Institutionen för kommunikation och information
Examensarbete i datavetenskap 30hp
C-nivå
Vårterminen 2011

Utvärdering av Artificiella Neurala Arkitekturer För Navigering

Ivar Lång

Utvärdering av Artificiella Neurala Arkitekturer För Navigering

Examensrapport inlämnad av Ivar Lång till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Mikael Thieme.

2011-06-03

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Utvärdering av Artificiella Neurala Arkitekturer För Navigering

Ivar Lång

Sammanfattning

Den klassiska approachen till navigering innefattar att agenten håller en intern representativ modell av omgivningen. Denna approach har emellertid många nackdelar, speciellt för dynamiska miljöer. En modernare approach är att förlita sig på den faktiska omgivningen istället för en modell av denna. Detta arbete presenterar en undersökning av navigeringsproblemet och hur väl det löses av agenter vars kontrollmekanismer utgörs artificiella neurala nätverk. Tillförlitligheten hos de två neurala arkitekturerna *Extended sequential cascaded network* och *Self-organized recurrent network* bestäms genom experiment. Det visas i experimenten att Extended sequential cascaded network är den mest tillförlitliga arkitekturen av de två när navigeringsproblemet skall angripas. Det visas även att Extended sequential cascaded network tränar fram ett helt reaktivt beteende i samtliga experiment. Slutsatsen som kan dras av detta är att svåra problem inte alltid kräver avancerade arkitekturer för att lösas tillfredsställande.

Nyckelord: AI, Artificial neural network, Extended sequential cascaded network, Navigation, Self-organization, YAKS

“Within thirty years, we will have the means to create superhuman intelligence. Shortly after, the human era will be ended.”

-Vernor Vinge
1993

Innehållsförteckning

1 Introduktion **1**

2 Bakgrund **3**
2.1 Klassisk och modern AI . . . 3
2.1.1 Navigering med hjälp av representation . . . 4
2.2 Navigeringsproblemet . . . 6
2.3 Artificiella neurala nätverk . . . 7
2.3.1 Extended sequential cascaded network (ESCN) . . . 8
2.3.2 Self-organized recurrent network (SORN) . . . 10
2.4 Artificiell evolution . . . 10
2.4.1 Genetiska algoritmer . . . 11
2.5 Simulering och verklighetsgapet . . . 14

3 Problemformulering **17**
3.1 Tidigare arbeten . . . 17
3.1.1 En robot som korsar verklighetsgapet (Jakobi) . . . 17
3.1.2 Neurala arkitekturer för minne och adaptivt beteende (Ziemke) . . . 18
3.1.3 Multipla T-korsningar för undersökning av fördröjt svar (Thieme) . . . 19
3.1.4 Vägplanering i realtid för dataspel (Graham) . . . 20
3.2 Syfte . . . 20
3.2.1 Delsteg 1: Testmiljö . . . 21
3.2.2 Delsteg 2: Utvärdering . . . 21
3.3 Metod . . . 21
3.3.1 Metod för delsteg 1: Implementering . . . 21
3.3.2 Metod för delsteg 2: Experiment . . . 22

4 Genomförande **23**
4.1 Genomförande av delsteg 1: Modifikation av simulator . . . 23
4.1.1 Implementering av Self-organized recurrent network . . . 23
4.1.2 Dynamiska testmiljöer . . . 24
4.2 Genomförande av delsteg 2: Experiment . . . 25
4.2.1 Miljöer . . . 27

5 Resultat **28**
5.1 Översikt . . . 28
5.2 Self-organized recurrent network (SORN) . . . 29
5.3 Extended sequential cascaded network (ESCN) . . . 33
5.4 Analys . . . 38

6 Slutsats och diskussion **40**
6.1 Framtida arbete 41

Referenser **42**

A Rörelsebanor och aktiveringsvärden **47**
A.1 Self-organized recurrent network 47
A.2 Extended Sequential cascaded network 55

1 Introduktion

Traditionellt navigerar intelligenta agenter (en benämning på autonoma, självstyrande, entiteter) genom att förlita sig på en intern representativ modell som beskriver omgivningen. Genom att representera modellen internt har agenten möjlighet att planera en väg från en plats till en annan, även om slutmålet är utom synhåll. För att få en bättre bild över vad en representativ modell är kan denna liknas vid en karta, vilken beskriver för agenten var det är möjligt och inte möjligt att röra sig.

För vissa tillämpningar inom artificiell intelligens (AI) är det ibland önskvärt att använda representativa modeller för navigering, medan det för andra tillämpningar snarare är en nackdel. Exempelvis kan AI för dataspel gynnas av att använda representativa modeller då miljöerna är relativt enkla, medan det för robotar ofta medför nackdelar. Inom robotiken kan det vara svårt att alltid låta en robot ha en representativ modell tillgänglig. Detta kan bero på att miljön inte är känd på förhand eller att den förändras över tid. Frekventa förändringar i miljön är mycket svåra att modellera och det är svårt att hålla modellen konsistent (Alnajjar, Zin & Murase, 2009). Exempelvis har robotar som skickas ut i rymden dåliga möjligheter att kunna hålla en modell av omgivningen internt, då miljön sällan är känd på förhand.

Miljörepresentationer lämpar sig bättre för statiska miljöer, det vill säga miljöer som inte förändras över tid (Graham, McCabe & Sheridan, 2005). Brooks (1991) lägger fram flera argument mot representativa modeller och varför de inte är önskvärda att använda inom robotiken. Abstraktionsnivån bestäms implicit av representationen och gör det omöjligt för roboten att ta beslut om något som inte återspeglas i representationen. De representativa modellerna är förenklingar av omvärlden och återspeglar därför inte verkligheten till fullo (Brooks, 1990, 1991). Brooks föreslår att istället förlita sig på själva miljön än en representation av denna. Genom att med sensorer läsa av omgivningen kan den faktiska omgivningen användas som en typ av modell, där alla detaljer finns med. Agenten förlitar sig på så vis inte längre på en förenklad intern representativ modell och den begränsande abstraktionsnivån är inte längre ett problem.

I denna rapport presenteras en undersökning av två arkitekturer för artificiella neurala nätverk och hur väl de fungerar som kontrollsystem hos en agent, vars uppgift är att navigera genom miljöer av olika grader av komplexitet. Agenten utgörs av en simulerad miniatyrrobot av typen Khepera. Då ingen intern representativ modell finns tillgänglig används sensorer för att läsa av omgivningen. Det artificiella neurala nätverket har som uppgift att tolka sensordata och förse agenten med styrinstruktioner. I undersökningens fokus står de två arkitekturerna Extended sequential cascaded network (ESCN, Ziemke, 1999, 2000) och Self-organized recurrent network[1] (SORN, Jakobi, 1997a, 1998a).

Härnäst, i kapitel 2 introduceras navigationsproblemet, neurala arkitekturer och andra begrepp som är centrala för undersökningen. Kapitel 3 presenterar arbetets syfte delsteg, metoder samt

[1] Namnet Self-organized recurrent network är lånat från Thieme (2002), då Jakobi inte uttryckligen namnger arkitekturen.

tidigare arbeten som ligger till grund för undersökningen. I kapitel 4 presenteras undersökningens genomförande och experimentens upplägg. Resultaten av experimenten presenteras sedan i kapitel 5 och slutligen i kapitel 6 följer en diskussion där slutsatser kring arbetet dras.

2 Bakgrund

Detta kapitel syftar till att ge läsaren en djupare förståelse av bakgrunden som detta arbete vilar på och en förståelse för några av de begrepp som används inom artificiell intelligens. Kapitlet inleds i 2.1 med en introduktion till området artificiell intelligens, där den klassiska approachen till AI jämförs med moderna approacher. I 2.2 beskrivs navigeringsproblemet, vilket är det centrala problemet i undersökningen. I 2.3 följer en presentation av artificiella neurala nätverk (ANN) samt några arkitekturer som är intressanta för undersökningen. I sektion 2.4 presenteras begreppet artificiell evolution, där fokus ligger på genetiska algoritmer. Slutligen, i sektion 2.5, introduceras läsaren till träning i simulation och varför simulering ibland är att föredra framför att träna en robot i verkliga världen. Men som det kommer visas är träning i simulation inte helt utan risker.

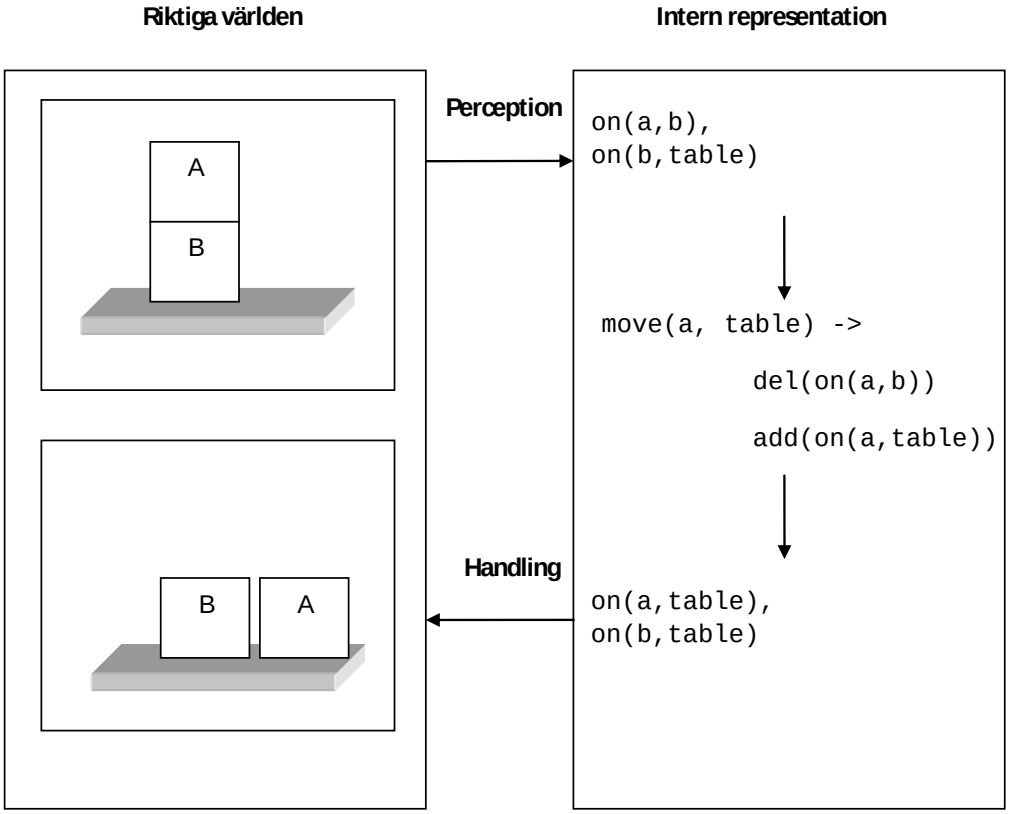
2.1 Klassisk och modern AI

Inom klassisk AI togs den så kallade physical symbol system hypothesis fram, där det påstås att varje intelligent system behöver manipulera datastrukturer av fysiska symboler för att kunna uppvisa intelligens (Brooks, 1991; Russel & Norvig, 2010). Symbolsystemen blir i viss mening representationer för det som det intelligenta programmet skall ta beslut om. Figur 2.1 illustrerar detta.

