UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tom Vodopivec

SPODBUJEVALNO UČENJE NA PROBLEMU VOZIČKA S PALICO

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Branko Šter

Ljubljana, 2011

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01755/2011
Datum: 01.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOM VODOPIVEC**

Naslov: **SPODBUJEVALNO UČENJE NA PROBLEMU VOZIČKA S PALICO**
REINFORCEMENT LEARNING ON THE CART-POLE PROBLEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Na krmilnem problemu vozička s palico uporabite dve metodi spodbujevalnega učenja: Q-učenje in ASE/ACE. Opišite programsko implementacijo krmilnega problema z dodanim grafičnim vmesnikom in obeh metod. Ovrednotite metodi glede uspešnosti reševanja problema pri različnih parametrih.

Mentor:
prof. dr. Branko Šter

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Tom Vodopivec,

z vpisno številko 63050129,

sem avtor diplomskega dela z naslovom:

Spodbujevalno učenje na problemu vozička s palico.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Branka Štera
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 19.9.2011 Podpis avtorja:

Zahvala

Za ves trud, pomoč, potrpljenje in ker mi je bil vedno na razpolago, ko sem imel kakšno težavo, se zahvaljujem mentorju izr. prof. dr. Branku Šteru.

Predvsem velika hvala moji družini, ker so mi študij sploh omogočili in me v vseh teh letih popolnoma podpirali – hvala mama, tata in sestrica.

Hvala še vsem bližnjim iz »Stesk tja proti Pklu«, ker so tako lepo skrbeli zame v času pisanja diplome. Predvsem pa še ena velika zahvala gre moji Maši, ki je bila ves čas moj neizmeren vir energije in spodbude. Brez tebe bi bilo mnogo težje, hvala Maša!

KAZALO

Povzetek

Cilj diplomske naloge je bil implementacija dveh algoritmov spodbujevalnega učenja na računalniški simulaciji krmilnega problema ter njuno ovrednotenje. Spodbujevalno učenje je ena izmed vrst strojnega učenja. Združuje principe reševanja problemov dinamičnega programiranja ter nadzorovanega učenja. Kot preizkusni sistem je bil izbran krmilni problem vozička s palico, ki je na tem področju širše uveljavljen za namene vrednotenja algoritmov učenja. Izmed algoritmov spodbujevalnega učenja smo izbrali dva predstavnika metod učenja po časovnik razlikah. Ta skupina algoritmov združuje metode dinamičnega programiranja in metode Monte Carlo. Prvi je algoritem za Q-učenje, drugi pa algoritem učenja z asociativnim iskalnim elementom in adaptivnim kritični elementom, ki spada med algoritme akter-kritik. Za dosego cilja smo razvili orodje za izvajanje eksperimentalnih preizkusov s simulacijo učenja na testnem problemu. Osnovno vodilo pri implementaciji tega orodja je bila modularnost in splošna uporabnost. Definirali smo lastno metodo vrednotenja uspešnosti, s katero smo ovrednotili algoritma na zelo širokem naboru parametrov simulacije. Ocenili in primerjali smo še računsko zahtevnost izbranih algoritmov.

Ključne besede:

spodbujevalno učenje, krmilni sistem vozička s palico, ovrednotenje algoritmov, metoda vrednotenja uspešnosti

Abstract

The main goal of this thesis was the evaluation and implementation of two types of reinforcement learning algorithms on a computer-simulated control problem. Reinforcement learning is a branch of machine learning which combines principles of dynamic programming and supervised learning for problem solving. For the benchmark system we chose the cart-pole control problem as it is widely used in this field for testing the efficiency of learning algorithms. Out of the reinforcement learning methods we chose two algorithms for temporal difference learning. This type of learning uses methods of dynamic programming and Monte Carlo methods. The first chosen algorithm is Q-learning, the second is an actor-critic algorithm which is called learning by associative search element and adaptive critic element. In the purpose of achieving our goal, we developed a computer application for the experimental testing of the simulation of learning on a benchmark system. Our aim was to make this tool as modular and reusable as possible. We defined a different method of performance evaluation which was used to evaluate both learning algorithms on a wide set of simulation parameters. We also measured the computational performance of both algorithms.

Key words:

reinforcement learning, cart-pole control problem, algorithm evaluation, performance evaluation method

1 Uvod

Strojno učenje je ena izmed osrednjih vej računalniškega področja umetne inteligence. Cilj, ki ga skuša doseči, je zelo splošen – da bi se računalniški sistem znal iz danih podatkov sam naučiti nekega koristnega obnašanja. Zaradi splošnosti se že učinkovito uporablja na mnogih področjih, kot so npr. računalniška razpoznava govora, razpoznava pisave, iskanje povzročiteljev bolezni, napovedovanje borznih indeksov pri ekonomiki, krmiljenje avtomatskih linij in robotov v tovarnah, nadzorovanje letalskih sistemov, ipd. Znanje, ki ga strojno učenje skuša doseči, v groben delimo na razpoznavo zapletenih vzorcev in oblik, razlikovanje med posameznimi vzorci glede na skupne značilnosti in inteligentno odločanje. S slednjim se ukvarja veja strojnega učenja, ki jo imenujemo spodbujevalno učenje.

Področje spodbujevalnega učenja je deležno veliko pozornosti s strani strokovnjakov, ker je eno izmed bolj splošnih oblik učenja in tako omogoča reševanje širokega spektra problemov. Aplikativna uporaba zajema npr. igranje iger, upravljanje letenje helikopterja, razvrščanje premikov dvigal, pa vse do ciljnega marketinga in robotskega igranja nogometa. Eden izmed odmevnih dosežkov spodbujevalnega učenja je bilo računalniško samostojno odkritje svetovno najboljše strategije igranja igre backgammon, ki je obrnila na glavo več stoletij razvijajoče se človeške strategije.

Kljub vsemu je aplikativna uporaba teh metod še vedno nizka v primerjavi s teoretičnimi dosežki. Drugače povedano, marsikaj bi lahko že dosegli, če bi vedeli, kaj kje uporabiti. Kot pri vseh prehodih iz teorije v prakso, je tukaj vrzel, ki jo skušamo zapolniti z eksperimentalnim preizkušanjem obstoječih metod pri reševanju določenih problemov. Naša raziskava se tako osredotoča na dve metodi spodbujevalnega učenja, ki ju bomo skušali ovrednotiti z vidika uspešnosti reševanja praktičnega problema. Na tak način upamo, da bomo dosegli ugotovitve, ki bi lahko služile kot izhodiščne točke za nadaljnje raziskave ali celo praktično uporabo. Za dosego tega cilja bomo skušali razviti uporabno računalniško orodje, ki bi v prihodnje omogočalo lažje opravljanje tovrstnih raziskav.

V nadaljevanju diplomskega dela bomo v drugem poglavju podali osnove spodbujevalnega učenja. Opisali bomo standardni model, naredili kratko primerjavo med glavnimi metodami strojnega učenja ter obrazložili način reševanja problemov. Navedli bomo osnovne metode spodbujevalnega učenja ter prikazali ključne razlike med njimi. V tretjem poglavju bomo podrobneje opisali dve metodi, ki smo ju izbrali

za implementacijo in ovrednotenje. Opisali bomo izbran problem reševanja ter podali nekatere matematične definicije in enačbe, ki so nujno potrebne za razumevanje in implementacijo. V četrtem poglavju bomo opisali implementacijo spodbujevalnega učenja v okolju. V petem poglavju bomo navedli podrobnosti simulacije omenjenega modela ter opisali razvoj celotnega orodja za opravljanje eksperimentalnih preizkusov. V šestem poglavju bomo obrazložili razvito metodologijo vrednotenja uspešnosti algoritmov učenja ter jo predstavili na praktičnih primerih. V sedmem poglavju so navedene ugotovitve o časovni in pomnilniški zahtevnosti implementiranih algoritmov. V predzadnjem poglavju pa so opisani rezultati eksperimentalnih meritev uspešnosti algoritmov pri različnih vrednostih dejavnikov okolja in parametrov učnih metod. Delo bomo zaključili z ovrednotenjem naše metodologije, algoritmov in razvitega orodja ter navedli smernice za nadaljnje delo.

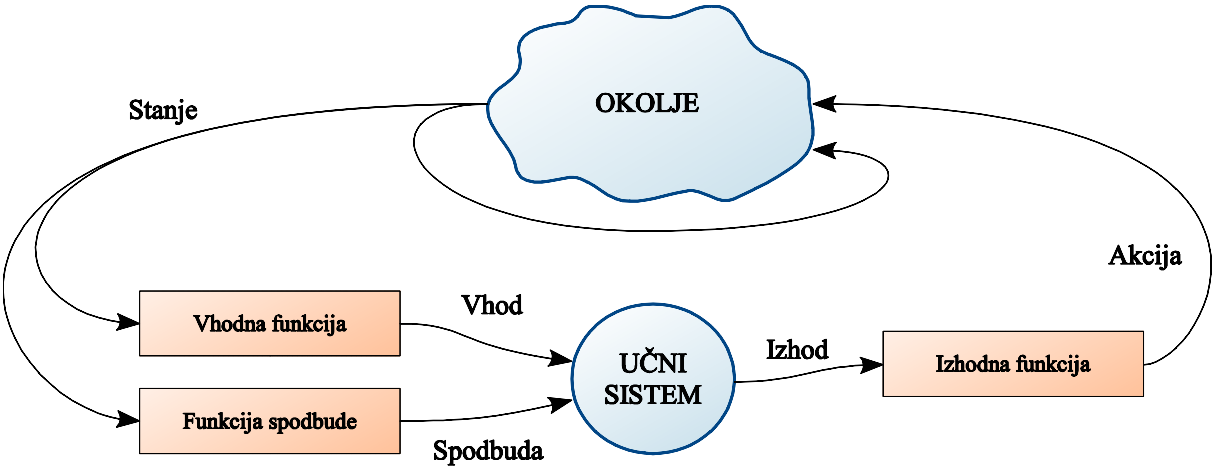
2 Spodbujevalno učenje

Prvi opis spodbujevalnega učenja lahko podamo z enostavnim primerom vožnje kolesa [10]. Sistemu spodbujevalnega učenja zadamo nalogo, naj vozi kolo ter mu naročimo le, naj ne pade na tla. Sistem začne voziti kolo tako, da naključno zavija levo in desno. Znajde se v primeru, ko je kolo nagnjeno za 45° v eno stran. Takrat izbere, da bo zavil v levo. Takoj za tem pade na tla. Nauči se, da v primeru nagnjenosti za 45° v to stran ne sme zaviti levo. Nato poskusi še enkrat. Začne voziti kolo in spet se znajde v enakem primeru, ko je kolo nagnjeno za 45° v isto stran. Takrat se odloči, da bo zavil desno. Takoj za tem se ravno tako zvrne na tla. Nauči se, da v tem primeru tudi zavoja v desno ne sme narediti, da torej karkoli naredi v primeru nagnjenosti kolesa za 45° v to stran ni dobro in zato sklepa, da že stanje nagnjenosti za 45° v to stran ni dobro in s tem tudi vsak zavoj, ki neposredno pripelje v to stanje. Tako pri naslednjem poskusu pride do stanja nagnjenosti za 40° v isto stran ter se odloči zaviti v levo. Takoj za tem spet preide v stanje nagnjenosti 45° in se nauči, da zavoj v levo pri 40° nagnjenosti ni dobra izbira – že prej, kot je dejansko padel na tla. Ob dovolj velikem številu **poskusov** se bo po teh principih naposled le naučil voziti kolo brez padca na tla.

Cilj spodbujevalnega učenja je torej naučiti neko entiteto (stvar), kako se *dobro* obnašati v nekem **okolju**. Kaj je *dobro*, ji pove okolje v obliki **kazni** ali **nagrade**. V navedenem primeru smo ji naročili, naj ne pade na tla, torej je ob vsakem padcu prejela kazen, ki ji je dala vedeti, da se mora tega izogibati. Tak način učenja temelji na načelih behavioristične psihologije, ki pravi, da karkoli živa bitja delajo, mislijo ter čutijo, lahko smatramo kot vzorce obnašanja, ki jih lahko natančno definiramo. Spodbujevalno učenje je torej metoda, ki skuša določiti vzorce obnašanja na neki entiteti, z učenjem na opisan način. Vsako entiteto ali sistem, ki se nekaj uči, imenujemo **učni sistem.** Kazen in nagrado pa negativna in pozitivna **spodbuda**. Od tod izvira ime spodbujevalnega učenja. Povzamemo lahko še drugače: spodbuda je odraz, kako dobro okolje sprejema obnašanje učnega sistema, učni sistem pa s poskušanjem spreminja svoje obnašanje tako, da bi bila spodbuda čim boljša (čim več nagrad in čim manj kazni).

2.1 Model spodbujevalnega učenja

V standardnem modelu spodbujevalnega učenja nastopa interakcija med učnim sistemom in okoljem, kot prikazuje shema na sliki 2.1 [13]. Ta interakcija poteka v obliki zaznavanja **stanja** okolja s strani učnega sistema ter izbire **akcije**, ki naj se v okolju izvede. Izbrana akcija spreminja okolje in ta sprememba se povrne v obliki spodbude za učni sistem. Funkcija spodbude skrbi za ustrezno interpretacijo trenutnega stanja okolja v vrednost spodbude (npr. ko je kolo zvrnjeno na tla, sporoči kazen). Izhodna funkcija preslikuje izhod učnega sistema v dejansko akcijo za okolje (npr. izbrana je bila prva akcija, kar pomeni zavij desno). Vhodna funkcija določa spoznavnost okolja (možnost opazovanja), ki je lahko delna ali polna.



Slika 2.1. Standardni model spodbujevalnega učenja.

2.2 Primerjava z drugimi učenji

Spodbujevalno učenje uvrščamo med glavne metode strojnega učenja, ki je osrednja veja področja umetne inteligence. Drugi dve glavni metodi strojnega učenja sta nadzorovano ter nenadzorovano učenje [1]. Te tri metode v osnovi ločimo glede na to, **kaj** naj se naučijo, torej po cilju učenja. V nadaljevanju podajamo kratek opis vseh treh, v tabeli 2.1 pa so prikazane ključne razlike med njimi.

Nadzorovano učenje

Nadzorovano učenje imenujemo tudi učenje z učiteljem. V tem primeru učni sistem učimo na množici vprašanj z ustreznimi odgovori (na parih vhodno-izhodnih vzorcev) z namenom, da se nauči pravilno odgovoriti tudi na vprašanja, ki jih še ni videl. Primer je optično prepoznavanje znakov: sistem učimo na veliki množici slik črke A ter

mu povemo, da je to črka A; naučiti pa se mora prepoznavati to črko tudi na tistih slikah, ki jih še ni videl. Algoritem nadzorovanega učenja mora torej znati izluščiti zakonitosti pri parih vhodno-izhodnih vzorcev, ki jih ima na razpolago, ter jih čim bolj posplošiti na poljuben vhodni vzorec. Povratno informacijo predstavlja napaka, ki je enaka razliki med napovedano vrednostjo in pravo. Z velikostjo napake merimo uspešnost nadzorovanega učenja. Ta vrsta učenja se uporablja za reševanje problemov klasifikacije – razvrščanja vzorcev v kategorije, ter regresije – iskanja relacij med spremenljivkami z atributi.

Nenadzorovano učenje

Nenadzorovano učenje imenujemo tudi učenje brez učitelja. Na razpolago imamo množico vzorcev brez nikakršnih označb. Tako imamo le množico vrednosti, v katerih skušamo najti ter povzeti ključne prvine – izluščiti informacijo. Pojem napake ali kazni ne obstaja, zato tudi ni povratne informacije ter ni uspešnosti. Uporablja se pri iskanju avtokorelacije na množici vzorcev – iskanje skritih struktur ter pravil, in pri razvrščanju vzorcev v gruče (ang. clustering – naj opozorimo, da to ni enako razvrščanju v kategorije [11]). Take metode pridejo v poštev na področju statistike ter podatkovnega rudarjenja.

Spodbujevalno učenje

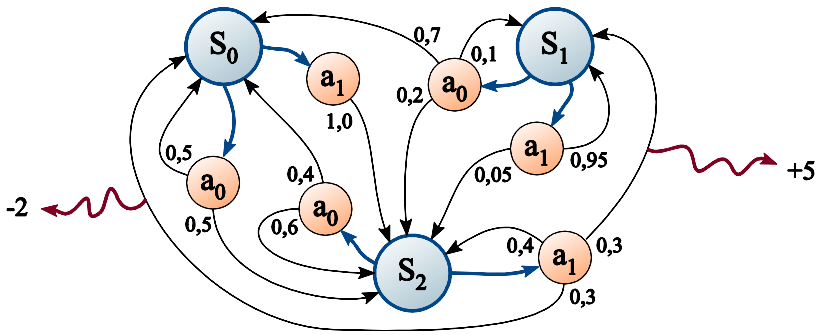
Pri spodbujevalnem učenju nimamo na razpolago učnih vzorcev, temveč samo interakcijo med učnim sistemom in okoljem v obliki izbrane akcije ter povrnjene spodbude, kot že prikazano. Ker je cilj učnega sistema pridobiti čim večjo vrednost spodbude po metodi poskusa in napake (ang. trial and error), med procesom učenja vedno išče ravnovesje med izkoriščanjem do zdaj pridobljenega znanja (da se obnaša tako, da je najbolj nagrajen) in raziskovanjem prostora (da bi se naučil nekaj novega, da bi bil potem še bolj nagrajen). Zaradi narave tega učenja pogosto merimo uspešnost sočasno s postopkom učenja. To metodo uporabljamo v primerih, ko poznamo model okolja, ampak je analitična rešitev nedostopna, in takrat, ko ne poznamo modela okolja. Uporablja se predvsem na področjih krmiljenja (teorije regulacij) in problemih optimizacije.

Tabela 2.1. Vrste strojnega učenja.

Metoda strojnega učenja	Temelj učenja	Cilj učenja	Merjenje uspešnosti	Uporaba
Nadzorovano	Množica parov vhodno-izhodnih vzorcev	Iskanje odvisnosti med vhodno ter izhodno množico vzorcev	Statistično po učenju	Klasifikacija (razpoznavanje vzorcev), regresija
Nenadzorovano	Množica vzorcev	Izluščiti informacijo iz množice vzorcev	Ni	Avtokorelacija, razvrščanje v gruče
Spodbujevalno	Interakcija z okoljem	Se naučiti obnašanja, ki bo najbolje sprejeto v okolju	Sočasno z učenjem	Ko je model okolja neznan ali analitična rešitev ni dostopna

2.3 Okolje in funkcija spodbude

Okolje je dinamičen sistem, ki ga lahko modeliramo z Markovskimi procesi. Iz tega razloga, po formalni definiciji, učni sistem spodbujevalnega učenja rešuje Markovski odločitveni problem [6]. Ta vsebuje štiri komponente: stanja, akcije, verjetnosti prehajanja med stanji ter vrednosti spodbud. Primer je prikazan na sliki 2.2. Vidimo, da obstajajo preslikave iz parov stanj in akcij v spodbude. Te preslikave določa **funkcija spodbude**, ki glede na trenutno stanje in izbrano akcijo vrača učnemu sistemu spodbudo v obliki skalarne vrednosti. Učni sistem, ki opazuje stanja, prejema spodbude ter izvaja akcije, mora za dosego čim večje vrednosti spodbude sam ugotoviti (se naučiti) odvisnosti med stanji, akcijami in spodbudami. Iz napisanega je razvidno, da funkcija spodbude definira cilj učnega sistema [10].



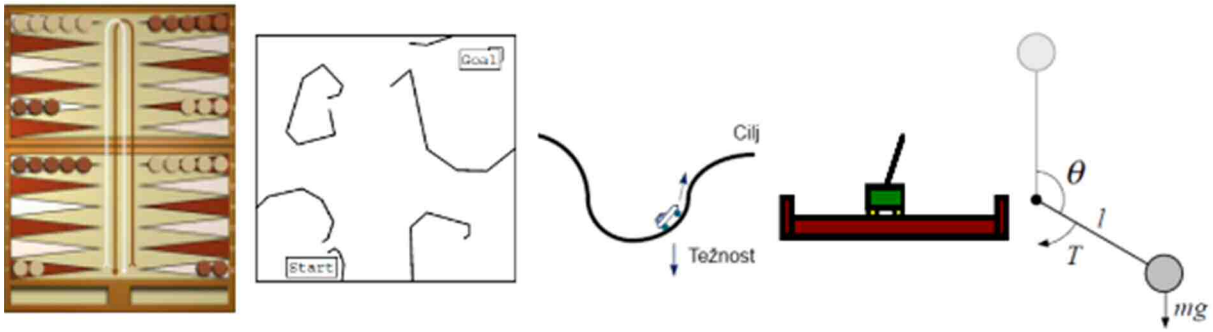
Slika 2.2. Primer markovskega procesa.

Vrste problemov

Glede na funkcijo spodbude lahko ločimo nekatere tipične probleme, ki jih rešuje učni sistem. Primeri so prikazani na sliki 2.3, od leve proti desni: igra backgammon, iskanje najkrajše poti, avto na hribu (opisan v nadaljevanju), voziček s palico, inverzno nihalo. Ena vrsta problemov so *problemi čiste zakasnjene spodbude*. Tukaj je spodbuda v vseh stanjih enaka 0, razen v nekem končnem stanju. Če je takrat spodbuda pozitivna, je to cilj, če je negativna, pa je stanje, ki se ga je potrebno izogibati. Taka primera sta igranje igre backgammon ter krmilni problem vozička s palico, ki bo podrobneje opisan v podpoglavju 3.3.

Še ena vrsta problemov, ki jo omenimo, so t.i. problemi *najkrajše poti do cilja* (ang. shortest path). V tem primeru je spodbuda v vsakem stanju enaka -1 (kazen), le v končnem stanju je 0. Učni sistem skuša dobiti čim več spodbude (čim manj kazni), zato bo skušal najti najkrajšo pot do končnega stanja. Primer je t.i. problem avta na hribu (ang. car on the hill problem [10]). Avtomobil je postavljen v luknjo med dve strmini in mora doseči vrh ene od strmin, vendar nima dovolj navora, da bi lahko prišel na vrh samo s potiskom naprej, temveč mora izkoristiti zalet z druge strmine. Učni sistem ima informacijo o poziciji in hitrosti avta ter izbira med akcijami pospeševanje naprej, nazaj ali nikamor.

Omenimo, da obstajajo še druge vrste problemov, pri katerih ni vedno nujno, da učni sistem išče največje vrednosti spodbud, lahko išče tudi najmanjše ali neke vmesne vrednosti [10].



Slika 2.3. Primeri problemov, ki se jih tipično rešuje s spodbujevalnim učenjem.

2.4 Vrednostna funkcija

Do zdaj smo povedali, na kakšen način poteka interakcija med učnim sistemom in okoljem ter da učni sistem stremi k čim večji vsoti spodbude. Nismo pa povedali, kako naj se nauči izbirati *dobre* akcije. Z vidika učnega sistema nismo še definirali,

kaj *dobra* akcija sploh je. Tako vpeljemo pojem **strategije** izbire akcij ter **vrednosti** stanja. Strategija določa, katera akcija naj bo izbrana v vsakem posameznem stanju – torej je preslikava iz stanj v akcije, na osnovi vrednosti stanja. Vrednost nekega stanja je definirana kot vsota vseh dobljenih spodbud, če bi v tem stanju začeli in sledili neki strategiji do zadnjega stanja. Optimalna strategija je torej taka preslikava iz stanj v akcije, ki ima največjo vsoto spodbud pri poljubnem začetnem stanju in pri izvajanju akcij po tej strategiji do končnega stanja [10]. Po tej definiciji je vrednost stanja neposredno pogojena s strategijo. **Vrednostna funkcija** tako določa vrednosti za vsako posamezno stanje – je preslikava iz stanj v vrednosti stanj. Na začetku učenja ni optimalna in tako pridemo do ključnega problema spodbujevalnega učenja, ki se glasi: kako razviti algoritem, ki učinkovito poišče optimalno vrednostno funkcijo. Za iskanje približka si lahko pomagamo s poljubnim funkcijskim aproksimatorjem (npr. več-nivojskim perceptronom, preslikovalno tabelo, ipd.) [10]. Ko so znane optimalne vrednosti stanj, lahko iz njih strategijo razberemo na enostaven način.

2.5 Iskanje približka vrednostne funkcije in osnovne metode

Učni sistem izvede akcijo v nekem trenutku, ki lahko povzroči določeno spodbudo šele po daljšem času. V tem času izvede še določeno število akcij, torej kaj naj se nauči oz. na kakšen način naj ugotovi, katera akcija je bila kriva za takšno spodbudo? To je osnovno vprašanje, ki ga rešujejo vse metode spodbujevalnega učenja. V nadaljevanju bomo te metode samo na kratko povzeli, podroben opis pa presega namen tega dela.

Osnovne algoritme spodbujevalnega učenja, ki jih bomo mi omenili, razvrščamo v dve skupini, kot prikazuje tabela 2.2. Prva skupina je **dinamično programiranje**, ki je dobro razvita formalna matematična teorija raziskovanja enostavnih problemov. Spreminjanje približkov vrednosti poteka po iteracijah na podlagi drugih že približanih (ocenjenih) vrednosti tako, da ni potrebno čakanje na končni izid. Dobljena rešitev problema je optimalna. Težava je v tem, da zahteva poznavanje modela (verjetnosti prehajanja stanj ter vrednosti pričakovanih spodbud) in ni praktično uporabna pri zelo velikih ali zveznih prostorih stanj. V osnovi sem štejemo dve vrsti algoritmov: **iteracijo vrednosti**, kjer iščemo vrednostno funkcijo in nato iz nje izluščimo strategijo izbire akcij, ter **iteracijo strategije**, kjer neposredno iščemo strategijo.

Druga skupina metod je **učenje po časovnih razlikah**, ki je kombinacija pristopov Monte Carlo in dinamičnega programiranja [19]. Lastnosti Monte Carlo se odražajo v tem, da lahko učenje poteka že samo na podlagi izkušenj, dobljenih iz interakcije z okoljem, brez poznavanja modela okolja. Iz dinamičnega programiranja pa uporablja prej omenjeno prednost, da za spreminjanje približkov vrednosti ni potrebno čakanje na končni izid. S to metodo dobimo skoraj-optimalno rešitev problema. Ker ni zahteve po poznavanju modela okolja, je uporabna tudi pri kompleksnih problemih z zveznimi stanji ali velikim številom stanj. Sem uvrščamo algoritem **Q-učenja**, ki se zgleduje po iteraciji vrednosti in ga nadalje opišemo v podpoglavju 3.1, ter **učenje akter-kritik**, ki je primerljivo z iteracijo strategije ter ga opišemo v podpoglavju 3.2.

