálnu hodnotu:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad (1.5)$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad (1.6)$$

Všetky stratégie π, pre ktoré platí $V^*(s) = V^{\pi}(s)$ ($Q^*(s,a) = Q^{\pi}(s,a)$) nazývame *optimálne stratégie*. Úlohu nájsť optimálnu stratégiu teda môžeme zredukovať na úlohu nájdenia optimálnej hodnotovej funkcie. Na riešenie tejto úlohy sa využívajú *Bellmanove rovnice*. Bellmanova rovnica pre V^{π} má rekurzívny tvar:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \qquad (1.7)$$

kde $T(s,a,s')$ predstavuje pravdepodobnosť prechodu z s do s' pomocou akcie a,

$R(s,a,s')$ odmenu, ktorú za tento prechod získame. Spojením (1.5) a (1.7) dostávame Bellmanovu rovnicu optimality:

$$V^*(s)=\max_a \sum_{s'} T(s,a,s')[R(s,a,s')+\gamma V^*(s')] \qquad (1.8)$$

Poznáme niekoľko vhodných metód na počítanie V^*, ktoré sú postavené na učení sa modelu prostredia, čiže hľadaniu funkcií T a R [Smart02]. Avšak toto učenie vyžaduje veľké množstvo dát a v meniacom sa prostredí môže byť komplikované. Preto sa v praxi používa inkrementálne učenie, ktoré sa iteratívne učí priamo funkciu V^*. V nasledujúcej časti uvedieme niekoľko takýchto algoritmov.

1.6 Algoritmy

V tejto časti predstavíme niekoľko základných reinforcement learning algoritmov. Sú to tabuľkové algoritmy, čiže hodnotová funkcia je reprezentovaná ako pole hodnôt. Táto vlastnosť ich robí nepoužiteľnými pre riešenie väčších problémov, ničmenej sú nepostrádateľným základom pre ďalšie, komplexnejšie algoritmy.

1.6.1 TD Learning

Temporal Difference Learning algoritmy sú RL algoritmy, ktoré sa iteratívne učia funkciu $V^*(s)$ pomocou odmeny a dočasných rozdielov medzi po sebe idúcimi stavmi. Upravovanie $V(s)$ v smere $V(s')$ je vlastne spätná propagácia odmeny. Suttonov TD(θ) algoritmus používa aktualizačné pravidlo (update rule) v tvare:

$$V(s_t)\leftarrow V(s_t)+\alpha[r_{t+1}+\gamma V(s_{t+1})-V(s_t)] \qquad (1.9)$$

kde α je parameter určujúci rýchlosť učenia, používaný aj v mnohých iných učiacich metódach v umelej inteligencii a γ je už vyššie spomínaný diskontný faktor.

Zovšeobecnením algoritmu TD(θ) je TD(λ) algoritmus, ktorého aktualizačné pravidlo má tvar:

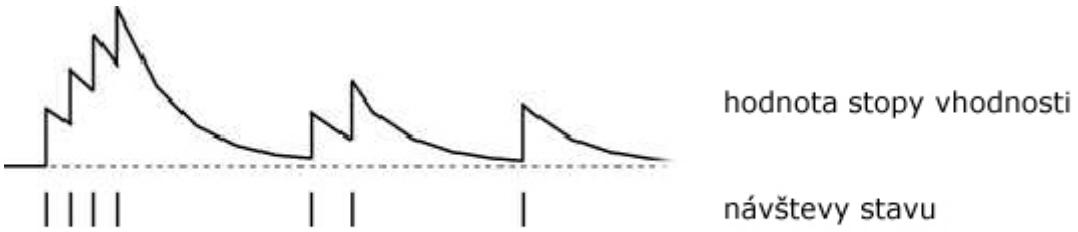
$$V(s_t)\leftarrow V(s_t)+\alpha[r_{t+1}+\gamma V(s_{t+1})-V(s_t)]e_t(s_t) \qquad (1.10)$$

Pre každý stav sme zaviedli novú premennú e, ktorú nazývame *stopa vhodnosti* (eligibility trace). Stopa vhodnosti v sebe zaznamenáva informáciu o

návšteve stavu. Všetky stopy vhodnosti sú na začiatku nainicializované na 0 a po každom kroku sa stopy vhodnosti pre všetky stavy aktualizujú podľa pravidla:

$$e_t(s)=\begin{cases}\gamma\lambda e_{t-1}(s) & ak\ s\neq s_t\\ \gamma\lambda e_{t-1}(s)+1 & ak\ s=s_t\end{cases} \qquad (1.11)$$

kde γ je diskontný faktor a λ $(0 \leq \lambda \leq 1)$ je parameter poklesu vhodnosti. Všetkým stavom klesne každý časový krok stopa vhodnosti $\gamma\lambda$-krát, iba práve navštívenému stavu stúpne navyše o 1 (obr. 1.2).



Obrázok 1.2: Príklad priebehu akumulovanej stopy vhodnosti počas behu TD(λ) algoritmu

V TD(λ) algoritme sa aktualizácia $V(s)$, podľa aktualizačného pravidla 1.10 prevádza každý časový krok pre všetky stavy. Zmenené však budú len stavy, ktoré už boli nedávno navštívené (ktorých stopa vhodnosti > 0) pričom veľkosť zmeny je priamo úmerná počtu návštev a nepriamo úmerná času uplynutého od poslednej návštevy.

Stopu vhodnosti, ako sme ju definovali (1.11) voláme *akumulačná stopa vhodnosti*, pretože jednotlivé návštevy zvyšujú jej hodnotu. Niekedy sa však ukazuje ako lepšia *nahradzujúca stopa vhodnosti*, ktorú aktualizujeme takto:

$$e_t(s)=\begin{cases}\gamma\lambda e_{t-1}(s) & ak\ s\neq s_t\\ 1 & ak\ s=s_t\end{cases} \qquad (1.12)$$

Hoci rozdiel medzi nahradzujúcou a akumulačnou stopou nie je veľký, v niektorých úlohách môže znamenať podstatný rozdiel v rýchlosti učenia [Sutton98].

1.6.2 Q-learning

Jeden z najdôležitejších pokrokov v teórií reinforcement learningu spravil Watkins v roku 1989, keď predstavil svoj Q-learning algoritmus [Watkins89]. Ako

vyplýva aj z názvu, Q-learning je algoritmus učiaci sa funkciu $Q(s,a)$. Algoritmus v čase t aktualizuje hodnotu $Q(s_t,a_t)$ po tom ako agent vykoná akciu a_t, čím sa zmení stav z s_t na s_{t+1} a dostane odmenu r_{t+1}. Aktualizačné pravidlo pre Q-learning je:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \qquad (1.13)$$

Začínajúc s náhodnými hodnotami Q-funkcie, algoritmus iteratívne aplikuje aktualizačné pravidlo a za istých rozumných podmienok, konverguje k optimálnej hodnote $Q^*(s,a)$ [Watkins92]. Optimálna stratégia π^* je potom tá, ktorá si vyberie akciu s najvyššou hodnotou $Q^*(s,a)$:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \qquad (1.14)$$

Q-learning je *off-policy* algoritmus, čo znamená, že sa učí optimálnu hodnotovú funkciu nezávisle od stratégie, ktorou sa pri učení riadi. To nám umožňuje experimentovať s rôznymi stratégiami, ktorými sa agent môže riadiť, bez starostí o konvergenciu algoritmu k optimálnej funkcii. Jediné, čo algoritmus od stratégie požaduje, je to, aby bola Q-funkcia priebežne aktualizovaná pre všetky dvojice stav-akcia. Inými slovami, aby algoritmus konvergoval k optimálnej Q-funkcii, potrebujeme neustále objavovanie (exploration).

1.6.3 SARSA

Sarsa je *on-policy* verzia Q-learningu. Názov algoritmu je technická skratka z postupnosti „State, Action, Reward, next State, next Action“. K typickej štvorici $(s_t, a_t, r_{t+1}, s_{t+1})$, ktorú na učenie používa Q-learning, Sarsa teda pridáva ďalšiu akciu a_{t+1}, ktorú vyberie podľa stratégie odvodenej od súčasnej hodnoty Q-funkcie. Pri jej aktualizácii tak už nemusíme vyberať maximálnu hodnotu Q-funkcie v stave s_{t+1}.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \qquad (1.15)$$

Pretože Sarsa je on-policy algoritmus, aj konvergencia Q-funkcie k optimalite je závislá na stratégii. Ak by sme použili greedy stratégiu (greedy vzhľadom ku Q-funkcii), aktualizačné pravidlo by vyzeralo rovnako ako pri Q-learningu, ničmenej

konvergenciu by to ešte nezaručilo, pretože takáto stratégia by nezaručovala neustále objavovanie. Aby sme zaručili konvergenciu algoritmu Sarsa, musíme si zvoliť *GLIE stratégiu*. GLIE (Greedy in the Limit of Infinite Exploration) stratégia je každá stratégia spĺňajúca tieto dve podmienky [Singh00]:

1. Každá akcia je vykonávaná nekonečne veľa krát v každom stave, ktorý je navštívený nekonečne veľa krát
2. Stratégia sa blíži ku greedy stratégii

Príklad takejto stratégie, je napríklad ε-greedy stratégia, pre ktorú platí

$$\varepsilon_t(s)=\frac{c}{n_t(s)} \qquad (1.16)$$

pre každý stav s, časový krok t, konštantu c ($0 < c < 1$), $n_t(s)$ označuje počet návštev agenta v stave s.

2. Použitie reinforcement learningu v robotike

Idea poskytnutia vysoko-úrovňovej špecifikácie cieľa ako vstupu pre metódy umelej inteligencie namiesto podrobného popisu, ako tento cieľ dosiahnuť, je pre programátorov veľmi príťažlivá. Pri učení s učiteľom je tento cieľ vyjadrený vo forme dvojíc <vstup,výstup>, ktoré slúžia ako vzor pre ďalšie vstupy. Reinforcement learning túto ideu posúva ešte o stupeň vyššie. Stačí nám určiť stavy, do ktorých sa chceme dostať, priradiť k nim príslušnú odmenu, naopak, stavom, ktoré sú nežiadúce priradiť trest (zápornú odmenu), a nechať agenta učiť sa na základe skúsenosti s prostredím. Táto vlastnosť robí reinforcement learning vhodným pre programovanie robotických systémov, kde častokrát nemáme k dispozícii korektné výstupy pre jednotlivé vstupy, ale vieme povedať, kedy robot splnil danú úlohu a zaslúži si odmenu.

V skutočnosti však reinforcement learning nie je až taký dokonalý spôsob riešenia robotických úloh, ako sa na prvý pohľad môže zdať. Pri aplikácii metód reinforcement learningu na reálne úlohy totiž narážame na problémy, ktoré nám znemožňujú akékoľvek učenie. Preto sa aj podstatná časť odborných prác ohľadne reinforcement learningu venuje týmto problémom a metódam na ich prekonanie. Zvyčajne sa to dosahuje zapojením metód učenia s učiteľom do reinforcement learningu. To sa javí ako prirodzená voľba, pretože aj ľudské učenie je kombináciou týchto dvoch prístupov. Mnoho sa naučíme zo skúseností, ale veľa aj pozorovaním a napodobňovaním iných. A práve robotika je odbor, ktorý sa najviac snaží simulovať človeka a jeho správanie. V tejto časti najprv predstavíme niektoré z najvýznamnejších úspechov reinforcement learningu v oblasti riadenia a robotiky, potom uvedieme hlavné problémy aplikácie reinforcement learningu na robotické úlohy a spravíme krátky prehľad súčasných reinforcement learning algoritmov.

2.1 Aplikácie Reinforcement Learningu

Jedným z prvých veľkých a asi aj najznámejším úspechom reinforcement

learningu je TD-Gammon, program hrajúci stolovú hru backgammon na úrovni najlepších svetových hráčov [Tesauro95]. Ako už s názvu vyplýva, používal Temporal Difference Learning, konkrétne TD(λ) algoritmus s funkčnou aproximáciou. Tento algoritmus predstavil v roku 1992 Gerry Tesauro a jeho úspech vzbudil väčší záujem o reinforcement learning. A hoci bol reinforcement learning prístup použitý pri viacerých aplikáciach v oblasti hier, interakcii človeka s počítačom a dokonca aj v ekonomike, najväčšie využitie našiel práve v oblasti robotiky a riadenia.

2.1.1 Aplikácie RL v oblasti riadenia

Jednou takou aplikáciou reinforcement learningu je riadenie autonómnej helikoptéry. Riadenie helikoptéry je náročný problém s asymetrickou, nelineárnou a viacrozmernou dynamikou. Vo všeobecnosti je považovaný za omnoho zložitejší ako riadenie lietadla s pevnými krídlami. Bagnell a Schneider ako prví túto úlohu riešili reinforcement learningom [Bagnell01]. Na nájdenie dobrej stratégie použili *policy search* metódy, čiže metódy, ktoré prehľadávajú priamo priestor stratégii. Výsledky boli povzbudzujúce, avšak v ďalšom vývoji už nepracovali priamo s reinforcement learningom.