Inom klassisk AI var det inte populärt (dock inte helt uteslutet) att förkroppsliga de intelligenta agenterna i form av robotar. Forskningen fokuserade istället på att undersöka enskilda aspekter av intelligens i isolation. Detta var emellertid något som senare skulle ändras då robotar i mitten på 80-talet började ses som det bästa sättet att undersöka intelligenta beteenden (Brooks, 1990; Russel & Norvig, 2010; Ziemke, 2001).

När robotar väl började bli mer populära låg fokus ofta på vilken funktionalitet kontrollsystemen skulle brytas ned i (se figur 2.2). Brooks (1991) menar emellertid att den mänskliga intelligensen är för komplex för att brytas ner i funktioner. Under sent 1980-tal introducerade därför Brooks (1986) en alternativ approach där robotens beteenden (eller kompetenser, så kallade *task-achieving behaviors*) istället låg i fokus (se figur 2.3). I samband med detta presenterar Brooks (1986, 1990, 1991) också den så kallade subsumption architecture, vilken är tätt kopplad till idén kring nedbrytning till beteenden istället för funktioner.

En stor skillnad mellan nedbrytning av funktionalitet och nedbrytning av beteenden är att en ensam beteendemodul ensamt kan generera ett beteende för hela systemet. Beteendemoduler fungerar således i isolation, medan de funktionsmoduler som användes inom klassisk AI behövde kombineras för att få systemet nå önskat beteende. Att hitta rätt kombination av funktionsmoduler för att uppnå ett visst beteende är svårt och det försvåras ytterligare då gränssnitten mellan modulerna ofta blir väldigt komplexa (Brooks, 1991). Det område inom robotiken som fokuserar på beteenden brukar benämnas som beteendebaserad robotik (*eng. behavior-based robotics*) (Wahde, 2007).



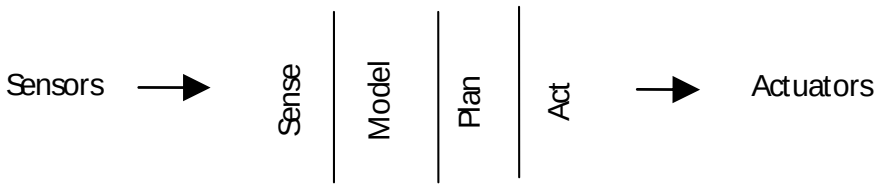
Figur 2.1: Illustration över symbolsystem. Riktiga världen visas på vänster sida och representationen på höger sida. (Efter Aktius, 2007, s. 15).

Brooks (1990) gör påståendet att physical symbol system hypothesis, på vilken mycket av forskningen inom klassisk AI var baserad på är, som Brooks (1990, sid. 1) uttrycker det, ”fundamentally flawed”. AI som baseras på representativa modeller begränsas av en abstraktionsnivå som implicit bestäms av modellen. Brooks (1990, 1991) menar därför att en förkroppsligad (*eng. embodied*) agent som är situerad i den riktiga världen bör använda världen som modell, istället för en representation av denna. The physical grounding hypothesis som presenteras av Brooks (1990, sid. 3) säger att ”... to build a system that is intelligent it is necessary to have its representations grounded in the physical world”. Den situerade agenten måste därför använda sensorer och aktuatorer för att interagera med världen.

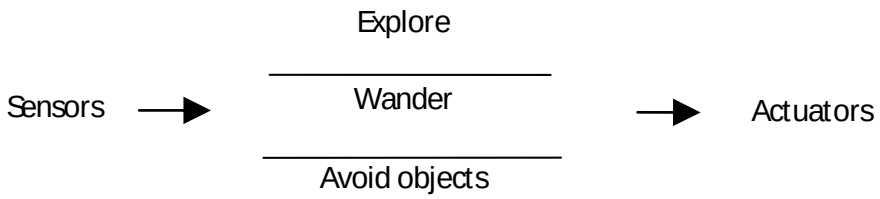
2.1.1 Navigering med hjälp av representation

Navigering med hjälp av en representativ modell är ett (klassiskt) alternativ till den modernare approach som annars presenteras här (dvs. att tolka sensordata med artificiella neurala nätverk och använda omgivningen som modell).

En representativ modell beskriver miljön i vilken agenten befinner sig i. De används främst då miljön är känd eller delvis känd på förhand. I statiska miljöer beskriver ofta modellen hela miljön



Figur 2.2: Nedbrytning av ett robotkontrollsystem till funktionsmoduler, enligt klassisk AI. (Efter Brooks, 1986; modifierad enl. Búason, 2002).



Figur 2.3: Nedbrytning av ett robotkontrollsystem till beteendemoduler. (Efter Brooks, 1986; modifierad enl. Búason, 2002).

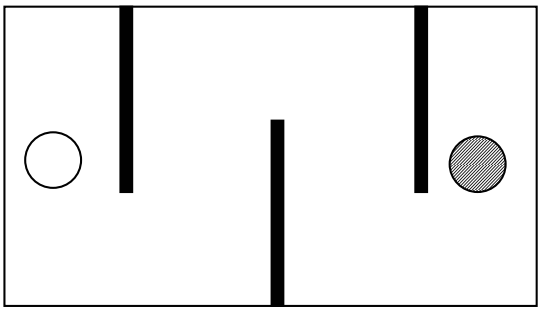
och är därför, i någon mening, pålitlig. I dynamiska miljöer är det dock mycket svårt att hålla modellen uppdaterad på ett sådant vis att denna exakt återspeglar hur miljön ser ut i varje tidssteg. Att endast förlita sig på representativa modeller i dynamiska miljöer kan orsaka stora problem (Graham et al, 2005; Graham, 2006). När det gäller robotar menar även Alnajjar et al (2009) att det är mycket svårt att modellera frekventa förändringar i den fysiska miljön.

Dynamiska miljöer är i många fall att likställa med delvis kända miljöer. De dynamiska elementen i samband med att det inte är möjligt att hålla modellen uppdaterad gör miljön oförutsägbar (även om en viss grundstruktur kan bibehållas). Algoritmer som exempelvis D* (Dynamic A-star) är framtagna speciellt för att hantera miljöer som är dynamiska och delvis kända (Stentz, 1994). D*-algoritmen anses dock av flera (Graham et al, 2005; Tomlinson, 2004) vara beräkningsmässigt för dyr att användas i praktiken.

I praktiken tar en representativ modell ofta formen av en graf, på vilken en stor mängd algoritmer[1] för grafsökning kan appliceras (Buckland, 2005; Cormen, Leiserson & Rivest, 1990; DeLoura, 2000; Korf, 1998 m.fl.). Genom att söka och planera en väg genom grafen planeras även en väg genom miljön. Det är med hjälp av modellen möjligt för agenten att planera vägar till platser som inte är inom synhåll eller som tidigare inte har besökts. För vissa tillämpningar av AI (t.ex. AI för dataspel), är ett av de största målen att upprätthålla en sorts illusion av intelligens. Med detta i åtanke är det kanske värt att ifrågasätta om det är ett trovärdigt och intelligent beteende att kunna navigera i miljöer som tidigare inte besökts, med ett optimalt[2] resultat.

[1] För ett verk som på ett bra sätt sammanfattar flera användbara sökalgoritmer för AI hänvisas läsaren till Korf (1998).

[2] Med optimalt menas här den kortaste och således effektivaste vägen till målet.



Figur 2.4: En statisk miljö där en agent navigerar från en plats (vit cirkel) till en annan (mörk cirkel).

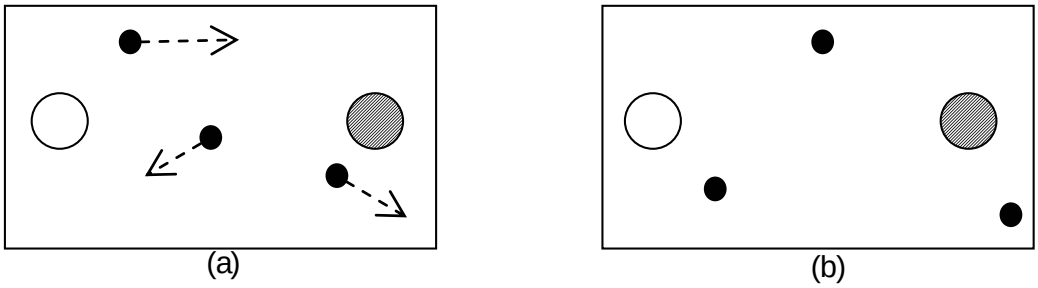
A*-algoritmen (Hart, Nilsson & Raphael, 1968)[3] är en väl ansedd sökalgoritm för statiska miljöer. Den är en vidarebyggnad på den kända Dijkstras algoritm (Dijkstra, 1959), vilken används för att finna kortaste vägen mellan två noder i en graf. A* använder sig av en (till problemet specifik) heuristisk beräkning för att öka sannolikheten att färre noder expanderas och att målet nås på kortare tid. Korf (1998) beskriver uträkningen för kostnaden från startnod till nod n enligt den generella formeln $f(n) = g(n) + h(n)$, där $g(n)$ är kostnaden från startnod till nod n och $h(n)$ är den uppskattade heuristikkostnaden från nod n till målnod. Således är $f(n)$ den uppskattat kortaste vägen från startnod till målnod, givet att heuristikuträkningen är *admissible*, innebärande att den aldrig överskattar kostnaden.

2.2 Navigeringsproblemet

Navigering är ett återkommande begrepp i denna undersökning och är därför värt att presentera i detalj. Att navigera innebär (i detta sammanhang) att ta sig från en plats till en annan i sin miljö (se figur 2.4). Detta kan verka ytterst trivialt vid första anblick, men under ytan döljer det sig flera problem som behöver hanteras. Miljöer kan, som är fallet i denna undersökning, vara dynamiska. Detta tillför ytterligare komplexitet, då även aspekten av tid spelar in. En dynamisk miljö, till skillnad från en statisk, förändras med tiden. Miljön vid tiden t behöver inte nödvändigtvis se samma ut vid $t + 1$ (se figur 2.5). De dynamiska inslagen behöver varken vara många eller frekventa för att göra miljön dynamisk, men ovissheten om miljön förblir samma eller ej orsakar ändå problem.