Tabela 2.2. Osnovne metode in algoritmi spodbujevalnega učenja.

Metoda	Algoritmi	Optimalna rešitev	Uporaba pri kompleksnih problemih
Dinamično programiranje	Iteracija vrednosti Iteracija strategije	Da	Ne
Učenje po časovnih razlikah	Q-učenje Učenje akter-kritik	Ne	Da

Dinamično programiranje

Matematično področje dinamičnega programiranja se zgleduje po dveh osnovnih principih, s katerima lahko najdemo odgovor na vprašanje, postavljeno na začetku tega podpoglavja. Ta dva principa sta bila že prikazana na primeru vožnje s kolesom (na začetku poglavja 2). Prvi pravi, da če v nekem stanju ob neki akciji dobimo npr. kazen, se je v prihodnosti bolje izogibati tej akciji v danem stanju. Drugi princip pa pravi, da če vse možne akcije v nekem stanju vodijo v kazen, potem je to stanje slabo in se mu je bolje izogibati. Učni sistem se bo tako naučil, da je vsaka akcija, ki v prihodnosti pripelje v to stanje, slaba in se ji bo tudi izogibal (v primeru nagrade je seveda ravno obratno) [10]. Tako se pri vsakem koraku informacija o dobrih ter slabih stanjih in akcijah širi na sosednja stanja in akcije – širi se po prostoru stanj in akcij. Tem korakom navadno pravimo iteracije.

Na začetku je približek optimalne vrednostne funkcije slab. Z drugimi besedami, preslikave iz stanj v vrednosti stanj niso veljavne. Glavni cilj je tako, da dobimo pravo preslikavo. Ko to dosežemo, lahko enostavno izluščimo optimalno strategijo.

Relacijo med spodbudo $r(x_t)$ v trenutnem stanju okolja x_t, trenutnim približkom optimalne vrednostne funkcije $V(x_t)$ ter približkom optimalne vrednostne funkcije ob naslednjem koraku $V(x_{t+1})$ podaja enačba (2.1), ki se imenuje Bellmanova enačba [10]:

$$V(x_t) = r(x_t) + \gamma V(x_{t+1}) \tag{2.1}$$

Proces učenja predstavlja iskanje njene rešitve. Vrednost γ (gama) **je faktor zmanjševanja teže prihodnjih spodbud** na intervalu [0,1]. Če je gama 0, je vednost posameznega stanja definirana samo s takojšnjo spodbudo ob izvedbi neke akcije iz tega stanja – to pomeni, da bi bil učni sistem sposoben reševati samo popolnoma trenutne probleme. Bližji kot je 1, večja je teža prihodnjih spodbud in načeloma tudi težji je problem – več časa potrebujemo, da dobimo povratno informacijo o spodbudi. Če je končno število stanj znano, ne potrebujemo zmanjševati teže prihodnjih spodbud ($\gamma = 1$). V primeru, da je število stanj neznano ali neskončno, pa mora biti njegova vrednost nujno manjša od 1, da zagotovimo konvergenco k optimalnim vrednostim stanj sicer bi bila vsota spodbud za vsako stanje neskončna.

2.6 Raziskovanje prostora in izkoriščanje znanja

Pri enačbi (2.1) mora, po definiciji, optimalna vrednostna funkcija zadoščati celotnemu prostoru stanj. To pomeni, da če želimo dobiti optimalno rešitev, moramo določiti optimalne vrednosti za vsa stanja. Proces učenja se premika skozi prostor stanj ter določa približke optimalnih vrednosti obiskanih stanj po korakih. Z dovolj velikim številom korakov lahko na tak način ugotovi optimalne vrednosti obiskanih stanj. Toda, če določenih stanj nikoli ni obiskal, ne more podati približka njihove optimalne vrednosti, zato začetna trditev tega odstavka ne more biti izpolnjena. Iz tega razloga moramo v učni proces vpeljati pojem **raziskovanja**. Raziskovanje je definirano kot namerna izbira akcije, za katero učni sistem meni, da ni najboljša, z namenom dobiti znanje neobiskanih ali malo obiskanih stanj. Prostor stanj mora torej biti dovolj raziskan, če hočemo dobiti optimalno ali skoraj-optimalno rešitev.

Iz povedanega sledi, da vedno iščemo kompromis med izbiro trenutnih najboljših vrednosti – izkoriščanju znanja, in iskanju še boljših vrednosti tako, da raziskujemo prostor. Zaradi tega razloga je pomembno, da uporabljamo take metode raziskovanja prostora, ki prinesejo največ novega znanja za čim manjšo ceno – čim več dolgoročne nagrade za ceno čim manj kratkoročne kazni.

3 Izbrane metode in okolje

V našem delu smo se pri izbiri metod zgledovali po članku Barta in sod. [2]. Implementirali smo algoritme učenja z asociativnim iskalnim elementom in adaptivnim kritičnim elementom (ang. associative search element / adaptive critic element – **ASE/ACE**), kot opisano v tem članku. To je metoda akter-kritik znotraj učenja po časovnih razlikah. Za preizkusni problem smo izbrali model vozička s palico. Ta krmilni problem se obširno uporablja pri vrednotenju uspešnosti algoritmov učenja. Uporaben je zaradi svoje podobnosti z marsikaterim realnim problemom (tako je na primer bil uporabljen kot prvotni preizkusni sistem za nadaljnjo implementacijo sistema za stabilizacijo podmornice [3]).

Kot drugi algoritem spodbujevalnega učenja smo izbrali algoritem Q-učenja, ki posnema iteracijo vrednosti z učenjem po časovnih razlikah. Pri obeh algoritmih učenja uporabljamo kot funkcijski aproksimator preslikovalno tabelo ter diskretiziramo prostor po metodi *Boxes* [2]. Omogočeno imamo polno spoznavnost okolja. Pri krmilnem problemu uporabljamo dvopoložajni krmilnik (ang. bang-bang controller [21]). To pomeni, da ima učni sistem v vsakem stanju na izbiro dve akciji.

3.1 Q-Učenje

Iteracija vrednosti

Metoda iteracije vrednosti popravlja približke optimalne vrednosti stanj po iteracijah, dokler se vrednosti stanj ne spreminjajo več. Takrat so dosežene optimalne vrednosti in s tem je najdena optimalna vrednostna funkcija. Izvedbo ene iteracije bomo v nadaljevanju imenovali tudi **korak učenja**. Opisane izračune opravlja z enačbo (3.1), ki je izpeljana iz enačbe (2.1) ob upoštevanju, da je uporabljen funkcijski aproksimator preslikovalna tabela [10]:

$$\Delta w_t = \max_a \big(r(x_t, a) + \gamma V(x_{t+1})\big) - V(x_t) \tag{3.1}$$

Ob vsaki iteraciji se izračunajo spremembe parametrov Δw_t za vsa stanja, pri čemer je a izvedena akcija v stanju x_t in $r(x_t, a)$ je prejeta spodbuda ob izvedbi te akcije v tem stanju. Navedeno je mogoče samo na tak način, da izvedemo v enem koraku vse

akcije v vsakem stanju, kar pa ni možno brez točnega poznavanja modela ali na realnem problemu, kot že navedeno v podpoglavju 2.5.

Q-vrednosti

Algoritem za Q-učenje deluje po principu iteracije vrednosti, le da ob posamezni iteraciji, namesto izračuna približkov optimalnih vrednosti vseh stanj, računa približek le za eno stanje ter za eno izbrano akcijo v tem stanju (princip metod Monte Carlo [19]). V tem primeru ni potrebno znanje modela in lahko izvajamo učenje tudi na realnem sistemu. Po dovolj velikem številu iteracij – številu korakov – ravno tako dosežemo neko skoraj-optimalno vrednost. Q-učenje namesto iskanja preslikav iz stanj v vrednosti stanj išče preslikave iz parov stanj/akcij v vrednosti imenovane *Q*-vrednosti [10]. Posledično namesto funkcije vrednosti ima t.i. *Q*-funkcijo. V vsakem stanju hrani po eno *Q*-vrednost za vsako akcijo tega stanja. Te vrednosti so, tako kot vrednosti stanja, seštevek vseh spodbud ob izvedbi določene akcije in nadaljnjemu sledenju dani strategiji. V kontekstu Q-učenja je vrednost nekega stanja definirana kot največja *Q*-vrednost, ki se nanaša na to stanje – drugače povedano: vrednost nekega stanja je vezana na najboljšo akcijo v tem stanju. Po tej definiciji je enostavno iz Bellmanove enačbe izraziti enačbo (3.2) za Q-učenje:

$$Q(x_t, a_t) = r(x_t, a_t) + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1}) \tag{3.2}$$

Enačbo za spremembo uteži (*Q*-vrednosti) po korakih učenja izpeljemo iz enačbe (3.1) in je prikazana z enačbo (3.3) [10]:

$$\Delta w_t = \alpha * \left[\left(r(x_t, a_t) + \gamma \max_{a_{t+1}} Q(x_{t+1} a_{t+1})\right) - Q(x_t, a_t)\right] \tag{3.3}$$

Naključnost in raziskovanje prostora smo vnesli z dodajanjem šuma pri izbiri izhodne akcije a_t ter pragovno (stopničasto) izhodno funkcijo, kot to prikazuje enačba (3.4) za dve možni akciji:

$$a_t = \begin{cases} 1, & Q(s(t), 0) + \check{s}um(t) < Q(s(t), 1) \\ 0, & \text{sicer} \end{cases} \tag{3.4}$$

Šum je definiran kot naključna vrednost pri enakomerni porazdelitvi na intervalu od $[-\beta, +\beta]$. Parameter β torej določa stopnjo raziskovanja prostora.

Vse koeficiente, ki nastopajo pri tem učenju, v nadaljevanju imenujemo **parametri Q-učenja** in so naslednji:

$\alpha > 0$	faktor spreminjanja Q-vrednosti
$\beta > 0$	parameter raziskovanja prostora
$\gamma \in [0,1]$	faktor zmanjševanja teže prihodnjih spodbud

3.2 Učenje ASE/ACE

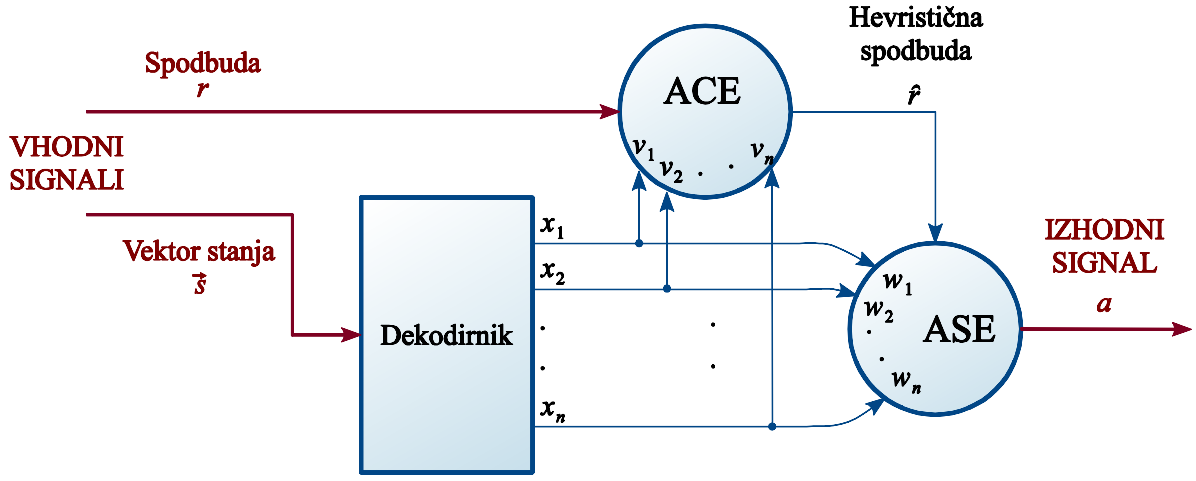
Učenje akter-kritik

Ta metoda posnema princip iteracije strategije v smislu, da je učenje osnovano na sledenju in istočasnemu poskušanju izboljšanja strategije. Poimenovanje izvira iz dejstva, da sta pri tej metodi prisotna dva samostojno delujoča elementa – dve strukturi, ki ločujeta funkcijo vrednosti in strategijo izbire akcij. Ena struktura je **akter**, ki ponazarja strategijo izbire akcij – torej izbira izhodne akcije učnega sistema. Druga je **kritik**, ki je dejansko približek optimalne funkcije vrednosti in deluje po principu učenja po časovnih razlikah. Kritik sprotno ocenjuje vrednosti akterja tako, da na svojem vhodu sprejema spodbudo iz okolja ter trenutno stanje, na izhod pa daje svojo **hevristično spodbudo** – to je spodbuda, ki temelji na podlagi svojih preteklih izkušenj – in služi kot vhodni signal v akterja. To pomeni, da je kritik tisti element, ki nadzira vse učenje, saj akter ne prejema več spodbude iz okolja. Model učnega sistema ASE/ACE, ki deluje po tem principu, je prikazan na sliki 3.1. Izbrana aproksimacijska funkcija je preslikovalna tabela, zato akter (ASE) in kritik (ACE) hranita približke optimalnih vrednosti v seznamih **uteži**. ACE hrani približek optimalnih vrednosti stanj z utežmi $\boldsymbol{v}$, ASE pa hrani približek optimalne strategije izbire akcij z utežmi $\boldsymbol{w}$. Dekodirnik, prikazan na shemi, deluje kot funkcija diskretizacije prostora z namenom, da lahko uporabljamo preslikovalne tabele kot funkcijski aproksimator. Na svoj vhod vektor stanj ter ga preslika v vrednosti izhodnih signalov tako, da ima v nekem trenutku le en signal vrednost 1, ostali pa 0 – dejansko razdeli prostor stanj na diskretne dele ter nam pove, v katerem delu se trenutno nahaja.

Sledi upravičenosti

Učenje ASE/ACE uporablja poleg pristopa akter-kritik še dodatni mehanizem, ki ga imenujemo sledi upravičenosti (ang. eligibility traces [2]). To je eden od osnovnih mehanizmov spodbujevalnega učenja na katerem sloni metoda učenja po časovnih razlikah [19]. Idejna zasnova pravi, da ko pride do nekega dogodka (v obliki pozitivne ali negativne spodbude), bi lahko zasluge za pojavitev tega dogodka pripisali le stanjem in akcijam, ki so pripeljale do njega. **Sledi** so torej vrednosti, ki izražajo začasno

pogostost nekega dogodka v smislu obiskanosti nekega stanja ali izbiranja akcije. Na ta način beležijo, katera stanja in akcije so **upravičene** do kazni ali spodbude, ko pride do nekega dogodka – določajo za koliko in katere uteži bodo spremenile vrednosti pri vsakem koraku učenja. Pri tem predpostavljamo, da so časovno bolj oddaljena stanja in akcije manj upravičena do sprememb uteži, zato uvedemo **koeficient upadanja sledi** λ (ang. trace decay rate [2]), ki določa stopnjo upadanja upravičenosti s časom. To je vrednost na intervalu med [0,1]. Zadnje obiskano stanje ima upravičenost enako 1, predzadnje pa $\lambda\gamma$, ipd.



Slika 3.1. Model učnega sistema ASE/ACE, prirejenega za reševanje problema vozička s palico.

Enačbe učenja ASE/ACE

V tem razdelku bomo podali enačbe učenja algoritma ASE/ACE ob upoštevanju dvopoložajnega krmilnika ter dekodirnika, ki vrača le en signal z vrednostjo 1, kot že opisano prej. Podrobna razlaga ni namen našega dela in jo lahko najdemo v omenjenem članku Barta in sod. [2].

Barto in sod. so za določitev izhodne akcije uporabili stopničasto pragovno funkcijo pri dodanem šumu z normalno (Gaussovo) porazdelitvijo. Mi pa smo uporabili sigmoidno funkcijo za določitev trenutnega praga $a'(t)$ na intervalu [0,1], ki ga uporablja stopničasta pragovna izhodna funkcija z vhodnim šumom z enakomerno porazdelitvijo na intervalu [0,1], kot prikazujeta sliki 3.2 ter 3.3. Izhod učnega sistema ASE/ACE na enak način opisujeta tudi enačbi (3.5) in (3.6). Vrednost i je številka signalne linije dekodirnika, ki ima stanje 1 – to je trenutno stanje okolja. $w_i(t)$ je vrednost uteži strategije za izbiro akcij za trenutno stanje okolja (v elementu ASE), $a(t)$ je trenutna izbira akcije učnega sistema.

Hevristična spodbuda ACE je podana z enačbama (3.7) in (3.8), pri čemer je $v_i(t)$ trenutna vrednost uteži vrednostne funkcije za trenutno stanje okolja (v elementu ACE), $p(t)$ je napoved spodbude, $p(t-1)$ je napoved v prejšnjem koraku učenja, $r(t)$ je trenutna spodbuda iz okolja, $\hat{r}(t)$ pa je trenutna izhodna hevristična spodbuda kritika. Sledi spreminjamo po enačbah (3.9) in (3.10). Tukaj je $\bar{x}_i(t)$ vrednost sledi trenutnega stanja, $e_i(t)$ vrednost sledi trenutne akcije v trenutnem stanju. Potrebno je poudariti, da morajo biti pred vsakim poskusom vrednosti vseh sledi enake 0. Končno lahko podamo izračun spreminjanja uteži trenutnega stanja $v_i(t)$ in uteži trenutne akcije v trenutnem stanju $w_i(t)$ z enačbama (3.11) in (3.12).

Vse koeficiente, ki nastopajo pri tem učenju, v nadaljevanju imenujemo **parametri učenja ASE/ACE** in so naslednji:

$\alpha > 0$	faktor spreminjanja uteži akcij (ASE, strategije izbire akcij)
$\beta > 0$	faktor spreminjanja uteži stanj (ACE, vrednostne funkcije)
$\gamma \in [0,1]$	faktor zmanjševanja teže prihodnjih spodbud
$\delta > 0$	koeficient spreminjanja sledi izbire akcij
$\lambda > 0$	koeficient spreminjanja sledi obiskanosti stanja

Seznam vseh enačb učenja ASE/ACE, na katere se sklicujemo v besedilu:

$$a'(t) = \frac{1}{1 + e^{-w_i(t)}} \tag{3.5}$$

$$a(t) = \begin{cases} 1, & \check{s}um(t) > a'(t), \\ 0, & sicer \end{cases} \tag{3.6}$$

$$p(t) = v_i(t) \tag{3.7}$$

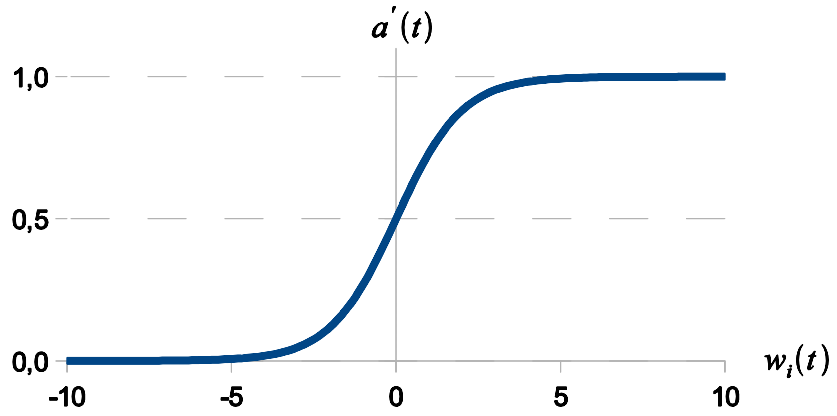
$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) \tag{3.8}$$

$$\bar{x}_i(t+1) = \lambda \bar{x}_i(t) + (1-\lambda) \tag{3.9}$$

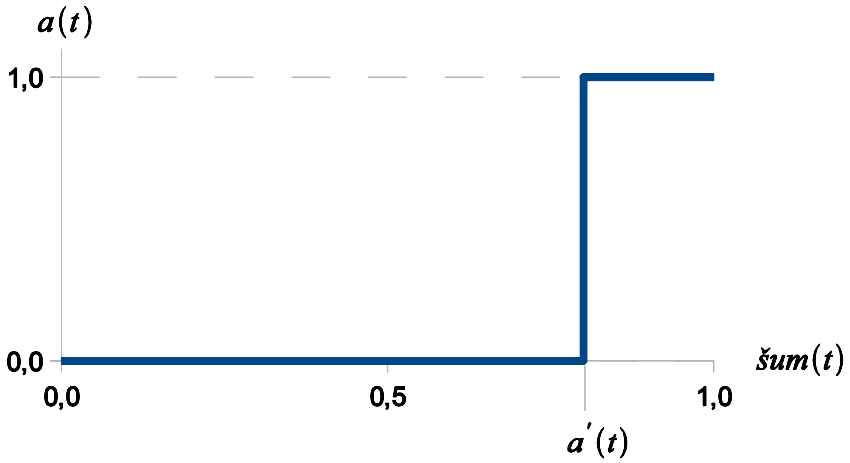
$$e_i(t+1) = \delta e_i(t) + (1-\delta) a(t) \tag{3.10}$$

$$v_i(t+1) = v_i(t) + \beta \hat{r}(t) \overline{x}_i(t) \tag{3.11}$$

$$w_i(t+1) = w_i(t) + \alpha \hat{r}(t) e_i(t) \tag{3.12}$$



Slika 3.2. Sigmoidna funkcija za določitev praga izhodne funkcije ASE/ACE.



Slika 3.3. Pragovna funkcija izhoda učnega sistema z učenjem ASE/ACE.

3.3 Krmilni problem vozička s palico

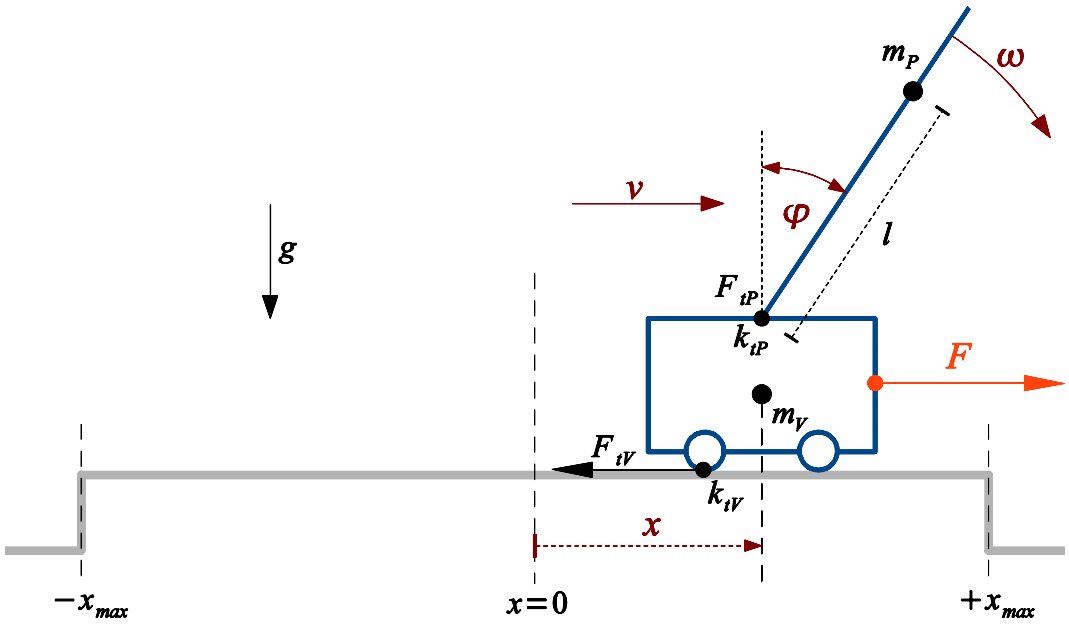
Sistem vozička s palico

Sistem sestavljajo voziček, na katerem je vpeta prosto vrteča palica (inverzno nihalo), ter podlaga z omejeno dolžino. Prikaz sistema je na sliki 3.4. Voziček se lahko premika v eni dimenziji v obe smeri po podlagi. Krmilnik lahko aplicira silo na voziček vzporedno s podlago (označena z oranžno barvo) ter zaznava stanje sistema. To stanje določajo količine, ki so označene z rdečo barvo.

Predpostavili bomo, da je težišče palice vedno na koncu, zato bomo razdaljo do težišča mase palice v nadaljevanju imenovali kar dolžina palice. Poleg navedenih količin delujeta v sistemu še pospešek vozička a ter kotni pospešek α, ki ju bomo opisali v nadaljevanju.

Seznam vseh količin je sledeč:

x	odmik vozička od sredine podlage [m]	m_V	masa vozička [kg]
v	hitrost vozička [m/s]	m_P	masa palice [kg]
φ	odklon palice od navpične lege [°]	g	težni pospešek [m/s^2]
ω	kotna hitrost palice [°/s]	F_{tP}	sila trenja vpetja palice [N]
F	potisna sila [N]	F_{tV}	sila trenja vozička [N]
x_{max}	polovica dolžine podlage [m]	k_{tP}	koeficient trenja vpetja palice
g	razdalja do težišča mase palice [m]	k_{tV}	koeficient trenja med podlago in vozičkom



Slika 3.4. Sistem vozička s palico.

Učni sistem in funkcija spodbude

Učni sistem se mora naučiti držati palico v navpičnem položaju z apliciranjem ustrezne sile, brez da zapelje voziček čez rob podlage. Ker uporabljamo dvopoložajni, krmilnik se akcija krmilnika interpretira kot potisk levo ali potisk desno s fiksno velikostjo sile. Izhod iz učnega sistema je tako izbrana smer potiska, vhod v učni sistem pa je trenutno stanje sistema vozička s palico, ki je določeno z odmikom in hitrostjo vozička ter odklonom in kotno hitrostjo palice.

Izbran problem spada v skupino problemov izogibanja z zakasnjeno spodbudo, kot že omenjeno v podpoglavju 2.3. Funkcija spodbude vrne spodbudo -1 (kazen), ko je **odklon palice večji ali enak 90°** ali ko se je **voziček odmaknil čez rob**

podlage. V vseh ostalih stanjih vrača spodbudo 0. Prehod v stanje, ko učni sistem dobi kazen, imenujemo **neuspeh**. Iz povedanega sledi, da je vsota vseh spodbud (kazni) v določenem času nasprotno enaka številu neuspehov.

Časovna simulacija fizikalnega sistema

Postopek simulacije sistema vozička s palico po časovnih korakih je sledeč [9]. Vsak časovni korak predpostavimo, da ima prispevek sile trenja N_C enak predznak kot prejšnji interval (na začetku simulacije predpostavimo, da je pozitiven). Najprej izračunamo α po enačbi (3.13). Nato z dobljeno vrednostjo izračunamo N_C po enačbi (3.14). Če se predznak N_C spremeni, izračunamo spet α z upoštevanjem drugega predznaka. Nato izračunamo še a po enačbi (3.15). Funkcija sgn() je funkcija predznaka.