Narozdiel od nich, tím ľudí zo Standfordskej univerzity pod vedením Andrewa Nga, pracuje na probléme riadenia helikoptéry dlhší čas so značnými úspechami.[1] Takisto používajú policy search metódy, konkrétne PEGASUS, algoritmus na priame hľadanie dobrej stratégie vo veľkých priestoroch stavov [Ng00]. Keďže helikoptéra lieta vo veľkých rýchlostiach pomerne stabilne, väčšiu výzvu pre nich predstavovali manévre v nízkej rýchlosti, hlavne vznášanie sa helikoptéry na mieste. Pomocou reinforcement learningu sa im podarilo naučiť helikoptéru prevádzať niekoľko akrobatických manévrov, ako napr. salto vpred, bočné rolovanie, lievik s chvostom vo vnútri aj vonku a ďalšie [Abbeel07]. Tiež sa im podarilo naučiť helikoptéru vznášať sa na mieste [Ng03] a to aj vrtuľou dole [Ng04].

Reinforcement learning bol úspešne aplikovaný aj na riadenie systému

1 Súhrn práce tohto tímu na vývoji učiacich sa programov riadenia helikoptéry s viacerými publikáciami a ukážkovými videami môžete nájsť na http://heli.stanford.edu.

výťahov. Ide o optimalizačný problém, kde máme budovu s viacerými poschodiami, niekoľko výťahov a ľudí čakajúcich na výťah. Úlohou je minimalizovať čas čakania ľudí na výťah. Táto úloha nie je triviálna a aj s využitím heuristík sa môžu vyskytnúť situácie, v ktorých musí človek čakať na výťah príliš dlho. Preto sa na túto úlohu Crites a Barto pokúsili nasadiť multiagentový reinforcement learning [Crites98], kde každý agent ovláda jeden výťah. Pretože priestor stavov je spojitý, rozhodli sa o použitie funkčnej aproximácie s neurónovou sieťou ako funkčným aproximátorom. Crites a Barto testovali 2 architektúry – paralelnú, kde majú všetky agenty jednu neurónovú sieť a decentralizovanú, kde má každý agent svoju vlastnú sieť. V oboch prípadoch agenty dostávajú spoločnú odmenu, ktorá závisí od akcií všetkých agentov a teda aj v decentralizovanej architektúre sa agenty nepriamo učia spolupracovať. Tento prístup bol aplikovaný v simulácii s 10 poschodiami a 4 výťahmi (kde je odhadovane 10^{22} stavov) a vo všetkých meraniach obe architektúry prekonali všetky dovtedy známe algoritmy na riadenie výťahov.

2.1.2 Aplikácie RL v oblasti robotiky

Zapojenie učiacich metód do robotiky bol určite prelomový moment. Existuje mnoho robotických problémov, v ktorých konvenčné riadiace algoritmy aj s použitím komplikovaných heuristík nedosahujú výsledky porovnateľné s učiacimi algoritmami. Reinforcement learningu, ako triede takýchto učiacich algoritmov, preto bola venovaná značná pozornosť. V robotike sa reinforcement learning osvedčil najmä pri učení motorických schopností robotov. Inšpirovaní učením dieťaťa, ktoré sa učí chodiť, behať alebo aj chytať a prenášať veci, aplikovali vedeckí pracovníci na svojich robotov reinforcement learning. Končatiny týchto robotov majú zvyčajne niekoľko stupňov voľnosti a nie je jednoduché nájsť správnu stratégiu na jednotlivé činnosti.

Najfrekventovanejší motorický problém je asi učenie chôdze. Je to prvá činnosť, ktorú by sme od robota očakávali, či už ide o dvojnohého humanoidného alebo štvornohého zvieracieho robota. Existuje niekoľko úspešných aplikácií reinforcement learningu na tento problém. Benbrahim a Franklin úspešne použili vlastnú verziu reinforcement learningu s neurónovými sieťami na naučenie chôdze

dvojnohého robota [Benbrahim97]. Morimoto a aj Tedrake používajú vo svojich prácach reinforcement learning, aby naučili dvojnohého robota správne umiestniť zdvihnutú nohu. Využívajú pritom tzv. Poincarého mapu, ktorá modeluje opakujúci sa vzor chôdze [Morimoto04], [Terdrake05]. Morimoto s Doyom zase pomocou hierarchického reinforcement learningu naučili jednoduchého trojdielneho robota vstávať zo zeme [Morimoto01]. Významný je aj úspech Kohla a Stonea, ktorí cez Policy Gradient metódy reinforcement learningu naučili štvornohého robota Sony Aibo stratégiu, pomocou ktorej sa pohyboval rýchlejšie, ako pomocou všetkých ručne upravovaných parametrizovaných stratégii a zaradil sa medzi najlepšie stratégie dosiahnuté učením [Kohl04]. Robot sa navyše optimálnu stratégiu naučil úplne sám, bez akéhokoľvek ľudského zásahu. Podobný úspech s tým istým robotom dosiahli aj Fidelman a Stone, keď ho naučili chytať loptu, medzi predné nohy a hlavu [Fidelman04]. Oba tieto algoritmy boli nasadené na Aibovi v medzinárodnej súťaži RoboCup 2004. V tejto oblasti je spomenutiahodná aj práca Stonea, Suttona a Kuhlmanna [Stone05], v ktorej použili reinforcement learning algoritmus Sarsa(λ) s funkčnou aproximáciou na agentov v simulácii robotického futbalu s výsledkami, ktoré prekonali mnohé iné prístupy.

Mnoho iných výskumov a prác používalo princípy a algoritmy reinforcement learningu, avšak ich celkový prehľad by zabral celý priestor tejto práce, čo už nie je našim cieľom.

2.2 Problémy reinforcement learningu v robotických aplikáciach

V nasledujúcom texte popíšeme problémy súvisiace s reinforcement learningom a naznačíme možné riešenia, ktoré podrobnejšie predstavíme v ďalšej časti práce.

2.2.1 Veľký alebo spojitý priestor stavov a akcií

Algoritmy uvedené v 1. kapitole uchovávajú hodnotovú funkciu v poli, predpokladajú teda diskrétny priestor stavov a akcií rozumnej veľkosti. V robotike sa ale s takýmito podmienkami stretávame len pri úplne jednoduchých úlohách.

Väčšina úloh pracuje so spojitým alebo príliš veľkým priestorom stavov, obzvlášť úlohy s viacrozmernou reprezentáciou stavu, kde je počet stavov exponenciálny od rozmeru stavu. V takýchto prípadoch agent navštevuje stále nové a nové stavy, s ktorými dovtedy nemal žiadnu skúsenosť. Keďže na úspešné naučenie sa blízko-optimálnej stratégie potrebujeme navštíviť každý stav viac ako raz [Kearns99], príliš veľký priestor stavov v praxi znamená neschopnosť navštíviť všetky stavy a naučiť sa tak rozumnú stratégiu.

Problém so spojitým priestorom stavov sa môže riešiť diskretizáciou stavového priestoru. Nie vždy ale vieme povedať, aká diskretizácia je pre danú úlohu vhodná. Ak budeme diskretizovať príliš hrubo, môžeme do jedného stavu zlúčiť stavy, ktoré k sebe nepatria. Na druhej strane, ak budeme diskretizovať príliš jemne, skončíme s veľkým počtom stavov. Obe možnosti znemožňujú robotovi naučiť sa rozumnú stratégiu. Existuje niekoľko algoritmov, ktoré sa pokúšajú o rozumnejšiu diskretizáciu. Parti-game algoritmus [Moore95] sa napríklad snaží dynamicky zjemňovať diskretizáciu v kritických oblastiach, ak učenie zlyháva. Continuous U Tree algoritmus [Uther98] používa na nájdenie správnej diskretizácie stavového priestoru techniky rozhodovacích stromov. Podobný algoritmus sme úspešne použili v rámci tejto práce (v časti 3. Experiment) na základné robotické úlohy.

Avšak problém veľkého stavového priestoru bol v predchádzajúcich rokoch najčastejšie riešený pomocou funkčnej aproximácie. Aproximovať môžeme hodnotovú funkciu, kde sa namiesto kompletnej tabuľky snažíme upravovať parametre funkcie. Na „úpravu“ týchto parametrov sa môže použiť niektorá z metód učenia s učiteľom (napr. neurónové siete). Ďalšia možná aproximácia je aproximácia stratégie. O aproximačných technikách pojednávame podrobnejšie v ďalších kapitolách.

2.2.2 Čiastočne pozorovateľné prostredie

Teória reinforcement learningu, tak ako sme ju predstavili v prvej kapitole predpokladá úplne pozorovateľné prostredie. Avšak v mnohých robotických aplikáciach je stav pozorovaný robotom len čiastočne. Napríklad v Robocupe môže

nastať situácia, keď kamera robota sníma loptu, ale nie súperovu bránku. Takéto rozhodovacie procesy nazývame čiastočne pozorovateľné Markovove rozhodovacie procesy (Partially Observable Markov Decision Process – POMDP). Na rozdiel od MDP v nich nevieme s určitosťou povedať v akom stave sa práve nachádzame. Namiesto stavu máme len pozorovanie a musíme si tak udržiavať pravdepodobnostnú distribúciu cez všetky stavy, ktorá nám hovorí s akou pravdepodobnosťou je ktoré pozorovanie ktorý stav. Na to, aby sme vedeli tieto pravdepodobnosti určovať, potrebujeme si pamätať niekoľko predchádzajúcich stavov a akcií. Takúto krátkodobú pamäť môžeme modelovať aj pomocou *stôp vhodnosti* (viď 1.6), ale ako vhodnejšia sa ukazuje reprezentácia pomocou *rekurentných neurónových sietí,* špeciálne pomocou architektúry *dlhej krátkodobej pamäte* (Long Short-Term Memory – LSTM).

Vo všeobecnosti je však nájdenie optimálnej stratégie pre POMDP prostredia PSPACE-úplný problém [Papadimitriou87]. Na riešenie takýchto úloh teda potrebujeme aproximačné algoritmy. Jedna skupina týchto algoritmov je založená na už spomínanej funkčnej aproximácii. V nich aproximujeme hodnotovú funkciu alebo celú stratégiu nejakou funkciou, ktorú aktualizujeme ako prechádzame priestorom stavov. Iný druh aproximácie predstavuje hierarchický reinforcement learning. Napríklad Hierarchický Q-learning [Wiering97] rozloží priestor stavov POMDP na menšie regióny, v ktorých rieši jednotlivé úlohy už ako MDP.

Dobrý prehľad aproximačných algoritmov na riešenie POMDP spravil Milos Hauskrecht v [Hauskrecht00].

2.2.3 Určenie riadiacej stratégie

Hoci reinforcement learning má tú ambíciu naučiť sa optimálnu stratégiu bez akýchkoľvek počiatočných znalostí o svete, niekedy by nám zapojenie takýchto znalostí pomohlo skrátiť dobu učenia. Pri off-policy algoritmoch, teda pri algoritmoch, kde je riadiaca stratégia nezávislá na učiacej sa stratégii, môžeme tieto počiatočné znalosti využiť pri výbere a tvorbe riadiacej stratégie.

Niektoré robotické úlohy sú napríklad definované tak, že robot dostáva odmenu až po vykonaní dlhej sekvencii akcií. Ide o tzv. *riedku odmenu*. Trvá dlhú

dobu, kým robot príde do stavu s odmenou a ešte dlhšiu, kým sa táto odmena spätne rozšíri do predchádzajúcich stavov. Existujú techniky, ktoré robota povzbudzujú v častejšom navštevovaní stavov s odmenou. Iné techniky sa naopak snažia posilniť spätné šírenie odmeny (napr. pomocou stôp vhodnosti).

2.3 Funkčná aproximácia hodnotovej funkcie

Pri implementácii algoritmov reinforcement learningu v robotike veľmi často narážame na problém veľkého alebo spojitého priestoru stavov. Jeden z prvých nápadov, ako tento problém riešiť, bolo použiť funkčnú aproximáciu, známu to metódu zo štatistiky a strojového učenia na modelovanie distribúcie dát. Na reprezentáciu hodnotovej funkcie sa namiesto tabuľky použije parametrizovaná funkcia s parametrami $\vec{\Theta_t}$, ktoré aktualizujeme pomocou metód učenia s učiteľom. Najčastejšie ide o neurónové siete, gradient-descent metódy alebo CMAC (Cerebellar Model Articulation Controller).