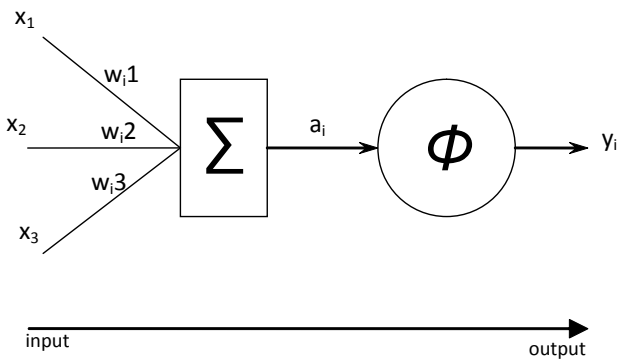
Ett av de traditionella sätten att lösa navigationsproblemet på, genom representativa modeller, blir på grund av detta ett mycket ineffektivt sätt hantera navigering på. Det behövs därför ett alternativt sätt att lösa navigationsproblemet på. Brooks (1991) föreslår en approach där miljön betraktas som själva modellen. Agenten avläser omgivningen med sensorer och tolkar sensordata för att producera styrinstruktioner. Sättet som sensordata tolkas kan variera. Exempelvis tolkar Brooks (1986) sensordata en mängd tillståndsmaskiner. I undersökningen som presenteras här tolkas sensordata emellertid med artificiella neurala nätverk.

[3] För en mindre formell beskrivning av A*-algoritmen se Buckland (2005).



Figur 2.5: En dynamisk miljö i två tidssteg, t (a) och t+1 (b). Dynamiska objekt (hinder) illustreras med svarta cirklar. Streckade pilar indikerar riktning som dynamiska objekt rör sig.

2.3 Artificiella neurala nätverk



Figur 2.6: En artificiell neuron med tre inkommande kopplingar. Summan av de inkommande kopplingarnas vikter ger värdet a_i och neuronen producerar med aktiveringsfunktionen ϕ output y_i.

Artificiella neurala nätverk (ANN) är beräkningsmodeller implementerade i mjukvara eller hårdvara som syftar till att ta vara på egenskaper som återfinns hos biologiska nervsystem. Ett artificiellt neuralt nätverk byggs upp av många, enkla, beräkningselement som kopplas samman. Dessa beräkningselement kallas noder eller neuroner (egentligen artificiella neuroner, se figur 2.6), men ges traditionellt olika namn beroende på var i nätet de är placerade. Inputneuroner kallas de neuroner som blir matade med information från omgivningen (exempelvis med data från sensorer). Outputneuroner kallas de neuroner som påverkar omgivningen genom att skicka en outputström tillbaka till omgivningen (exempelvis till en motor). Slutligen finns också så kallade gömda (eller interna) neuroner. Dessa kommunicerar uteslutande med andra neuroner och är således ”gömda” från omgivningen (Floreano & Mattiussi, 2008).

Det finns flera egenskaper hos artificiella neurala nätverk som gör att de ofta uppskattas för olika sorters tillämpningar. Floreano & Mattiussi (2008) beskriver flera av dessa egenskaper, som bl.a. robusthet, flexibilitet och generaliserbarhet. Att ANN har möjligheten att kunna generalisera innebär att efter att ha tränats med ett antal exempel kunna ta beslut om liknande, men inte tidigare påträffade, fall. Detta gör ANN väldigt användbara i vissa situationer.

I biologiska nervsystem kan neuroner vara antingen hämmande (*eng. inhibitory*) eller stimulerande (*eng. excitatory*). Artificiella neuroner kan dock vara både och, vilket i praktiken innebär att de kan ge ifrån sig både positiva och negativa outputsignaler, där negativa fungerar hämmande och positiva fungerar stimulerande (Floreano & Mattiussi, 2008). Varje artificiell neuron (hädanefter bara neuron) i ett nätverk har en koppling till minst en annan neuron, antingen som inkommande eller utgående koppling. Varje neuron i har en inputsignal ai, en outputsignal y_i och en aktiveringsfunktion ϕ. Dessutom har också varje koppling en vikt w. Inputsignalen a_i beräknas genom att summera alla inputsignaler:

$$a_i = \sum_{j=1}^{N} w_{ij} x_j \tag{2.1}$$

Där w_{ij} är inkommande koppling till neuron i från neuron j och x_j är aktiveringsvärdet från neuron j.

En av de enklaste aktiveringsfunktionerna (och den tidigaste, enl. Floreano & Mattiussi, 2008) är stegfunktionen, vilken avger 1 om den totala inputsignalen (summerad enligt formel 2.1) överstiger tröskelvärdet ϑ, annars avger den 0:

$$\phi(a_i) = \begin{cases} 1: & \sum_{j=1}^{N} w_{ij} x_j > \vartheta \\ 0: & annars \end{cases} \tag{2.2}$$

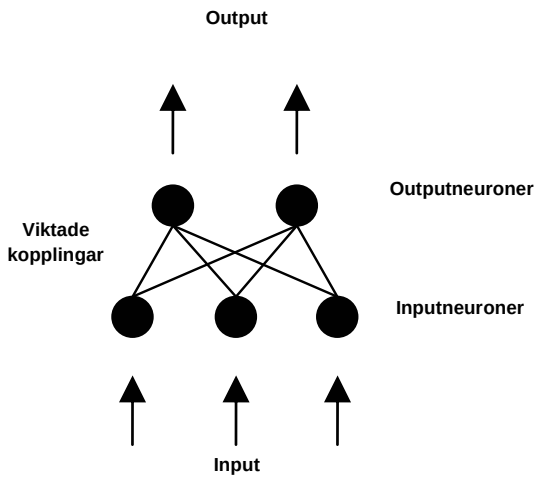
I vissa fall krävs det att output från neuroner kan anta andra värden än 0 och 1. Exempelvis kan strömmen till en motor behöva ligga emellan dessa extremvärden för att en robot skall nå önskad hastighet. En stegfunktion har därför vissa begränsningar och ett alternativ till denna är den sigmoida funktionen. Den sigmoida funktionen ger ett reellt tal inom ett visst intervall, ofta [0, 1], vilket ges av funktionen

$$\phi(a_i) = \frac{1}{1 + e^{-a_i}} \tag{2.3}$$

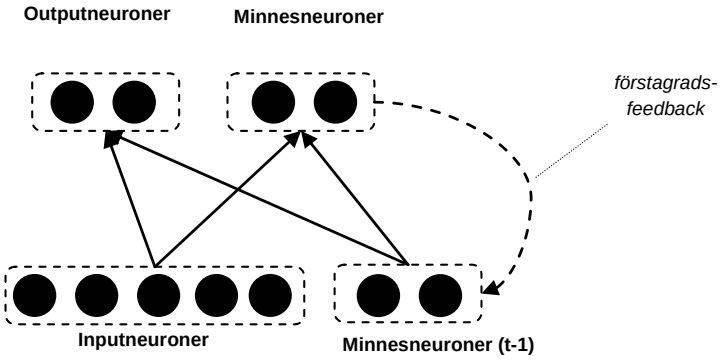
Det finns många sorters arkitekturer för ANN. Två av dessa, feedforward- och rekurrenta (återkopplande) nät, presenteras kort här då de är av betydelse för kommande delar av rapporten. Nät av typen feedforward är relativt simpla och har ett helt reaktivt beteende (dvs. de agerar inte utifrån minne eller genom återkoppling av tidigare output), vilket kan göra dem otillräckliga för vissa tillämpningar (t.ex. enligt Graham et al, 2005). Figur 2.7 illustrerar en feedforward-arkitektur utan gömda neuroner, ett så kallat perceptron-nät. Rekurrenta nät (figur 2.8) har i motsats till feedforward-nät återkoppling ”tillbaka”, vilket medför ett visst korttidsminne som kan användas i ett följande tidssteg. Rekurrenta nät har med framgång använts av flera som kontrollmekanismer och styrsystem för robotar (Búason & Ziemke, 2003; Jakobi, 1997a; Ziemke, 1999 m.fl.).

2.3.1 Extended sequential cascaded network (ESCN)

Extended sequential cascaded network (figur 2.10) är en arkitektur som beskrivs av Ziemke (1999) och bygger vidare på Sequential cascaded network (SCN, figur 2.9). ESCN beskrivs som ett nätverk uppbyggd av två nätverk; ett funktionsnätverk och ett kontextnätverk. Kontextnätverket be-



Figur 2.7: Artificiellt neuralt nätverk med feedforward-arkitektur (utan gömda lager, ett så kallad perceptronnät).

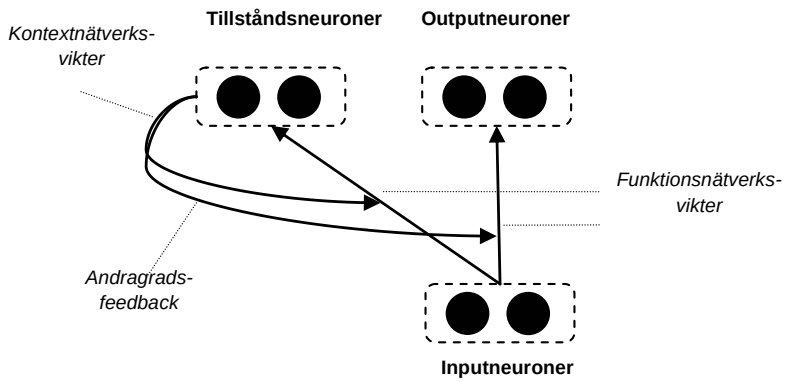


Figur 2.8: Ett enkelt rekurrent (återkopplande) artificiellt neuralt nätverk. (Efter Ziemke, 1999).

räknar om funktionsnätets vikter för nästa tidssteg, med hjälp av tillståndsneuronerna. ESCN använder emellertid en beslutsneuron[4] (vad som skiljer ESCN mot SCN) för att åstadkomma selektiv feedback (Ahlén, 2003; Ziemke, 1999).

Både SCN och ESCN använder så kallad feedback av andra ordningen (alt. av högre ordningen), till skillnad från enkla rekurrenta nätverk (figur 2.8) som använder feedback av första ordningen (Ziemke, 1999). När feedback av första ordningen används tas vanligtvis viss neuronoutput och matas tillbaka som input i ett senare tidssteg, vanligtvis nästa (observera minneslagren i figur2.8). Ziemke (1999) beskriver feedback av andra graden som en förändring av vikterna (i funktionsnätverket) där den sensomotoriska mappningen beräknas om. Förändringarna av vikterna är beroende av tillståndsneuroner (observera tillståndsneuronerna i figur 2.9 och 2.10).

[4]Bergfeldt & Linåker (2002) använder en gating-node för att åstadkomma liknande funktionalitet i den multilagerarkitektur som de presenterar.