Stanje sistema spreminjamo po času z uporabo Eulerjeve metode numerične integracije [21]. Odmik in hitrost vozička ter odklon in kotno hitrost palice računamo po enačbah (3.16), pri čemer je parameter Δt časovno trajanje koraka simulacije. Fizikalna natančnost simulacije je odvisna od te vrednosti. V nadaljevanju jo poimenujemo parameter časovnega koraka simulacije vozička s palico.

Enačbe dinamike sistema vozička s palico:

$$\alpha = \frac{g\sin(\varphi) + \cos(\varphi)\left\{\frac{-F - m_p l\omega^2[\sin(\varphi) + k_{tV}\,\mathrm{sgn}(N_C v)\cos(\varphi)]}{m_V + m_P} + k_{tV} g\,\mathrm{sgn}(N_C v)\cos(\varphi)\right\} - \frac{k_{tP}\omega}{m_P l}}{l\left\{\frac{4}{3} - \frac{m_P\cos(\varphi)}{m_V + m_P}[\cos(\varphi) - k_{tV}\,\mathrm{sgn}(N_C v)]\right\}} \tag{3.13}$$

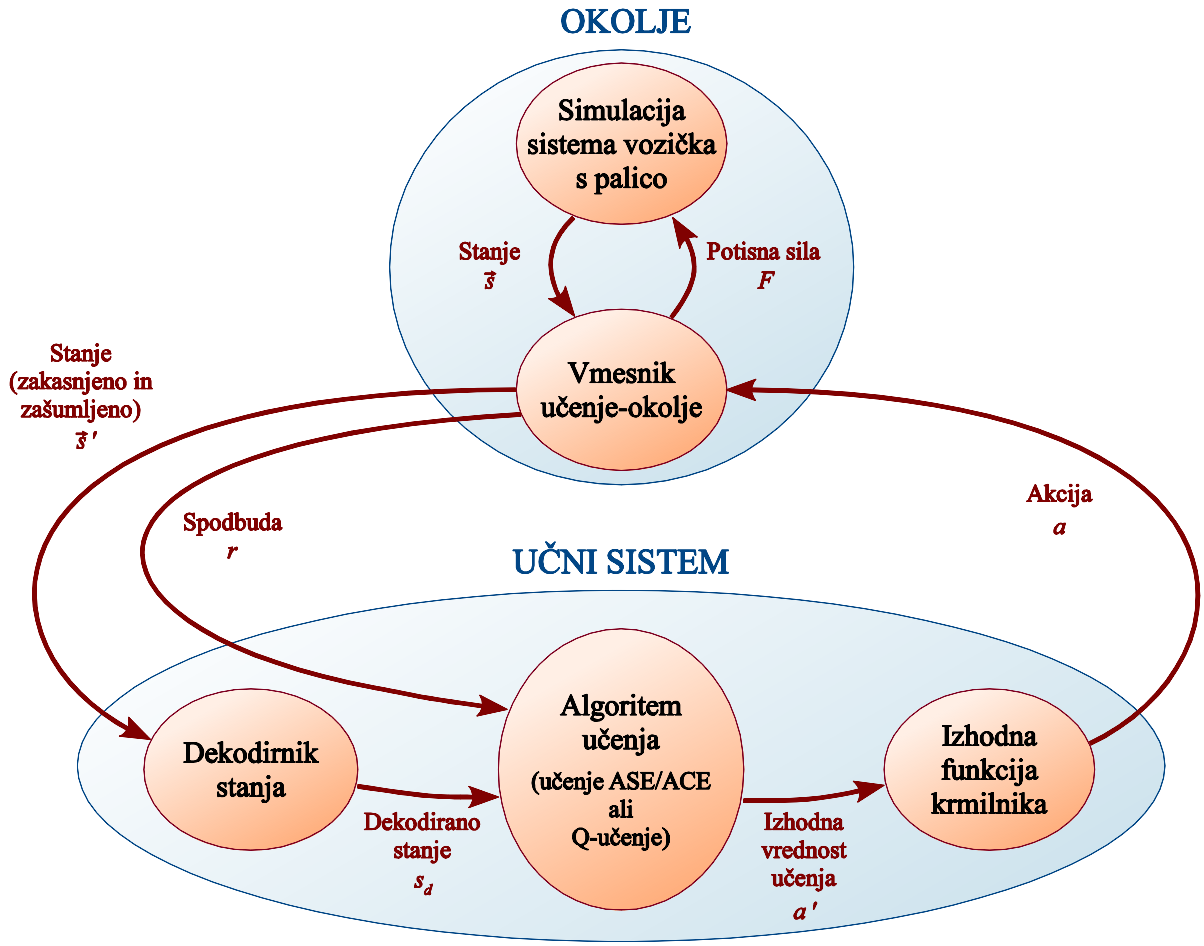
$$N_C = (m_V + m_P)g - m_P l(\alpha\sin(\varphi) + \omega^2\cos(\varphi)) \tag{3.14}$$

$$a = \frac{F + m_P l(\omega^2\sin(\varphi) - \alpha\cos(\varphi)) - k_{tV} N_C\,\mathrm{sgn}(N_C v)}{m_V + m_P} \tag{3.15}$$

$$\begin{aligned} x_{t+1} &= x_t + \Delta t * v_t \\ v_{t+1} &= v_t + \Delta t * a \\ \varphi_{t+1} &= \varphi_t + \Delta t * \omega_t \\ \omega_{t+1} &= \omega_t + \Delta t * \alpha \end{aligned} \tag{3.16}$$

4 Simulacija učnega sistema v okolju

Metodi spodbujevalnega učenja, ki smo ju opisali v poglavju 3, združimo z modelom vozička s palico v skupen sistem. Implementacijo programske kode tega sistema smo poimenovali **simulacija učnega sistema v okolju**. Na sliki 4.1 so prikazani posamezni programski sklopi simulacije ter podatkovni tokovi med njimi.



Slika 4.1. Programski sklopi ter podatkovni tokovi učnega sistema v okolju.

Celotna simulacija je razdeljena na pet glavnih sklopov. Učni sistem tvorijo dekodirnik stanja, izhodna funkcija krmilnika ter algoritem učenja, ki deluje po metodi Q-učenja ali učenja ASE/ACE. Simulacija dinamike sistema vozička s palico ter vmesnik med simulacijo in učnim sistemom (v nadaljevanju tudi: vmesnik učenje-okolje) tvorita okolje. Učni sistem pošilja v okolje izbrane akcije, od njega pa prejema spodbudo ter stanje, ki je lahko zakasnjeno in/ali zašumljeno. Simulacijo izvajamo v

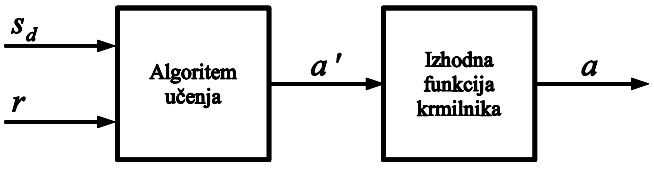
zanki po časovnih korakih. Posamezen časovni korak imenujemo tudi korak učenja. Slika 4.1 prikazuje izvedbo enega koraka učenja.

Glavno vodilo pri pisanju programske kode je bilo ohranjanje modularnosti posameznih sklopov simulacije, zato smo program napisali kot samostojne enote programske kode z uporabo objektnega programiranja. Vsako enoto se uporablja preko dveh skupin procedur: nastavitvene procedure ter procedure za izvajanje. Prva skupina omogoča začetno nastavitev podatkovnih struktur in spreminjanje ter nastavljanje vseh dejavnikov, ki vplivajo na potek izvajanja simulacije (npr. teža palice, faktor spreminjanja uteži, ipd.). Druga skupina pa se uporablja za izvajanje korakov učenja, torej za spreminjanje stanja okolja, učenja algoritma ter prenosa podatkov med posameznimi enotami.

Podrobnosti o izvajanju korakov učenja v zanki bomo opisali v poglavju 5, v nadaljevanju tega poglavja pa se bomo posvetili opisom posameznih enot na sliki, njihovi implementaciji, delovanju ter proceduram.

4.1 Algoritem učenja in izhodna funkcija krmilnika

Osrednji del učnega sistema je algoritem učenja, ki lahko deluje po metodi učenja ASE/ACE ali Q-učenja. Delovanje obeh je bilo podrobno opisano v prejšnjem poglavju. Vhod algoritma učenja predstavljata spodbuda iz okolja r in dekodirana vrednost stanja s_d. Izhod algoritma učenja je zvezna spremenljivka a'. Izhodna funkcija krmilnika ni nujen gradnik sistemov učenja v okolju, toda ker ima v našem primeru model sistema vozička s palico dvopoložajni krmilnik, jo potrebujemo za preslikavo zveznega izhoda algoritma učenja a' v celoštevilčni izhod učnega sistema a z vrednostjo 0 ali 1. Slika 4.2 prikazuje omenjena sklopa v stilu t.i. črnih škatel [21] – zaključene programske enote, ki sprejemajo vhodne vrednosti kot parametre klica funkcije, ki vrača izhodno vrednost.



Slika 4.2. Algoritem učenja ter izhodna funkcija krmilnika predstavljeni kot t.i. črni škatli.

Posamezen algoritem je implementiran s tabelami uteži in sledi ter procedurami za učenje. Procedure za učenje spreminjajo vrednosti v tabelah po ustreznih enačbah. Uteži in sledi ne beležimo po času, zato je velikost tabel po tej dimenziji enaka 1 –

beležimo samo zadnjo vrednost. Dekodirana vrednost stanja s_d pomeni indeks v tabelah, zato njena velikost množice vrednosti določa velikost tabel. To pomeni, da je velikost posamezne tabele odvisna le od tega, na koliko delov razdelimo prostor stanj. To je število n in ga v nadaljevanju imenujemo tudi število diskretnih stanj prostora. Izhodna funkcija krmilnika je implementirana z odločitvenim stavkom med pragom a' ter naključno vrednostjo z enakomerno porazdelitvijo na določenem intervalu.

V tabeli 4.1 so prikazane procedure za uporabo, z rdečo barvo pa so označeni nastavljivi parametri. Ker je v našem primeru spodbuda -1 samo ob neuspehu, ves preostali čas pa enaka nič, smo zaradi optimizacije računske zahtevnosti enačbe učenja razdelili na dve ločeni proceduri. V nadaljevanju bomo navedli razlike pri implementaciji posameznega algoritma učenja povezanega z izhodno funkcijo krmilnika.

Tabela 4.1. Procedure programske kode algoritmov učenja.

		Opis	**Vhodni parametri**	**Izhodne vrednosti**
Nastavitvene procedure	**Dodelitev virov**	Dodelitev pomnilnika za podatkovne strukture	Število diskretnih stanj prostora n	-
	Nastavitev parametrov	Nastavitev vrednosti parametrov enačb učenja	$\alpha, \beta, \lambda, \delta, \gamma$	-
	Nastavitev začetnega stanja	Nastavitev začetnih vrednosti tabel in notranjih spremenljivk	Začetne vrednosti tabel	-
Procedure za izvajanje	**Korak učenja**	Sprememba uteži in sledi pri spodbudi 0	Številka stanja sistema s_d	Izbrana akcija a
	Kaznovanje	Sprememba uteži in sledi pri spodbudi -1	Vrednost spodbude r	-

Učenje ASE/ACE

Ta algoritem hrani za vsako stanje štiri vrednosti, zato je implementiran s štiri tabelami velikosti n:

- uteži za ASE, $w(s)$
- uteži za ACE, $v(s)$
- sledi za ASE, $e(s)$
- sledi za ACE, $\bar{x}(s)$

Pomnilniška zahtevnost je tako $4n$. Ob vsakem koraku učenja se izračunajo nove vrednosti za vse elemente v tabelah, zato je računska zahtevnost posameznega koraka učenja reda $O(4n)$ po notaciji računske zahtevnosti.

Izbira akcije je prikazana s kratko psevdokodo:

```
a' = sigmoidna_funkcija ( w[s_d] )
x  = naključna_vrednost ( med 0 in 1 )
Če ( x < a' )
  a = 0
Sicer
  a = 1;
```

Pri čemer je s_d dekodirana vrednost trenutnega stanja. Naključna spremenljivka ima enakomerno porazdelitev.

Q-Učenje

Algoritem Q-učenja računa t.i. vrednosti Q za pare stanj in akcij. Ker sta v našem primeru možni dve izhodni akciji, implementiramo dvo-dimenzionalno tabelo velikosti $2 * n$:

- vrednosti Q za akcijo 0; $Q(s, 0)$
- vrednosti Q za akcijo 1; $Q(s, 1)$

Pomnilniška zahtevnost je $2n$. Ob vsakem koraku učenja se izračuna vrednost Q samo za trenutno stanje s_d, zato je računska zahtevnost posameznega koraka učenja reda $O(1)$. To je ena od bistvenih razlik med obema algoritmoma za učenje. Meritve časov računanja so podane v poglavju 7.

Psevdokoda za izbiro akcije:

```
a' = Q[s_d][1] - Q[s_d][0]
x  = naključna_vrednost ( med -beta in +beta )
Če ( x < a' )
  a = 0
Sicer
  a = 1;
```

Tudi v tem primer ima naključna spremenljivka enakomerno porazdelitev.

Procedure za uporabo programskega sklopa za Q-učenje so podobne kot za učenje ASE/ACE in so navedene v tabeli 4.1, razlike so naslednje:

- Ni vhodnih parametrov δ in λ pri nastavitvah, temveč samo α, β, λ.
- Po definiciji v tabelah ne spreminjamo vrednosti uteži in sledi, temveč vrednosti Q ter posledično pri kaznovanju nimamo ponovne nastavitve sledi.

4.2 Dekodirnik stanja

Dekodirnik stanja pove, v katerem delu prostora stanj se sistem trenutno nahaja. Kot vhod sprejme vektor trenutnega stanja $\vec{s}$ (slika 4.3), ki je sestavljen iz štirih zveznih spremenljivk – štirih dimenzij: odmika vozička x, hitrosti vozička v, kotnega odklona palice φ in kotne hitrosti palice ω. Izhodna vrednost s_d je nenegativno celo število in služi kot indeks v tabelah algoritma učenja, kot že omenjeno v prejšnjem podpoglavju. Ker funkcija dekodirnika preslika navedene zvezne spremenljivke v diskretno spremenljivko, pravimo, da dekodirnik opravlja funkcijo diskretizacije prostora stanj.

Slika 4.3. Dekodirnik stanja predstavljen kot črna škatla.

Število diskretnih stanj prostora ter meje delitve posamezne dimenzije so poljubni. Mi smo se zgledovali po Bartu in sod. [2] ter tako privzeli $n = 162$ diskretnih stanj prostora. Pri delitvi prostora po posameznih dimenzijah smo naredili manjšo nadgradnjo tako, da lahko poljubno nastavljamo največji odmik vozička x_{max}, zato se meje delitve izračunajo sorazmerno glede na to vrednost. Meje po dimenzijah so naslednje:

- odmik vozička, 3 intervali: $\pm x_{max}/3$, $\pm\infty$ m
- hitrost vozička, 3 intervali: $\pm 0{,}5$, $\pm\infty$ m/s
- odklon kota, 6 intervalov: 0, ± 1, ± 6, $\pm\infty$ °
- kotna hitrost, 3 intervali: ± 50, $\pm\infty$ °/s

Računsko učinkovita vsebuje množico odločitvenih stavkov in vsoto vrednosti za vsako dimenzijo posebej.

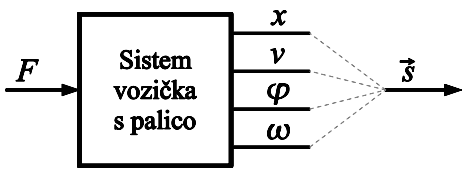
V tabeli 4.2 sta prikazani proceduri za uporabo, z rdečo barvo pa je označen nastavljiv parameter. Na največji odmik vozička se v nadaljevanju sklicujemo tudi kot na parameter širine podlage (ki je dvakratnik te dolžine).

Tabela 4.2. Procedure programske kode dekodirnika stanja.

		Opis	**Vhodni parametri**	**Izhodne vrednosti**
Nastavitvene procedure	**Nastavitev začetnega stanja**	Nastavitev notranjih spremenljivk	Največji odmik vozička x_{max}	Število diskretnih stanj prostora n
Procedure za izvajanje	**Diskretizacija stanja**	Preslikava iz zveznega v diskretno stanje	Vektor stanja sistema $\vec{s}$	Številka stanja sistema s_d

4.3 Simulacija sistema vozička s palico

V tem programskem sklopu poteka fizikalna simulacija sistema vozička s palico po diskretnih časovnih korakih. Na sliki 4.4 so prikazane vhodno/izhodne spremenljivke programskega sklopa fizikalne simulacije ter posamezne spremenljivke vektorja stanja.



Slika 4.4. Fizikalna simulacija sistema vozička s palico predstavljena kot črna škatla.

Tabela 4.3 prikazuje procedure za uporabo, z rdečo barvo so označeni nastavljivi parametri. Ob klicu procedure *simuliraj dinamiko fizikalnega sistema* se na podlagi trenutnega stanja $\vec{s}(t)$ izračunajo vrednosti novega stanja $\vec{s}(t + \Delta t)$ z upoštevanjem vhodne sile F ter želenega časovnega intervala Δt po enačbah, ki so navedene v poglavju 3.3. V istem poglavju so tudi navedeni vsi parametri sistema vozička s palico.

Časovna natančnost fizikalne simulacije

Časovni interval fizikalne simulacije Δt ne vpliva na spreminjanje vrednosti v tabelah algoritma učenja. Drugače pa je pri fizikalni simulaciji dinamike vozička. Zaradi uporabljene Eulerjeve metode, nenatančnost fizikalne simulacije narašča z velikostjo Δt. Iz tega razloga smo implementirali vhodni parameter *najdaljši dovoljen časovni interval* za posamezen izračun novega stanja Δt_{max}. Če je ob klicu procedure za simulacijo dinamike želen časovni interval Δt večji od Δt_{max}, se izračun novega stanja razdeli na več izračunov pri krajšem intervalu $\Delta t'$. Število teh izračunov se določi tako, da je $\Delta t' < \Delta t_{max}$.

Tabela 4.3. Procedure programske kode simulacije fizikalnega sistema vozička s palico

		Opis	**Vhodni parametri**	**Izhodne vrednosti**
Nastavitvene procedure	**Nastavitev parametrov**	Nastavitev vrednosti parametrov fizikalnega sistema in pred-izračuni	Parametri sistema vozička s palico, najdaljši dovoljen časovni interval Δt_{max}	-
	Nastavitev začetnega stanja	Nastavitev začetnih vrednosti notranjih spremenljivk	-	-
Procedure za izvajanje	**Simuliraj dinamiko fizikalnega sistema**	Izračun novega stanja po fizikalnih enačbah	Vektor stanja $\vec{s}(t)$ potisna sila F, časovni interval Δt	Vektor stanja $\vec{s}(t + \Delta t)$

Implementacija trenja

Sila trenja je lahko pri določenih parametrih mnogo večja od potisne sile. Ko je voziček v stanju mirovanja, lahko to privede do situacije, da je prispevek sile trenja v nasprotno smer potisne sile večji od prispevka potisne sile in se tako voziček premakne iz stanja mirovanja v smer sile trenja – kar je seveda napačno. Da se izognemo tej težavi, je potrebno ločeno izračunati prispevek sile trenja k hitrosti ter prispevek ostalih sil k hitrosti in z odločitvenim stavkom preveriti, kateri je večji. Če je prispevek sile trenja večji, potem nastavimo hitrost na 0; tako pravilno ustavimo voziček in preprečimo, da bi se premaknil v napačno smer. Na tak način smo zagotovili pravilnost simulacije neodvisno od velikosti parametrov. Podobno velja za silo trenja vrtišča palice.

Optimizacija računske zahtevnosti

Enačbe dinamike sistema vsebujejo veliko računsko potratnih operacij, kot so deljenja in kotne funkcije. Več parametrov se ne spreminja med simulacijo (npr. masa vozička, težni pospešek, ipd.). Zato smo enačbe izpeljali tako, da se čim več računanja opravi samo enkrat, ob nastavitvi novih vrednosti parametrov. To so t.i. pred-izračuni ob klicu procedure za nastavitev parametrov. Zaradi tega je potrebnih manj računskih operacij za sprotni izračun koraka simulacije. Pred-izračuni so označeni s P na enačbah (4.1). Pri preostalih sprotnih izračunih smo izpostavili vse faktorje I, ki nastopajo več kot enkrat – enačbe (4.2). Tako je optimizirana oblika enačb dinamike, ki se izvedejo ob vsakem časovnem intervalu simulacije, podana z dvema enačbama (4.3).

Pridobljeno število operacij na račun optimizacije je prikazano v tabeli 4.4, izmerjena časovna pohitritev pa je podana v poglavju 7. Optimizirane enačbe:

$$P_1 = \frac{k_{tV}}{m_V + m_P} \qquad P_4 = \frac{1}{m_V + m_P} \qquad P_7 = \frac{m_P l}{m_V + m_P}$$

$$P_2 = m_p l \qquad P_5 = \frac{4}{3} l \qquad P_8 = \frac{k_{tV}}{m_V + m_P}(m_V + m_P)g \tag{4.1}$$

$$P_3 = \frac{k_{tP}}{m_P l} \qquad P_6 = g k_{tV} \qquad P_9 = \frac{k_{tV}}{m_V + m_P} m_P l$$

$$I_1 = \omega^2 \qquad I_2 = F P_4 + P_7 I_1 \sin(\varphi)$$

$$I_3 = P_7 \cos(\varphi) \qquad I_4 = \operatorname{sgn}(N_C v) \tag{4.2}$$

$$\alpha = \frac{g \sin(\varphi) + \cos(\varphi)\,(P_6 I_4 - I_2 + \cos(\varphi)\, P_1 I_4) - P_3 \omega}{P_5 - I_3(\cos(\varphi) + k_{tV} I_4)} \tag{4.3}$$

$$a = I_2 - I_3\,\alpha + I_4(P_9(\sin(\varphi)\alpha - I_1 \cos(\varphi)) - P_8)$$

Tabela 4.4. Primerjava števila računskih operacij pred in po optimizaciji enačb dinamike.

	prvotno	optimizirano
kotne funkcije	2	1
korenske funkcije	0	1
deljenja	5	1
množenja	24	18
seštevanja in odštevanja	16	14
odločitveni stavki	2	5

Opombe k enačbam za optimizacijo računske zahtevnosti:

Enačbe za spreminjanje odmika, hitrosti, kota in kotne hitrosti po Eulerjevi metodi so enake kot v poglavju 3.3. V istem poglavju so tudi navedena poimenovanja vseh količin, ki nastopajo v enačbah.

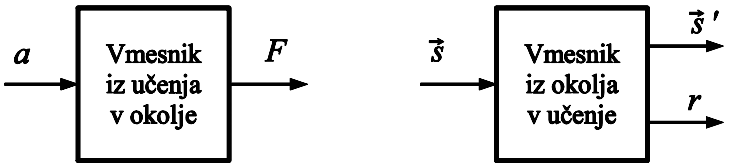
Pri funkciji predznaka sgn() smo predpostavili, da je prispevek trenja N_C vedno pozitiven [9], zato upoštevamo samo predznak hitrosti v. To funkcijo smo implementirali z dvema odločitvenima stavkoma.

V navedenih enačbah nastopajo kotne funkcije na več mestih vendar se pri implementaciji ob vsakem koraku izračunajo le enkrat. Poleg tega smo rezultat ene kotne funkcije izračunali po Pitagorovi trigonometrični identiteti [21]. S tem smo se znebili ene kotne operacije na račun pridobljene operacije kvadratnega korena ter odločitvenega stavka. Katere od teh računskih operacij se izvedejo hitreje, je odvisno od arhitekture posameznega procesorja, na katerem se koda izvaja, toda v večini primerov so kotne funkcije bolj potratne kot korenske, zato je naša izbira smiselna.

4.4 Vmesnik med učnim sistemom in simulacijo vozička s palico

Ta programski sklop je sestavljen iz dveh podsklopov, ki jih prikazuje slika 4.5: vmesnik iz učenja v okolje ter vmesnik iz okolja v učenje. Znotraj prvega podsklopa smo umestili funkcijo preslikave akcije a v potisno silo F, ki deluje po enačbi (4.4). V drugem podsklopu pa je funkcija spodbude oz. kaznovanja, ki preverja trenutno stanje okolja in deluje po opisu iz poglavja 3.3.

$$F = f(a) = \begin{cases} +F_{max}; a = 1 \\ -F_{max}; a = 0 \end{cases} \tag{4.4}$$



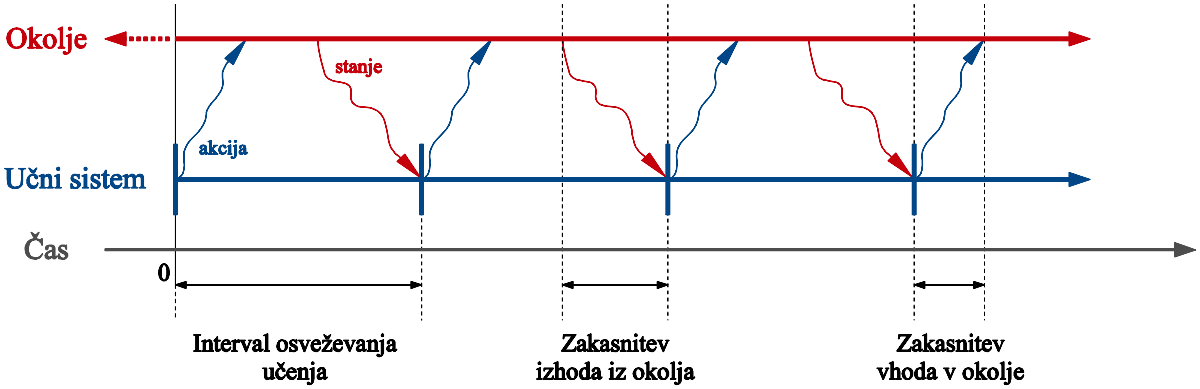
Slika 4.5. Oba sklopa vmesnika učenje-okolje predstavljena kot črni škatli.