Výhoda takejto reprezentácie je, že máme aproximovanú hodnotu hodnotovej funkcie aj pre stavy, ktoré sme ešte nenavštívili, ale sú „blízko“ pomerne dobre preskúmaným stavom. Táto vlastnosť robí aproximačné metódy veľmi lákavé na použitie v problémoch so spojitými doménami.

Ako však Boyan a Moore ukázali v [Boyan95], jednoduché nahradenie tabuľky funkčnou aproximáciou znamená zničenie záruky konvergencie. Dokonca aj v niektorých zdanlivo ľahkých prípadoch algoritmus diverguje. Hlavným dôvodom tohto nežiadúceho javu je iteratívny charakter reinforcement learning algoritmov. Preto malá chyba aproximácie, ktorá by v metódach učenia s učiteľom nevadila, sa započíta aj do nasledujúcej aproximácie. Po niekoľkých iteráciach algoritmu môže byť táto kumulovaná chyba už tak veľká, že aproximácia prestáva byť užitočná a učenie zlyháva.

Napriek týmto pesimistickým teoretickým zisteniam sa funkčná aproximácia v praxi dobre osvedčila. Existujú funkčné aproximátory, ktoré na konkrétnych problémoch, alebo na problémoch z konkrétnej domény, aproximujú viac než uspokojivo. (napr. TD-gammon [Tesauro95]; riadenie systému výťahov [Crites98];

problém plánovania úloh [Zhang95] a.i.). Vieme tiež, že Suttonov on-policy TD(λ) algoritmus konverguje, ak je použitý lineárny funkčný aproximátor [Tsitsiklis97]. Nedávno bol predstavený aj prvý TD algoritmus, ktorý aj pri použití nelineárneho funkčného aproximátora (ako je napr. neurónová sieť) konverguje k lokálnemu optimu [Sutton09].

Iný prístup predstavuje tiež pomerne nová metóda evolučnej funkčnej aproximácie [Whiteson06], ktorá vychádza z pozorovania, že úspech funkčnej aproximácie závisí od architektúry aproximátora, ktorý sa väčšinou musí vytvárať manuálne. Whiteson a Stone na nájdenie optimálnej architektúry aproximátora (v ich prípade neurónovej siete) použili klasické evolučné algoritmy. Výsledky boli porovnateľné, v niektorých prípadoch aj lepšie, ako ručne vytvorená architektúra. Nevýhoda tohto prístupu spočíva v dlhšej dobe učenia, ktorá je spôsobená tým, že reinforcement learning algoritmus sa musel niekoľkokrát zopakovať, kým sa evolvovala optimálna architektúra neurónovej siete.

2.4 Policy Learning

Zatiaľ čo v deväťdesiatych rokoch v praktickom reinforcement learningu zažívala úspechy funkčná aproximácia hodnotovej funkcie, posledné desaťročie sa do popredia dostal prístup priameho hľadania stratégie (policy learning).

Jedným zo spôsobov prehľadávania priestoru stratégii je vyskúšanie všetkých možností. Takéto prechádzanie priestorom musí mať nejaké usporiadanie alebo sklon, ktorý reflektuje naše znalosti o riešeniach v prehľadávanom priestore. Napr. *Levinove prehľadávanie* [Schmidhuber97] používa výpočtovú a kolmogorovskú zložitosť jednotlivých riešení ako mieru usporiadania. Littman zase do reinforcement learningu zaviedol branch-and-bound metódu [Littman94]. Na hľadanie optimálnej stratégie boli použité aj evolučné [Glickman01] a hierarchické [Baum95] prístupy. Ničmenej, žiadny z týchto prístupov nezaznamenal také výrazné úspechy ako *policy gradient metódy*.

Začínajúc priekopníckou prácou Gullapaliho, Franklina a Benbrahima [Gullapali94],[Benbrahim97] boli policy gradient metódy používané v celej rade

robotických úloh, ako v jednoduchých (napr. balancovanie žrde [Kimura98]), tak aj v zložitejších (napr. učenie chôdze [Terdrake05], [Kohl04]). Tieto metódy patria v súčasnosti k najefektívnejším reinforcement learning metódam pre riešenie úloh pracujúcich vo viacrozmerných spojitých priestoroch.

Policy gradient metódy predstavujú prístup, ktorý optimalizuje priamo parametrizovanú riadiacu stratégiu π_ϑ, s vektorom parametrov ϑ, pomocou *poklesu gradientu* (gradient descent). Stratégia π_ϑ musí byť v policy gradient metódach diferencovateľná vzhľadom na svoje parametre ϑ. Všeobecný cieľ optimalizácie stratégie v reinforcement learningu je optimalizácia parametrov ϑ tak, aby očakávaná odmena

$$J(\vartheta)=\mathrm{E}\left\{\sum_{k=0}^{H}\gamma^k r_k\right\} \qquad (2.1)$$

bola optimálna, pričom γ je diskontný faktor a r_k je odmena v k-tom kroku. H je takzvaný *horizont*. Pre epizodické úlohy je konečný, pre spojité úlohy je nekonečný. Pri nekonečnom horizonte je potrebná normalizácia.[2]

Parametre aktualizujeme podľa vzťahu:

$$\theta_{h+1}=\theta_h+\alpha_h\nabla_{\theta_h}J \qquad (2.2)$$

kde h je poradové číslo aktualizácie, α_h je rýchlosť učenia a $\nabla_{\theta_h}J$ je gradient stratégie. Za určitých podmienok[3] je garantovaná konvergencia prinajmenšom k lokálnemu minimu.

Gradient $\nabla_{\theta_h}J$ samozrejme nemáme priamo k dispozícii, ale pomocou dát, ktoré získame počas behu programu, môžeme túto hodnotu odhadovať. V posledných rokoch vzniklo mnoho rôznych algoritmov, ktoré počítajú tento odhad. Z nich najvýznamnejšie prístupy sú *konečno-rozdielové metódy* (finite-difference methods) a *metódy vierohodnostného pomeru* (likelihood ratio methods). Konečno-rozdielové metódy boli kvôli svojej jednoduchosti viackrát použité v robotike [Kohl04], [Terdrake05], [Geng05]. Na úspešnú implementáciu tejto

2 V tomto prípade normalizácia spočíva vynásobením očakávanej návratovej hodnoty normalizačným faktorom, ktorý volíme (1-γ) pre $\gamma < 1$.

3 Odhad gradientu je *nevychýlený* (unbiased estimator), $\sum_{k=0}^{\infty}\alpha_k\to\infty$ a $\sum_{k=0}^{\infty}\alpha_k^2=const$

metódy však potrebujeme mať určité znalosti o robotickom systéme, čo nie vždy máme. Preto sú metódy vierohodnostného pomeru prijateľnejším riešením. Tie sú založené na REINFORCE algoritme, vynájdenom Ronaldom Williamsom už začiatkom deväťdesiatych rokov [Williams92]. Súčasné metódy vierohodnostného pomeru tento algoritmus vylepšujú rôznym spôsobom. Oproti konečno-rozdielovým metódam majú viacero výhod, okrem iného aj rýchlejšiu konvergenciu a lepšie spracovanie zašumených dát. Aj kvôli týmto výhodám bol policy gradient learning s metódami vierohodnostného pomeru použité v mnohých robotických aplikáciach [Benbrahim97], [Gullapali94], [Kimura98], [Endo05].

Policy gradient metódy majú mnoho výhod. Medzi najvýznamnejšie patrí to, že oproti funkčnej aproximácii hodnotovej funkcie často vyžadujú menej parametrov pri učení, že reprezentácia stratégie môže byť zvolená zmysluplne vzhľadom na úlohu a môžeme do nej zapracovať predchádzajúce znalosti, a že v literatúre existuje pre odhadovanie gradientu viacero rôznych algoritmov, ktoré sú silne teoreticky podchytené. Výhodou je aj jemný prechod medzi stratégiami v procese učenia. Špeciálne v robotike častokrát požadujeme, aby zmeny v stratégii boli jemné. Príliš veľké skoky v stratégii by mohli byť pre robota nebezpečné a aj počiatočné nastavenia stratégie by sa mohli zmazať v jednom kroku.

Hlavné nevýhody policy gradient metód sú: zložitejšie použitie v off-policy nastavení, pomalšia konvergencia v diskrétnych problémoch a to, že nemáme zaručené dosiahnutie globálneho maxima. Podrobnejší popis policy gradient metód nájdeme v [Sutton99], [Aberdeen02], [Peters06], [Peters10].

2.5 Aktér-kritik metódy

Aktér-kritik metódy kombinujú predchádzajúce dva prístupy - aproximáciu hodnotovej funkcie a policy learning. Pre obidva majú separátnu pamäťovú štruktúru. Štruktúru pre stratégiu nazývame *aktér*, pretože ju používame na výber akcií, zatiaľčo odhadovanú hodnotovú funkciu nazývame *kritik*, kvôli tomu, že kritizuje akciu vybratú aktérom. Obidve štruktúry sú aktualizované oddelene, ale ich učenie prebieha simultálne v iteračnom algoritme.

Aj keď sa môžu zdať ako vyšší stupeň reinforcement learningu, aktér-kritik metódy v skutočnosti patria k prvým reinforcement learning algoritmom. Objavené už v osemdesiatych rokoch minulého storočia sa kvôli zvýšenému záujmu o funkčnú aproximáciu hodnotovej funkcie odsunuli do úzadia. Avšak po tom, čo sa napriek mnohým praktickým úspechom dokázali slabé teoretické vlastnosti funkčnej aproximácie a naopak objavili sa konvergenciu zaručujúce policy gradient algoritmy, aktér-kritik metódy prežili na počiatku 21. storočia malú renesanciu. Algoritmy kombinujúce praktické vlastnosti funkčnej aproximácie hodnotovej funkcie a teoretické záruky policy gradient metód sa v súčasnosti ukazujú ako najlepšie reinforcement learning algoritmy pre väčšinu problémov.

Ako prvý prišli s touto kombináciou Konda a Tsitsiklis [Konda00], ale významnejší pokrok prišiel až s použitím *prirodzeného gradientu* (natural gradient) na aktualizáciu parametrov stratégie. Prirodzený gradient odbúrava problémy normálneho gradientu s tzv. plošinami, ktoré sa niekedy vyskytujú na povrchu funkcie očakávanej odmeny, a ktoré viedli k pomalej konvergencii policy gradient algoritmov. Prirodzený gradient sleduje najstrmší vzostup funkcie vzhľadom na Fischerovu informačnú vzdialenosť. Už aj v metódach učenia s učiteľom sa tento prístup ukazoval byť omnoho efektívnejší ako klasický gradient [Amari98] a pri jeho zavedení do reinforcement learningu sa táto efektívnosť potvrdila [Kakade02], [Peters03]. Použitie prirodzeného gradientu v aktér-kritik algoritme priniesol povzbudivé výsledky aj v robotických aplikáciach [Peters08].

3. Experiment

V tejto časti budem demonštrovať použitie konkrétneho reinforcement algoritmu na konkrétnom probléme. Vybral som si dve príbuzné robotické úlohy – sledovanie chodby (corridor following) a vyhýbanie sa prekážkam (obstacle avoidance). Kvôli teoretickým problémom s funkčnou aproximáciou (ktoré som otestoval aj v praxi) som sa rozhodol ísť cestou diskretizácie stavového priestoru. Uniformná diskretizácia a následné použitie tabuľkového Q-learningu však konvergovali veľmi pomaly a preto som použil algoritmus, ktorý rozdeľuje priestor stavov dynamicky.

Reinforcement learning som aplikoval na robota Pioneer 3DX v simulácii prostredia Microsoft Robotics Studio. Preto najprv v krátkosti rozoberiem výhody a nevýhody simulácie ako takej, predstavím vývojové prostredie Microsoft Robotics Studio a robota, s ktorým som pracoval.

3.1 Simulácia

Vývoj robotických aplikácii je špecifický svojou hardvérovou závislosťou, čo má za následok výskyt problémov, ktoré sa v iných oblastiach informatiky nevyskytujú. Ak naša aplikácia pracuje napríklad s multiagentovými systémami, sme obmedzovaní počtom robotov, ktoré máme k dispozícii, pričom niektoré roboty môžu byť iba prototypy. Problém nastáva aj pri ladení softvéru, keď neodhalená chyba v programe môže spôsobiť vážne poškodenie hardvéru. Špecifické problémy majú viaceré učiace algoritmy, ktoré konvergujú až po veľkom počte epizód (rádovo stovky až tisícky epizód), čo je prakticky veľmi neefektívne, obzvlášť v testovacej fáze programu, keď hľadáme optimálne hodnoty konštánt.