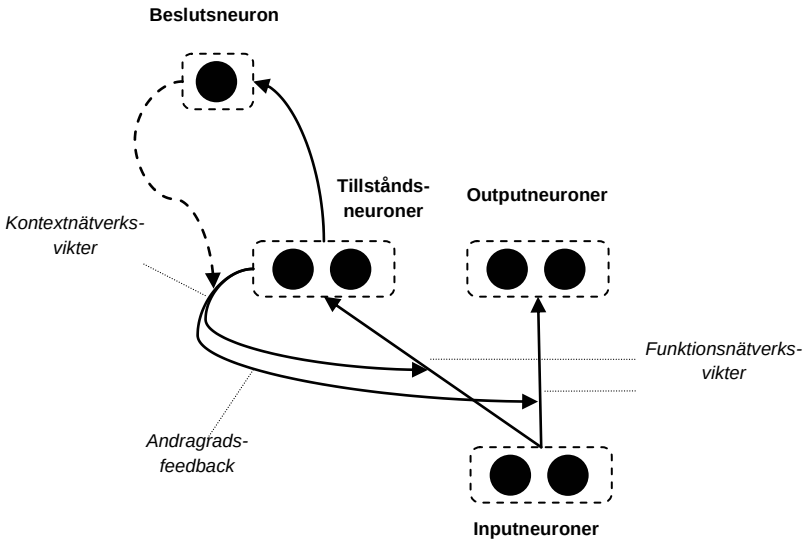
Figur 2.9: Sequential cascaded network. (Efter Ziemke, 1999).

2.3.2 Self-organized recurrent network (SORN)

Jakobi (1997a, 1998a) beskriver en evolutionär och självorganiserande approach till kontrollsystem för Khepera-roboten. Traditionellt är det endast vikterna som genomgår förändring vid belöningsbaserad, evolutionär inlärning. I den arkitektur som presenteras av Jakobi (1997a, 1998a) genomgår även själva kopplingarna evolution. Det vill säga att kopplingarna i nätet inte är förutbestämda av den som designar nätet, utan bestäms av agenten själv (under evolution). Arkitekturen (figur 2.11) använder inga gömda neuroner utan förlitar sig på interaktion mellan inputneuroner och outputneuroner. Arkitekturen har inte heller den tydliga lagerstruktur som återfinns hos t.ex. feedforward-arkitekturer. Inputneuroner kan alltså mycket väl vara kopplade till andra inputneuroner och outputneuroner kan återkoppla till enskilda inputneuroner, vilket gör arkitekturen rekurrent. Varje neuron kan maximalt ha tre inkommande kopplingar från andra neuroner i nätverket. Givet åtta sensorer och två motorer (dvs. 10 neuroner) resulterar detta i 30 potentiella vikter och kopplingar som genomgår evolution. Arkitekturens genetiska representation utgörs av en bitsträng av 140 bitar. Strängen kodar dock endast halva nätverket, vilket resulterar i att nätverket sedan speglas (Jakobi, 1996, 1997a). Anledningen till detta är på grund av att Jakobi (1996, 1997a, 1997b, 1998a, 1998b) fokuserar på så kallade minimala simulationer (se sektion 2.5). Att halvera storleken på den genetiska representationen leder till en snabbare träning, men på grund av den framtvingade symmetrin utesluts också potentiella lösningar.

2.4 Artificiell evolution

Många av idéerna kring artificiell evolution är inspirerade av naturlig evolution. Artificiell evolution skiljer emellertid sig från naturlig evolution på flera avseenden, men vad som framförallt skiljer dem åt är det faktum att artificiell evolution i de allra flesta fall har fördefinierade mål medan naturlig evolution är en kontinuerlig process utan distinkta mål (Floreano & Mattiussi, 2008). Centrala begrepp är population, mångfald, genetiskt arv och selektion. Population kan beskrivas som en samling individer. Mångfald betyder i sammanhanget att populationen av individer är varierad. Om alla individer skulle bestå av samma genetiska struktur skulle populationen inte utvecklas på



Figur 2.10: Extended sequential cascaded network. (Efter Ziemke, 1999).

ett tillräckligt bra sätt, vilket skulle leda till att evolutionen i det närmaste skulle fastna. Föräldrar för sina egenskaper vidare genom genetiskt arv och nya individer ärver egenskaper från sina föräldrar (Floreano & Mattiussi, 2008).

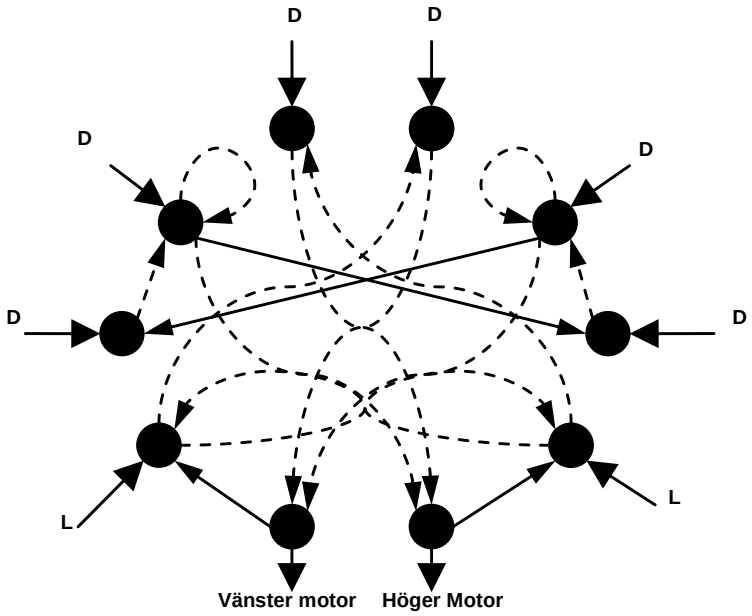
Genetiska algoritmer, en sorts söktekniker och optimeringstekniker som bygger på artificiell evolution, presenteras härnäst. De används ofta för att optimera konfigurationer av artificiella neurala nätverk. Men även om fokus hos detta arbete ligger på ANN bör det nämnas att genetiska algoritmer kan tillämpas på en stor mängd andra problem. Exempelvis har genetiska algoritmer med framgång använts för att optimera konfigurationer för Petri nets (Dahlbom, 2011).

2.4.1 Genetiska algoritmer

Genetiska algoritmer är i grunden söktekniker (optimeringstekniker), speciellt lämpade för icke-linjära problem som beror på flera variabler. En individ i en genetisk algoritm är en kandidatlösning och beskriver därför en möjlig lösning till problemet.

Inom genetiska algoritmer finns det många termer som står i analogi till termer som används inom naturlig genetik. En gen är det minsta byggblocket i en genetisk algoritm. Flera gener bildar en kromosom och en eller flera kromosomer bildar ett genom. Varje gen har en egen plats på kromosomen, benämnd locus. En gen kan förekomma i flera varianter, där varje variant kallas för allel. Alleler ger upphov till olika egenskaper och drag. Ponera att en människas hårfärg bestäms av endast en gen på kromosomen. Genens alleller ger då upphov till olika hårfärg. En organisms genetiska struktur, dess genotyp, utgörs av alla dess kromosomer. I praktiken består ett genom i en genetisk algoritm vanligtvis av endast en kromosom. Kromosom och genom är därför två termer som i det närmaste betyder samma sak (Floreano & Mattiussi, 2008; Mitchell, 1996).

En genetisk algoritm verkar i generationer, där en population genomgår förändring allteftersom evolutionen fortskrider. Förhoppningen är att varje ny generation skiljer sig något från den tidigare på ett sådant sätt att den nya har en aning bättre egenskaper. Ett värde tilldelas till varje individ



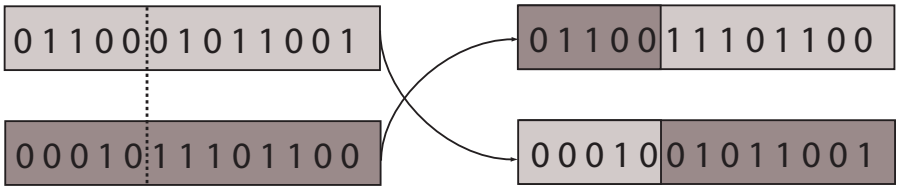
Figur 2.11: En möjlig variant av the ”Self-organized recurrent network” - SORN. Streckade pilar symboliserar negativa vikter och svarta pilar symboliserar positiva vikter. D symboliserar distanssensorer (IR) och L symboliserar ljuskänsliga (ambient) sensorer. (Efter Jakobi, 1997a, s. 347).

i populationen som beskriver hur väl en individ löser problemet. Detta värde benämns som individens fitness-värde. Sättet som fitness-värdet tilldelas är alltid problemspecifikt (Floreano & Mattiussi, 2008; Mitchell, 1996).

Det finns generellt inga garantier för att en efterföljande generation skall bestå av en mängd bättre lösningar (elitism kan dock garantera att efterföljande generation innehåller minst en mängd lika bra individer, se nedan). Genetiska algoritmer bygger på principen att en stark individ, med högt fitness-värde, har högre sannolikhet att (genom selektion) föra sina gener vidare till nästkommande generation. Detta är analogt till frasen survival of the fittest (Mitchell, 1996; Sims, 1994).

Evolution genom en genetisk algoritm kan ses som något som sker i flera steg. Det första, och kanske mest uppenbara, är att individer i populationen förökar sig och producerar avkomma (i form av nya individer). Reproduktion, överkorsning, sker med en överkorsningsoperator. Det finns flera olika typer, där enpunktsöverkorsning (*eng. single-point crossover*) är ett exempel på en enkel sådan (se figur 2.12).

Det nämndes tidigare att starkare individer har större chans att föröka sig. Detta upprätthålls genom en selektionsoperator. Det finns även flera olika typer av selektionsoperatorer, där rouletthjulsselektion och turneringsselektion är två exempel på vanligt förekommande. Kopplat till selektion är också mekanismen kallad elitism. Med elitism tas de n bästa individerna med till nästkommande generation. Elitism kan ge upphov till snabbare konvergering men kan också öka sannolikheten att algoritmen fastnar i ett lokalt optima, vilket kan vara svårt att ta sig ur (Floreano & Mattiussi, 2008).



Figur 2.12: Överkorsningoperator (enpunkts) tillämpas på två kromosomer (till vänster i bild). Överkorsningspunkten väljs slumpmässigt och barnen (till höger i bild) tar gener från båda föräldrarna.

Ett ytterligare steg som sker är mutation. Vid mutation förändras en eller flera gener hos individen, för att ibland bidra med en ny egenskap till populationen. Är det en god egenskap har den en större sannolikhet att föras vidare till nästkommande generationer än om egenskapen vore en nackdel. Eftersom mutation inte är beror på populationens mångfald kan mutation också hjälpa algoritmen ur ett lokalt optima, om denna har fastnat i ett sådant. Ett lokalt optima[5] motsvarar en mängd lösningar som upplevs som bra, men som inte är de bästa möjliga lösningarna, vilka utgörs av ett globalt optima (Floreano & Mattiussi, 2008; Mitchell, 1996).

Genetiska algoritmer används ofta för att hitta en god konfiguration av komplexa typer av system. Dessa system är i sammanhanget ofta artificiella neurala nätverk (Buckland, 2002; Mitchell, 1996; Parker & Zhiyi, 2003; Ziemke, 1999 m.fl.). ANN tillämpas inte sällan på problem vars natur gör det svårt att i förväg känna till korrekt output givet en viss input. I dessa fall används genetiska algoritmer som träningsapproach (känt som belöningsbaserad träning) för att hitta en ANN-konfiguration som löser problemet tillräckligt bra (Buckland, 2002, Mitchell, 1996).