Implementacija šumov in zakasnitev

Simulacijo okolja smo nadgradili z implementacijo šumov ter zakasnitev. Vmesnik iz učenja v okolje dodaja časovno zakasnitev akciji a ter šum potisni sili F (v nadaljevanju tudi: vhodna zakasnitev in vhodni šum). Vmesnik iz okolja v učenje dodaja šum vektorju stanja ter časovno zakasnitev spodbude in vektorja stanja (v nadaljevanju tudi: izhodni šum in izhodna zakasnitev). Vhodni šum je izražen v odstotkih naključnega nihanja vrednosti potisne sile F_{max}, izhodni šum pa v odstotkih naključnega nihanja trenutne vrednosti posamezne spremenljivke stanja. Naključna porazdelitev je enakomerna.

Ker je tudi spodbuda zakasnjena, lahko s spreminjanjem trajanja zakasnitve preizkušamo sposobnost napovedovanja algoritma učenja.

Na sliki 4.6 je na časovnem traku prikazan potek simulacije z upoštevanjem zakasnitev. Z vidika okolja je na začetku simulacije potisna sila enaka 0, dokler ne dobi na vhod prve akcije. Učni sistem pa predpostavlja ohranjanje začetnega stanja okolja, dokler ne dobi na vhod prvega vektorja stanja.



Slika 4.6. Interakcija med učnim sistemom in okoljem na časovnem traku z upoštevanjem zakasnitev.

Tabela 4.5 prikazuje procedure uporabe. Med nastavljive parametre, ki so označeni z rdečo, spadajo: vhodni in izhodni šum, vhodna in izhodna zakasnitev, največja velikost potisne sile F_{max} ter največji odmik vozička x_{max}. Parameter največjega odmika potrebuje funkcija spodbude za pravilno delovanje.

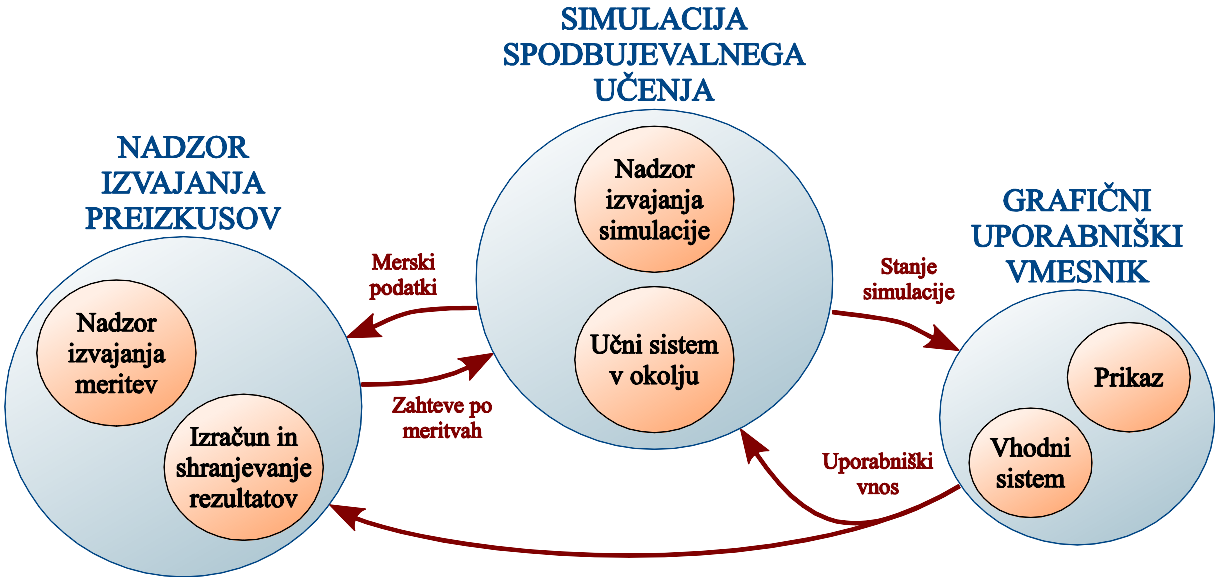
Tabela 4.5. Procedure programske kode vmesnika med učnim sistemom in okoljem.

		Opis	**Vhodni parametri**	**Izhodne vrednosti**
Nastavitvene procedure	**Nastavitev parametrov**	Dodelitev pomnilnika za zakasnitve in nastavitev notranjih spremenljivk	Šumi, zakasnitve, velikost sile F_{max}, največji odmik vozička x_{max}	-
	Nastavitev začetnega stanja	Nastavitev začetnih vrednosti pomnilnika za zakasnitve	-	-
Procedure za izvajanje	**Prenos iz učenja v okolje**	Določitev vhodne sile v okolje, dodajanje šuma in zakasnitve	Izhod okolja a	Potisna sila F
	Prenos iz okolja v učenje	Določitev spodbude, dodajanje šuma stanju in zakasnitve	Vektor stanja $\vec{s}$	Zašumljen in zakasnjen vektor stanja $\vec{s}'$, spodbuda r

5 Programska implementacija

Merjenje uspešnosti spodbujevalnega učenja pri različnih okoliščinah zahteva – poleg simulacije učnega sistema v okolju – razvoj programskih sklopov, ki te okoliščine določajo in ki omenjeno simulacijo uporabljajo kot orodje za opravljanje meritev in zbiranje rezultatov. Teoretična osnova ter implementacija simulacije učnega sistema v okolju je bila podrobno opisana v preteklih poglavjih, zato se bomo tukaj posvetili opisu preostalih sklopov aplikacije.

Na sliki 5.1 so prikazani vsi programski sklopi aplikacije. Izvajanje korakov učenja učnega sistema je implementirano v sklopu **nadzora izvajanja simulacije**, ki skupaj z omenjenim **učnim sistemom v okolju** tvorita zaključeno enoto **simulacije spodbujevalnega učenja**. Ta pošilja merske podatke ter sprejema zahteve po izvajanju simulacij in meritev od **nadzora izvajanja preizkusov**, kjer smo implementirali funkcije za **izračun in shranjevanje rezultatov** ter **funkcije za nadzor nad izvajanjem posameznih meritev**. **Grafični uporabniški vmesnik** dodatno razširja zmogljivosti aplikacije v smislu uporabnikove možnosti opazovanja stanja simulacije v realnem času ter vpliva nanjo.



Slika 5.1. Programski sklopi celotne aplikacije ter interakcija med njimi.

5.1 Nadzor izvajanja simulacije učnega sistema v okolju

V poglavju 4 je natančno opisano delovanje in implementacija posameznega koraka učenja. Hočemo simulirati učenje po času, zato moramo implementirati zaporedno izvajanje korakov učenja – iteracijo. To je ključna funkcionalnost **nadzora izvajanja simulacije učnega sistema v okolju** (v nadaljevanju tudi: nadzor izvajanja simulacije). Poleg iteracije korakov učenja je potrebno izpolniti še druge zahteve, zato navajamo seznam vseh nalog tega programskega sklopa:

- **diskretizacija časa**, to je iteracija korakov učenja ter določitev časovnega trajanja enega koraka,
- **omejitev simuliranega časa** določa, koliko korakov učenja se bo izračunalo,
- **nastavitev začetnega stanja okolja in učnega sistema** pred začetkom simulacije,
- **ponovno nastavljanje začetnega stanja okolja** ob neuspehu.

Časovno trajanje enega koraka učenja imenujemo **interval učenja**. Označen je na sliki 4.6 v poglavju 4.4. Interval učenja določa hitrost osveževanja algoritma učenja – to je število korakov učenja, ki jih algoritem naredi v simulirani sekundi. Iz navedena sledi, da je skupen simuliran čas enak zmnožku števila korakov učenja ter intervala učenja, zato smo ga poimenovali tudi **čas učenja**.

Simulacija spodbujevalnega učenja

Diagram poteka simulacije spodbujevalnega učenja prikazuje desna polovica slike 5.2. Implementirana je z zanko, ki izvaja korake učenja, dokler ne doseže zastavljenega časa učenja. Pred zanko je del za nastavitev začetnega stanja okolja in učnega sistema. Ob vsaki iteraciji preverjamo, če je prišlo do neuspeha. V tem primeru ponovno nastavimo začetno stanje okolja in tako simuliramo začetek novega poskusa učnega sistema, da obdrži stanje brez neuspeha.

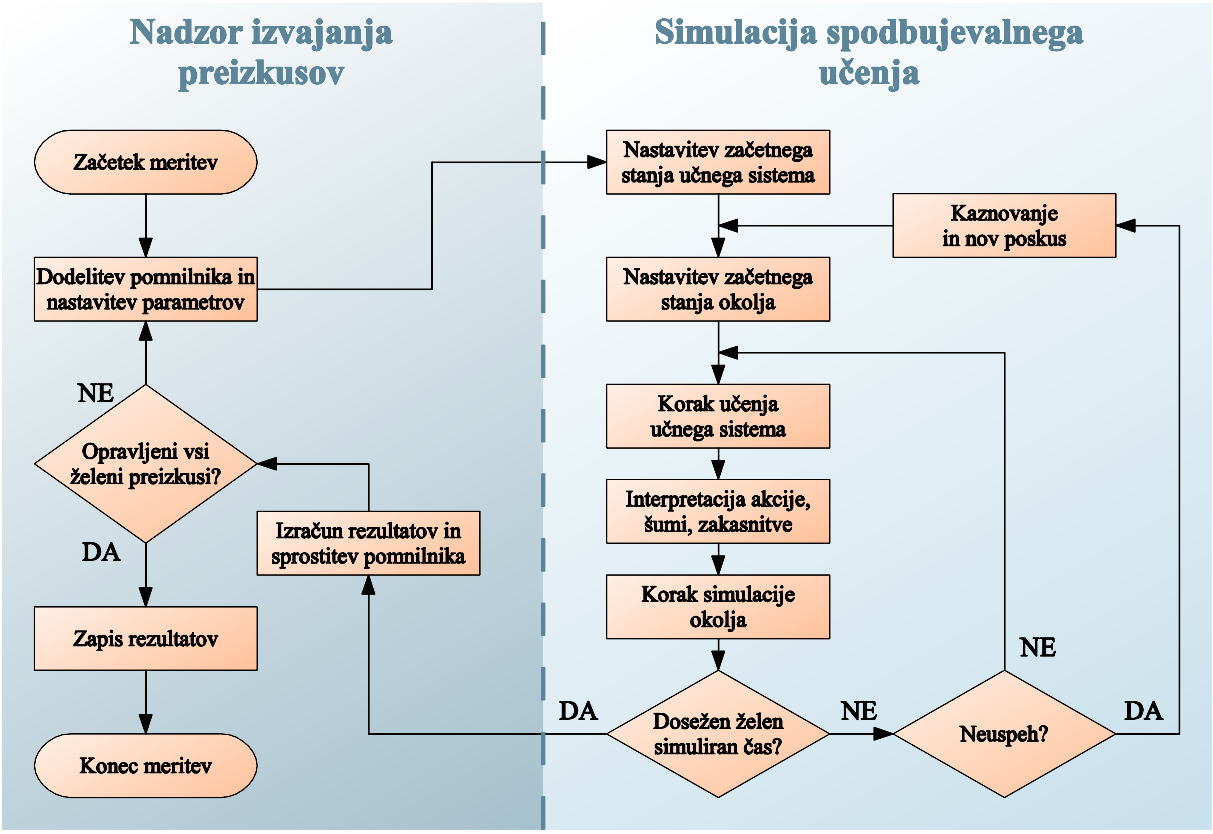
Nastavljivi parametri simulacije spodbujevalnega učenja so:

- interval učenja,
- čas učenja, podan v časovnih enotah ali v številu korakov učenja,
- začetno stanje okolja: začetni odmik vozička od sredine podlage ter začetni odklon palice od navpične lege,

- vsi parametri učnega sistema v okolju (sistema vozička s palico, vmesnika učenje-okolje, algoritma učenja).

5.2 Nadzor izvajanja preizkusov

Ta programski sklop določa okoliščine izvajanja preizkusov ter shranjuje njihove rezultate. Posamezne funkcije vidimo na sliki 5.2, kjer je prikazan diagram poteka izvajanja preizkusov. Na začetku vsakega preizkusa se najprej dodelijo viri – pomnilnik – za simulacijo in nastavijo želene vrednosti vseh nastavljivih parametrov simulacije spodbujevalnega učenja. Nato se simulacija izvede za omejen čas učenja. Po izvedbi se izračunajo in shranijo povprečne vrednosti rezultatov ter sprosti pomnilnik. Število preizkusov določi uporabnik.



Slika 5.2. Diagram poteka izvajanja preizkusov s simulacijo spodbujevalnega učenja.

Nadzor izvajanja meritev

V nadzoru izvajanje meritev je implementirana nastavitev želenih vrednosti vseh parametrov ter izvajanje meritev s simulacijo spodbujevalnega učenja. Ustrezni merski podatki so posredovani sklopu za izračun in shranjevanje rezultatov. V trenutku,

ko so opravljeni vsi preizkusi, se sporoči istemu sklopu, naj zapiše podatke na trdi disk.

Dodatne programske zmogljivosti:

- sprotni prikaz poteka preizkusov na standardnem izhodu,
- možnost izvajanja meritev samo računske zahtevnosti posameznih sklopov simulacije spodbujevalnega učenja,
- programska upočasnitev računanja iteracij simulacije na poljuben časovni interval z uporabo sistemskega ukaza za **zamrznitev procesa**. Tako je omogočeno izvajanje simulacije tudi v realnem času. To je bistvenega pomena za učinkovito interakcijo človeškega uporabnika s simulacijo, saj bi se sicer ta odvijala prehitro.

Izračun in shranjevanje rezultatov

V sklopu izračuna in shranjevanja rezultatov je implementirano sprotno shranjevanje in izračun rezultatov med posameznim preizkusom. Omogočeno je sprotno dinamično dodeljevanje in sproščanje pomnilnika, kar je bistveno za zmanjševanje porabe pomnilnika zaradi razmeroma velikega števila preizkusov ter merskih podatkov (podrobneje opisano v poglavju 6). Končna oblika shranjevanja izračunanih vrednosti je implementirana z zapisom tekstovnih datotek na trdi disk po opravljenih vseh preizkusih. Pri tem lahko uporabnik poljubno nastavi pot shranjevanja, imena datotek ter ločljivost zapisa rezultatov – to pomeni število točk po eni dimenziji pri morebitnem prikazu z grafom.

5.3 Grafični uporabniški vmesnik

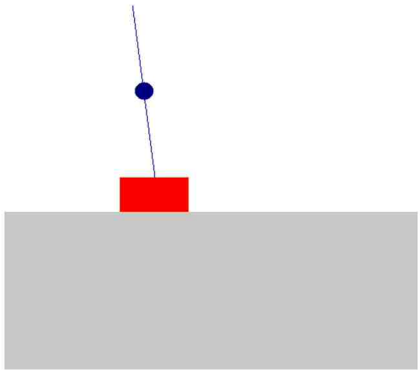
Grafični uporabniški vmesnik je namenjen lažji predstavitvi delovanja simulacije vozička s palico in simulacije spodbujevalnega učenja ter interakciji uporabnika s simulacijo v smislu možnosti apliciranja potisne sile. Na tak način je omogočeno preizkušanje odziva učnega sistema na uporabniške motnje. Na sliki 5.3 je primer zaslonskega posnetka prikaza trenutnega stanja simulacije.

Sklop je sestavljen iz vhodnega sistema ter dela za prikaz grafike. Vhodni sistem zajema uporabniški vnos tako, da preverja dogodke na vhodno/izhodnih napravah računalnika. Grafični prikaz pa deluje v zanki po naslednjih korakih:

- prejem novega stanja simulacije
- izračun nove slike stanja: razmestitev posameznih gradnikov po risalni površini

- izbris slike prikaza prejšnjega stanja
- prikaz nove slike stanja
- zamrznitev procesa za določen čas

Zamrznitev procesa za določen čas je nujno potrebna, da ne obremenimo polno centralne procesne enote. Ta čas določa število prikazanih slik na sekundo. Vse možnosti, ki jih vmesnik ponuja uporabniku, so opisane v naslednjem podpoglavju.



Slika 5.3. Zaslonski posnetek grafičnega prikaza aplikacije.

5.4 Uporaba programa

Nastavitvena datoteka

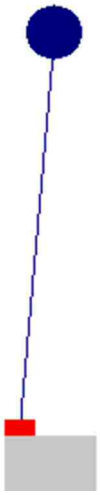
Aplikacijo smo poskušali razviti tako, da je čim bolj prilagodljiva. V ta namen smo implementirali funkcijo, ki ob zagonu prebere zunanjo tekstovno datoteko, iz katere razbere vrednosti parametrov simulacije ter nastavitve grafičnega uporabniškega vmesnika (npr. ločljivost prikaza, število slik na sekundo, ipd.). Datoteko zato poimenujemo **nastavitvena datoteka**. V njej lahko s poljubnim urejevalnikom besedil določimo vrednosti nastavljivih parametrov celotnega sistema. Spreminjanje vrednosti je trivialno zaradi pregledne oblike zapisa. V primeru, da omenjena datoteka manjka ali je zapis nepravilen, se parametri aplikacije nastavijo na določene privzete vrednosti.

Grafični vmesnik je zmožen prilagajanja prikaza glede na izbrano ločljivost ter na nastavljene vrednosti parametrov simulacije. Na sliki 5.4 je primer prikaza za dolgo palico s težiščem na koncu in veliko večjo maso od vozička, pri kratki dolžini podlage.

Potek izvajanja programa z uporabniškim vmesnikom

Ob zagonu aplikacije se začne izvajati učenje z največjo možno hitrostjo, ki je odvisna od računske moči centralne procesne enote. Takrat grafični prikaz prikazuje izredno

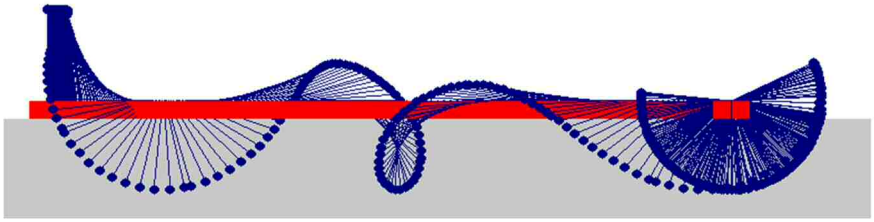
hitre premike po zaslonu, saj simulacija ni upočasnjena na resnični čas. Ko učni sistem doseže določeno število korakov brez da bi prišlo do neuspeha ali ko doseže določeno število neuspehov, se simulacija upočasni na resnični čas, tako da je omogočeno opazovanje obnašanja naučenega učnega sistema.



Slika 5.4. Grafični prikaz poljubne nastavitve parametrov.

Interakcija uporabnika

Uporabnik lahko vpliva na simulacijo tako, da potiska voziček v eno ali drugo smer z različno velikimi potisnimi silami. Kadarkoli med izvajanjem je mogoč izklop ali vklop učnega sistema. Na tak način lahko tudi preklapljamo med največjo hitrostjo simuliranja in simulacijo v resničnem času. Hitrost simulacije v resničnem času lahko povečujemo ter zmanjšujemo s faktorjem dva. Dodatna možnost je izklop in vklop brisanja slik prikaza grafičnega vmesnika. S tem lahko dosežemo, da ostane sled premikanja vidna na zaslonu, kot prikazano na sliki 5.5.



Slika 5.5. Grafični prikaz brez brisanja slik.

Primer uporabe z grafičnim uporabniškim vmesnikom

Navedli bomo praktičen primer, ki prikazuje sposobnost učnega sistema prilagajanju okolja. Ko učni sistem doseže določeno uspešnost in se hitrost simulacije nastavi na resnični čas, lahko skuša uporabnik z apliciranjem potisne sile spraviti sistem v stanje

neuspeha – voziček na rob podlage ali palico na tla. Če uporabnik naključno vnaša v okolje potisne sile, ki so manjše od potisne sile, ki jo daje učni sistem, potem ni bistvenega vpliva na stanje. V primeru pa, da uporabnik vnaša v okolje silo enako veliko kot potisno silo, ki jo daje učni sistem, se mora učni sistem prilagoditi tako, da začne v okolje ves čas pošiljati silo, ki je nasprotno enaka uporabnikovi – npr. če uporabnik potiska voziček vedno v levo s silo 10 N, bo moral učni sistem ves čas potiskati v desno s silo 10 N – saj samo tako lahko ohranja stanje brez neuspeha. S praktičnim poskusom ugotovimo, da se učni sistem v tem primeru prilagodi v povprečju po manj kot deset neuspehih.

Izvajanje meritev

Pri izvajanju meritev je grafični uporabniški vmesnik izključen, sej se program vedno izvaja brez zamrznitev s polno zmogljivostjo centralne procesne enote. Na standardni izhod se izpisuje potek meritev, tako da lahko razberemo, kolikšen delež je že bil opravljen do nekega trenutka. Po končanih meritvah se za vsak merski parameter zapiše tekstovna datoteka z rezultati. Iz teh datotek lahko z uporabo različnih računalniških aplikacij tvorimo bolj pregleden prikaz v obliki grafov in tabel.

5.5 Izvorna koda

Celoten program je napisan v programskem jeziku C++. Obsega približno 3000 vrstic strukturirane programske kode, znotraj katerih smo implementirali približno 30 procedur in 15 različnih objektov ter uporabili 10 knjižnic.

Programiranje je bilo objektno naravnano [21] z velikim poudarkom na modularnosti in ponovni uporabnosti programske kode, pri čemer smo se delno zgledovali po *The Reinforcement Learning Toolbox, Reinforcement Learning for Optimal Control Tasks* [16]. Zato je vsak sklop samostojna enota, kot smo podrobneje opisali v prejšnjem in tem poglavju. Izbrane so bile take knjižnice, ki delujejo na več različnih platformah (ang. cross-platform [21]), tako da se lahko izvorno kodo prevede in izvaja na različnih operacijskih sistemih. Poleg modularnosti smo stremeli tudi k čim boljši optimizaciji računske zahtevnosti posameznih sklopov.

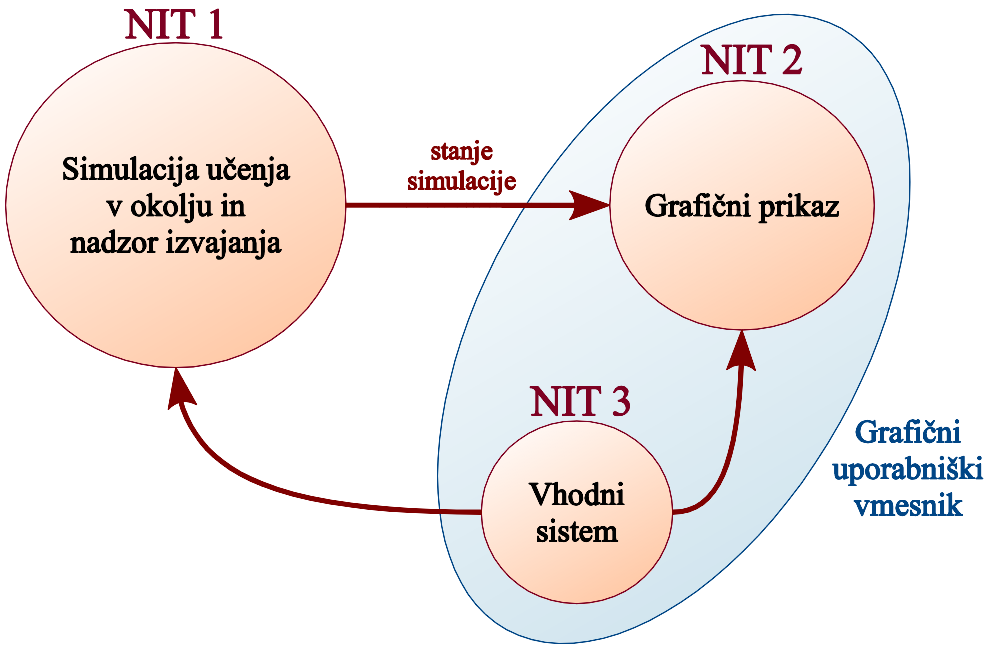
Simulacija v realnem času in omejitev števila slik zahtevata uporabo sistemskih klicev za zamrznitev procesa za določen čas. Zaradi tega smo programsko kodo razdelili na več niti, kot prikazuje slika 5.6. Tako smo omogočili časovno neodvisno izvajanje posameznih sklopov. Grafični prikaz lahko izvaja zanko osveževanja slike s polju-

bnim številom slik na sekundo – do neke zgornje meje, ne da bi pri tem vplival na hitrost osveževanja simulacije.

Uporabljene knjižnice in viri

Za pomoč in reference pri programiranju smo uporabljali spletne vire namenjene programskemu jeziku C++ [4,5]. Grafični uporabniški vmesnik ter večnitnost smo implementirali s knjižnico Simple DirectMedia Layer (SDL [18]). Za osvojitev znanja uporabe te knjižnice smo se zgledovali po učnih tečajih [14,17].

Pri implementaciji simulacije učnega sistema v okolju so nam bili v veliko pomoč primeri izvornih kod za simulacijo vozička s palico, vmesnika med okoljem ter učnim sistemom, algoritma učenja ASE/ACE in Q-učenja [7,8,12,15].



Slika 5.6. Interakcija niti.

Uporabljena programska oprema

Program smo razvili z orodjem Microsoft Visual Studio 2010 na operacijskem sistemu Windows 7. Izdelava diplomskega poročila, shem in grafov je bila narejena s programskimi paketi orodji Microsoft Office, OpenOffice ter paint.net. Pomožna orodja, ki smo jih uporabili so Adobe Reader X, Ghostscript, Ghostview, GSView, BibTeX, JabRef, Notepad++ in Google Chrome in Corel PDF Fusion.

6 Merjenje uspešnosti

Želimo ugotoviti vpliv vseh možnih dejavnikov na uspešnost algoritmov spodbujevalnega učenja. Dejavniki so izraženi kot vhodni parametri simulacije. Število kombinacij vrednosti vseh teh parametrov je ogromno, zato bomo merili vpliv posameznih izbranih parametrov tako, da bomo spreminjali vrednost samo izbranega parametra, vrednosti ostalih parametrov pa bodo ostale nespremenjene. Izvajanje meritev smo sistematično razčlenili na posamezne manjše sklope; premišljeno smo izbrali ustrezne merske vrednosti, ki verodostojno odražajo uspešnost algoritma in vse omenjene parametre smo razdelili po skupinah. Vse navedeno je podrobneje opisano v tem poglavju.

6.1 Posamezni sklopi in potek izvajanja meritev

Nekateri izmed sklopov meritev, ki jih bomo navedli v nadaljevanju, so že bili omenjeni v preteklih poglavjih. V tem razdelku jih bomo opisali z vidika izvajanja meritev ter jih umestili poleg novih izrazov. Potek izvajanja meritev ter delitev na sklope po obsegu izvajanja prikazuje diagram poteka na sliki 6.1.