Možným riešením týchto problémov je simulácia. Avšak žiadna simulácia nedokáže stopercentne simulovať skutočný svet. Všeobecný problém akýchkoľvek modelov sveta je paradoxne ich dokonalosť. V robotických simulátoroch sa táto nežiadúca dokonalosť vyskytuje najmä pri meraniach senzorov, ktoré bývajú v skutočnom svete nepresné a zašumené. Tento problém sa rieši úmyselným „kazením“ meraní v simulácii pridaním náhodného šumu. Sofistikovanejšie riešenie

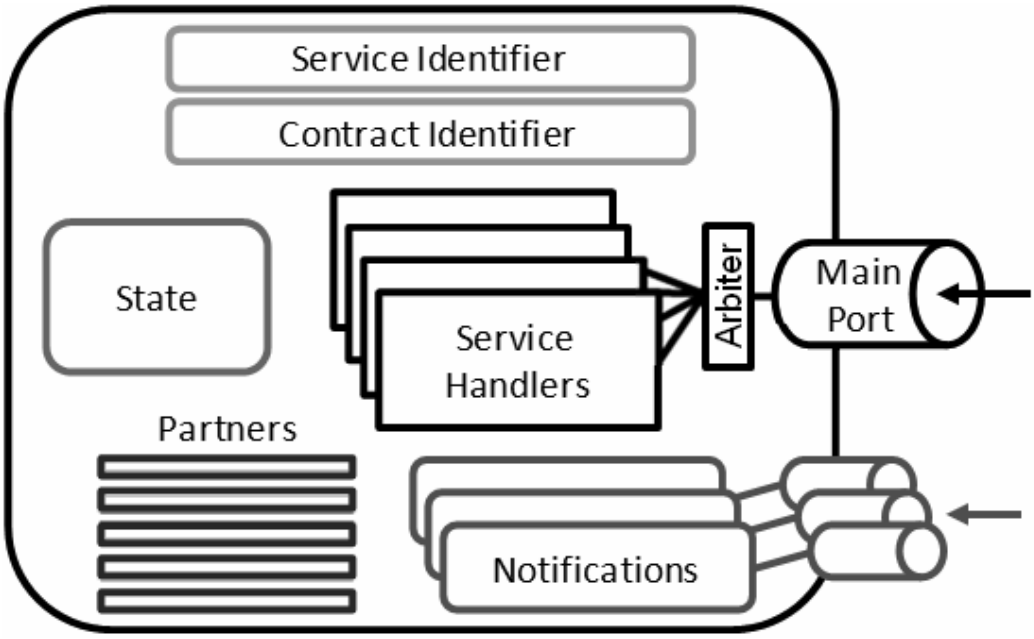
je upravovať merania v simulácii podľa meraní zo skutočného sveta. Tak či onak, prechod zo simulácie do skutočného sveta vyžaduje dodatočné úpravy konštánt a iných nastavení algoritmu, prípadne aj dotrénovanie robota na reálnych dátach.

3.2 Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio (MRDS) je vývojové prostredie určené na tvorbu robotických aplikácii pre rôzne hardvérové platformy. Skladá sa z niekoľkých základných častí.

3.2.1 Concurrency and Coordination Runtime (CCR)

V robotických aplikáciach je často potrebné vysporiadať sa s asynchrónnou komunikáciou medzi programom, viacerými vstupmi prichádzajúcimi z robotických senzorov a výstupmi riadiacimi robota. MRDS obsahuje knižnicu CCR, ktorá obsahuje triedy a metódy zabezpečujúce súbežnosť vlákien, ich koordináciu a posielanie správ medzi nimi. Programátor tak môže zložiť aplikáciu z nezávislých častí kódu, ktoré bežia súbežne a asynchrónne, bez nutnosti manuálnej tvorby vlákien, zámkov či semaforov.



Obrázok 3.1: Schéma služby v MRDS

3.2.2 Decentralized Software Services (DSS)

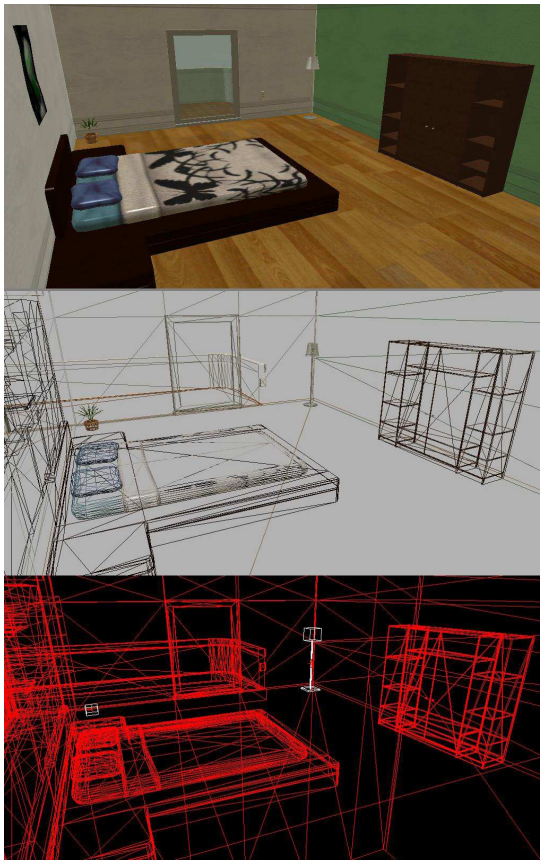
To, čo poskytuje CCR v rámci jedného procesu, poskytuje knižnica DSS v rámci viacerých procesov, ba dokonca počítačov. Aplikácia pracujúca s DSS pozostáva z paralelne bežiacich nezávislých služieb. Každá služba má svoj stav, ktorý môže meniť na základe správ, ktoré prijíma. Následne môže poslať notifikácie ďalším službám, ktoré sa prihlásili na odber. Priamejší spôsob komunikácie (nezávislý na udalosti) predstavuje partnerstvo služby s inými službami.

3.2.3 Visual Simulation Environment

MRDS poskytuje 3D simulačné prostredie s licencovaným fyzikálnym modelom AGEIA™ PhysX™ s interiérovými aj exteriérovými scénami a s niekoľkými preddefinovanými robotmi - LEGO NXT, iRobot Create, MobileRobots Pioneer 3DX a robotické rameno KUKA LBR3.

Všetky komponenty scény, vrátane robota, sú plne editovateľné, čo programátorovi umožňuje vytvárať simulácie a roboty aké potrebuje.

Pri simulácii môžeme jednoducho ovládať kameru a prepínať medzi rôznymi zobrazeniami scény (renderovacími módmi) – vizuálnym, drôteným a fyzikálnym (obrázok).

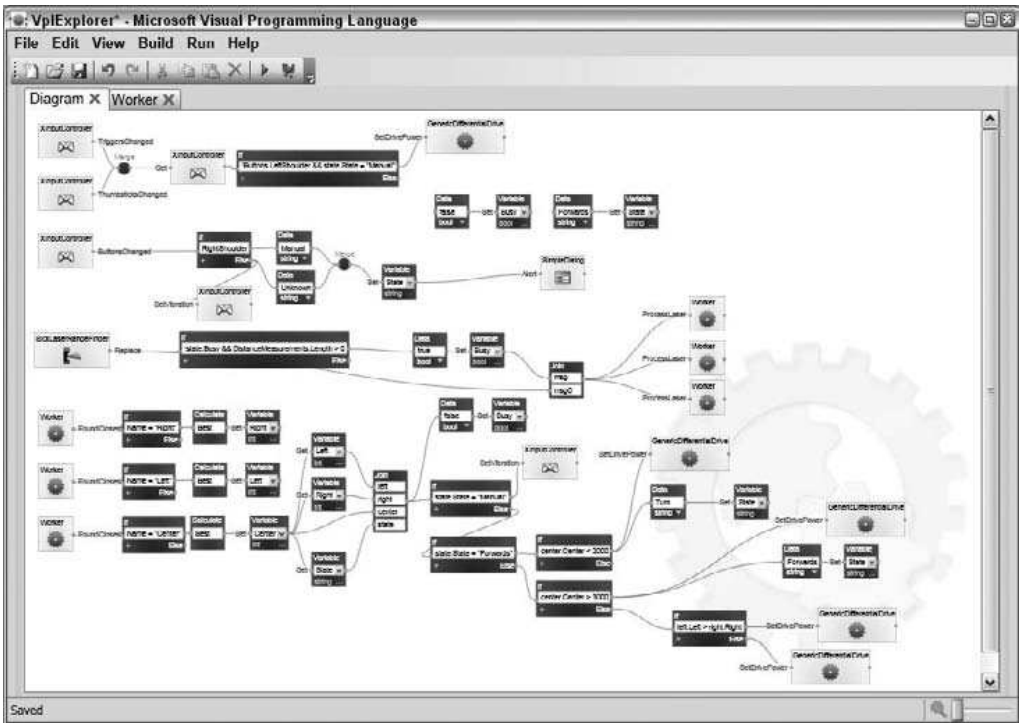


Obrázok 3.2: Renderovacie módy v MRDS

3.2.4 Visual Programming Language

Vizuálny programovací jazyk slúži ako grafické vývojové prostredie založené na toku dát. (namiesto toku príkazov, ako to je pri klasickom programovaní). Tento prístup môžeme prirovnať k výrobnej linke, kde sa v každom bloku vykoná operácia až keď tam dorazia dáta. Jednotlivé bloky VPL diagramu môžu byť služby,

výpočty, podmienky, alebo vnorené diagramy. Prechody medzi blokmi reprezentujú posielanie správ v DSS. VPL je vhodné pre používateľov, ktorí chcú vytvárať aplikácie, aj bez znalosti konkrétneho programovacieho jazyka, ako aj pre skúsených programátorov na znázornenie logiky programu.



Obrázok 3.3: Visual Programming Language – vývojové prostredie

Okrem vizuálneho programovacieho jazyka MRDS umožňuje programovať v jazykoch C#, C++, Visual Basic a Python.

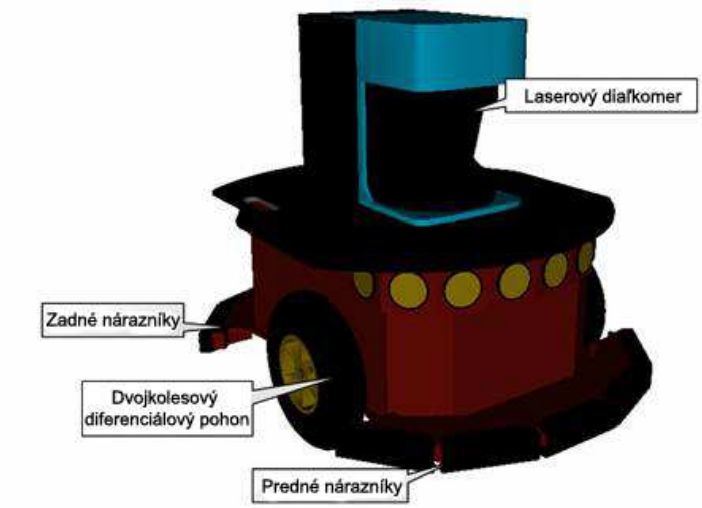
MRDS bolo úspešne používané pri viacerých robotických projektoch, ako napríklad pri programovaní autonómnych vozidiel, robotickej ruky a pri ďalších rozsiahlych aj menších robotických aplikáciach.

3.3 Pioneer 3DX

Na zadané úlohy som použil model robota Pioneer 3DX (obr. 3.4). V súčasnosti je to snáď najpopulárnejší edukačný robot. Ide o dvojkolesového robota s *diferenciálovým pohonom* kolies, čo znamená, že každé koleso je ovládané samostatným aktuátorom, čo okrem iného umožňuje aj rotáciu robota na mieste. Základná verzia Pioneeru obsahuje predné aj zadné nárazníkové senzory a ultrazvukové sonary. Okrem toho je robot rozšíriteľný o laserový diaľkomer,

kameru a ďalšie zaujímavé príslušenstvo.

Vo svojej implementácii som vychádzal z modelu robota a z prostredia, ktoré pre edukačné účely navrhol Trevor Taylor [Johns08]. Z neho som používal iba riadiaci systém, laserový diaľkomer a predný nárazník. Keďže simulácia v Microsoft Robotics Studio pracuje na princípoch decentralizovaného systému služieb (viď 3.2.2), všetky tieto časti pracovali nezávisle jedna od druhej.



Obrázok 3.4: Model robota Pioneer 3DX

3.4 Sledovanie chodby

V úlohe sledovania chodby od robota požadujeme plynulý pohyb po chodbách bez nárazov do stien. Zložitosť tejto úlohy závisí od robota a našich požiadavok, ktoré na neho kladieme. Existuje niekoľko riešení, ktoré operujú v reálnom svete a pracujú s kamerovým vstupom, ktorý sa spracováva a používa ako vstup do učiacich algoritmov alebo deterministických stratégii. Jednoduchšia verzia má na vstupe vzdialenosti k najbližším prekážkam. Tieto vzdialenosti pochádzajú najčastejšie z ultrazvukových alebo laserových senzorov. Na takúto verziu problému som sa pokúsil aplikovať reinforcement learning.