För evolution av kontrollsystem för robotar är det inte nödvändigtvis sant att överkorsning producerar bättre avkomma (Nolfi, 1996; Nolfi & Floreano, 2000; Ziemke, 1999). Meeden (1994, sid. 24) skriver följande angående överkorsning av artificiella neurala nätverk:

> When evolving networks, crossover would create a new set of weights by taking some weights from one successful network and the rest from another successful network. However, these two networks could be using very different strategies for solving the problem. Creating an offspring by recombining portions of their weights may result in an extremely poor alternative solution. In fact even if the two networks are employing the same strategy, it is probably instantiated in the weights in very different ways. So again, recombination may be quite unsuccessful.

Det är därför troligt att asexuell reproduktion (dvs. reproduktion genom mutation) är att föredra vid utformning av experiment rörande robotik med ANN. Slutligen nämner Risi, Hughes & Stanley (2010, s. 471) någonting som är värt att hålla i åtanke vid utformning av fitnessberäkningar "... the more ambitious the goal, the more likely it is that search can be deceived by suboptimal solutions".

[5] Benämnas ofta även som lokalt respektive globalt minima.

2.5 Simulering och verklighetsgapet

Som uppmärksammades i sektion 2.1 menar Brooks (1990, 1991) att agenten behöver vara situerad i, och interagera med, den riktiga världen för att ens ha en chans att visa på något sorts intelligent beteende (enkelt uttryckt; agenten behöver ta formen av en robot). En central aspekt hos robotar inom de modernare underområdena till AI (t.ex. evolutionär robotik) är att de skall ha förmåga att lära (Floreano & Mattiussi, 2008; Wahde, 2007). Det finns dock många problem förknippade med robotinlärning. Denna sektion lyfter fram några av dessa problem.

Beteenden som för oss människor framstår som triviala kan för en robot vara väldigt komplexa. Ett sådant beteende som har undersökts i stor utsträckning är det för att undvika hinder (Brooks, 1986; Jakobi, Husbands & Harvey, 1995; Nolfi, 1997 m.fl.). Evolutionära träningstekniker har givit robotar möjlighet att, i någon mening, själva bestämma hur de skall åstadkomma sådana beteenden. Detta tar emellertid lång tid då sökrymden normalt sett är oerhört stor (Jakobi, 1998b). Det tar i de allra flesta fall så lång tid att göra i realtid att denna approach ofta är utesluten.

Jakobi (1997a) beskriver experiment inom evolutionär robotik som, om de hade utförts i realtid, skulle tagit mer än 17 månader. När de utfördes i simulation tog de istället ca 4 timmar. Floreano & Modada (1994) tränade en robot till ett hinder-undvikande beteende, vilket tog 65 timmar i simulation. Vidare medför träning i den verkliga världen flera andra problem som måste lösas, som exempelvis strömförsörjning, slitskador, automatisering av fitnessutvärdering. (Jakobi, 1998b).

Simulering är ett alternativ till detta. Målet är då att träna ett kontrollsystem i simulering, för att sedan flytta över det tränade kontrollsystemet till en fysisk robot. Förhoppningen är att denna robot sedan skall uppvisa samma beteende som i simulering. Jakobi (1997a) använder begreppet verklighetsgap (*eng. the reality-gap*) och menar att om en robot som tränats i simulation skall visa samma beteende i verkligheten måste denna korsa verklighetsgapet. Men som bl.a. Brooks (1992), Jakobi et al (1995) och Jakobi (1997a, 1997b, 1998a, 1998b) påpekar finns allvarliga brister med att låta en robot tränas i simulering. Simulatorn behöver t.ex. vara noggrant designad för att modellera världen på ett tillräckligt bra sätt. Brooks (1992, sid. 2) är emellertid mycket kritisk:

> There is a real danger (in fact, a near certainty) that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing and actuation—it is very hard to simulate the actual dynamics of the real world.

Jakobi (1998b) menar att om hela omvärlden skall modelleras tillräckligt noga kommer simulationen bli oerhört komplex. Många av fördelarna med simulering kommer att gå förlorade. Tiden det tar att utveckla miljön tillsammans med att det är beräkningsmässigt för dyrt att köra simuleringarna kommer att överskugga fördelarna. Jakobi använder istället så kallade minimala simuleringar.

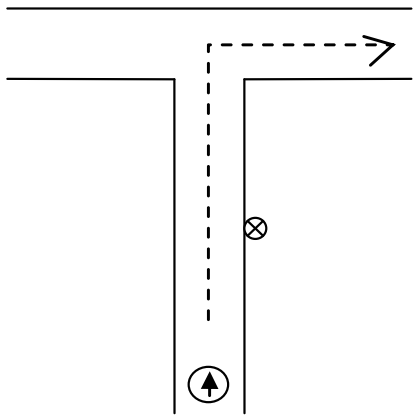
Grundidén med minimala simuleringar är att endast de delar av miljön som är avgörande för att roboten skall nå ett visst beteende modelleras. Dessa aspekter kallas för basaspekter (*eng. base-set aspect*s) och tillsammans utgör de basmängden (*eng. base-set*). Resterande delar av miljön benämns som implementationsaspekter (*eng. implementation aspects*). Implementationsaspekterna slumpas mellan varje körning i miljön för att undvika att roboten blir beroende av dessa. Om roboten inte utvecklar ett beroende av implementationsaspekterna kallas det för att roboten är base-set

robust. Nackdelen med minimala simuleringar, som Nolfi & Floreano (2002) uppmärksammar, är att den som designar simuleringen på förhand måste känna till (eller göra antaganden kring) vilka aspekter som skall tas med i basmängden.

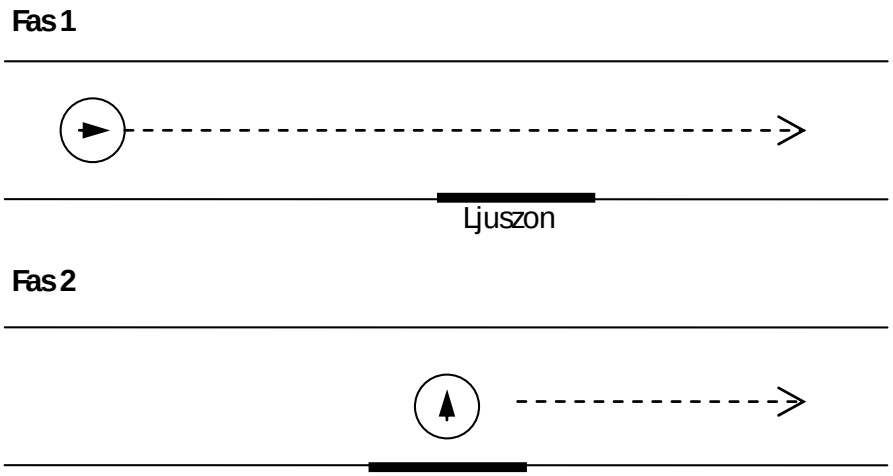
Brooks (1992) uppmärksammar att robotars sensorer ofta ger väldigt otydliga läsvärden. Sensorer i simuleringar ger enkelt perfekta läsningar, vilket helt enkelt är orealistiskt. Jakobi (1997a, 1998b) maskerar därför dessa skillnader genom att lägga på störningar på sensorläsningar under simuleringen. Jakobi kallar approachen för *envelope-of-noise*. Det har visat sig roboten kan korsa verklighetsgapet på bästa sätt när roboten tränas med en ungefärligt motsvarande mängd störningar som kan observeras hos den verkliga roboten. Om för mycket eller för lite störningar används under träningen har det också visats att den fysiska roboten inte beter sig på samma sätt som under simuleringen (Jakobi et al, 1995).

Jakobi (1997a, 1998a) demonstrerar minimala simuleringar genom att träna en Khepera-robot för T-korsningsproblemet genom att använda ”teleportering” (se figur 2.14). Enligt Jakobi (1997a) och Thieme (2002) tvingar teleportering agenten att explicit göra ett val om vilket håll den skall svänga åt. Detta innebär att agenten tvingas hålla ett internt tillstånd som beskriver det rätta hållet.

Minimala simuleringar eliminerar emellertid andra potentiella lösningar, som att exempelvis följa väggen ända till mål (Nolfi & Floreano, 2000). T-korsningsproblemet (även känt som the road-sign problem) går ut på att roboten placeras i botten av en miljö som tar formen av ett T (se figur 2.13). På vägen till vägskälet indikeras ”rätt håll” av en lampa placerad på en av korridorens väggar. Roboten kan emellertid inte se lampan när den når vägskälet utan måste komma ihåg vilket håll som indikerades av lampan. Det tränade kontrollsystemet laddades sedan ned till en fysisk robot, vilken testas i den riktiga världen. Roboten lyckades under testerna lösa T-korsningsproblemet i verkligheten på ett utmärkt sätt (Jakobi, 1997a). Resultaten av detta experiment presenteras vidare i sektion 3.1.1.



Figur 2.13: T-korsningsproblemet där ljuskällan är placerad på höger vägg.



Figur 2.14: En minimal simulation av T-korsningsproblemet. I fas 1 rör sig roboten genom korridoren och genom ljuszonen. När ljuszonen passeras teleporteras roboten till vägskälet (fas 2) för att tvingas välja ett håll. (Efter Jakobi, 1997a).

3 Problemformulering

Detta kapitel inleds med en presentation av tidigare arbeten som ligger till grund för den undersökning som presenteras här, i vilken en agent har som uppgift att navigera från startposition till målposition i miljöer av olika slag. Ett problem som försvåras ytterligare när miljöerna är dynamiska. Vidare följer undersökningens syfte och de delsteg som undersökningen innefattar. Slutligen presenteras de metoder som används för att lösa delstegen.

3.1 Tidigare arbeten

Här presenteras arbeten som ligger till grund för undersökningen i denna rapport. Flera av dessa arbeten använder Khepera-roboten, en typ av miniatyrrobot i sina undersökningar. Jakobi (1997a; 1997b; 1998a) gör experiment med bl.a. en Khepera-robot som tränas för T-korsningsproblemet i en simulatormiljö. Det undersöks sedan hur väl roboten klarar att korsa verklighetsgapet. Jakobi använder i detta experiment den självorganiserande arkitekturen SORN som kontrollsystem för roboten. Ziemke (1999) använder Khepera-roboten och utvärderar fem ANN-arkitekturer (däribland ESCN) i två olika experiment, där minne och adaptivt beteende undersöks. Vidare utvärderar Thieme (2002) fyra ANN-arkitekturer genom experiment med Khepera-roboten i sex olika varianter av den traditionella T-korsningsmiljön. Slutligen presenteras också arbetet av Graham (2006) och Graham et al (2003, 2005) som främst är riktat mot dataspel, men vars resultat rörande feedforward-nätverk ändå har viss relevans för undersökningen.