Sklopi po obsegu izvajanja od najmanjšega proti največjemu:

1. **Korak učenja** (v nadaljevanju: korak)

   To je najmanjša enota. V sklopu koraka se izvede ena iteracija izračunov simulacije spodbujevalnega učenja. Njegovo časovno trajanje je določeno z intervalom učenja.

2. **Poskus**

   To je število korakov med posamezno nastavitvijo začetnega stanja okolja ter **neuspehom**. Drugače povedano, vsakič, ko pride do neuspeha in kaznovanja učnega sistema, se trenutni poskus zaključi in začne se nov poskus.

3. **Tek**

   To je izvedba določenega števila korakov. Tek se vedno začne s poskusom številka 1. Število neuspehov je vedno za 1 manjše od števila poskusov, ker se zadnji poskus ne konča z neuspehom, temveč s prekinitvijo simulacije zaradi doseženega časa učenja. Čas učenja (število korakov) je znan ter spada med parametre simulacije,

število poskusov in neuspehov pa je neznano, zato smo ga uvrstili k merskim rezultatom.

4. **Preizkus nabora parametrov** (v nadaljevanju: preizkus nabora)

   To je izvedba več ponovitev tekov pri enakih vrednostih parametrov simulacije – enakem naboru parametrov. Pred začetkom tega preizkusa se nastavijo vrednosti parametrov, ki se med izvajanjem preizkusa ne spreminjajo. Število ponovitev tekov je znano in spada med parametre izvajanja meritev. Rezultat preizkusa so povprečne vrednosti vseh ponovitev, zato ta sklop opravlja nalogo statistične verodostojnosti merskih vrednosti.

5. **Preizkus vpliva izbranega parametra** na uspešnost algoritma učenja

   To je izvedba več preizkusov nabora, pri čemer spreminjamo vrednost samo enega izbranega parametra simulacije. Tako ugotavljamo na kakšen način se spreminja uspešnost algoritma učenja na določeni množici vrednosti izbranega parametra. Ta množica se razlikuje od parametra do parametra ter spada med parametre izvajanja meritev.

6.2 Parametri

V preteklih poglavjih smo za vsak programski sklop posebej navajali nastavljive parametre. Na tem mestu jih povzamemo ter razdelimo v skupine glede na to, kateremu sklopu pripadajo. Izpustili smo tiste, katerih vpliva ne bomo merili.

Seznam vseh parametrov meritev (skupine parametrov so obarvane z rdečo, posamezni parametri pa s črno):

- **Parametri izvajanja meritev**
  - število ponovitev tekov za posamezen preizkus nabora
  - množica vrednosti za vsak parameter simulacije, pri katerih bomo preizkušali vpliv
- **Parametri simulacije**
  - Parametri izvajanja simulacije
    - interval učenja
    - čas učenja ali število korakov
  - Parametri okolja
    - Parametri simulacije vozička s palico: dolžina palice (razdalja do težišča), masa palice, časovni korak simulacije vozička s palico

- Parametri vmesnika učenje-okolje: dolžina podlage, velikost sile potiska, vhodni in izhodni šum, vhodna in izhodna zakasnitev
- Parametri nastavitve začetnega stanja okolja: začetni odklon palice

➢ Parametri učenja

- Parametri učenja ASE/ACE: alfa (α), beta (β), gama (γ), sigma (δ), lambda (λ)
- Parametri Q-učenja: alfa (α), beta (β), gama (γ)

Število ponovitev tekov bomo določili pred pričetkom opravljanja meritev tako, da bo za vse meritve enak. Posamezno množico vrednosti preizkušanja vpliva parametra pa bomo določali sproti glede na dobljene rezultate. Merili bomo kakšen vpliv ima vsak parameter simulacije na uspešnost posameznega algoritma spodbujevalnega učenja. Rezultati so prikazani v poglavju 7.

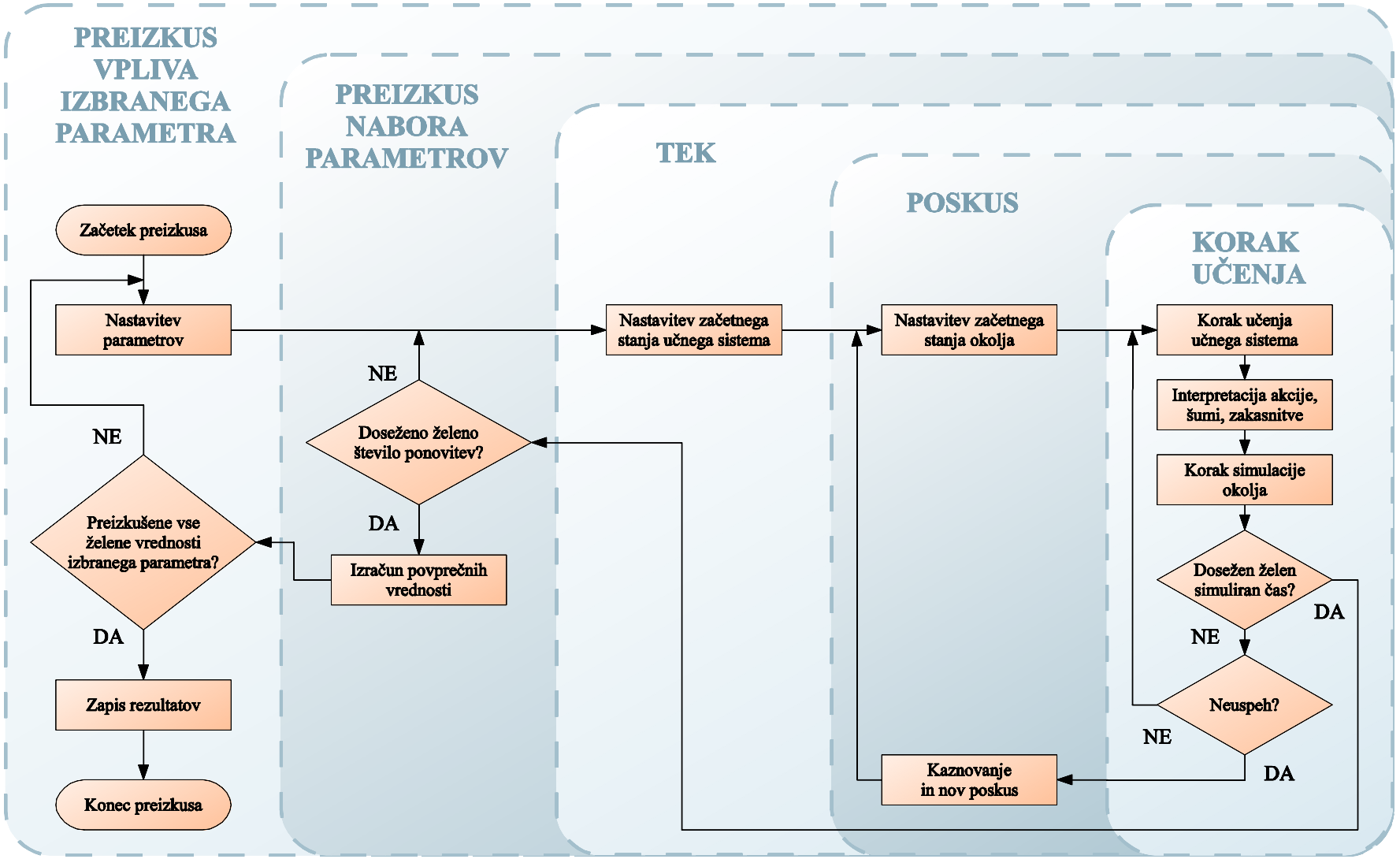
6.3 Pokazatelji uspešnosti

Za lažje razumevanje in primerjavo vpliva različnih parametrov na uspešnost algoritma učenja smo morali iz velikega števila tekov, poskusov in korakov pridobiti eno-številčne pokazatelje uspešnosti. V nadaljevanju je razloženo, na kakšen način smo prišli do njih.

Merski rezultat enega teka je število neuspehov in njihova časovna porazdelitev – časovni trenutki, ko je prišlo do posameznih neuspehov. Število in časovna porazdelitev neuspehov sta neposredno povezana s številom poskusov in trajanjem vsakega poskusa, kot že omenjeno v podpoglavju 6.1.

Z vidika samega okolja bi lahko merili povprečen odklon palice od kota 0° ali povprečen odmik vozička od sredine podlage. Toda ta dva pokazatelja bi bila nesmiselna, saj je učnemu sistemu pomembno le, da ne pride do neuspeha – to je padca palice ali prevelikega odmika vozička, saj le takrat dobi negativno spodbudo oz. kazen. Torej so v pogledu učnega sistema vsa druga stanja dobra, ne glede na velikost odklona palice ali odmika vozička.

Iz navedenega je razvidno, da imamo na razpolago merske podatke o številu neuspehov in njihovi časovni porazdelitvi pri več ponovitvah tekov z enakim časom učenja, iz katerih moramo pridobiti verodostojne eno-številčne pokazatelje uspešnosti za posamezen preizkus nabora. Kot verodostojen pokazatelj mislimo na to, da ne sme izkazovati trenutne ali delne uspešnosti (npr. najdaljše trajanje poskusa izmed vseh poskusov), temveč končno uspešnost po pretečenem času učenja ali povprečno tekom



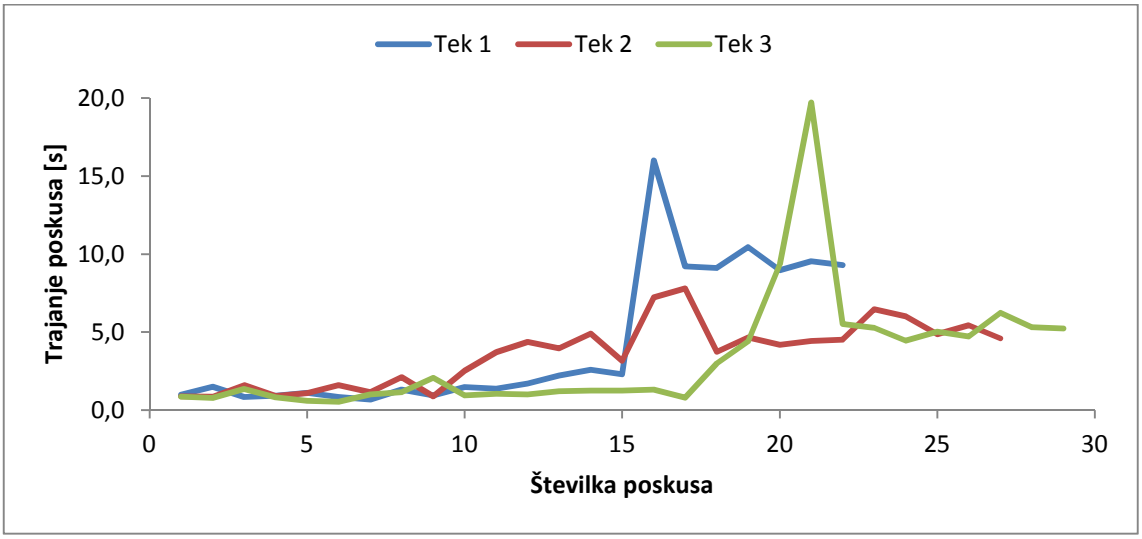
Slika 6.1. Diagram poteka meritev po sklopih izvajanja.

celotnega čas učenja. Pokazatelj mora biti verodostojen tudi v statističnem smislu – mora omogočati izračun povprečne vrednosti iz več ponovitev tekov, tako da se pri večanju števila ponovitev natančnost povprečnega izračuna tudi povečuje.

V naslednjih podpoglavjih sledijo obrazložitve dobrih in slabih pokazateljev ter naša izbira.

6.4 Trajanje in število poskusov

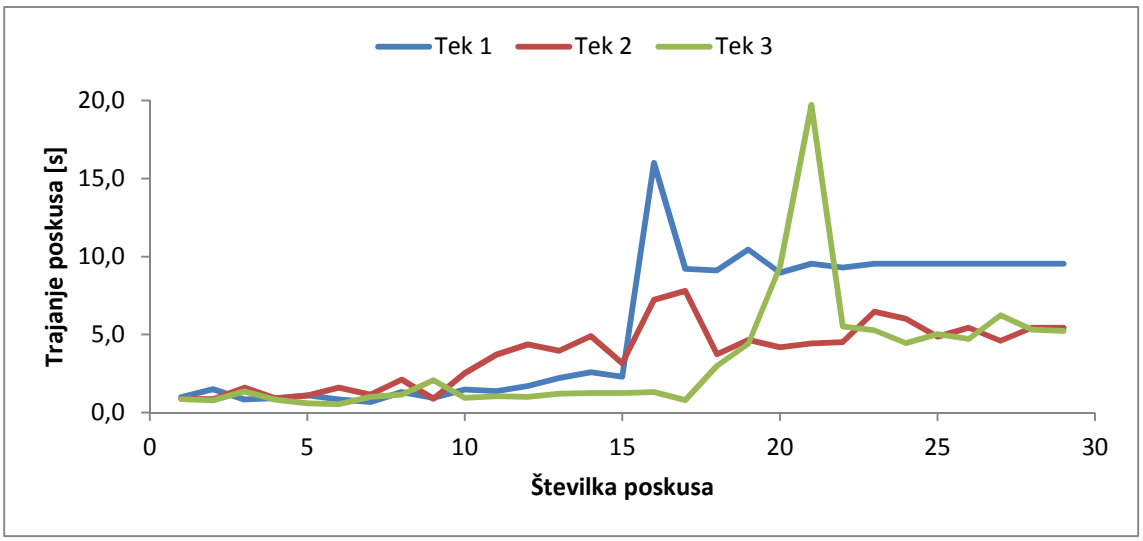
Prvi prikaz uspešnosti algoritma učenja, ki ga bomo opisali, je tak kot so ga uporabili Barto in sod. [2]. Opazujemo trajanje posameznih poskusov z naraščanjem števila poskusov v posameznem teku. Primer takega grafa za tri teke prikazuje slika 6.2. Na grafu je razvidno število vseh poskusov, iz trajanj posameznih poskusov pa je posredno razvidna tudi časovna porazdelitev neuspehov v času učenja. Algoritem učenja sčasoma postaja bolj uspešen. To je vidno v tem, da trajanje posameznega poskusa v povprečju narašča z naraščanjem števila poskusov. Za boljšo uspešnost smatramo, če je končno število poskusov v času učenja manjše, ter če trajanje posameznih poskusov z večanjem števila poskusov hitreje narašča.



Slika 6.2. Graf trajanja posameznih poskusov za tri teke.

Pridobitev eno-številčnega pokazatelja uspešnosti iz omenjenih vrednosti je težavna. Trajanje posameznih poskusov z večanjem števila poskusov ne narašča vedno, temveč niha. Zato trajanje najdaljšega poskusa – to je najbolj uspešnega poskusa – ni dober pokazatelj ne končne, ne povprečne uspešnosti. Iz istega razloga tudi trajanje zadnjega poskusa ni dober pokazatelj. Poleg tega je trajanje zadnjega poskusa vedno prekinjeno zaradi izteka časa učenja in ne zaradi neuspeha učnega sistema, kar naredi

to vrednost mersko nezanesljivo v vsakem primeru. Še večja težava je v tem, da je število poskusov na posamezen tek različno, zato povprečenje trajanja posameznega poskusa po več tekih ni mogoče. Barto in sod. [2] so skušali to težavo zaobiti z navideznim podaljšanjem števila poskusov, tako kot prikazuje graf na sliki 6.3.



Slika 6.3. Graf navideznega podaljšanja števila poskusov pri treh tekih.

Vsakemu teku so dodali toliko navideznih poskusov, da so jih vsi imeli enako število – tako so vsi teki imeli število poskusov enako teku z največ poskusi (na omenjenem grafu je to tek 2). Za trajanje navideznih tekov je bilo predpostavljeno, da je enako trajanju zadnjega teka ali predzadnjega teka, če je daljši. Po opisani metodi je izračun povprečne vrednosti iz več tekov možen, toda smiselnost teh rezultatov je izničena zaradi naslednjih dejstev:

- Kot smo že omenili, trajanje poskusov zelo niha. Iz tega razloga predpostavka, da imajo vsi bodoči poskusi enako ali vsaj približno enako trajanje kot zadnji poskus, ni dobra.
- Z navideznim dodajanjem poskusov istočasno navidezno povečamo čas učenja, ki ga je imel posamezen tek na razpolago. Tako računamo povprečne vrednosti iz tekov, ki so se navidezno izvajali različno dolgo, kar je seveda napaka. Tabela 6.1 prikazuje navidezno podaljšan čas učenja zaradi navideznega dodajanja števila poskusov.
- Ker navidezno povečamo število poskusov do števila poskusov teka z največ poskusi, se natančnost izračuna povprečne vrednosti ne povečuje z naraščanjem števila tekov, temveč pri vsakem številu tekov limitira proti drugačni vrednosti.

Iz navedenega zaključimo, da opisana metoda ni dobra in da po njej ne moremo dobiti zanesljivega eno-številčnega pokazatelja uspešnosti algoritma učenja.

Tabela 6.1. Navidezno dodajanje tekov ter podaljšanje časa učenja.

	Število dodanih navideznih tekov	**Navidezen celoten čas učenja [s]**
Tek 1	7	166,8
Tek 2	2	110,9
Tek 3	0	100,0

6.5 Povprečno trajanje poskusa

Za vsak tek lahko enostavno izračunamo povprečno trajanje enega poskusa po enačbi (6.1). Ta pokazatelj odraža povprečno uspešnost za čas učenja in ga lahko povprečimo po tekih. Sam po sebi pa ne vsebuje podatka o časovni porazdelitvi začetkov poskusov – to je pojavitev neuspehov – kar pa je bistvenega pomena. Npr. vzemimo 2 teka, oba z enakim številom neuspehov po enako dolgem času učenja. Pri prvem teku so se vsi neuspehi zgodili v prvih 10 % časa učenja in nato preostalih 90 % časa ni bilo nobenega. Pri drugem teku pa so se neuspehi dogajali enakomerno čez celoten čas učenja. Samoumevno je, da je bil prvi tek bistveno bolj uspešen kot drugi, kljub enakemu številu neuspehov, enakemu številu poskusov in enakemu povprečnemu trajanju enega poskusa. Tak primer prikazuje tabela 6.2 v podpoglavju 6.7. Torej moramo uporabiti še nek drugi pokazatelj, ki ga bomo opisali v nadaljevanju poglavja.

$$\text{Povprečno trajanje poskusa} = \frac{\text{čas učenja}}{\text{število poskusov}} \tag{6.1}$$

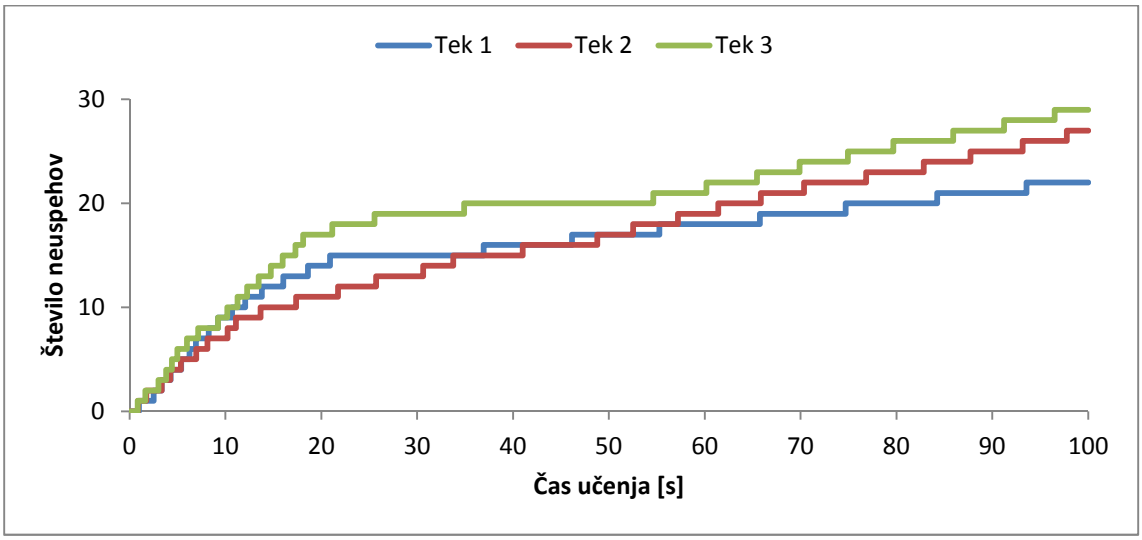
Poleg navedenega, zadnji poskus se ne konča z neuspehom, temveč s prekinitvijo zaradi izteka časa učenja (kot že omenjeno prej). To pomeni, da podatka o dejanskem trajanju tega poskusa nimamo, kar nekoliko pokvari natančnost izračuna povprečja trajanja enega poskusa. Težavi se izognemo, če namesto števila poskusov opazujemo število neuspehov.

6.6 Število neuspehov

Z vidika spodbujevalnega učenja lahko merimo uspešnost po seštevku vseh spodbud v času učenja. V okolju, kot smo ga mi implementirali, učni sistem prejema kazen -1 samo ob neuspehih, sicer pa je spodbuda vedno enaka 0. To pomeni, da je seštevek spodbud enak številu neuspehov v posameznem teku (le drugače predznačen). Število

neuspehov torej prikazuje končno uspešnost po določenem času učenja. Poleg tega ga lahko enostavno povprečimo po tekih. Ker pa je neposredno povezan s številom poskusov, nastopi ista težava – da sam po sebi ne vsebuje informacije o časovni porazdelitvi neuspehov.

Graf na sliki 6.4 prikazuje naraščanje števila neuspehov po času učenja za tri primere tekov. Iz oblike posamezne krivulje je razvidna časovna porazdelitev neuspehov posameznega teka. Bolj kot je krivulja podobna logaritemski, bolj smatramo, da je algoritem uspešen. To pomeni, da želimo, da bi bil čim večji delež neuspehov čim bolj na začetku časa učenja ter čim manjši delež neuspehov proti koncu časa učenja. Z izračunom ukrivljenosti krivulje lahko dobimo pokazatelj časovne porazdelitve neuspehov, ki skupaj s številom neuspehov odpravita prej omenjeno težavo. Ta pokazatelj je **časovna sredina neuspehov** .



Slika 6.4. Graf naraščanja števila neusephov po času učenja za tri teke.

6.7 Časovna sredina neuspehov

Ta pokazatelj je zasnovan na ukrivljenosti krivulje števila neuspehov po času učenja (slika 6.4). Izračunan je kot povprečna vrednost časov nastanka vseh neuspehov ter izražen v odstotkih časa učenja, po enačbi (6.2). V primeru linearne krivulje bo izračunana vrednost enaka 50 %, v primeru eksponentne krivulje se bo pomikala proti 100 % ter v primeru logaritemske oblike krivulje proti 0 %. Iz tega sledi, da manjša kot je vrednost tega pokazatelja, večja je uspešnost algoritma. Ta vrednost omogoča tudi enostavno povprečenje po tekih. Tabela 6.2 prikazuje primer, ko vsi do zdaj opisani pokazatelji pokažejo enako vrednost, le omenjeni pa pokaže bistveno razliko v uspešnosti.

$$\text{Časovna sredina neuspehov} = \frac{\sum_i^{\text{število neuspehov}} (\text{čas nastanka neuspeha})_i}{\text{število neuspehov}} * \frac{100\ \%}{\text{čas učenja}} \tag{6.2}$$

Tabela 6.2. Primer dveh različno uspešnih tekov z enakim številom poskusov in neuspehov.

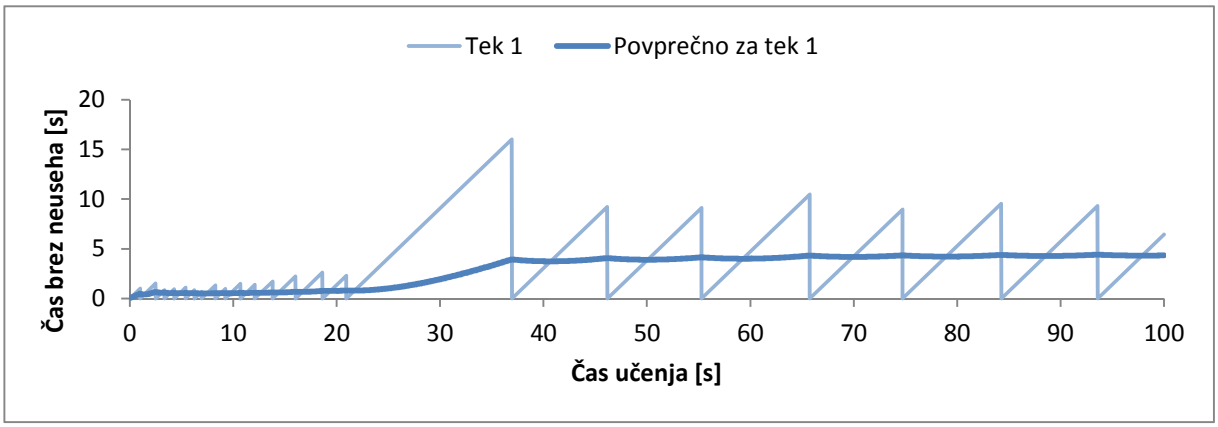
	Potek poskusov	Čas učenja [s]	Število poskusov	Povprečno trajanje poskusa [s]	Število neuspehov	Časovna sredina neuspehov [%]
Tek 1	9 poskusov s trajanjem vsak 1 sekundo ter nato 1 poskus s trajanjem 999 sekund	1000	10	100	9	4,5
Tek 2	10 poskusov s trajanjem vsak 100 sekund	1000	10	100	9	45

6.8 Povprečen čas brez neuspeha

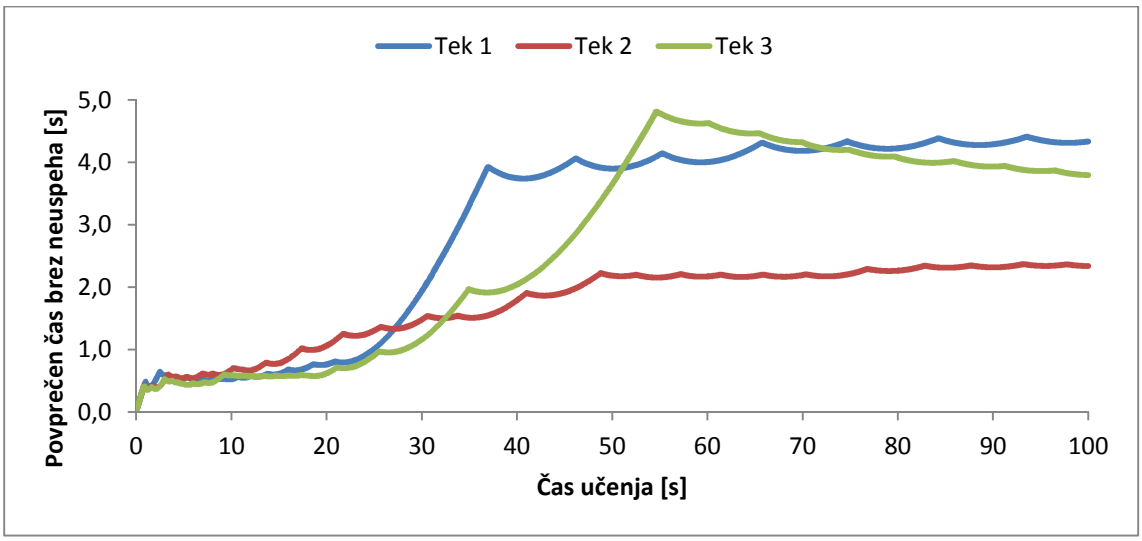
Na uspešnost algoritma učenja lahko gledamo tudi z vidika okolja. Kot pokazatelj uspešnosti čez celoten čas učenja nas zanima, koliko časa v povprečju sta bila palica in voziček v ravnotežju. To pomeni, koliko časa v povprečju ni prišlo do neuspeha. Nazobčana krivulja na grafu 6.5 prikazuje pretečen čas od zadnjega neuspeha za vsak trenutek časa. Tako je povprečna višina te krivulje do nekega trenutka enaka **povprečnemu času brez neuspeha** do istega trenutka. Izračunamo ga tako, da površino pod krivuljo delimo po času. Površino krivulje izračunamo iz vsote posameznih površin trikotnikov, katerih stranico določa časovni razmik med dvema neuspehoma – to je enako trajanju posameznega poskusa. Posledično lahko povprečno vrednost izračunamo po enačbi (6.3) in je izražena s časovno enoto v sekundah. V enačbi nastopa kvadratna funkcija, zato ta pokazatelj vsebuje, poleg informacije o končnem številu neuspehov, tudi informacijo o časovni porazdelitvi neuspehov v času učenja. Dobili dober pokazatelj, ki združuje lastnosti pokazateljev časovne sredine neuspehov ter števila neuspehov.

$$\text{Povprečen čas brez neuspeha} = \frac{\sum_i^{\text{število poskusov}} (\text{trajanje poskusa})_i^2}{\text{čas učenja}} * \frac{1}{2} \tag{6.3}$$

Povprečen čas brez neuspeha ne vsebuje informacije o tem, ali je pogostost neuspehov večja na začetku ali na koncu simulacije. V našem primeru to ni težava, saj je pričakovano, da pogostost neuspehov pada z naraščanjem časa učenja, ker postaja učni sistem vedno bolj uspešen. Tako se tudi izkaže pri meritvah. Graf na sliki 6.6 prikazuje povprečen čas brez neuspeha za tri teke.