3.4.1 Analýza úlohy

Stavový priestor

Ako vstup do môjho algoritmu (alebo v terminológii reinforcement learningu: na reprezentáciu stavu) slúžili merania laserového diaľkomera. Ten sa skladá z 361 laserových lúčov tvoriacich horizontálny poloblúk, čiže jeden laser pripadá na pol stupňa. Diaľkomer pracuje nezávisle na robotovi, každú sekundu vráti nové hodnoty merania v rozsahu od 0 po 8000mm. Ak by sme na

reprezentáciu stavu použili tieto nespracované hodnoty, mali by sme stavový priestor o veľkosti 8000^{361}. Taký veľký stavový priestor je nemožné v reálnom čase vôbec celý prejsť, nieto sa ešte niečo naučiť.

Jeden spôsob ako sa s tým vysporiadať je použiť funkčnú aproximáciu. Jednoduché použitie lineárneho funkčného aproximátora na Q-learning nezafungovalo, hodnota parametrov divergovala už po menej ako 100 epizódach. Skúsil som preto zmenšiť stavový priestor. Najprv som si všimol, že na charakterizáciu stavu nám stačia tri merania – na 0°, 90° a 180°, čiže vzdialenosti robota k stenám naľavo, vpredu a napravo. Ďalej som použil diskretizáciu na veľkosti vrátených meraní, ktoré som namapoval na interval 0..80. Dostal som tak stavový priestor 80^3.

Akcie

Použil som štandardné diskrétne akcie – vpred, vľavo a vpravo. Akcia vpred aktivovala pohyb dopredu, akcie vľavo a vpravo zase vykonávali otáčanie robota na mieste o 30°. Hoci som na tréning použil chodby s 90° a 180° zákrutami, otáčanie o 30° umožňuje robotovi plynulejší pohyb aj v priestoroch s nekonvenčnými chodbami. Taktiež to pre nás predstavuje väčšiu výzvu, nakoľko pre úspešne prejdenie 180° zákruty sa potrebuje robot naučiť vykonať sekvenciu šiestich otočení (hoci nie nutne po sebe).

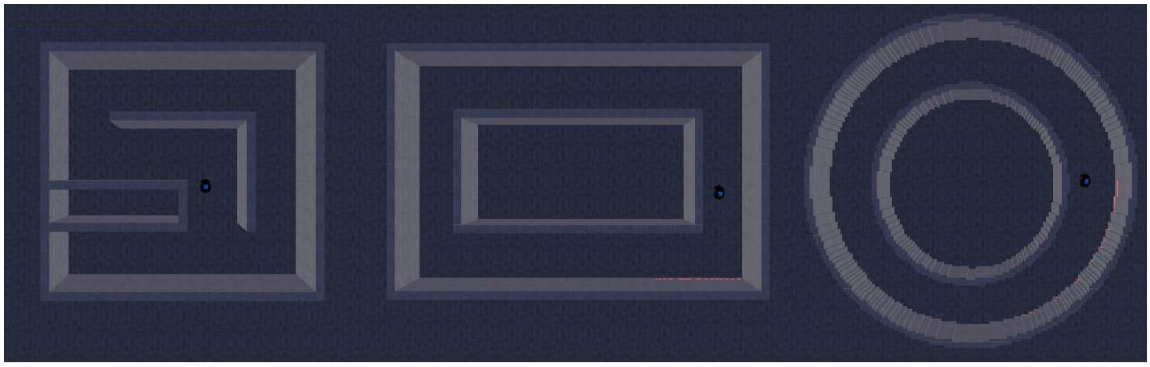
Odmena

Naším cieľom je naučiť robota chodiť po chodbách bez narazenia do steny. Existuje viac spôsobov ako v takejto úlohe určiť odmenu. Nepochybne robot musí dostať trest za každé narazenie do steny. K tomu môžeme pridať aj pozitívnu odmenu ak robot chodí v dostatočnej vzdialenosti od steny. Ďalším rozšírením by bolo pridanie nižšieho trestu aj za to, keď sa robot nachádza v blízkosti steny, aj keď do nej ešte nenarazil. S týmito druhmi odmien, jednotlivým limitom vzdialeností a výškami odmien som v priebehu učenia experimentoval.

Prostredie

Na úlohu sledovania chodby som si vytvoril jedno trénovacie a dve testovacie

prostredia (obr. 3.5). Trénovacie prostredie som zvolil s 90° a 180° zákrutami, a to tak, aby v ňom boli aj ľavotočivé aj pravotočivé zákruty a tiež aj slepá ulička, aby sa robot trénoval na rôzne situácie. Na testovanie som si pripravil jednoduchú obdĺžnikovú a kruhovú chodbu.



Obrázok 3.5: Zľava: trénovacie prostredie a 2 testovacie prostredia v úlohe sledovania chodby

3.4.2 Q-learning

Vďaka svojim konvergenčným zárukám je tabuľkový Q-learning stále lákavá možnosť pre použitie na menšie robotické úlohy. Pre našu stredne veľkú úlohu je však potrebné spraviť niekoľko vylepšení, aby sme rýchlosť konvergencie spravili aspoň trochu prijateľnou.

Prvé vylepšenie spočíva v ešte väčšom okresaní priestoru stavov. Už po uniformnej diskretizácii máme 80x80x80 stavov. Otázka je, či naozaj všetky potrebujeme. Bočné merania nám pomáhajú rozlišovať stavy kedy sme blízko a kedy ďaleko od steny. Ale trénovacie aj testovacie prostredia majú chodby široké len 2000mm, čiže bočné merania môžeme ohraničiť touto hodnotou. Zmenšíme tým priestor stavov na 20x80x20 stavov, čo je 16-krát menej ako v predchádzajúcom algoritme.

Druhá možnosť ako zlepšiť konvergenciu je posilniť spätné šírenie chyby. Čím máme viac rozdielnych stavov, tým menej návštev pripadne na každého z nich. Preto sa snažíme vydolovať maximum z každej jednej návštevy stavu. Jeden zo spôsobov ako toto spraviť predstavujú stopy vhodnosti. Algoritmus TD(λ) so stopami vhodnosti predstavený v 1.6.1 počíta s jednou odmenou na konci epizódy a po nej aktualizuje všetky navštívené stavy, na základe odmeny a „vzdialenosti“ stavu od odmeny. Keďže v našom algoritme dostával robot odmenu aj

počas behu epizódy (a to dosť často), prevzal som ideu spätného šírenia chyby a upravil ju. Použil som zásobník do ktorého som si ukladal navštívené stavy bez odmeny. Akonáhle robot dostane v stave odmenu alebo trest, vyberá postupne všetky stavy zo zásobníka a aktualizuje ich Q-hodnoty s koeficientom $1/t^2$, kde t je počet krokov stavu od odmeny. Takto sú teda priamo odmenené (resp. potrestané) stavy vedúce k odmene (trestu). Ako však poznamenáva Sutton a Barto v [Sutton98], stopy vhodnosti sa pri off-policy algoritmoch, ako je aj náš Q-learning, musia používať opatrnejšie, pretože počas trénovania nepoužívajú iba greedy akcie, ale z času na čas vykonajú náhodnú akciu. Ak dostaneme odmenu alebo trest po vykonaní náhodnej akcie, nemôžeme kredit za získanie odmeny (trestu) pripisovať aj predchádzajúcim stavom. Preto ak náš algoritmus vyberie náhodnú akciu, musí zároveň vyprázdniť zásobník s predchádzajúcimi stavmi.

Tréning prebiehal v epizódach ohraničených narazením robota do steny. Každú epizódu robot začínal na náhodnom mieste s náhodným otočením, aby boli všetky stavy navštívované s približne rovnakou pravdepodobnosťou. Ako riadiacu stratégiu som použil ε-greedy metódu, s ε = 0,2 pričom ε sa každou epizódou jemne znižovalo. Rýchlosť učenia (α) aj diskontný faktor (γ) som nastavil na 0,5.

3.4.3 Decision Tree algoritmus

Keďže tabuľkový Q-learning aj po viacerých vylepšeniach konvergoval pomaly a funkčná aproximácia s lineárnym aproximátorom divergovala, skúsil som alternatívnu metódu – dynamickú diskretizáciu stavového priestoru.

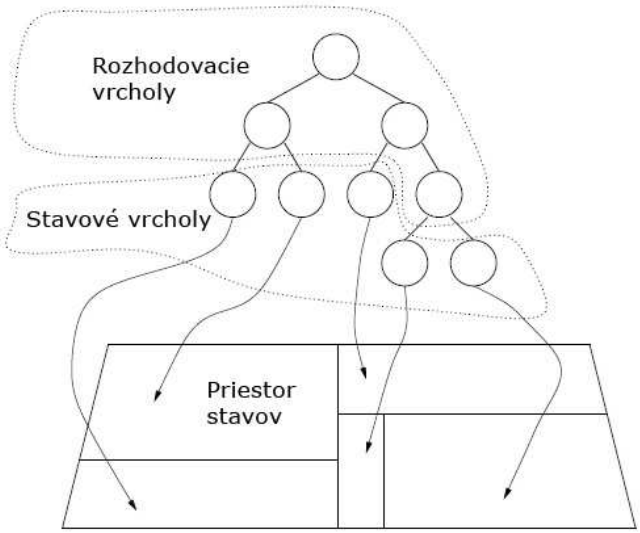
Pri uniformnej diskretizácii sa priestor stavov rozdelí na rovnako veľké časti, ktoré potom reprezentujú celú skupinu stavov s jednou hodnotou Q-funkcie. V tomto prístupe je potrebné dopredu určiť veľkosť jednej skupiny. Ak je príliš veľká, tak sa v nej môžu vyskytnúť stavy, ktoré k sebe nepatria (majú príliš rozdielnu optimálnu hodnotu Q-funkcie), ak je zase veľmi malá, diskretizácia stráca svoj zmysel, pretože týchto skupín stavov je stále veľmi veľa.

Existuje viac možností ako diskretizovať rozumnejšie (viď 2.2.1). V princípe sa všetky riešenia snažia diskretizovať neuniformne, teda s rôznou hustotou na rôznych miestach. Z tejto skupiny algoritmov som na úlohu sledovania chodby a

vyhýbaniu sa prekážok vybral Pyeattov a Howeov algoritmus [Pyeatt01] založený na rozhodovacom strome.

Rozhodovací strom je binárny strom, ktorý sa skladá z dvoch typov vrcholov – rozhodovacích a stavových. Stavové vrcholy sú listy stromu, ktoré obsahujú odhadovanú hodnotovú funkciu daného regiónu stavov. Keďže na učenie používame Q-learning, v každom stavovom vrchole udržiavame Q-funkciu pre každú akciu zvlášť. Tiež si v tomto liste ukladáme zoznam vstupných vektorov reprezentujúcich stavy, ktoré sme prešli a zoznam aktualizácií (updates) Q-funkcie pre tieto stavy. Aktualizácia Q-funkcie je definovaná ako

$$\Delta \mathrm{Q}(s_t, a_t) = \alpha[r_{t+1} + \gamma \max_a \mathrm{Q}(s_{t+1}, a) - \mathrm{Q}(s_t, a_t)] \quad (3.1)$$



Obrázok 3.6: Príklad rozhodovacieho stromu rozdeľujúceho priestor stavov na regióny variabilnej veľkosti

Rozhodovacie vrcholy sú vnútorné vrcholy stromu, ktoré rozdeľujú priestor stavov na regióny, vždy podľa jednej premennej z n-tice premenných reprezentujúcich stav (obr. 3.6).

V našom algoritme začíname so stromom s jedným vrcholom reprezentujúcim celý priestor stavov. Ako robot beží, prechádza rôznymi stavmi a získava spätnú väzbu v podobe odmeny, v liste sa tieto informácie uchovávajú až pokiaľ nedosiahne istú veľkosť. Vtedy prebehne test, či má byť rozdelený stavový vrchol. Ak áno, určia sa hranice delenia a vrchol sa zmení na rozhodovací vrchol s dvoma novými stavovými vrcholmi. Takto postupne strom narastá do hĺbky rozdeľujúc priestor stavov na menšie a menšie regióny.