3.1.1 En robot som korsar verklighetsgapet (Jakobi)

Jakobi (1997a; 1997b; 1998a) genomför experiment med en Khepera-robot för att undersöka huruvida en robot som tränats i en minimal simulation (med teleportation, se figur 2.14) kan klara att korsa verklighetsgapet (se kap. 2.5). Jakobi använder i experimenten den självorganiserande arkitekturen (SORN) som också undersöks i denna rapport.

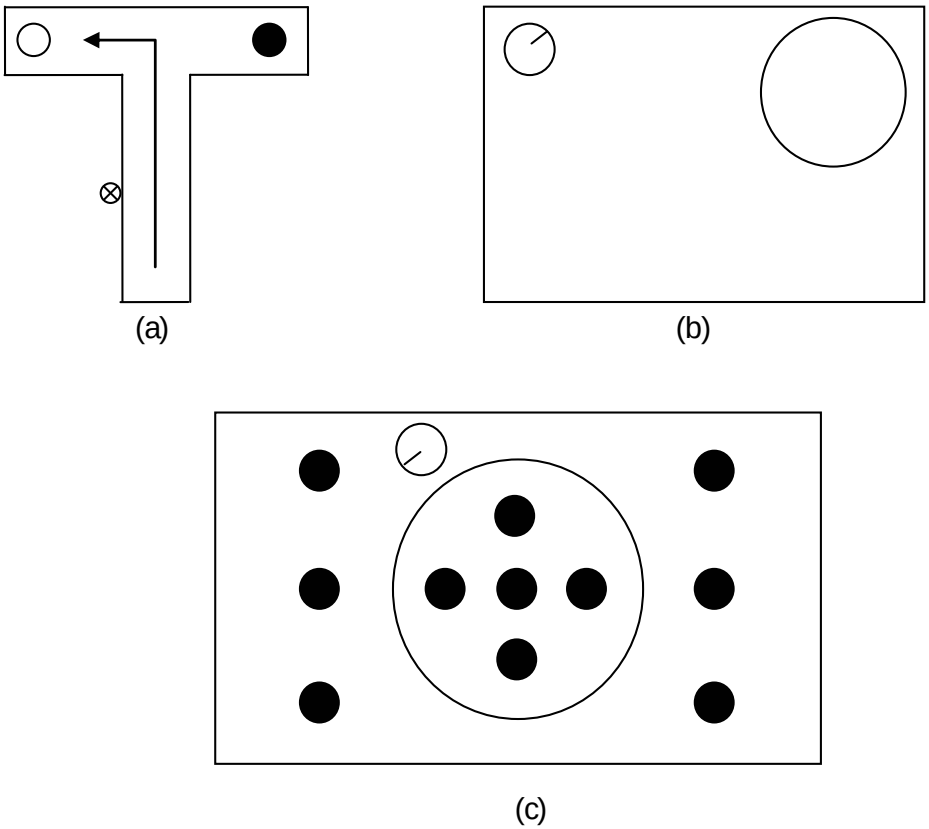
För att försäkra sig om att kontrollsystemet som evolveras är base set robust varieras implementationsaspekterna, dvs. de aspekter som roboten inte får bli beroende av. Det som varieras för varje körning i miljön inkluderar bl.a. (enl. Jakobi 1998a, sid 27):

- Korridorernas bredd mellan 13 och 23 cm.
- Robotens startriktning mellan ± 22,5 grader i riktning nedåt.
- Storleken och riktningen av det belysta området i korridoren mellan 2 och 12 cm.
- Längden av korridoren under fas 1 mellan 40 och 60 cm (se figur 2.14).
- Konstanter i formler som rör de modellerade sensorerna.

Roboten tränades med en genetisk algoritm över 1000 generationer där en populationsstorlek av 100 individer användes. Varje individ fick 10 försök på sig att lösa problemet och varje försök varade i 15 sekunder. Efter träningen i simulation (som tog ca fyra timmar) fördes kontrollsystemet över till en fysisk robot. Den fysiska roboten läts sedan försöka lösa T-korsningsproblemet i verkligheten under totalt 60 försök som utfördes i sekvens. För var tjugonde försök ändrades korridorbredden (11 cm, 18 cm respektive 23 cm) och för var tionde försök ändrades sidan som lampan var placerad på. Roboten svängde åt rätt håll i samtliga försök och lyckades därmed korsa verklighetsgapet.

3.1.2 Neurala arkitekturer för minne och adaptivt beteende (Ziemke)

I undersökningen av Ziemke (1999) utvärderas fem arkitekturer (däribland SCN och ESCN) för artificiella neurala nätverk i två experiment. I det första experimentet har agenten som uppgift att navigera till ett cirkulärt område i miljön och, efter agenten tagit sig innanför området, hålla sig kvar innanför så länge som möjligt (se figur 3.1b). I det andra experimentet är agentens uppgift att samla de svarta objekt som ligger inom den stora mittcirkeln och undvika de svarta objekt som ligger utanför (figur 3.1c). I samtliga körningar fick agenten en slumpad startposition och vinkel. Intressant att notera är att i fallet ESCN tilldelades inte agenten några poäng (dvs. fitnesspoäng) i de tidssteg där beslutsneuronen (se figur 2.10) var aktiv. Detta var, enligt Ziemke (1999), för att uppmuntra minimal användning av beslutsneuronen. Trots detta var det ESCN som i båda experimenten nådde bäst resultat.



Figur 3.1: Tre miljöer för experiment. (a) visar en T-korsning (T-maze) med en lampa (väster vägg) som indikerar "rätt håll". (b) visar den första experimentmiljön från Ziemke (1999). (c) visar den andra experimentmiljön från Ziemke (1999).

3.1.3 Multipla T-korsningar för undersökning av fördröjt svar (Thieme)

I undersökningen av Thieme (2002) utvärderas fyra arkitekturer för artificiella neurala nätverk. Nätverket fungerar som styrsystemet hos en (simulerad) Khepera-robot och robotens uppgift är att svänga åt rätt håll i en t-korsning (se figur 3.1a). Det enda hjälpmedlet agenten har är en lampa som indikerar åt vilket håll agenten skall svänga. Lampan är placerad innan korsningen och agenten har följaktligen inte tillgång till indikatorn när den väl skall svänga. Detta testar agenternas möjlighet att utföra uppgifter med fördröjt svar (*eng. delayed response*), vilket är det konventionella sättet att undersöka korttidsminne (Thieme, 2002; Ziemke & Thieme, 2002). Fyra arkitekturer utvärderas i totalt sex varierade T-korsningsmiljöer. Arkitekturerna i undersökningen utgörs av ett feedforward-nätverk, ett enkelt rekurrent nätverk, Sequential cascaded network (SCN) och Extended sequential cascaded network (ESCN).

Thieme (2002, s. 12) definierar i sammanhanget tillförlitlighet (*eng. reliability*) som ”... the percentage of trials in which the agent actually reaches the goal within a certain number of steps without running into any dead ends”. Tabell 3.1 visar resultaten av Thiemes (2002) undersökning. ESCN visar på bäst tillförlitlighet med 99,4 %. Värt att uppmärksamma är att det enkla rekurrenta

nätverket (likt det som visas i figur 2.8) visar på bättre tillförlitlighet än SCN. SCN visar således på betydligt sämre tillförlitlighet än ESCN, med bara strax över 4 procentenheter bättre det helt reaktiva feedforward-nätverket.

Tabell 3.1: Genomsnittliga tillförlitligheter för de fyra arkitekturerna i Thiemes (2002) undersökning. (Tabell efter Thieme, 2002).

Arkitektur	**Genomsnittlig tillförlitlighet**
Feedforward-nätverk	87,1 %
Enkelt rekurrent nätverk	92,5 %
Sequential cascaded network	91,2 %
Extended sequential cascaded network	99,4 %

3.1.4 Vägplanering i realtid för dataspel (Graham)

Graham och Graham et al beskriver i flera publikationer (Graham et al, 2003; Graham et al, 2005; Graham, 2006) en approach till navigering utan representation. Den är främst riktad mot AI för dataspel och kallas real-time pathfinding. Approachen bygger på att använda själva miljön som modell, istället för att förlita sig på en traditionell representativ modell (vissa likheter med Brooks (1991) idéer kan observeras). Sensorer används för att i realtid läsa av omgivningen. Sensordata matas till ett artificiellt neuralt nätverk, vilket producerar styrinstruktioner till agenten. Träning av ANN görs belöningsbaserat med en genetisk algoritm. Träningen sker på traditionellt vis genom att tillämpa den genetiska algoritmen på nätets vikter (och inte på den faktiska strukturen av nätet). I experimenten använder Graham ett feedforward-nätverk med ett gömt lager.

Experimenten (Graham et al, 2005; Graham, 2006) kan sammanfattas i tre steg. Det första är att lära agenten att röra sig mot ett mål. Det andra är att lära agenten att undvika hinder. Det tredje är att kombinera de båda tidigare stegen för att lära agenten att röra sig mot ett mål samtidigt som hinder undviks. Att träna agenten för de två första stegen var tämligen enkelt, men det tredje steget visade sig vara betydligt svårare. Graham et al (2005) påstår, att ett ANN med bara ett gömt lager inte är kapabelt att lära sig denna typ av beteende, vilket visar på vissa nackdelar med feedforward-nätverk för denna typ av tillämpning.

3.2 Syfte

Arbetet syftar till att utvärdera två arkitekturer för artificiella neurala nätverk, med avseende på hur tillförlitliga de är som kontrollmekanismer för agenter som navigerar. De två arkitekturerna i fråga är Extended sequential cascaded network (ESCN) och Self-organized recurrent network (SORN).

Agentens uppgift är att navigera genom miljön för att ta sig från startposition till målposition. Kolliderar agenten med en vägg eller ett hinder betraktas det som att agenten har misslyckats med uppgiften. Uppgiften betraktas som lyckad om agenten når målet inom en bestämd tid. Tillförlitligheten mäts som en procentsats av antalet lyckade försök att utföra agentens uppgift (likt Thiemes (2002) sätt att mäta). Valet av arkitekturer grundas i att en redan bevisat tillförlitlig arki-

tektur för denna typ av tillämpningar (dvs. ESCN) jämförs med en självorganiserande arkitektur (dvs. SORN).

I tidigare undersökningar (Thieme, 2002; Ziemke, 1999) har undersökta arkitekturer haft en bestämd intern struktur (dvs. att kopplingar och neuroner förblir samma) och endast nätens vikter har genomgått evolution. Undersökningen är först och främst intressant då en av de bästa arkitekturerna från tidigare undersökningar, ESCN, jämförs med en arkitektur av mer självorganiserande typ, SORN. Självorganisering lyfts ofta fram som en önskvärd egenskap hos en robot, då det minskar risken för mänskliga designfel och felaktiga mänskliga antaganden (Bergfeldt & Linåker, 2002; Nolfi & Floreano, 2000). Ingen tidigare undersökning av dessa båda arkitekturer har gjorts[1], vilket medför en viss originalitet i arbetet. Resultaten från undersökningen är främst av värde för utveckling av framtida kontrollsystem för virtuella agenter och robotar.