Slika 6.5. Graf časa brez neuspeha ter povprečnega časa brez neuspeha po času učenja.



Slika 6.6. Povprečen čas brez neuspeha po času učenja za tri teke.

6.9 Primerjava in povprečenje pokazateljev

Primerjava pokazateljev uspešnosti posameznih tekov

Grafi na slikah 6.2, 6.4, 6.5 in 6.6 v prejšnjih podpoglavjih se vsi nanašajo na isto meritev treh tekov. Tako lahko opazujemo uspešnost tekov iz različnih zornih kotov. Ker hočemo primerjati, kateri je boljši, jih moramo razvrstiti po uspešnosti. To je

težka naloga, če opazujemo le krivulje na grafih. Navedeno upravičuje smiselnost uporabe opisanih eno-številčnih pokazateljev uspešnosti.

Izbrani pokazatelji uspešnosti teka po pretečenem času učenja so: število neuspehov, časovna sredina neuspehov in povprečen čas brez neuspeha. V tabeli 6.3 so prikazani z modro barvo za omenjene tri teke. Slabi pokazatelji, opisani v prejšnjih podpoglavjih, so prikazani z rdečo barvo. S pomočjo teh vrednosti lahko podane tri teke enostavno razvrstimo po uspešnosti. Tek 1 je bistveno bolj uspešen od ostalih dveh po vseh pokazateljih, razen po najdaljšem trajanju poskusa, kar potrjuje neustreznost tega pokazatelja. Z vidika povprečnega trajanja poskusa je tek 2 boljši od teka 3. Ampak tek 3 ima kljub večjemu številu neuspehov boljši povprečni čas brez neuspeha kot tek 2. Razlog je v časovni sredini neuspehov, ki je pri teku 3 bistveno nižja. Iz navedenega zaključimo, da je tek 3 bolj uspešen kot tek 2, čeprav ima nižje povprečno trajanje poskusa. Tako je potrjena neustreznost tudi povprečnega trajanja poskusa kot pokazatelja uspešnosti.

Tabela 6.3. Primer izračuna povprečja ter odklona pokazateljev uspešnosti.

	Število neuspehov	**Časovna sredina neuspehov [%]**	**Povprečen čas brez neuspeha [s]**	**Povprečno trajanje poskusa [s]**	**Najdaljše trajanje poskusa [s]**
Tek 1	22	27,0	4,3	4,3	16,0
Tek 2	27	38,0	2,3	3,6	7,8
Tek 3	29	31,4	3,8	3,3	19,7
Povprečje	**26,0**	**32,1**	**3,5**		
Odklon [%]	11,3	14,1	24,2		

Pokazatelji uspešnosti nabora parametrov

Uspešnost preizkusa nabora parametrov je izražena s povprečnimi vrednostmi pokazateljev uspešnosti, ki so izračunani iz več ponovitev tekov. Kot pomožne pokazatelje izračunamo tudi standardne odklone ter jih izrazimo v odstotkih povprečne vrednosti. Odklone računamo po enačbi (6.4).

Odkloni vsebujejo informacijo o tem, kolikšen delež ponovitev tekov doseže nizko uspešnost ter kolikšen delež doseže visoko uspešnost relativno na vse ponovitve. Večji kot je odklon, večji je delež tekov, ki ni dosegel visoke uspešnosti. Iz tega vidika lahko smatramo odklon kot pokazatelj zanesljivosti učnega sistema, da bo pri danih okoliščinah dosegel enako ali višjo uspešnost, kot je izračunana povprečna vrednost. Iz

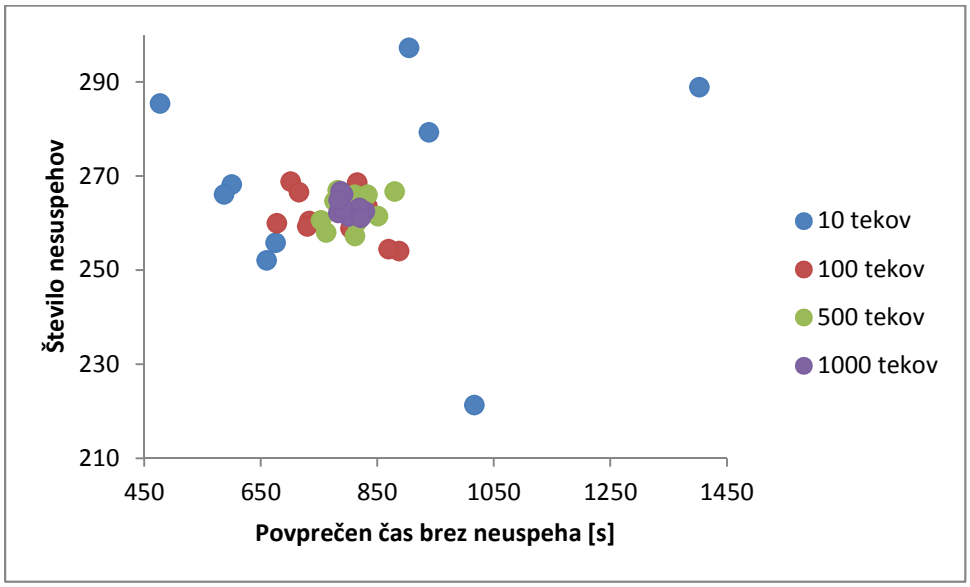
navedenega sledi, da je manjši odklon odraz večje zanesljivosti in posledično boljše uspešnosti.

$$\text{Odklon} = \sqrt{\frac{\sum_i^{\text{število tekov}}(\text{vrednost teka}_i - \text{povprečna vrednost})^2}{\text{število tekov}}} * \frac{100\ \%}{\text{povprečna vrednost}} \quad (6.4)$$

Izračunane vrednosti za primer treh ponovitev tekov prikazuje tabela 6.3. Razvidno je, da ima največji odklon pokazatelj povprečen čas brez neuspeha. To je pričakovano, ker v njegovem izračunu nastopa kvadratna funkcija.

6.10 Število ponovitev tekov

Statistično verodostojnost zagotavljamo z izračunom povprečnih vrednosti iz več tekov. Ne vemo pa, kolikšno naj bo število teh ponovitev, da bo verodostojnost dovolj velika. Za kriterij smo določili nihanja med izračuni povprečnih vrednosti enako-številčnih sklopov tekov. Nihanje pri določenem številu tekov izračunamo iz deset povprečnih vrednosti pri enakem številu tekov. Upadanje razpršenosti povprečnih vrednosti z naraščanjem števila ponovitev tekov prikazuje slika 6.7. Nihanja povprečne vrednosti pri različnih številih ponovitev pa prikazuje tabela 6.4 in so izražena v odstotkih povprečne vrednosti.



Slika 6.7. Razpršenost povprečnih vrednosti v odvisnosti od števila ponovitev tekov.

Tabela 6.4. Nihanja povprečnih vrednosti pri različnih številih ponovitev tekov.

	Nihanje povprečne vrednosti [%]		
	števila neuspehov	časovne sredine neuspehov	povprečnega časa brez neuspeha
10 tekov	7,8	13,1	31,9
100 tekov	1,9	5,4	9,1
500 tekov	1,3	2,3	4,7
1000 tekov	**0,7**	**1,3**	**2,0**

Pri 1000 ponovitvah tekov smo bili z nihanji zadovoljni, zato pri večjem številu ponovitev nismo opravili meritev. Poleg tega, bi bilo večje število računsko že zelo zahtevno za naš testni sistem (podrobnosti o časovni zahtevnosti so opisane v poglavju 7). Tako smo za vse meritve določili, da bomo vsak preizkus nabora parametrov izvedli s 1000 ponovitvenimi teki.

Opombe:

Meritve nihanj, prikazane na sliki 6.7 in tabeli 6.4, so bile izvedene s privzetimi vrednostmi parametrov, ki so opisane v poglavju 8.1. Pri drugačnih vrednostih parametrov bi meritve pokazale tudi drugačna nihanja. Toda to ni bistvenega pomena, saj nihanja vedno padajo z večanjem števila ponovitev, ne glede na vrednosti parametrov.

Prikazane vrednosti nihanj so dejansko standardni odkloni, izračunani z enačbama (6.5). Poimenovanje nihanje smo uporabili z namenom, da ne bi bralec zamešal odklonov, opisanih v prejšnjem podpoglavju ter odklonov oz. nihanj, o katerih smo pisali v tem podpoglavju.

$$\text{povprečje povprečij} = \frac{\sum_i^{10}(\text{povprečna vrednost})_i}{10}$$

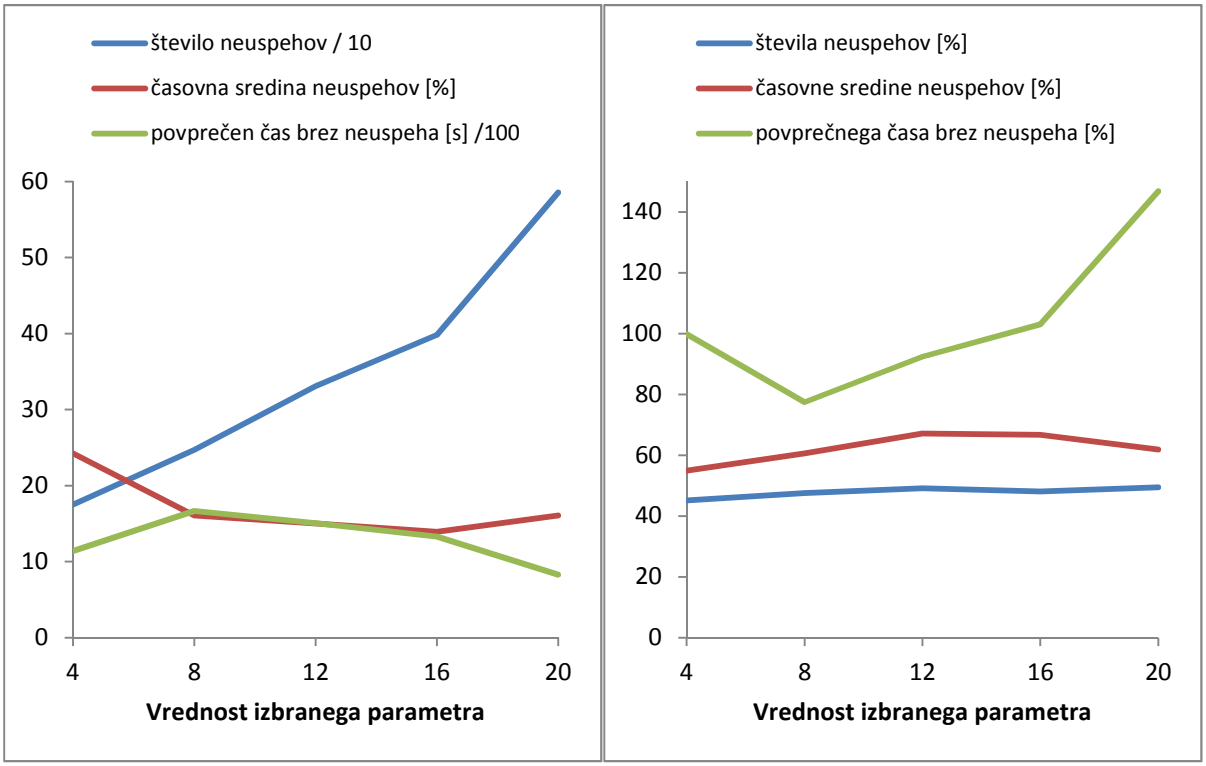
$$\text{Nihanje} = \sqrt{\frac{\sum_i^{10}((\text{povprečna vrednost})_i - \text{povprečje povprečij})^2}{10}} * \frac{100\ \%}{\text{povprečje povprečij}} \tag{6.5}$$

6.11 Preizkus vpliva parametra

Iz vsega opisanega v tem poglavju povzamemo, da vpliv nekega parametra predstavljajo pokazatelji uspešnosti nabora parametrov pri različnih vrednostih tega parametra.

Primer preizkusa vpliva nekega poljubnega parametra na uspešnost algoritma učenja tako predstavljata grafa na sliki 6.8. Levi graf prikazuje povprečne vrednosti pokazateljev uspešnosti normirane tako, da je omogočen skupen prikaz. Desni graf pa prikazuje njihove odklone.

Število grafov in krivulj za prikaz vpliva enega parametra na uspešnost algoritma z vidika vseh pokazateljev uspešnosti je razmeroma veliko. Poleg tega bomo pri meritvah v nadaljevanju primerjali dva algoritma učenja. To bi privedlo do nepraktičnega števila grafov, zato bomo to v nadaljevanju omejili, da bomo prikazali le tiste, ki podajajo največ informacije. V večini primerov bomo prikazali le graf povprečnega časa brez neuspeha, ker ta pokazatelj odraža lastnosti obeh drugih pokazateljev, kot že opisano v podpoglavju 6.8. Ostale pokazatelje pa bomo komentirali opisno, v primeru, da bodo prikazovali drugačne rezultate.



Slika 6.8. Vpliv izbranega parametra na uspešnost učnega sistema. Na **levem** grafu so prikazane **vrednosti pokazateljev** uspešnosti povprečne vrednosti, **na desnem pa odkloni** povprečnih vrednosti za primer izbranega parametra.

7 Časovna in pomnilniška zahtevnost

Časovna zahtevnost izvajanja meritev in število korakov učenja

Izvedba vseh želenih meritev mora biti opravljena v nekem sprejemljivem času. Časovna zahtevnost ene meritve je odvisna od števila potrebnih ponovitev tekov, števila korakov učenja ter procesorskega časa, potrebnega za izračun enega koraka. Število ponovitev smo že določili v poglavju 6.10. Za število korakov pa želimo, da bi bilo čim večje, zato ga bomo določili na podlagi časovne zahtevnosti izračuna enega koraka.

Meritev časa izvajanja zelo velikega števila korakov je pokazala, da se z uporabo algoritma za Q-učenje en korak izvede v približno 0,4 μs, z uporabo algoritma za učenje ASE/ACE pa v 1,7 μs. Ker je časovna zahtevnost pri Q-učenju nižja, smo pri tem algoritmu preizkušali vplive parametrov bolj natančno, to pomeni na večji množici vrednosti parametra. Z upoštevanjem, da bomo preizkusili približno 10 do 30 vrednosti pri približno 20 parametrih pri vsakem algoritmu in da bomo za vsako meritev naredili 1000 ponovitev smo določili **en milijon korakov na tek**. To število se uporablja pri večini ostalih meritev in pri intervalu učenja 0,01 s določa **10000 s simuliranega časa učenja**. Skupen računski čas za izvedbo vseh meritev znašal približno 20 dni na enem procesorskem jedru, na našem testnem sistemu s tremi jedri pa malo manj kot teden dni. Testni sistem bo opisan v nadaljevanju.

Časovne zahtevnosti posameznih programskih sklopov

Navedli smo, da ima korak učenja različno časovno zahtevnost pri uporabi algoritma za Q-učenje ter algoritma za učenje ASE/ACE. To je bilo pričakovano, saj imata algoritma računske zahtevnosti različnih redov, kot opisano v poglavju 4.1. Iz tega razloga nas je zanimalo dejansko razmerje v časovni zahtevnosti simulacije z enim in drugim algoritmom. Poleg tega nas je zanimal tudi faktor pohitritve, ki smo ga dosegli z optimizacijo računanja simulacije vozička s palico, o kateri smo pisali v poglavju 4.3. V te namene smo opravili meritve časovnih zahtevnosti posameznih sklopov simulacije ter jih prikazali v tabeli 7.1. Iz meritev ugotovimo, da **potrebuje algoritem za učenje ASE/ACE približno 26-krat več računskega časa kot algoritem za Q-učenje**. To je zelo pomemben dejavnik, ki ga ne smemo zanemariti

pri končni primerjavi obeh algoritmov. Ker je časovna zahtevnost simulacije vozička s palico večja od algoritma za Q-učenje, je dosežena 22-odstotna pohitritev tega sklopa na račun optimizacije, dober rezultat.

Tabela 7.1. Čas procesiranja enega koraka učenja.

	Čas procesiranja [μs]
Algoritem za Q-učenje	0,053
Algoritem za učenje ASE/ACE	1,384
Ne-optimizirana fizikalna simulacija vozička s palico	0,127
Optimizirana fizikalna simulacija vozička s palico	0,099

Poraba delovnega pomnilnika

Delovni pomnilnik se dodeli pred vsakim tekom ter sprosti po koncu teka. Shranjeni ostanejo le izračuni pokazateljev uspešnosti za vsak tek ter izračuni povprečnih vrednosti in njihovi odkloni. Pomnilnik za shranjevanje povprečnih vrednosti se dodeli pred vsakim preizkusom nabora parametrov ter sprosti po koncu preizkusa, ko se rezultati zapišejo na trdi disk. S sprotnim sproščanjem smo dosegli, da je količina potrebnega pomnilnika bistveno manjša in je tako reda *O(število korakov na tek + število tekov)*. Poraba pri enem milijonu korakov in 1000 ponovitvah je približno 40 MB, pri 10 milijonih korakov pa približno 400 MB. Količino bi lahko še znižali, če ne bi prikazovali uspešnosti za vsak trenutek časa učenja ter če ne bi računali odklonov.

Testni sistem

Vsi preizkusi so bili izvedeni na osebnem računalniku. Uporabljen operacijski sistem je Windows 7, 64-bitna različica. Procesna enota je tri-jedrni AMD Athlon 2 X3 450, s taktom delovanja 3,2 GHz. Količina delovnega pomnilnika je 4 GB, vrste DDR3.

8 Rezultati meritev uspešnosti

V poglavju 6 je bila opisana metodologija izvajanja meritev, v tem poglavju pa bomo prikazali rezultate opravljenih meritev. Navedli bomo ugotovitve vplivov parametrov simulacije na uspešnost krmiljenja vozička s palico pri uporabi algoritma za Q-učenje ter algoritma za učenje ASE/ACE.

8.1 Privzete vrednosti parametrov

Vpliv posameznih parametrov simulacije na uspešnost ugotavljamo tako, da spreminjamo vrednost le enega izbranega parametra na enkrat, kot že opisano. Vrednosti ostalih parametrov – tistih, ki ne spreminjamo – imenujemo privzete vrednosti in so prikazane v tabelah 8.1. Pri njihovi določitvi smo se večinoma zgledovali po Bartu in sod. [2]. Parametri, katerih vpliv bomo ugotavljali, so obarvani rdeče. Vrednosti v tabeli veljajo za vse meritve v nadaljevanju, razen, kjer je izrecno navedeno drugače. Z meritvami smo najprej poskušali poiskati vrednosti parametrov učenja, pri katerih je uspešnost boljša. V tabeli so prikazane pri parametrih učenja v desnem stolpcu. Nato smo s temi vrednostmi izvedli še vse ostale meritve, od vključno podpoglavja 8.3 naprej.

Ker je interval učenja enak 10 ms, lahko na vseh prikazanih grafih naredimo pretvorbo iz časa v število korakov z množenjem s faktorjem 100 – razen tam, kjer je uporabljen drugačen interval učenja (bo posebej navedeno).

Poudarimo, da je dolžina palice dejansko razdalja od vpetja na vozičku do težišča palice, kot prikazano na sliki 3.4 v podpoglavju 3.3. Poleg tega je največji odmik vozička, brez da pride do neuspeha, polovica dolžine podlage.

8.2 Parametri učenja

Ker smo želeli čim bolj povečati uspešnost algoritmov, smo najprej ugotavljali vpliv parametrov učenja in skušali poiskati vrednosti, pri katerih je ta največja. Uporabili smo princip t.i. požrešne metode [21]: iščemo najboljšo vrednost enega parametra pri uporabi že dobljenih najboljših vrednosti ostalih parametrov ter to ponavljamo dokler ne izračunamo izboljšanjih vrednosti za vse parametre. V naših okoliščinah to ni dober način za iskanje najboljše uspešnosti, ker je zelo verjetno, da zaidemo v nek lokalni maksimum. Uporabiti bi morali takšno metodo, ki bolje preišče prostor – ki

bolje preišče možne kombinacije vrednosti, npr. z uporabo evolucijskega programiranja [21]. Ker pa namen našega dela ni bil optimizacija parametrov učenja, temveč samo ugotavljanje njihovega vpliva, bo izbrana metoda zadostovala.

Tabele 8.1. Privzete vrednosti vseh parametrov simulacije in izboljšane vrednosti parametrov učenja.

Parametri okolja	Privzeta vrednost
Masa vozička	1,0 kg
Masa palice	0,1 kg
Dolžina palice	0,5 m
Koeficient trenja vozička	0,030
Koeficient trenja palice	0,005
Težni pospešek	9,81 m/s^2
Velikost sile potiska	10 N
Dolžina podlage	4,8 m
Vhodni šum	0%
Izhodni šum	0%
Vhodna zakasnitev	0 ms
Izhodna zakasnitev	0 ms
Začetni odklon palice	0°
Časovni korak simulacije vozička s palico	enak intervalu učenja

Parametri izvajanja simulacije	Privzeta vrednost
Interval učenja	10 ms
Čas učenja	10000 s

Parametri Q-učenja	Privzeta vrednost	Izboljšana vrednost
Alfa (α)	0,5	0,3
Beta (β)	0,001	0,00001
Gama (γ)	0,999	0,9993

Parametri učenja ASE/ACE	Privzeta vrednost	Izboljšana vrednost
Alfa (a)	10	37
Beta (β)	0,5	0,4
Gama (γ)	0,995	0,9993
Sigma (δ)	0,98	0,98
Lambda (λ)	0,98	0,98

Parametri meritev	
Število ponovitev	1000
Množica vrednosti preizkusa vpliva vsakega parametra	Se razlikuje od parametra do parametra

Parametri diskretizacije prostora
Število diskretnih stanj prostora je 162.
Meje delitve za posamezne dimenzije so navedene v podpoglavju 4.2.

Pri iskanju najboljših parametrov učenja naletimo še na naslednjo težavo. Pri različnih parametrih okolja in izvajanja so tudi optimalne vednosti parametrov učenja različne. To pomeni, da bomo učenje optimizirali le na trenutno določene okoliščine – to so privzete vrednosti parametrov v tabelah 8.1. Pri nekih drugih okoliščinah pa bo lahko zaradi optimizacije uspešnost slabša. To potrjujejo meritve v nadaljevanju.

Q-učenje

Najprej smo poiskali vrednost parametra učenja alfa pri katerem je uspešnost najboljša, nato pri gama ter nato pri parametru raziskovanja prostora – beta. Najboljšo vrednost parametra alfa smo določili pri različnih vrednostih nekaterih parametrov. To prikazuje graf na sliki 8.1. Vidimo, da je pri povečanju zakasnitve vhoda z 0 ms na 10 ms najboljša vrednost alfa drugačna kot sicer. Povečanje intervala učenja z 10 ms na 20 ms pa ni imelo bistvenega vpliva. To potrjuje prejšnjo trditev o vplivu vrednosti drugih parametrov na optimalne vrednosti parametrov učenja.