Kľúčová otázka pri tomto prístupe je kedy a ako rozdeľovať stavové vrcholy. Na túto otázku náš algoritmus odpovedá pomocou štatistických metód. Vypočítame si priemer a smerodajnú odchýlku zoznamu aktualizácií. Ak je absolútna hodnota priemeru menšia ako dvojnásobok smerodajnej odchýlky, znamená to, že jednotlivé

vstupy vo vrchole reprezentujú stavy s veľkou diferenciou a treba ich rozdeliť do dvoch separátnych vrcholov. Za premennú podľa ktorej budeme rozdeľovať vyberieme tú, ktorá má najväčší rozptyl v zozname vstupných vektorov. Do nového rozhodovacieho vrchola vložíme strednú hodnotu tejto premennej a rozdelíme zoznam vstupných vektorov spolu so zoznamom aktualizácií podľa tejto hodnoty.

Algoritmus Decision Tree

1. V čase t prijmi vstupný vektor v_t reprezentujúci aktuálny stav
2. Pomocou vektora v_t nájdi v rozhodovacom strome list reprezentujúci stav s_t
3. S pravdepodobnosťou 1 - ε vyber akciu a s najväčšou hodnotou $Q(s_t,a)$, s pravdepodobnosťou ε vyber náhodnú akciu a
4. Vykonaj akciu a, prijmi odmenu r_{t+1} a vektor v_{t+1}
5. Vypočítaj aktualizáciu $\Delta Q(s_t,a)$ a s vektorom v_t ju vlož do vrchola reprezentujúceho stav s_t
6. Rozhodni či má byť stav s_t rozdelený do dvoch stavov
   (a) Ak dĺžka zoznamu vstupov je menšia ako minimálna dĺžka zoznamu vstupov, potom split := false
   (b) Inak
      i. vypočítaj priemer μ a smerodajnú odchýlku σ zoznamu aktualizácií
      ii. ak $|\mu| < 2\sigma$ potom split := true
      iii. inak split := false
7. Ak split, tak rozdeľ vrchol
8. Použi v_{t+1} ako vstup do ďalšieho cyklu algoritmu

Decision Tree algoritmus v úlohe sledovania chodby

Na reprezentáciu stavu sme tentokrát použili 5 meraní diaľkomera, ktoré sme si predpripravili podobne ako v predchádzajúcom algoritme, aby veľkosť stavového priestoru bola 20x40x80x40x20. Minimálnu dĺžku zoznamu vstupov sme po viacerých testovaniach nastavili na hodnotu 60. Stavový vrchol v rozhodovacom strome sa nerozdelil, skôr ako jeho zoznam vstupov nedosiahol túto dĺžku. Ostatné nastavenia a parametre boli také isté ako pri Q-learningu.

Q-funkcia pre jednotlivé regióny je uložená v listoch nie explicitne, ale ako zoznam aktualizácií, teda jednotlivých prírastkov. Ku každej aktualizácii sme si zapamätali aj akciu, na ktorú sa aktualizácia viaže. Pre výpočet konkrétnej hodnoty $Q(s,a)$ sa potom sčítali všetky aktualizácie vo vrchole reprezentujúcom stav s, ktoré sa viazali s akciou a. Kvôli výpočtovej efektívnosti sme si však túto hodnotu nechávali uloženú explicitne a v prípade pridania novej aktualizácie alebo rozdelenia vrcholu sme ju prepočítali.

Zaujímavé je aj pozorovanie, že výsledný strom býva pomerne dobre vyvážený. Vyplýva to zo spôsobu, ako rozdeľujeme vrcholy a tiež aj z toho, že priestor stavov robot z dlhodobého hľadiska prechádza rovnomerne. Vyváženosť stromu je dôležitá pre plynulý beh programu a výpočtovú zložitosť, ktorá je takto logaritmická od veľkosti stromu, a teda v konečnom dôsledku aj od veľkosti stavového priestoru.

Je dobré si uvedomiť, že diskretizácia je aproximáciou. Čiže nemusíme prejsť všetky stavy prostredia na to, aby sme vedeli dostatočne dobre odhadnúť hodnotovú funkciu. Na dobrú aproximáciu hodnotovej funkcie pre jeden konkrétny stav stačí, aby sme prešli dostatočne veľa „podobných“ stavov. Táto vlastnosť aproximácie má za následok niekedy až drasticky rýchlejšiu konvergenciu k optimálnej (alebo aspoň suboptimálnej) hodnotovej funkcii.

3.5 Testovanie

3.5.1 Metodika testovania

Pretože cieľom úlohy nie je nič viac než naučiť robota chodiť po chodbách bez narazenia do steny, miera úspešnosti sa stanovuje ľahko – počet „krokov“ robota pred narazením do steny. Nehľadeli sme pri tom na „kvalitu“, či robot chodí plynule stredom chodby, alebo či naopak chodí od steny ku stene; či nechodí len po jednom úseku hore-dole a pod. Z toho, ako bola definovaná odmena ani nevyplývalo, že by sa robot mal naučiť konkrétny štýl pohybu. Ničmenej, pri viacerých testovaniach sa ukazovalo, že kvalita chôdze rastie spolu s kvantitou.

Pretože robot pracuje v prostredí Microsoft Robotics Studio len ako súbor jednotlivých decentralizovaných služieb (diferenciálový pohon, laserový diaľkomer, nárazník), v ktorom je aj náš algoritmus len služba vytvárajúca partnerstvá s ostatnými, vznikajú nám isté nepravidelnosti. Dĺžka jedného kroku robota napríklad nie je konštantná, pretože pri akcii vpred posielame službe diferenciálneho pohonu iba správu, aby aktivoval obidve kolesá. Potom sa spustí countdown, ktorý zabezpečí, že ostaneme v danom stave nejakú dobu. Dĺžka tejto prestávky býva približne od 250ms do 300ms, úplne presne sa však nedá predpovedať, pretože je závislá na počte notifikácii v službe diaľkomera. Táto variabilita potom vedie k drobným rozdielom dĺžky krokov robota.

Iný príklad predstavujú rozdiely v meraniach diaľkomera. Keďže diaľkomer a pohon kolies pracujú navzájom asynchrónne, diaľkomer posiela dáta, ktoré nameral počas vykonávania aktuálnej akcie. Ak sa robot otáča doľava o 30 stupňov, diaľkomer môže svoje meranie vykonať jeden krát na dvadsiatich, inokedy na desiatich stupňoch.

Kvôli týmto nepravidelnostiam trasa robota nebýva nikdy úplne rovnaká ani keď štartuje z rovnakej počiatočnej pozície. Aby sme vedeli korektne vyhodnotiť úspešnosť robota, musíme ho spúšťať viackrát z tej istej pozície a brať do úvahy priemernú hodnotu počtu krokov jednotlivých pokusov.

Ďalším problémom, s ktorým sme sa pri testovaní stretli je „zasekávanie sa“

robota na jednom mieste, teda nekonečné oscilovanie akcií vľavo, vpravo. Pri trénovaní tento problém vyriešila ε-greedy stratégia, ktorá s nejakou malou pravdepodobnosťou z času na čas vykonala náhodnú akciu. Pri testovaní sme ale nemohli vyberať náhodné akcie, tie by nám značne poškodili korektnosť testov. Greedy stratégia, ktorou sme sa pri testovaní riadili, vždy vyberá akciu s najväčšou hodnotovou funkciou. A ak sme z niektorého stavu prešli vľavo do stavu, ktorý mal najväčšiu hodnotovú funkciu pre akciu vpravo (alebo obrátene), dostali sme sa do nekonečného cyklu. V niektorých prípadoch sa robotovi z tejto situácie pomohli dostať práve už spomínané nepravidelnosti. Avšak pri Decision Tree algoritme už drobné nepravidelnosti v meraniach nehrali rolu, pretože stavy s podobnými hodnotami boli väčšinou zoskupené v jednom regióne s rovnakou Q-funkciou. Preto sme museli greedy stratégiu modifikovať, a to zavedením pravidla, že ak je robot zacyklený, vynásobime veľkosť uhla otočenia náhodným číslom z intervalu <1,2>. Toto pravidlo v drvivej väčšine prípadov pomohlo dostať robota zo zacyklenia.

3.5.2 Oblasti testovania

Pri vyhodnocovaní výsledkov je potrebné vedieť, čo sa chceme z testovania dozvedieť. Dá sa očakávať, že „múdrejší“ algoritmus bude vykazovať lepšie výsledky, alebo že úspešnosť Q-learningu bude priamo úmerná počtu navštívených stavov. Viac ako tieto očakávané výsledky nás zaujímajú výsledky, ktoré nevieme dopredu len tak ľahko predpovedať. Preto sme si stanovili tieto dve oblasti, ktoré chceme otestovať a vyhodnotiť:

1. Porovnanie úspešnosti robota na pravouhlých a v okrúhlych chodbách
2. Otestovanie robota natrénovaného na úlohu sledovania chodieb v prostredí s prekážkami a robota natrénovaného na úlohu vyhýbaniu sa prekážok v protredí s chodbami.

3.6 Výsledky

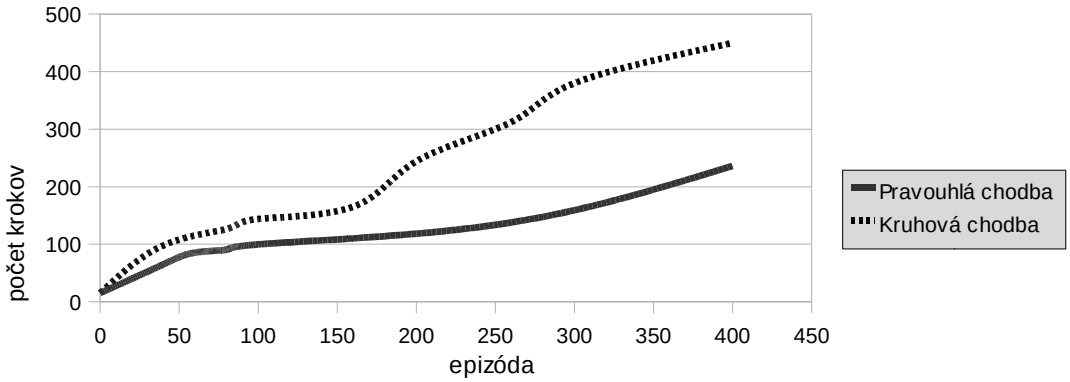
Výsledky po testovaní boli vcelku uspokojivé, miestami prekvapivé, a skoro vždy prinášali so sebou otázky o príčinách ich úspechu či neúspechu. Môžeme však skonštatovať, že vo všetkých testovacích prostrediach sa nám podarilo priblížiť sa

k optimálnej stratégii pre úlohu sledovania chodby.

3.6.1 Úspešnosť v závislosti na testovacom prostredí

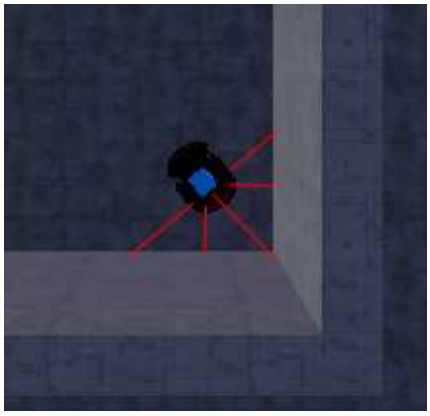
Decision Tree algoritmus

V pravouhlých chodbách sa musí robot naučiť chodiť rovno a v prípade potreby zatočiť o 90 stupňov. V oválnych chodbách zase robot musí zatáčať oveľa častejšie, ale o menší uhol. Keďže tréning robota prebiehal v prostredí pravouhlých chodieb, zdajú sa byť pri testovaní zvýhodnené. Ukazuje sa však, že to tak nemusí byť a robot naučený Decision Tree algoritmom podával lepšie výsledky v kruhovej



Obrázok 3.7: Graf úspešnosti robota v pravouhlej a kruhovej chodbe pri učení Decision Tree algoritmom.

chodbe ako v pravouhlej (obr. 3.7). Asi najrozhodujúcejším faktorom, ktorý znížil výsledky v pravouhlej chodbe sú samotné pravé uhly. Pri testovaní sme si totiž všimli, že robot má problémy s pravouhlými rohmi. Veľmi často sa stáva, že ak sa ocitne v blízkosti rohu, tak sa nakoniec vyberie priamo do jeho stredu a narazí. Tento fenomén sa dá vysvetliť tým, že senzor vpredu je aj v priebehu učenia najrozhodujúcejší, čo sa týka zisku trestu. Pri piatich senzoroch je meranie predného senzora väčšie ako jeho dvoch susedných a približne rovnaké ako dvoch bočných (obr. 3.8). Preto sa rozhodne ísť dopredu. Keď sa mu to stane na prvej zákrute, z potenciálnych 200 krokov robot spraví pribižne len 20.