3.2.1 Delsteg 1: Testmiljö

För att undersökningen skall kunna genomföras behövs en lämplig testmiljö där en robot kan simuleras. Testmiljön modellerar lämpligen en mängd delar av roboten (ett *base set*) tillräckligt noggrant för att resultaten skall ha någon relevans för robotiken som område. Testmiljön behöver också stödja önskade nätverksarkitekturer (ESCN och SORN) för att undersökningen skall kunna genomföras. Även stöd för dynamiska miljöer är ett krav (dvs. miljöer som förändras över tid).

3.2.2 Delsteg 2: Utvärdering

Tillförlitligheten hos arkitekturerna bestäms genom att göra en statistisk utvärdering i den testmiljö som tagits fram. Tillförlitligheten kan beskrivas som antalet lyckade försök av agenten att utföra sin uppgift av det totala antalet försök (dvs. ett genomsnittligt tillförlighetsvärde). De statistiska data som insamlas bör också utgöra en god grund för analys.

3.3 Metod

Här presenteras de metoder som används för att realisera delstegen som beskrivs ovan. Metod för delsteg 1 är implementering. Som Testmiljö används den befintliga robotsimulatorn YAKS. Metod för delsteg 2 är experiment, där experiment utförs i testmiljön, med avsikt att samla in data för utvärdering.

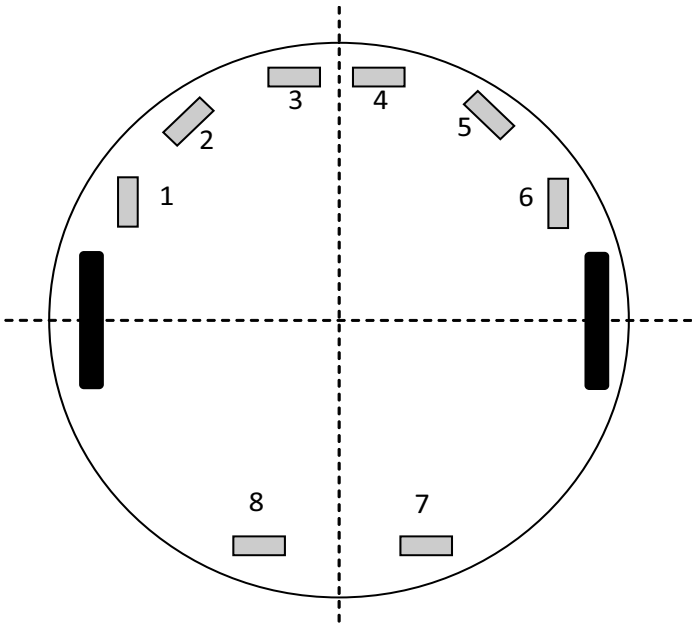
3.3.1 Metod för delsteg 1: Implementering

Testmiljön utgörs av den befintliga robotsimulatorn YAKS[2] (Yet Another Khepera Simulator). YAKS är en simulator för Khepera-roboten och stödjer både artificiella neurala nätverk och genetiska algoritmer och har med framgång använts i tidigare arbeten (t.ex. Búason, 2002; Búason & Ziemke, 2003; Ziemke(1999)). YAKS är förberedd för ESCN (Ahlén, 2003, sid. 20), vilket gör simulatorn mer fördelaktig. Simulatorn stödjer emellertid inte SORN-arkitekturen och därför

[1] Denna slutsats dras efter omfattande eftersökning.

[2] YAKS tillgänglig från `http://freshmeat.net/projects/yaks/`.

behöver modifikationer göras till simulatorn för att stödja denna arkitektur. Khepera-roboten är cirkulär med en diameter på cirka 55 millimeter och är utrustad med åtta distanssensorer (sex fram och två bak), två ljussensorer och två motorer, en för varje hjul (se figur 3.2). Till Khepera-roboten kan fler moduler kopplas, vilket ger ytterligare funktionalitet, som exempelvis marksensor och kamera (Ahlén, 2003; Búason & Ziemke, 2003; Floreano & Mattiussi, 2008 m.fl.).



Figur 3.2: Khepera-roboten med åtta sensorer (1-8) och två hjul (i svart). (Efter Jakobi, 1997, sid. 338).

3.3.2 Metod för delsteg 2: Experiment

Experiment har använts som en metod i flera tidigare undersökningar, varför det också är motiverat att använda i detta arbete. Här utvärderas de två arkitekturerna (ESCN och SORN) genom att köras i miljöer av ökande komplexitet. Komplexiteten (alt. svårighetsgraden) bestäms av antalet hinder och graden av dynamik i miljön. En miljö med två statiska hinder klassas som enklare än en miljö med fyra dynamiska hinder (som följaktligen rör på sig).

I experimentet används en arkitektur per agent. För varje miljö tränas en agent (robot) separat (dvs. en agent som drivs av en arkitektur tränas specifikt för en miljö). Under agenternas träning tillämpas en genetisk algoritm. Den bästa individen från den sista generationen antas representera hur väl arkitekturen låter agenten navigera i miljön. Den bästa individen för respektive arkitektur och miljö går vidare för utvärdering. Utvärdering sker genom att köra varje tränad agent i sin respektive miljö ett bestämt antal gånger. Under körningarna sker här även insamling av data för vidare analys.

4 Genomförande

Detta kapitel beskriver arbetets genomförandefas. Kapitlet inleds med sektion 4.1, innehållandes en beskrivning av de modifikationer som gjorts till simulatorn för att den skall stödja krävd funktionalitet. En detaljerad beskrivning av implementationen av SORN tas upp och en beskrivning av hur dynamiska miljöer har åstadkommits. Slutligen följer en beskrivning av experimentens upplägg i 4.2.

4.1 Genomförande av delsteg 1: Modifikation av simulator

Det finns i grund och botten två problem som måste lösas i simulatormiljön. Dessa är att (1) implementera stöd för önskade nätverksarkitekturer och (2) implementera stöd för dynamiska element i miljöer. YAKS stödjer, som tidigare uppmärksammats (se sektion 3.3.1), redan ESCN. SORN stöds emellertid inte, vilket leder till att arkitekturen behöver implementeras. Följande sektion beskriver implementeringen av SORN och vad som skiljer den mot originalarkitekturen som beskrivs av Jakobi (1997a). Efter detta följer även en beskrivning av hur miljöerna gjordes dynamiska.

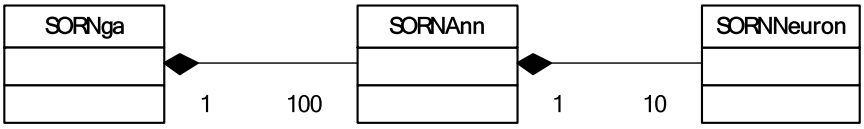
4.1.1 Implementering av Self-organized recurrent network

Till att börja med bör det nämnas att implementationen som beskrivs här skiljer sig i flera avseenden från originalarkitekturen (dvs. den arkitektur som beskrivs av Jakobi, 1997a). Således blir implementationen som beskrivs här en sorts variant på originalarkitekturen, då den fortfarande behåller flera av aspekterna som gör originalarkitekturen relativt unik. Denna sektion syftar främst till att lyfta fram skillnaderna och likheterna mellan originalarkitekturen och denna variant.

Originalarkitekturen använder en bitsträng som genotyp, vilken sedan översätts till ett helt nätverk (eller snarare ett halvt nätverk, vilket förklaras senare). Således är det enkelt att tillämpa både mutations- och överkorsningsoperatorer på en individ. De experiment som görs i denna undersökning använder dock inte någon överkorsningsoperator, varför endast mutation stöds av SORN-implementationen. Se kap 2.4.1 för en motivering till varför mutation är att föredra framför överkorsning.

Implementationen av SORN i detta arbete har emellertid en mer objektorienterad approach (klassdiagram visas i figur 4.1) och använder inte en bitsträng för att representera nätverket. I fokus står de tre klasserna SORNga, SORNAnn och SORNNeuron. SORNga håller en population av individer, vilka i själva verket utgörs av instanser av SORNAnn. Vidare håller SORNAnn tio instanser av SORNNeuron, motsvarande åtta inputneuroner och två outputneuroner. Mutation av nätverket sker genom anrop till metoder som gjorts synliga i gränssnittet hos SORNAnn.

Originalarkitekturen genomgår evolution på ett speciellt sätt, genom att endast evolvera halva nätet och sedan spegla det. Detta är möjligt då nätet kan delas på mitten, vilket bildar två exakta



Figur 4.1: UML klassdiagram som visar SORN-implementationen. Medlemmar och metoder har utelämnats.

spegelbilder av varandra (Jakobi, 1997a). Att evolvera nätet på detta sätt ger upphov till en viss symmetri (se figur 2.11). I implementationen som presenteras här evolveras dock hela nätet, vilket självfallet inte ger upphov till samma symmetri. Originalarkitekturen speglas för att simuleringen skall gå snabbare (Jakobi, 1996, 1997a), men nackdelen är att många potentiella lösningar utesluts då roboten får ett symmetriskt beteende. Eftersom detta arbete inte fokuserar på att göra snabba simuleringar var det inte heller lockande att utesluta potentiella lösningar av denna anledning.

Implementationen grundas i tanken att varje neuron har en egen prioritet (som är disjunkt från övriga neuroners prioritet), vilken avgör aktiveringsordningen i nätet. En neuron med lägre prioritet aktiveras före en neuron med högre prioritet. För enkelhetens skull mappas neuronens prioritetsvärde direkt mot dess index. Det vill säga att neuron 1 aktiveras före neuron 2 osv. De två outputneuronerna använder en sigmoid aktiveringsfunktion (formel 2.3) medan inputneuronerna använder en stegfunktion (formel 2.2). Tröskelvärdena hos neuronerna förändras, likt kopplingarna, under träningen.

Varje neuron kan ha maxialt tre inkommande kopplingar från andra neuroner i nätet, vilket stämmer överrens med originalarkitekturen. Grannmatrisen[1] har storleken 10×3, med 10 rader och 3 kolumner. Varje position består av ett par, innehållandes inkommande neuronindex och kopplingens vikt. Det bör tilläggas att det är fullt möjligt att en eller flera loopar[2] existerar i nätet. Varje inkommande neuronindex är i intervallet [-1, 9], där -1 tolkas som en icke-existerande koppling.

I originalarkitekturen består genomet av 140 gener som kan förändras med genetiska operatorer. Här är genantalet istället (motsvarande) 70 (inkommande kopplingar (30) + kopplingsvikter (30) + tröskelvärden (10)). Notera att även inkommande kopplingar från neuronindex -1 (dvs. icke-existerande kopplingar) samt outputneuroner ändå kan få sin vikt respektive sitt tröskelvärde muterat. Detta är sannolikt en brist, då en möjlig mutation av en gen i genotypen inte alltid kommer att återspeglas i den slutliga fenotypen (dvs. nätverket). Det nya värdet hos en gen efter en mutation slumpas uniformt i dess interval. Inkommande kopplingar slumpas uniformt i intervallet [-1, 9] och kopplingsvikter i intervallet [-2, 2]. Tröskelvärden slumpas uniformt i intervallet [-1, 1]. Intervallen för respektive gener följer originalarkitekturen (Jakobi, 1997a).