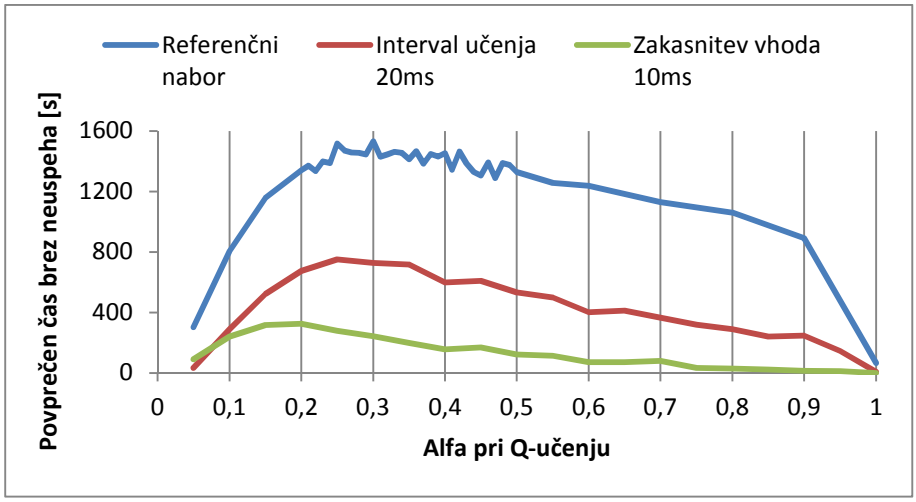
Pri merjenju parametra gama smo preizkusili, če imajo podoben medsebojni vpliv tudi parametri učenja. Da je to res, potrjujeta grafa na sliki 8.2, ki prikazujeta meritve pri alfa = 0,3 ter alfa = 0,8. Za najboljšo vrednost pri privzetih parametrih smo tako izbrali vrednosti **alfa = 0,3** in **gama = 0,9993**.

Pri teh dveh vrednostih smo preizkusili še parameter beta, kot prikazujeta grafa na sliki 8.3. Uspešnost Q-učenja narašča z zmanjševanjem beta. Ta parameter določa stopnjo raziskovanja prostora, kot opisano v podpoglavju 3.1. Pri beta = 0 postane sistem deterministično, zato dobimo vedno isti rezultat, ne glede na število ponovitev. Najboljšo uspešnost smo tako dosegli pri **beta = 0,00001**. Grafov števila neuspehov ter časovne sredine neuspehov nismo prikazali, ker kažejo na zelo podobne optimalne vrednosti.

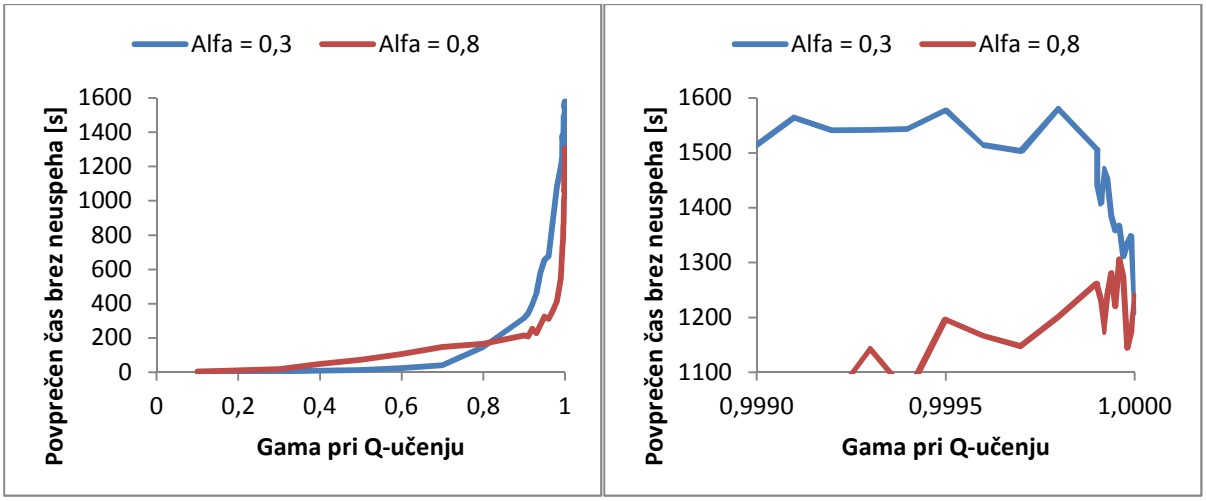
Učenje ASE/ACE

Najprej smo določili najboljšo vrednost parametra alfa, nato beta, gama, sigma in nazadnje parametra lambda. Na podlagi meritev, prikazanih na grafih slike 8.4, smo določili vrednosti **alfa = 37** in **beta = 0,4**. Grafi na slikah 8.5 in 8.6 pa prikazujejo meritve za gama, sigma in lambda. Kot najboljše vrednosti smo določili **gama = 0,9993**, **sigma = 0,98** ter **lambda = 0,98**. Izboljšane vrednosti sigma in lambda se ne razlikujejo od privzetih. To potrjuje, da smo predhodno izbrane najboljše vrednosti alfa, beta in gama optimizirali le na to točno določeno vrednost sigma in lambda (ker je zelo majhna verjetnost, da smo po naključju izbrali privzete vrednosti ravno enake optimalnim vrednostim). Učenje je zelo občutljivo na gama in sigma, saj neustrezna vrednost teh parametrov hitro zniža uspešnost na blizu 0.

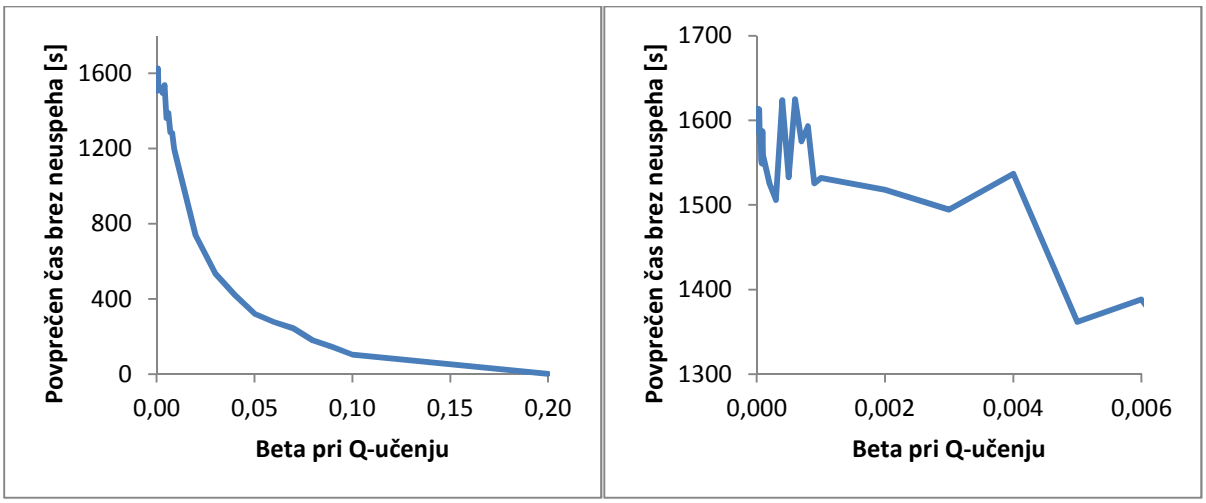
Obratno je pri alfa in lambda, ko uspešnost limitira proti mnogo višji vrednosti. Faktor zmanjševanja teže prihodnjih spodbud – gama – ima enako najboljšo vrednost kot pri Q-učenju. To verjetno zaradi tega, ker je po definiciji odvisen od dejavnikov okolja in ne od algoritma učenja (opisan je bil v poglavju 2).



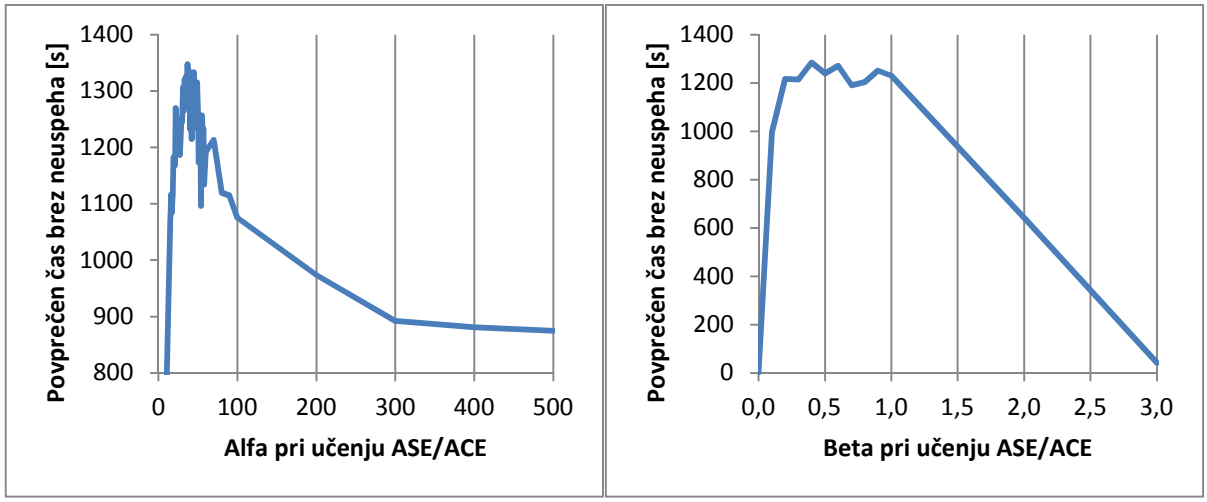
Slika 8.1. Vpliv parametra alfa na uspešnost Q-učenja pri različnih parametrih okolja.



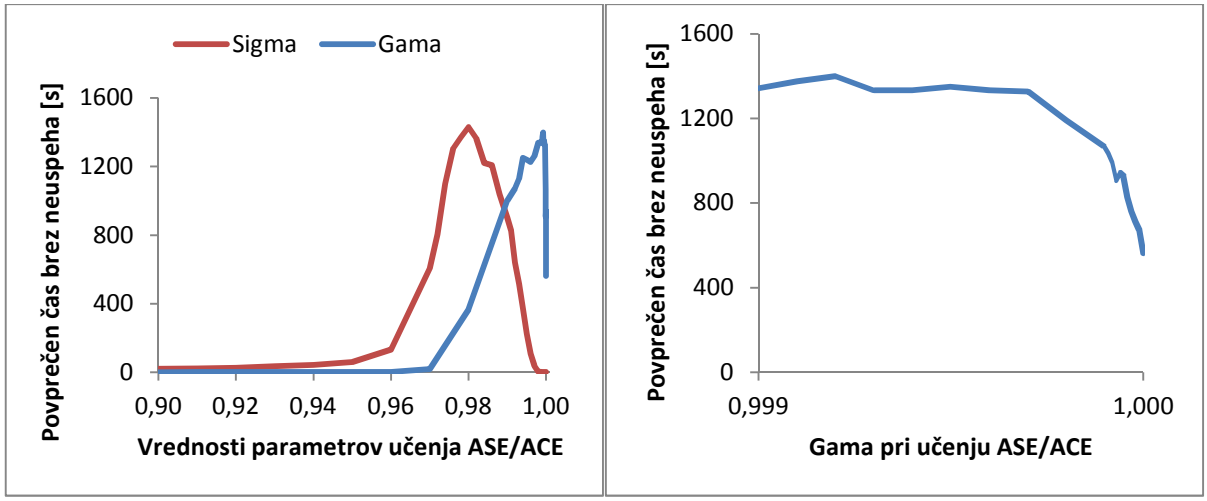
Slika 8.2. Vpliv parametra gama na uspešnost Q-učenja pri različnih vrednostih alfa.



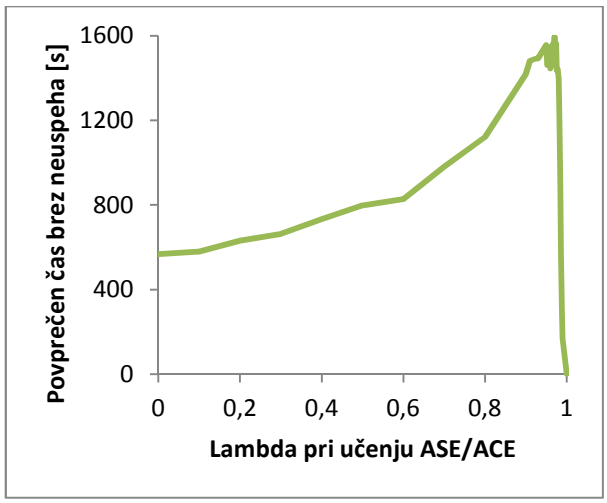
Slika 8.3. Vpliv parametra beta na uspešnost Q-učenja.



Slika 8.4. Vpliv parametra alfa (levo) in beta (desno) na uspešnost učenja ASE/ACE.



Slika 8.5. Vpliv parametrov gama in sigma na uspešnost učenja ASE/ACE.

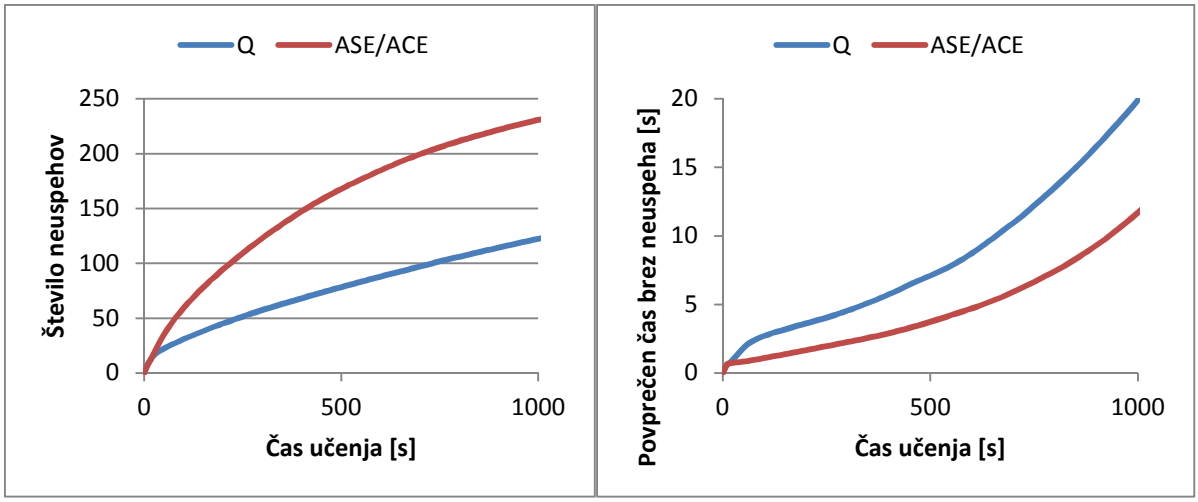


Slika 8.6. Vpliv parametra lambda na uspešnost učenja ASE/ACE.

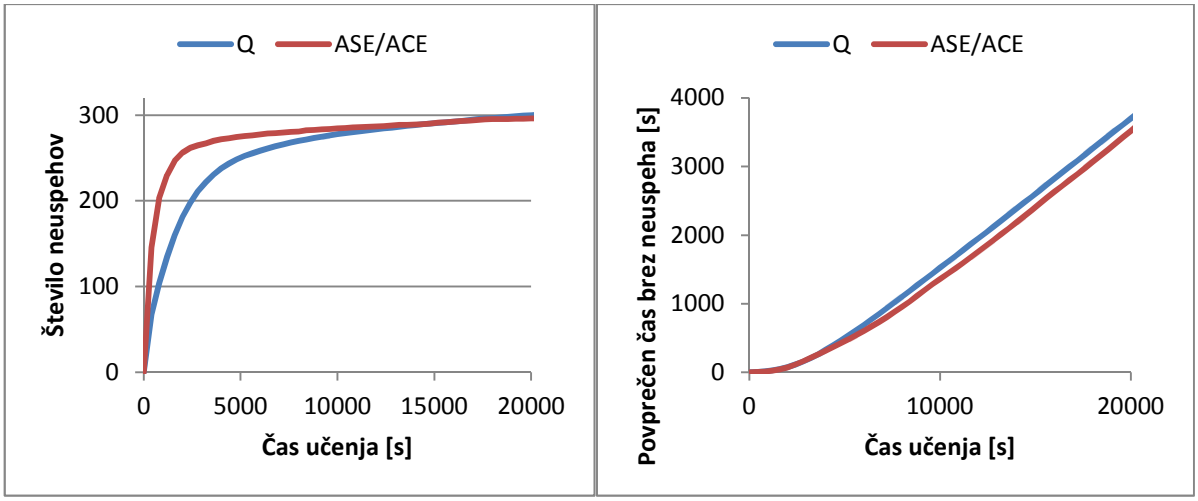
8.3 Čas učenja

Od te točke naprej smo pri vseh meritvah uporabljali izboljšane vrednosti parametrov učenja. Primerjali smo uspešnost obeh algoritmov pri različnem simuliranem času na razpolago za učenje. Pri razmeroma kratkem času 1000 s (100000 korakov), je Q-učenje tudi do dvakrat bolj uspešno od učenja ASE/ACE, prikazano na grafih slike 8.7. Po času učenja 20000 s (2 milijona korakov) sta oba algoritma približno enako dobra – grafi na sliki 8.8. Po razmeroma dolgem času učenja 400000s (40 milijonov korakov) pa vidimo, da je učenje ASE/ACE bistveno bolj uspešno kot Q-učenje – grafi na sliki 8.9. Pri Q-učenju število neuspehov limitira k linearno naraščajoči asimptoti, pri učenju ASE/ACE pa k vodoravni asimptoti. To pomeni, da bi razlika v uspešnosti obeh algoritmov še naraščala z daljšanjem časa učenja kar je zelo pomemben dejavnik pri končni primerjavi obeh algoritmov.

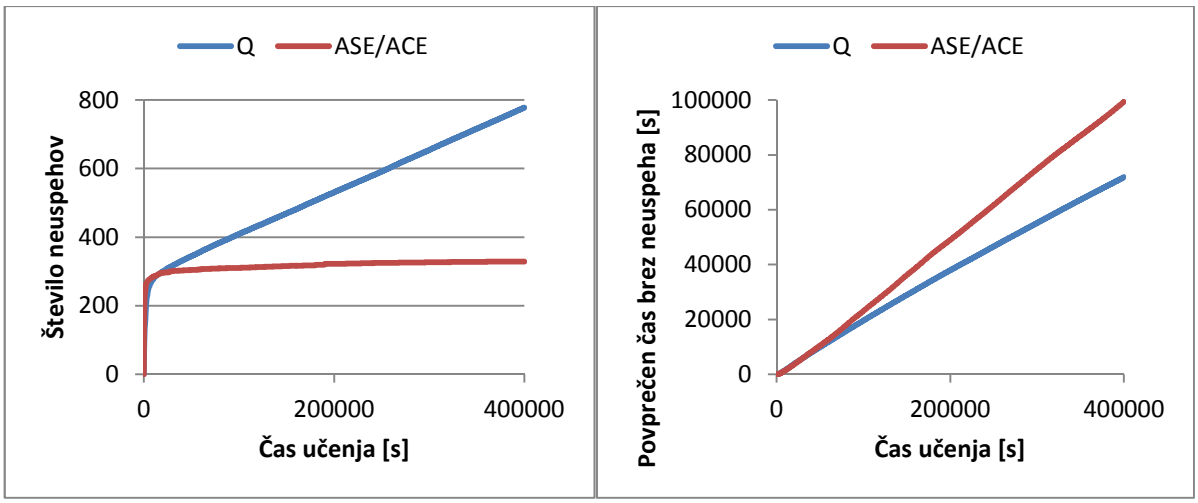
Odklonov in časovne sredine neuspehov nismo prikazovali na grafih, zato so v strnjeni obliki podani v tabeli 8.2. Časovna sredina neuspehov pri Q-učenju pri dolgem času učenja naraste, pri ASE/ACE pa ravno obratno. To potrjuje prejšnjo trditev o asimptotah. Tabela še prikazuje, da so pri daljšem času učenja odkloni pri Q-učenju večji, pri ASE/ACE pa v povprečju manjši. Razlog za to podajata grafa na sliki 8.10. Ta grafa prikazujeta odstotni delež tekov (izmed 1000 ponovitev), ki so dosegli določeno raven uspešnosti za čas učenja 20000 s ter 400000 s. Vidimo, da se pri daljšem času učenja pri Q-učenju izoblikujeta dve skupini tekov – ena, ki dosega izredno visoko uspešnost (30 % tekov) ter ena, ki dosega izredno nizko uspešnost (50 % tekov). Pri učenju ASE/ACE je prisotna le skupina, ki dosega izredno visoko uspešnost (30 % tekov), uspešnost po ostalih tekih pa je razporejena enakomerno. Iz grafov tudi vidimo, da je pri obeh učenjih z povečanjem časa učenja zelo narasel delež tekov, ki doseže skoraj največjo možno uspešnost (to pomeni, da je povprečen čas brez neuspeha skoraj enak času učenja).



Slika 8.7. Uspešnost obeh algoritmov učenja pri času učenja 1000 s (100000 korakov).



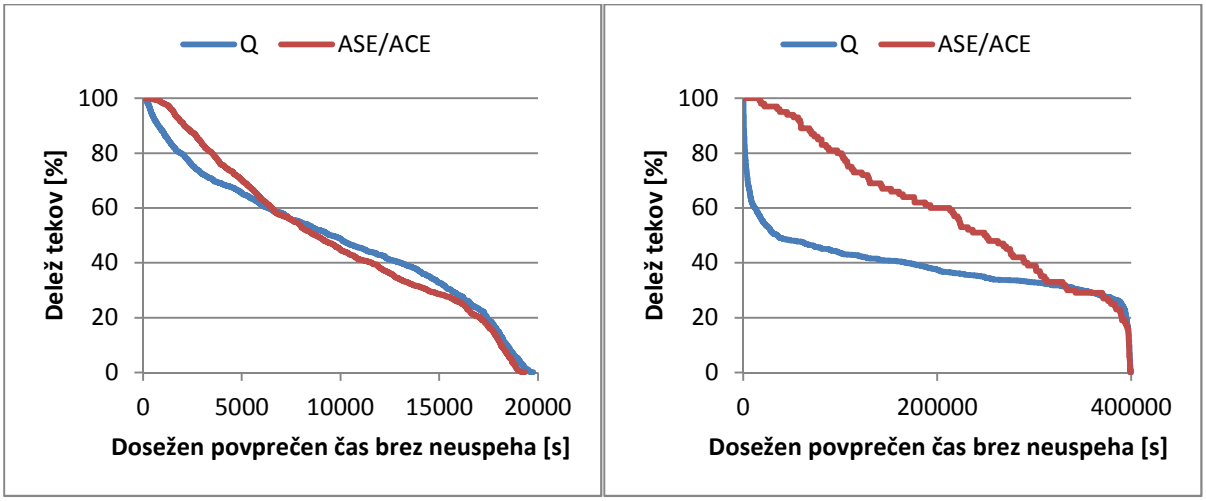
Slika 8.8. Uspešnost obeh algoritmov učenja pri času učenja 20000 s (2 milijona korakov).



Slika 8.9. Uspešnost obeh algoritmov učenja pri času učenja 400000 s (40 milijonov korakov).

Tabela 8.2. Časovna sredina neuspehov in odkloni pri različnih časih učenja.

Čas učenja [s]	Algoritem učenja	Časovna sredina neuspehov [%]	Odklon povprečnega časa brez neuspeha [%]	Odklon števila poskusov [%]
10000	**ASE/ACE**	7,5	89,7	48,2
	Q	16,1	88,5	49,6
100000	**ASE/ACE**	5,6	81,4	43,2
	Q	11,3	99,5	54,2
400000	**ASE/ACE**	3,0	70,1	46,3
	Q	15,2	118,6	156,7



Slika 8.10. Odstotni delež tekov, ki doseže določeno uspešnost pri času učenja 20000 s (levo) ter času učenja 400000 s (desno).

8.4 Parametri okolja

V nadaljevanju bodo opisane meritve vpliva vseh parametrov okolja. Tiste, za katere smatramo, da so bolj pomembne, bodo izpostavljene v svojih podpoglavjih. Izmed teh parametrov smo najprej preverili, ali časovna natančnost fizikalne simulacije vozička s palico vpliva na uspešnost algoritmov učenja. Spreminjali smo časovni korak te simulacije od enako dolgega kot interval učenja do 10-krat krajšega. Ugotovili smo, da ne vpliva na meritve, zato smo še naprej uporabljali privzeto vrednost.

Masa in dolžina palice

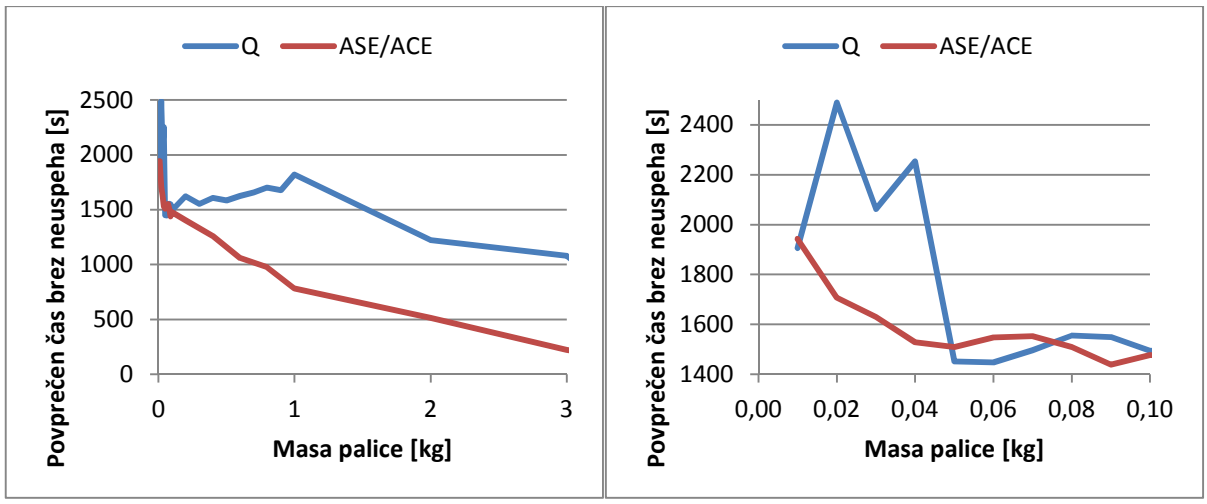
Grafi na sliki 8.11 prikazujejo meritve pri spreminjanju mase palice. Pri masi večji od 0,1 kg je uspešnost obeh algoritmov v povprečju slabša, toda Q-učenje je manj občut-

ljivo in celo izboljšuje uspešnost do določene povečane mase. Pri masi manjši od 0,1 kg je uspešnost obeh algoritmov boljša, Q-učenje je tudi tukaj boljše, vendar ima velika nihanja že pri majhnih spremembah mase.