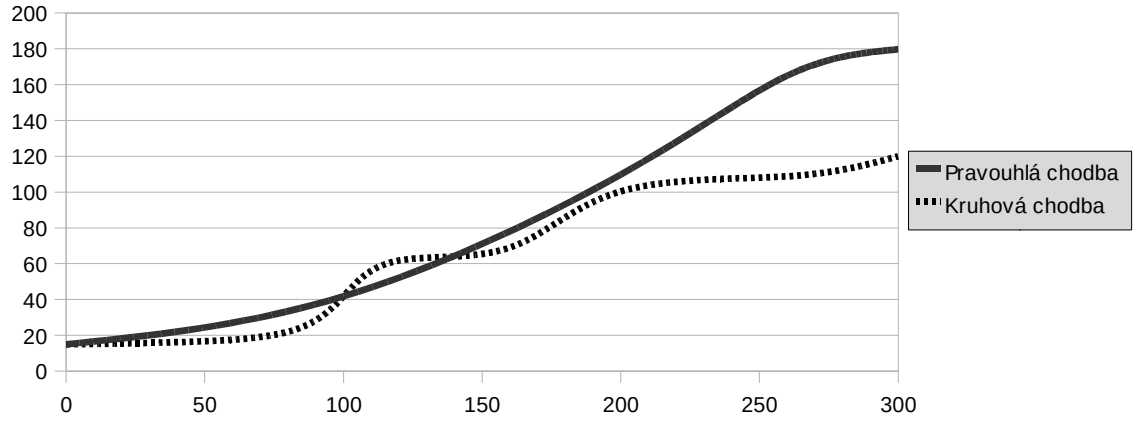


Obrázok 3.8: Situácia v rohu so znázornenými meraniami definujúcimi stav

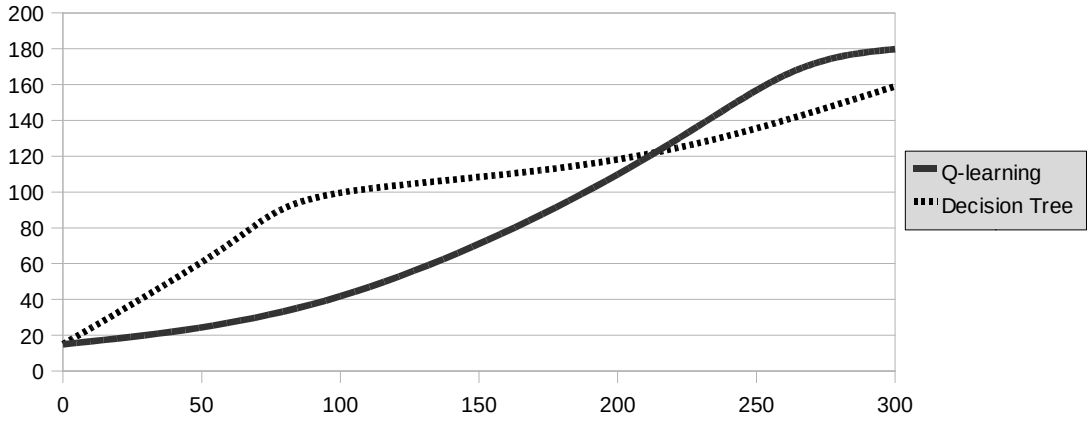
Niekoľko takýchto pokusov potom pokazí celkový testovací priemer. Tento jav sa dá priamočiaro zovšeobecniť tvrdením, že čím ostrejší uhol steny zvierajú, tým väčšia pravdepodobnosť, že robot do tohto rohu narazí. Z tohto tvrdenia potom ľahko vidíme, prečo podáva robot v kruhovej chodbe také dobré výsledky.

Q-learning

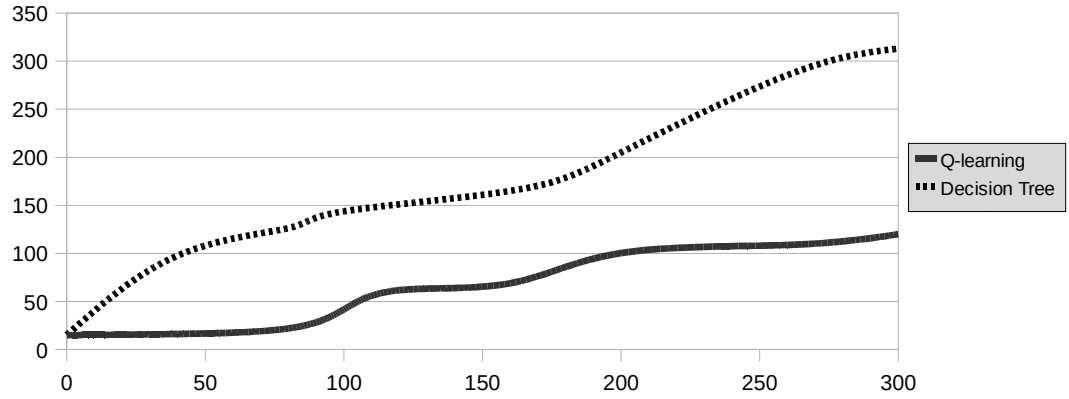
Pri Q-learningu sme však používali iba tri laserové merania, čiže problém s rohom robot nemal. A hoci sa ťarbavý tabuľkový Q-learning zlepšoval len veľmi pomaly, nakoniec v pravouhlej chodbe predstihol nielen svoje výsledky z kruhovej chodby (obr. 3.9), ale aj Decision Tree algoritmus z pravouhlej chodby (obr. 3.10). V kruhovej chodbe ostal víťazom Decision Tree algoritmus (obr. 3.11).



Obrázok 3.9: Graf úspešnosti robota v pravouhlej a kruhovej chodbe pri učení Q-learningom



Obrázok 3.10: Porovnanie úspešnosti Q-learningu a Decision Tree algoritmu v pravouhlej chodbe

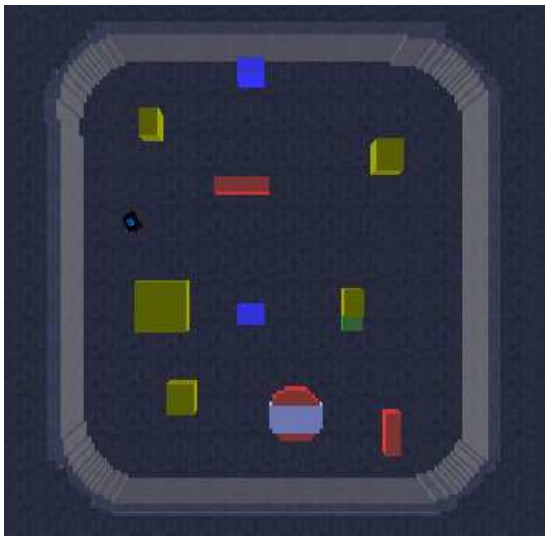


Obrázok 3.11: Porovnanie úspešnosti Q-learningu a Decision Tree algoritmu v kruhovej chodbe

3.6.2 Vyhýbanie sa prekážkam

Vyhýbanie sa prekážkam je obľúbená robotická úloha pre mobilné roboty. Existuje veľa elegantných algoritmov na túto úlohu. Čo nás ale zaujíma viac je súvis s úlohou sledovania chodby. Ako problém reinforcement learningu sú tieto úlohy takmer identické – robot dostáva trest za narazenie a odmenu za chodenie v dostatočnej vzdialenosti od akéhokoľvek objektu. Steny chodby teda môžu byť vnímané ako veľmi dlhé prekážky. Robot naučený vyhýbať sa prekážkam by sa teoreticky mal vedieť pohybovať aj po chodbe bez nárazu. Otázka je, či to platí aj opačne.

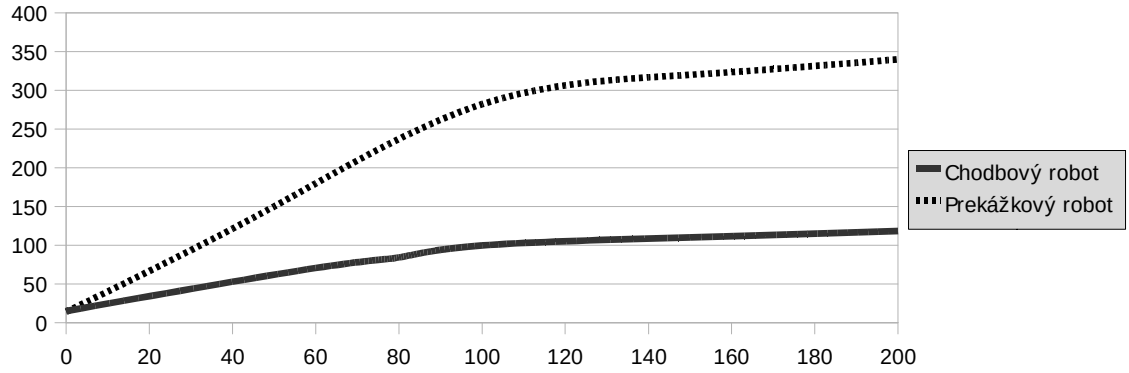
Na testovanie sme si vytvorili ďalšie prostredie bez chodieb, ale s prekážkami. Aby sme sa vyhli problému s pravouhlými rohami, zaoblili sme ich (obr. 3.12). Zobrali sme si dvoch robotov – jedného natrénovaného na sledovanie chodby a druhého na obchádzanie prekážok, otestovali sme ich najprv na svojej úlohe, potom na tej druhej a nakoniec sme porovnali úspešnosť obidvoch robotov v obidvoch prostrediach.



Obrázok 3.12: Prostredie pre testovanie úlohy vyhýbaniu sa prekážkam

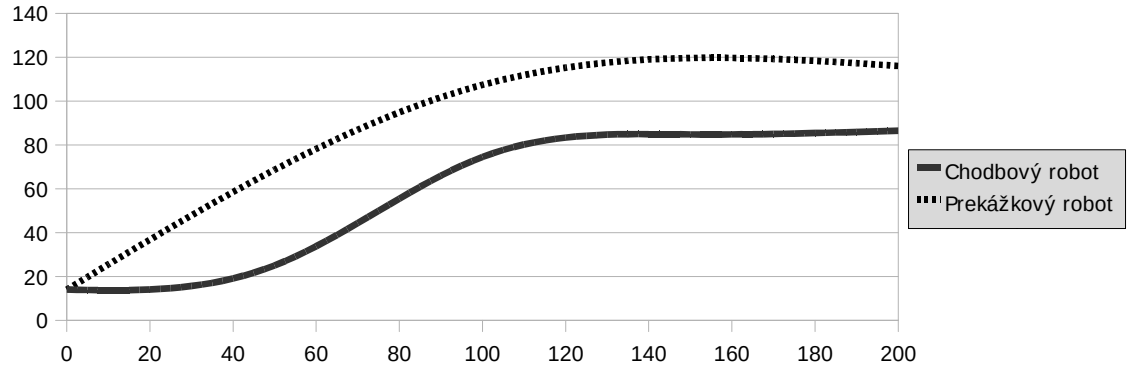
V tejto časti som na trénovanie používal už len Decision Tree algoritmus.

V prostredí s prekážkami sa ukázalo, že robot natrénovaný na sledovanie chodby síce dokáže obchádzať prekážky, avšak v celkovom hodnotení bol menej úspešný ako robot natrénovaný na vyhýbanie sa prekážkam (obr. 3.13). S časti za to môže ohraničená reprezentácia stavu – v úlohe sledovania chodby sme osekali laserové merania aby sedeli na dĺžku chodby, čo sa nám v prostredí s otvoreným priestorom vypomstilo a aproximácia bola nepresnejšia. Takisto sa v tomto prostredí mohol „chodbový“ robot stretnúť so stavmi, ktoré doteraz nenavštívil (napríklad, že predný a bočný laser boli nízke hodnoty – dve rôzne prekážky – a stredný laser bol vysoká hodnota – medzera medzi prekážkami).



Obrázok 3.13: Porovnanie úspešnosti robota trénovaného na úlohu sledovania chodby a robota trénovaného na úlohu vyhýbania sa prekážok v prostredí s prekážkami bez chodieb

V opačnom prípade ale „môj dom, môj hrad“ neplatilo. „Prekážkový“ robot dokázal sledovať chodbu bez väčších problémov, poradil si aj s pravouhlými rohmi a v celkovom hodnotení tak predbehol robota trénovaného špeciálne na sledovanie chodby (obr. 3.14).



Obrázok 3.14: Porovnanie úspešnosti robota trénovaného na úlohu sledovania chodby a robota trénovaného na úlohu vyhýbania sa prekážok v prostredí s chodbami bez prekážok

3.7 Diskusia

Voľba Decision Tree algoritmu bola len logickým pokračovaním v snahe vylepšovať algoritmus Q-learning. A hoci som v teoretickej časti uviedol príklady mnohých lepších algoritmov, ako sa ukázalo, na základné robotické úlohy stačia aj jednoduchšie algoritmy. V prostredí kruhovej chodby sa Decision Tree s päťrozmerným stavovým priestorom po niekoľkých stovkách epizód blížil k optimálnej stratégii a v niektorých pokusoch robot prešiel bez narazenia celý kruh aj 5-6 krát.