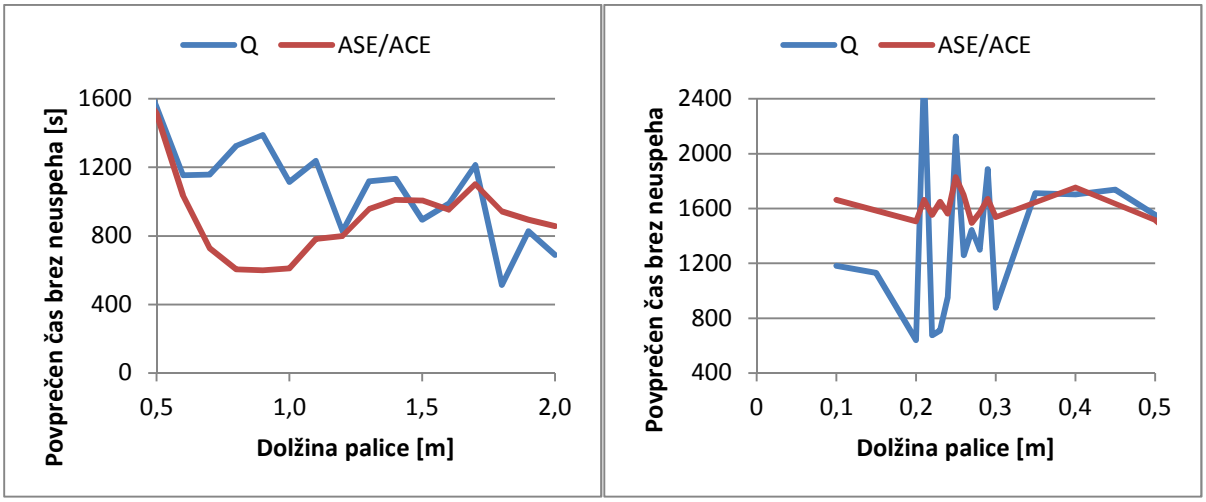
Grafa na sliki 8.12 prikazujeta uspešnost pri spreminjanju dolžine palice. Pri palici daljši od 0,5 m je uspešnost v povprečju slabša pri obeh algoritmih. Pri krajši od 0,5 m je pri učenju ASE/ACE v povprečju enaka, pri Q-učenju pa pada in zelo niha pri zelo majhnih spremembah dolžine palice.

Zaradi vseh teh nihanj pri majhnih spremembah vrednosti parametrov okolja domnevamo, da imata ta dva parametra močan vpliv na optimalne vrednosti parametrov učenja. Ker so nihanja pri Q-učenju mnogo večja, je možno, da je ta vpliv še močnejši pri tem algoritmu ali pa, da smo vrednosti parametrov Q-učenja preveč optimizirali za privzete vrednosti. Navedene lastnosti se izkazujejo tudi pri več meritvah, prikazanih v nadaljevanju. **Tukaj poudarimo, da prikazana nihanja uspešnosti niso posledica statističnih nihanj.** To pomeni, da tudi če naredimo poljubno število ponovitev prikazanih meritev, se bodo rezultati razlikovali kvečjemu za nekaj odstotkov, tako kot je to opisano v podpoglavju 6.10.

Števila neuspehov in časovne sredine neuspehov ne prikazujemo, ker podajajo isto informacijo kot povprečen čas brez neuspeha. Z naraščanjem uspešnosti algoritma učenja standardni odklon povprečnega časa brez neuspeha pada, odklona drugih dveh pokazateljev pa naraščata, in obratno. Tako je pri vseh meritvah, zato teh vrednosti tudi v nadaljevanju ne bomo prikazovali.



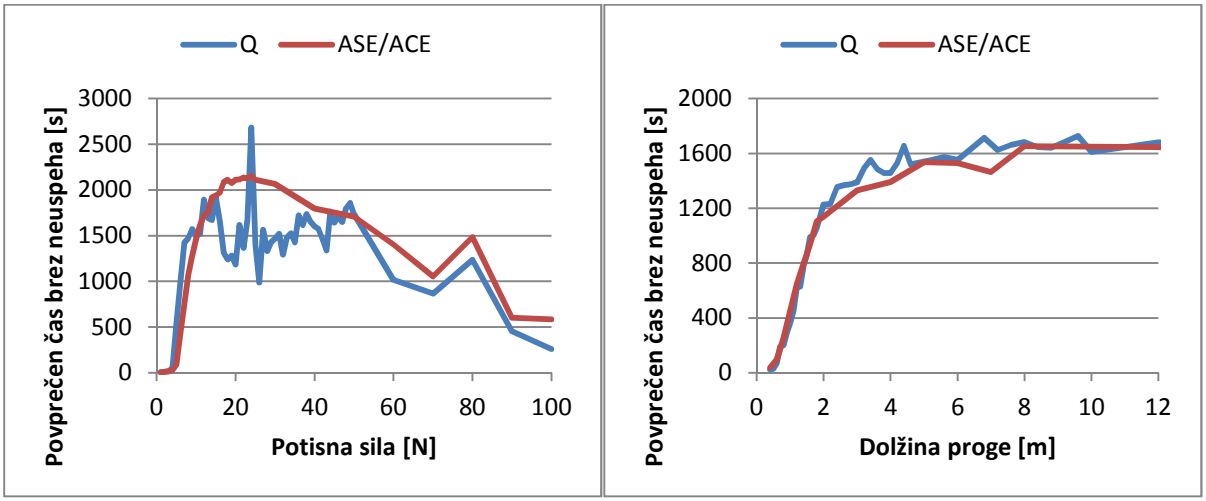
Slika 8.11. Vpliv mase palice na uspešnost algoritmov učenja.



Slika 8.12. Vpliv dolžine palice na uspešnost algoritmov učenja.

Potisna sila

Levi graf na sliki 8.13 prikazuje uspešnosti pri spreminjanju velikosti potisne sile. Z manjšanjem potisne sile pod 10 N se manjša tudi uspešnost obeh algoritmov, pri čemer lahko Q-učenje z manjšo silo zagotavlja enako uspešnost kot ASE/ACE. Z večanjem sile do določene meje uspešnost učenja ASE/ACE narašča, nato pa začne upadati, ker postane aplicirana sila že prevelika za privzet interval učenja. Uspešnost Q-učenja pri večanju sile je v povprečju zaznavno nižja od uspešnosti učenja ASE/ACE in zelo niha že pri majhnih spremembah, tako kot pri prejšnjih meritvah.



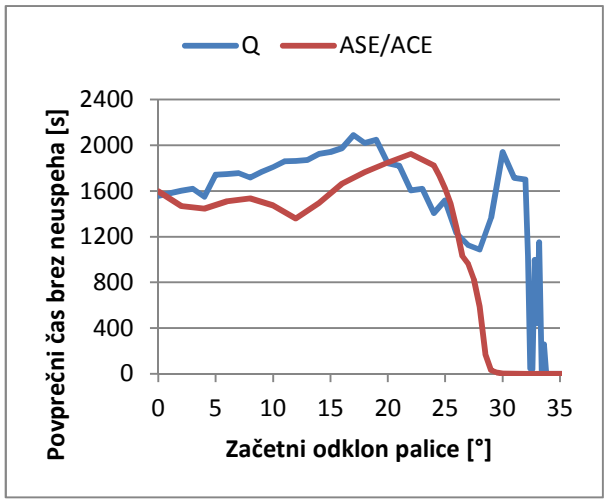
Slika 8.13. Vpliv potisne sile (levo) in dolžine podlage (desno) na uspešnost algoritmov učenja.

Dolžina podlage

Desni graf na sliki 8.13 prikazuje uspešnosti pri spreminjanju dolžine podlage. Ta parameter ima neposreden vpliv na funkcijo spodbude. Z daljšanjem podlage tako lajšamo delo učnemu sistemu, ker manjšamo verjetnost neuspehov zaradi prevelikega odmika vozička. Zato na grafu vidimo pričakovano limitiranje uspešnosti k vodoravni asimptoti. Višino asimptote določa težavnost uravnovešanja palice. Tukaj sta oba algoritma enakovredna.

Začetni odklon palice

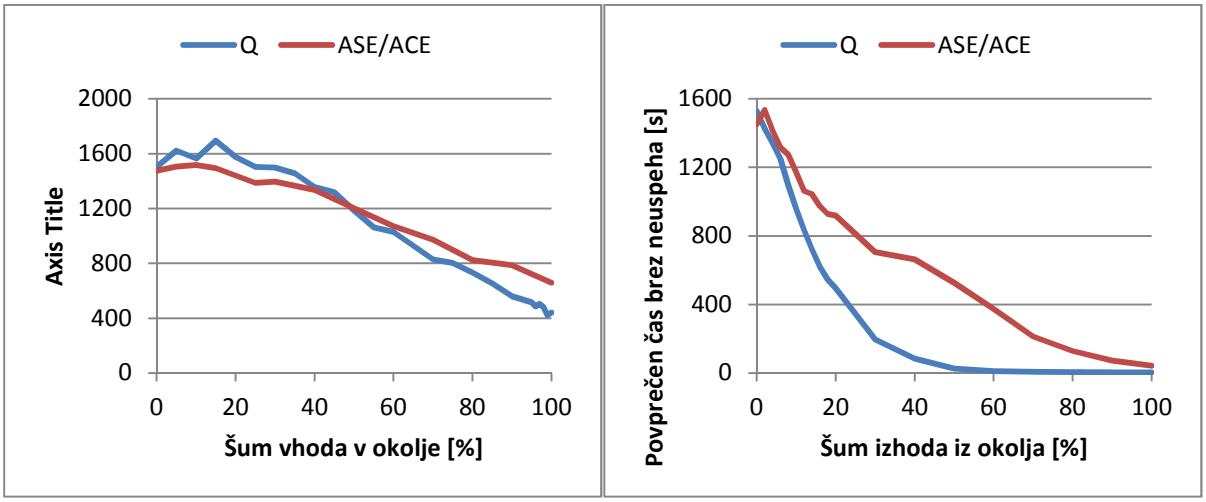
Začetni odklon palice od navpične lege pomaga algoritmu učenja bolje opisati prostor stanj. To se do neke mere odraža v boljši uspešnosti. Na grafu slike 8.14 je vidno, da z večanjem odklona uspešnost pri Q-učenju dobro narašča, pri učenju ASE/ACE pa nekoliko manj – naraste šele pozneje. Ko je palica že na začetku poskusa toliko nagnjena, da jo težko naravnamo, uspešnost zelo hitro pade na 0. Q-učenje je v tem primeru uspešno še za nekaj stopinj odklona več kot učenje ASE/ACE, pri prehodu pa se pojavijo velika nihanja, kot že pri drugih meritvah.



Slika 8.14. Vpliv začetnega odklona palice na uspešnost algoritmov učenja.

8.5 Šum

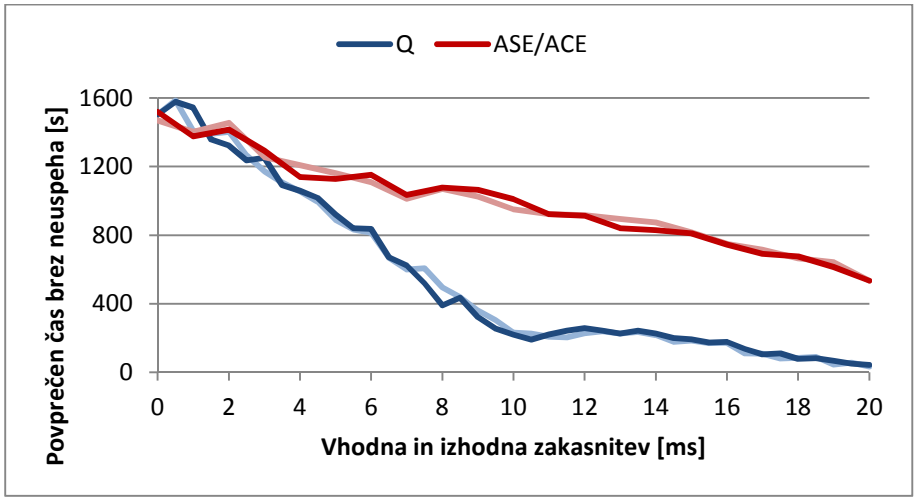
Na sliki 8.15 so prikazane uspešnosti pri spreminjanju šumov. Majhne vrednosti šuma v nekaterih primerih izboljšajo uspešnost, ker algoritmu učenja pomagajo preiskovati prostor stanj. Pri vhodnem šumu je do približno 50-odstotnega šuma Q-učenje bolj uspešno, pri še večjem pa učenje ASE/ACE. Pri izhodnem šumu je bistveno večja razlika v prid učenju ASE/ACE, ki ima pri 50-odstotni vrednosti šuma še vedno približno tretjino začetne uspešnosti, medtem ko je uspešnost Q-učenja že blizu 0.



Slika 8.15. Vpliv šuma vhoda v okolje (levo) ter izhoda iz okolja (desno) na uspešnost algoritmov učenja.

8.6 Zakasnitev

Če zanemarimo statistično nihanje, so rezultati meritev vhodne ter izhodne zakasnitve iz okolja enaki in so prikazani na grafu slike 8.16. Proces učenja v okolju je sklenjena krožna zanka prenosa informacij med učnim sistemom in okoljem (stanja in akcije – slika 2.1 v poglavju 2). Ena interakcija med njima – en obhod zanke – traja določen čas. Brez zakasnitev je ta čas enak vrednosti intervala učenja. Na uspešnost učnega sistema vpliva količina tega časa, ne pa, kako je razporejen znotraj zanke. Zaradi tega je vseeno, ali je zakasnitev na vhodu v okolje ali na izhodu iz okolja, v obeh primerih dobimo enake rezultate. V dodatno podporo tej trditvi navajamo, da smo opravili meritve tudi pri različnih sočasnih kombinacijah vhodne in izhodne



Slika 8.16. Vpliv vhodne (svetli krivulji) in izhodne zakasnitve (temni krivulji) na uspešnost algoritmov učenja.

zakasnitve ter dobili vedno enake rezultate.

Z večanjem zakasnitev pa se čas posameznega obhoda zanke daljša, zato se tudi uspešnost učenja manjša. Povedano drugače – zakasnitve podaljšujejo čas, ki je potreben, da učni sistem dobi povratno informacijo o spodbudi, zato z njimi dejansko preverjamo sposobnost čistega napovedovanja algoritma učenja.

Z vnosom zakasnitev spreminjamo časovne spremenljivke sistema, zato smatramo, da so to eni izmed parametrov okolja, pri katerih je uspešnost učenja najbolj odvisna od vrednosti parametrov učenja, sodeč po enačbah za algoritme učenja iz poglavja 3.

Na omenjenem grafu je razvidno, da je učenje ASE/ACE bistveno bolj odporno na zakasnitve kot Q-učenje.

8.7 Interval učenja

Tako kot navedeno pri zakasnitvah, interval učenja vpliva na čas, potreben za pridobitev povratne informacijo iz okolja. Zato lahko daljšanje intervala primerjamo z dodajanjem zakasnitev. Poleg tega, kot že opisano v prejšnjih poglavjih, se vsaka meritev izvaja pri omejenem simuliranem času učenja. Ta čas je enak produktu števila korakov učenja ter trajanju posameznega koraka – intervalu učenja, zato se z daljšanjem intervala število korakov sorazmerno zmanjšuje ter obratno. To pomeni, da s spreminjanjem vrednosti tega parametra krmilni problem dvakratno otežujemo ali lajšamo.

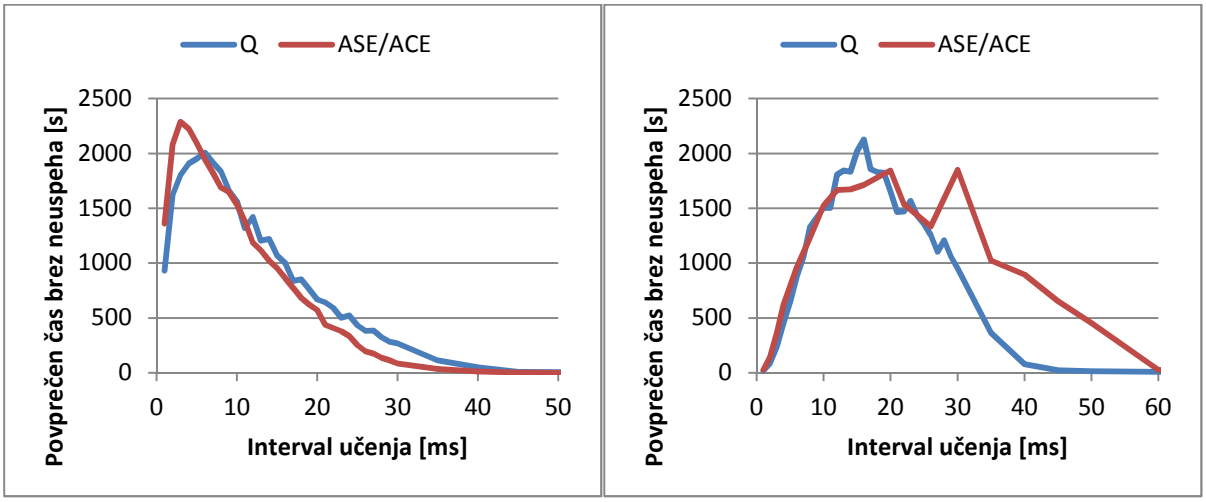
Zaradi navedenega smo opravili meritve na dva načina. Prve skupino meritev smo opravili tako, da smo omejili simuliran čas na 10000 s, kot prih vseh do zdaj. V tem primeru je število korakov učenja različno za vsako vrednost parametra intervala učenja. Pri drugih meritvah pa smo omejili število korakov na 1 milijon. V tem primeru je simuliran čas učenja različen za vsako vrednost parametra.

Prve meritve so prikazane na levem grafu slike 8.17. Pri daljšanju intervala uspešnost obeh algoritmov pada, saj imata na razpolago manj korakov za učenje. V tem primeru je Q-učenje nekoliko boljše od ASE/ACE. Pri krajšanju intervala pa uspešnost narašča do neke meje ter nato strmo pade proti 0. Ta meja je na točki, ko je izboljšanje uspešnosti zaradi večjega števila korakov nasprotno enako poslabšanju zaradi neustreznih parametrov učenja – število korakov do povratne informacije o spodbudi se zelo poveča, zato bi morale biti vrednosti parametrov učenja drugačne.

Ta meja je manjša pri učenju ASE/ACE, zato je pri kratkih intervalih dosežena uspešnost višja od uspešnosti Q-učenja.

Desni graf na sliki 8.17 prikazuje drugo skupino meritev. V tem primeru z daljšanjem intervala uspešnost narašča, ker ima učni sistem na razpolago več simuliranega časa – več časa za opazovanje okolja. Narašča pa le, dokler je izboljšanje uspešnosti zaradi tega časa večje od poslabšanja uspešnosti zaradi zakasnitve povratne informacije. Od te meje dalje uspešnost pada, pri čemer je učenje ASE/ACE boljše. S krajšanjem intervala uspešnost pada, ker ima učni sistem na razpolago manj simuliranega časa za opazovanje okolja ter zaradi neustreznih vrednosti parametrov učenja, kot opisano prej.

Tako kot pri zakasnitvah, tudi tukaj spreminjamo časovne spremenljivke sistema. Iz tega razloga je pri spreminjanju trajanja intervala učenja uspešnost učenja zelo odvisna od vrednosti parametrov učenja. Pri večanju intervala je potrebno parametre učenja nastaviti tako, da upoštevajo manjše število korakov do končne spodbude, pri krajšanju intervala pa ravno obratno.



Slika 8.17. Vpliv trajanja intervala učenja na uspešnost algoritmov učenja pri fiksnem času učenja 10000s (levo) in fiksnem številu korakov 1 milijon (desno).

9 Sklepne ugotovitve

V tem delu smo se na kratko seznanili s področjem spodbujevalnega učenja ter nekaterimi osnovnimi metodami njegove uporabe. Naš glavni cilj je bila implementacija sistema spodbujevalnega učenja s krmilnim problemom ter ovrednotenje dveh primerov algoritmov. V ta namen smo razvili simulacijo omenjenega sistema ter modularno orodje, ki to simulacijo uporablja za izvajanje eksperimentalnih meritev. Za boljše izvajanje preizkusov in razumevanje rezultatov smo tudi razvili metodologijo vrednotenja uspešnosti.

Za vrednotenje smo izbrali algoritem Q-učenja [20], in algoritem učenja ASE/ACE, ki so ga predstavili Barto in sod. [2]. Ugotovili smo, da so glavne prednosti algoritma za Q-učenje predvsem boljša uspešnost pri razmeroma kratkem simuliranem času učenja, bistveno nižja računska zahtevnost, ki je za celoten red manjša od zahtevnosti algoritma učenja ASE/ACE, saj je neodvisna od števila diskretnih stanj prostora, ter enostavnejša implementacija. Glavna slabost algoritma Q-učenja je, da z naraščanjem simuliranega časa razmeroma hitro doseže največjo raven možnega pridobljenega znanja, pri ASE/ACE pa te ravni nismo dosegli. To pomeni, da razlika med uspešnostjo učenja ASE/ACE in uspešnostjo Q-učenja narašča vedno bolj v prid ASE/ACE z daljšanjem časa na razpolago za učenje. Pri daljšanju časa učenja ima Q-učenje še to težavo, da postaja vedno večja verjetnost, da bo posamezen poskus dosegel izredno visoko ali izredno nizko uspešnost. Poleg navedenih so še druge prednosti algoritma učenja ASE/ACE bistveno večja odpornost na zakasnitve ter šume in manjša občutljivost pri spreminjanju vrednosti dejavnikov okolja na ne-optimalne vrednosti parametrov učenja. Slednje pri Q-učenju povzroča velika nihanja že pri majhnih spremembah nekaterih dejavnikov okolja (npr. mase ali dolžine palice), kar pomeni, da je potrebno pazljivo določiti vrednosti parametrov učenja za dan problem. Pri spremembah frekvence osveževanja učenja pa sta oba algoritma približno enako dobra.

Pridobljene ugotovitve lahko nudijo izhodiščno točko pri načrtovanju krmilnikov s spodbujevalnim učenjem v smislu, da dajejo osnovne smernice pri izbiri ustreznega algoritma za dan problem in okoliščine. Razvito orodje za izvajanje preizkusov je še na zelo primitivni stopnji, vendar lahko služi kot ogrodje za razvoj bolj zmogljivega orodja, ki bi na enostavnejši način omogočalo bolj splošno preizkušanje metod učenja nad različnimi problemi (po vzoru [16]). Opisana metodologija vrednotenja uspešnosti

se lahko uporablja v navezi s poljubnim algoritmom učenja tudi pri drugih problemih zakasnjene spodbude ter omogoča bolj natančno in verodostojno interpretacijo uspešnosti z več vidikov kot pa običajna splošno uporabljena metoda, ki so jo uporabili tudi Barto in sod. [2].

Za konec podajamo še nekaj smernic za nadaljnje delo. Izvajanje omenjenih preizkusov je časovno zelo potratno zaradi računske zahtevnosti simulacije spodbujevalnega učenja. To bi lahko pohitrili z uporabo paralelnega procesiranja, npr. na grafičnih procesnih enotah. Z vidika vrednotenja uspešnosti, podaja naše delo ugotovitve le za zelo omejene okoliščine. To bi bilo potrebno dopolniti s preizkušanjem medsebojnih vplivov različnih dejavnikov; uporabo drugačnih funkcij za aproksimacijo vrednosti stanja okolja (npr. z uporabo nevronskih mrež) ter z reševanjem težjih problemov – takih, ki so čim bolj podobni realnim. Tako bi lahko npr. v problem krmiljenja s palico vpeljali nove dejavnike kot so neravna površina podlage ali njeno sprotno spreminjanje (simulacija premikanja), časovne blokade motorjev ter preslikava akcije učnega sistema v napetost na elektromotorjih namesto v potisno silo. V končni fazi bi lahko opravljali poskuse tudi na realnem sistemu ali pa prenesli na realni sistem algoritem, naučen na simulaciji. Zaključujemo z zamislijo, da bi lahko za iskanje optimalnih vrednosti parametrov učenja uporabljali evolucijske metode ter mogoče celo med samim postopkom učenja uvedli sprotno prilagajanje teh vrednosti glede na spreminjanje dejavnikov okolja. Na tak način bi učni sistem sam iskal optimalne vrednosti svojih lastnih parametrov učenja ter tako razširil raziskovanje prostora še v to smer.

Literatura in viri

[1] Taiwo Oladipupo Ayodele, Types of Machine Learning Algorithms, 2010.

[2] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, zv. SMC-13, št. 5, str. 835-846, september/oktober 1983.

[3] Ciril Bohak, Luka Čehovin, and Marko Toplak, "Stabilizacija podmornice s spodbujevalnim učenjem," Poročilo seminarske naloge 2006.

[4] (2011, September) C programming.com - Your Resource for C and C++ Programming. Dostopno na: http://www.cprogramming.com/

[5] (2011, September) cplusplus.com - The C++ Resources Network. Dostopno na: http://www.cplusplus.com/

[6] Peter Dayan and Christopher J. C. H. Watkins, Reinforcement learning, 2001.

[7] David J. Finton. (2011, September) Controller-less Driver For the Cart-Pole Problem. Dostopno na: http://pages.cs.wisc.edu/~finton/poledriver.html

[8] David J. Finton. (2011, September) Q-Learning Controller for the Cart-Pole Problem. Dostopno na: http://pages.cs.wisc.edu/~finton/qcontroller.html

[9] Răzvan V. Florian, Correct equations for the dynamics of the cart-pole, 2005.

[10] Mance E. Harmon and Stephanie S. Harmon, Reinforcement Learning: A Tutorial, 1996.

[11] Nejc Ilc, "Primerjava metod za razvrščanje vzorcev v gruče," Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana, diplomska naloga 2009.

[12] Jiří Iša. (2011, September) Jiri Isa: Projects - Cart-pole system. Dostopno na: http://artax.karlin.mff.cuni.cz/~isa_j1am/projects/cart_pole/

[13] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, zv. 4, str. 237-285, 1996.

[14] (2011, September) Lazy Foo' Productions. Dostopno na: http://www.lazyfoo.net/SDL_tutorials/

[15] Rémi Munos in Andrew Moore. (2011, September) Cartpole. Dostopno na: http://www.cmap.polytechnique.fr/~munos/variable/cartpole.html

[16] Gerhard Neumann, "The Reinforcement Learning Toolbox, Reinforcement Learning for Optimal Control Tasks," Institut für Grundlagen der Informationsverarbeitung (IGI), Graz, doktorska disertacija 2005.

[17] (2011, September) SDL: Tutorials - GPWiki. Dostopno na: http://content.gpwiki.org/index.php/SDL:Tutorials

[18] (2011, September) Simple DirectMedia Layer. Dostopno na: http://www.libsdl.org/

[19] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction.*: MIT Press, 1998.

[20] Christopher J.C.H. Watkins, "Learning from Delayed Rewards," Cambridge University, Cambridge, doktorska disertacija 1989.

[21] Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org

A Seznam slik

B Seznam tabel