V pravouhlej chodbe sa ukazoval lepší Q-learning, ale to len vďaka tomu, že používal trojrozmernú reprezentáciu stavu, čo mu pomohlo v situáciach v rohu. Decision Tree algoritmus s trojrozmerným stavom som už nestihol natrénovať a otestovať, dá sa však predpokladať, že by predčil v tomto prostredí Q-learning.

Z týchto výsledkov celkom jasne vidno, že pri definovaní reinforement learning problému musíme mať nejaké znalosti o prostredí, pre ktoré robota trénujeme. Samozrejme, tieto znalosti najlepšie nadobudneme skúšaním a analýzou výsledkov jednotlivých algoritmov.

Z druhého experimentu s prekážkami vyplýva aký vzťah majú problémy vyhýbaniu sa prekážok a sledovania chodby. Tak, ako sme si tieto úlohy definovali v rámci tejto práci (čo nie je neštandartná definícia), je problém sledovania chodby iba špecifickým prípadom problému vyhýbania sa prekážok. To nás zase privádza k všeobecnejšej myšlienke, že niekedy riešenie jedného problému je efektívnejšie, ak riešime na prvý pohľad iný, ale v podstate všeobecnejší problém.

V prvých dvoch kapitolách sme sa pokúsili o prierez reinforcement learningom. Nebolo však možné zahrnúť všetky metódy reinforcement learningu. Nedostali sme sa veľmi k multiagentovému reinforcement learningu, tiež sme s priestorových príčin vynechali nový zaujímavý prístup k riešeniu viacerých podobných úloh – algoritmus SKILLS, kde sa robot snaží z naučených informácií extrahovať nejaké všeobecnejšie schopnosti, ktoré by sa dali použiť v iných úlohach. Podobne, ale v rámci jednej úlohy, sa snaží relačný reinforcement learning o hľadanie vzťahov medzi jednotlivými stavmi za účelom zlepšiť škálovateľnosť reinforcement learningu.

Záver

V tejto práci sme v stručnosti zhrnuli súčasne „know-how“ reinforcement learningu. Uviedli sme príklady úspešného použitia reinforcement learningu v robotických aplikáciach a na koniec sme sami jednu experimentálnu aplikáciu navrhli, naprogramovali, otestovali a vyhodnotili. Experiment pritom skončil úspešne a ponúkol nám niektoré nové pohľady na robotické úlohy.

Prínos tejto práce je predovšetkým v spracovaní veľkého množstva materiálov o reinforcement learningu do krátkeho prehľadu, ktorý v slovenčine dlhodobo chýba. Práca by mohla slúžiť ako vstupná brána do sveta reinforcement learningu pre ďalších záujemcov. S množstvom bibliografických odkazov je k tomu dobre pripravená.

Pokiaľ viem, aj moje experimenty sú v istom zmysle priekopnícke. Ich prínos je hlavne v poukázaní na spôsob navrhovania a programovania reinforcement learning algoritmov, ale tiež aj v niektorých inovatívnych modifikáciach v konkrétnych robotických úlohách.

Reinforcement learning je aj napriek mnohým úspechom niektorými odborníkmi pre svoju jednoduchosť odsúvaný do úzadia. Avšak súčasný reinforcement learning už nie je len tabuľka, MDP a dynamické programovanie. Je to celá škála algoritmov, heuristických metód, inovatívnych prístupov, s nádejnými výhliadkami do budúcnosti. Množstvo článkov a vedeckej práce za posledné dve desaťročia dokazuje, že myšlienka reinforcement learningu ešte stále fascinuje mnohých ľudí a asi to tak minimálne ešte jedno desaťročie ostane.

Zoznam použitej literatúry

[Abbeel07] ABBEEL, P. et al. 2007. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Advances in Neural Information Processing Systems,* vol. 19*,* MIT Press, 2007.

[Aberdeen02] ABERDEEN, D. – BAXTER, J. 2002. Scaling Internal-State Policy-Gradient Methods for POMDPs. In *Proc. ICML-02.*

[Amari98] AMARI, S. 1998. Natural Gradient Works Efficiently In Learning. *Neural Computation.* vol. 10, pp. 251 – 276.

[Bagnell01] BAGNELL, J. A. – SCHNEIDER, J. G. 2001. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proc. ICRA*, 2001, pp.1615-1620.

[Benbrahim97] BENBRAHIM, H. – FRANKLIN, J. A. 1997. *Biped dynamic walking using reinforcement learning*. Doctoral Dissertation.University of New Hampshire.

[Boyan95] BOYAN, J. A. – MOORE, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. In *Neural Information Processing Systems.* Vol. 7, MIT Press, 1995

[Crites98] CRITES, R. H. – BARTO, A. G. 1998. Elevator Group Control Using Multiple Reinforcement Learning Agents. In *Machine Learning*. pp.235-262

[Endo05] ENDO, G. et al. 2005. Learning CPG sensory feedback with policy gradient for biped locomotion for a full-body humanoid. In *The International Journal of Robotics Research*. vol. 27. no. 2, pp. 213 - 228

[Fidelman04] FIDELMAN, P. – STONE, P. 2004. Learning Ball Acquisition on a Physical Robot. In *ISRA 2004.*

[Geng05] GENG, T. – PORR, B. – WÖRGÖTTER, F. 2005. Fast biped walking with a reflexive controller and real-time policy searching. In *Advances in Neural Information Processing Systems,* vol. 18*,* MIT Press, 2006.

[Gullapali94] GULLAPALI, V. – FRANKLIN, J. A. – BENBRAHIM, H. 1994. Acquiring robot skills via reinforcement learning. In *Control Systems*, IEEE , vol.14, no.1, pp.13-24, Feb 1994.

[Hauskrecht00] HAUSKRECHT, M. 2000. Value-Function Approximations for Partially Observable Markov Decision Processes. In *Journal of Artificial Intelligence Research,* vol. 13, pp. 33-94, 2000.

[Johns08] JOHNS, K. – TAYLOR, T. 2008. *Professional Microsoft Robotics Developer Studio.* Birmingham, UK: Wrox Press Ltd., 2008, ISBN:0470141077 9780470141076

[Kaelbling96] KAELBLING, L. P. et al. 1996. Reinforcement Learning: A Survey. In *Journal of Artificial Intelligence Research,* vol. 4, pp.237-285, 1996.

[Kakade02] KAKADE, S. 2002. A Natural Policy Gradient. In *Neural Information Processing Systems*,

vol.15, pp.1531-1538

[Kearns99] KEARNS, M. – SINGH, S. 1999. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Neural Information Processing Systems,* vol. 12, pp. 996-1002.

[Kimura98] KIMURA, H. – KOBAYASHI, S. 1998. *Reinforcement Learning for Conti-nuous Action using Stochastic Gradient Ascent*.

[Kohl04] KOHL, N. – STONE, P. 2004. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Robotics and Automation*, *Proceedings. ICRA '04. 2004 IEEE International Conference on* , vol. 3, pp. 2619- 2624

[Konda00] KONDA, V. R. – TSITSIKLIS, J. N. 2000. Actor-Critic Algorithms. In *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143-1146, 2003.

[Littman94] LITTMAN, M. L. 1994. Memoryless policies: Theoretical limitations and practical results. In *Proc. Third Int. Conf. on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 238-245, MIT Press, 2004.

[Moore95] MOORE, A. W. 1995. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. In *Machine Learning*, vol. 21, no. 3, pp. 199-233

[Morimoto01] MORIMOTO, J. – DOYA, K. 2001. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. In *Robotics and Autonomous Systems,* vol. 36, no. 1, pp.37-51.

[Morimoto04] MORIMOTO, J. et al. 2004. A Simple Reinforcement Learning Algorithm For Biped Walking. In *Robotics and Automation*, *Proceedings. ICRA '04. 2004 IEEE International Conference on* , vol. 3, pp. 3030-3035.

[Ng00] NG, A. Y. – JORDAN, M. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. 16th Conf. Uncertainty Artif. Intell.,* p. 1.

[Ng03] NG, A. Y. et al. 2003. Autonomous helicopter flight via Reinforcement Learning. In *Advances in Neural Information Processing Systems*, vol. 16, MIT Press, 2004.

[Ng04] NG, A. Y. et al. 2004. Autonomous inverted helicopter flight via Reinforcement Learning. In *Proc. ISER, 2004*, pp.363-372.

[Papadimitrou87] PAPADIMITRIOU, CH. – TSITSIKLIS, J. N. 1987. The Complexity of Markov Decision Processes. In *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441-450.

[Peters03] PETERS, J. – VIJAYAKUMAR, S. – SCHAAL, S. 2003. Reinforcement Learning for Humanoid Robotics. In *Proc. IEEE-RAS International Conference on Humanoid Robots.*

[Peters06] PETERS, J. – SCHAAL, S. 2006. Policy Gradient Methods for Robotics. In *Intelligent Robots and Systems*, pp. 2219-2225.

[Peters08] PETERS, J. – SCHAAL, S. 2008. Natural Actor-Critic. In *Neurocomputing*, vol. 71, pp. 1180-

1190.

[Peters10] PETERS, J. – BAGNELL, J. A. 2010. Policy Gradient Methods. In *Springer Encyclopedia of Machine Learning*, ISBN 978-0-387-30768-8

[Pyeatt01] PYEATT, L. D. – HOWE, A. E. 2001. Decision Tree Function Approximation in Reinforcement Learning. In *Proc. 3rd International Symposium on Adaptive Systems*, pp. 70-77

[Schmidhuber97] SCHMIDHUBER, J. – ZHAO, J. – WIERING, M. 1997. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. In *Machine Learning*, vol. 28, no. 1, pp. 105-130.

[Singh94] SINGH, S. P. – JAAKKOLA, T. – JORDAN, M. 1994. Learning without state-estimation in partially observable Markovian decision processes. In *Proc. ICML*, 1994, pp.284-292.

[Singh00] SINGH, S. P. et al. 2000. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. In *Machine Learning*, vol. 39, no. 1, pp. 287-308.

[Smart02] SMART, W. D. 2002. *Making Reinforcement Learning Work on Real Robots*. Doctoral Disertation, 2002. Brown University Providence.

[Stone05] STONE, P. – SUTTON, R. S. – KUHLMANN, G. 2005. Reinforcement Learning for RoboCup-Soccer Keepaway. In *Adaptive Behavior,* vol. 13, no. 3, pp. 165–188.

[Sutton98] SUTTON, R. S. – BARTO, A. G. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, ISBN 978-0-262-19398-6.

[Sutton99] SUTTON, R. S. et al. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proc. NIPS*, 1999, pp.1057-1063.

[Sutton09] SUTTON, R. S. et al. 2009. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *Proc. NIPS*, 2009, pp.1204–1212.

[Terdrake05] TERDRAKE, R. – ZHANG, T. W. – SEUNG, H. S. 2005. Learning to Walk in 20 Minutes. In *Proc. Yale Workshop on Adaptive and Learning Systems*

[Tesauro95] TESAURO, G. J. 1995. Temporal Difference Learning and TD-Gammon. In *Commun. ACM*, vol. 38, no. 3, pp. 58-68.

[Tsitsiklis97] TSITSIKLIS, J. N. – VAN ROY, B. 1997. An analysis of temporal-difference learning with function approximation. In *Automatic Control,* vol.42, no.5, pp.674-690.

[Uther98] UTHER, W. T. B. – VELOSO, M. M. 1998. Tree Based Discretization for Continuous State Space Reinforcement Learning. In *Proc. Sixteenth National Conference on Artificial Intelligence (AAAI)*. Cambridge, MA: MIT Press, 1998.

[Watkins89] WATKINS, CH. J. C. H. 1989. Learning from Delayed Rewards. PhD thesis, University of Cambridge, Psychology Department.

[Watkins92] WATKINS, CH. J. C. H. – DAYAN, P. 1992. Q-learning. In *Machine Learning,* vol. 8, no. 3, pp. 279-292.

[Wiering97] WIERING, M. – SCHMIDHUBER, J. 1997. HQ-Learning. In *Adaptive behavior,* vol. 6, pp. 219-246.

[Williams92] WILLIAMS, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning,* vol. 8, no. 3, pp. 229-256.

[Whiteson06] WHITESON, S. – STONE, P. 2006. Evolutionary Function Approximation for Reinforcement Learning. In *Journal of Machine Learning Research,* vol. 7, pp. 877-917.

[Zhang95] ZHANG, W. – DIETTERICH, T. G. 1995. A Reinforcement Learning Approach to Job-Shop Scheduling. In *Proc. International Joint Conference on Artificial Intelligence*, pp. 1114-1120.