4.1.2 Dynamiska testmiljöer

Alla miljöerna i experimenten använder sig av väggar som hinder. Målet med undersökningen är inte att bara undersöka agentens förmåga att undvika hinder (dvs. collision avoidance), utan även förmågan att navigera genom miljön. Det är önskvärt att använda väggar som rör sig i en förutbestämd riktning istället för mindre, separata, hinder som rör sig i godtycklig riktning (se figur 2.5) när navigationsproblemet undersöks. Detta eftersom att i det senare fallet kan agenten

[1] Begreppet används informellt här då en grannmatris egentligen har storleken n × n (Rosen, 2007).
[2] En loop är i grafteori en kant som ansluter en nod till sig själv (Rosen, 2007).

utveckla ett beteende som går ut på att åka rakt mot målet, stanna vid ett hinder och vänta på att det försvinner ur vägen, för att sedan fortsätta mot målet. Om agenten istället befinner sig i en miljö med en tydlig struktur (vilket anses vara fallet när väggar används som hinder) måste den anstränga sig mer för att hitta en väg till målet (dvs. planera en väg). Även om väggarna rör sig anses miljöns struktur hållas mer konsekvent än om det istället hade varit hinder som rört sig i godtycklig riktning.

Då YAKS inte stödjer rörliga väggar behöver ett alternativ hittas. Det visar sig att YAKS stödjer rörliga, runda, hinder där hindrens radie, hastighet och riktning kan specificeras. Alla rörliga hinder har samma radie, men de kan ha en individuell hastighet och riktning. Genom att placera flera hinder i rad och ge dem dessa samma hastighet och riktning är det möjligt att skapa väggar.

4.2 Genomförande av delsteg 2: Experiment

Experimenten syftar i huvudsak till att samla in statistik över hur väl arkitekturerna låter agenten navigera i miljöerna. Resultat från experimenten presenteras och analyseras i kapitel 5.

Experimenten inbegriper utöver två arkitekturer även fyra miljöer som tagits fram för att undersöka arkitekturernas tillförlitlighet i både statiska och dynamiska miljöer. Vidare undersöks även två olika fitnessberäkningar. Experimenten är således 16 till antalet och benämns SORN-[1,2]-[1,4] samt ESCN-[1,2]-[1,4] där exempelvis ESCN-1-4 betecknar experimentet där ESCN tränas och körs med fitnessfunktion 1 i miljö 4.

Två fitnessfunktioner (dvs. fitnessberäkningar) används i experimenten. Den första, f_1, specificerar väldigt lite om hur agenten bör agera för att nå målet. Med f_1 belönas agenten om den når målet och straffas om den kolliderar. Vidare uppmuntras agenten att nå målet på så kort tid som möjligt. När agenten når målet tilldelas den därför en bonus som grundas i hur många tidssteg det är kvar av epoken. Om agenten inte lyckas nå målet medan den inte heller krockar ges den 0 fitness i slutet av epoken.

$$f_1(i) = \sum_{epok=0}^{antalEpoker} \begin{cases} 2000 + (NrOfSteps - step_i)\, vid\, nått\, mål \\ -2000\, vid\, kollision \\ annars\, 0 \end{cases} \tag{4.1}$$

Den andra funktionen, f_2, specificerar emellertid mer information än f_1. Den liknar f_1 då den belönar agenten vid nått mål och straffar agenten vid kollision. Där f_1 ger varken belöning eller straff för att inte nå målet utan kollision ger istället f_2 ett straff som beror på hur långt bort från målet agenten är vid slutet av epoken. Att befinna sig väldigt nära målet vid avslutad epok skulle därför generera ett väldigt litet straff. Förhoppningen med detta är att agenten skall utveckla ett beteende som leder denna i rätt riktning

$$f_2(i) = \sum_{epok=0}^{antalEpoker} \begin{cases} 2000 + (NrOfSteps - step_i)\, vid\, nått\, mål \\ -2000\, vid\, kollision \\ annars\, -(euklidiska\, avståndet\, från\, i\, till\, mål) \end{cases} \tag{4.2}$$

För både f_1 och f_2 gäller följande; i är en individ i populationen och stepi är tidssteget när i når målet. Vidare gäller att $0 \leq step \leq NrOfSteps$ där $NrOfSteps$ är 400, då varje epok varar 400 tidssteg. Notera även att vid nått mål eller vid kollision avbryts den aktuella epoken och nästa epok påbörjas.

Experimenten inleds med att träna de 16 agenterna (agenten från varje experiment) i miljöerna. Träningen sker med en genetisk algoritm, där en population av 100 individer tränas under 1000 generationer. Varje individ körs i miljön under 20 epoker (dvs. varje individ har 20 försök per generation att ta sig till målet). Samtliga epoker under en generation spelar in på det slutliga fitness-värdet för en individ. Varje epok varar 400 tidssteg, där ett tidssteg motsvarar 100 millisekunder. En epok motsvarar således 40 sekunders körning. Parametrarna har lånats från flera tidigare experiment med Khepera-roboten. Exempelvis använder både Thieme (2002) och Jakobi (1997a) en populationsstorlek av 100 individer som evolveras över 1000 generationer. Vidare kör Ziemke (1999) simulationer som varar i 400 tidssteg.

En elitism-mekanism sparar de tio bästa individerna oförändrade till nästkommande generation. Resterande 90 platser i populationen fylls sedan genom reproduktion (betydande att den genetiska algoritmen inte är s.k. steady-state (Davis, 1991), den använder inte överlappande populationer). Reproduktion sker asexuellt (dvs. endast genom mutation) med en sannolikhet på 5 %. Vidare tillämpas turneringsselektion för att välja ut individer för reproduktion. Sannolikheten för att den starkare av två slumpvis utvalda individer skall gå vidare för reproduktion är 80 %.

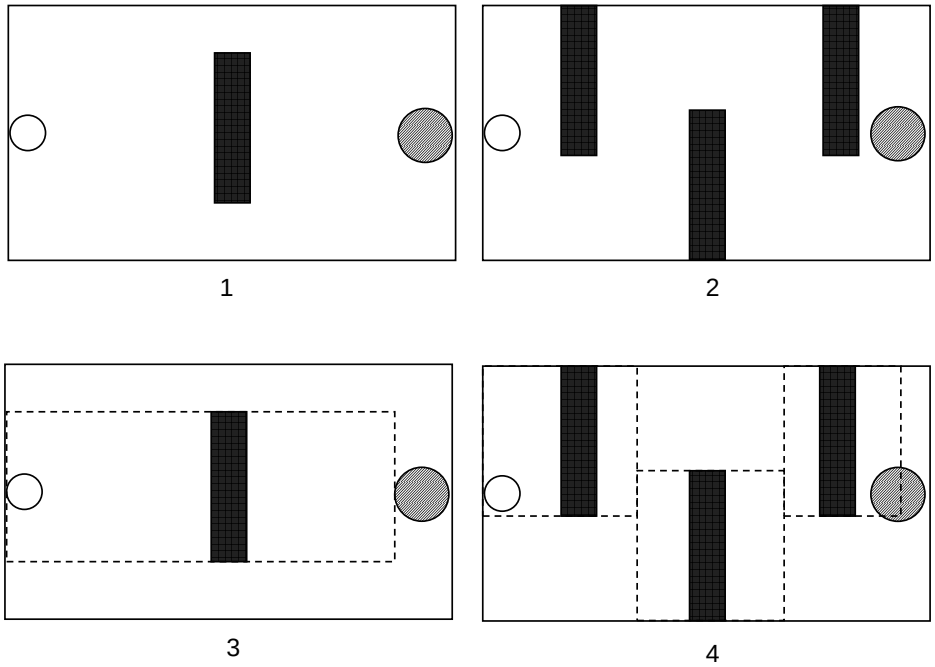
I början av varje körning i miljön slumpas agentens startriktning mellan [-90°, 90°] där 0 grader avser en riktning rakt fram (åt höger, i riktning mot målet i samtliga miljöer). I miljö 3 och 4 (se följande sektion) slumpas även hindrens startriktning (0° eller 180°) och dess hastighet mellan [1, 8]. Dessa aspekter bedöms som implementationsaspekter och slumpas för varje ny epok. Detta för att undvika att agenten blir beroende av en viss förutbestämd detalj som inte nödvändigtvis finns i den riktiga världen.

Khepera-roboten använder i experimenten bara distans-sensorer, där sex sensorer är placerade i robotens front och två är placerade baktill. I inställningarna till simulatorn specificeras det att sensorstörningar (*eng. noise*) på upp till 5 %. Detta för att ta hänsyn till inexakta sensorer och störningar som kan finnas hos en riktig robot. Som Jakobi et al (1995) beskriver bör störningsnivån först observeras hos fysiska robotar. Men denna möjlighet fanns inte, varför ett (i det närmaste godtyckligt) värde på 5 % valdes.

Efter att varje agent har tränats i sin respektive miljö utvärderas dem genom att låta dem försöka navigera genom sin miljö under 400 tidssteg under 1000 repetitioner. Sättet att mäta tillförlitlighet är inspirerat av Thiemes (2002) sätt att mäta. Tillförlitligheten är helt enkelt en procentsats som beskriver antalet lyckade försök av antalet totala försök.

Slutligen kan det nämnas att experimenten uppskattningsvis tog 24 timmar[3] att köra i sekvens, där agenternas träning tog den mesta av tiden. Tiden för att köra själva utvärderingarna är näst intill försumbar i sammanhanget. Träningen av ESCN tog upp tre fjärdedelar av tiden, vilket tyder på att ESCN är beräkningsmässigt avsevärt dyrare att träna jämfört med SORN.

[3] På en persondator med processortypen Intel Core 2 Duo E4400 2.00 Ghz.



Figur 4.2: De fyra miljöerna. Miljö 1 och 2 är statiska. Miljö 3 och 4 är dynamiska, vilket indikeras av de streckade områdena som symboliserar området inom vilka väggarna kan röra sig. Vit cirkel symboliserar startposition och streckad cirkel symboliserar mål.

4.2.1 Miljöer

Miljöerna (figur 4.2) är fyra till antalet och syftet med dess utformning är att de skall vara av ökande komplexitet. De två första är statiska och anses som mindre komplexa. De resterande två miljöerna är dynamiska och anses därför som mer komplexa (och således svårare). Anledningen till att även statiska miljöer undersöks är för att undersökningen inte skall bli helt svartvit. Även om agenten inte lyckas navigera genom alla dynamiska miljöer kan det ändå vara av intresse att få en uppfattning om hur den löser enklare miljöer. Miljö 3 och 4 är dynamiska varianter av miljö 1 respektive miljö 2.

Samtliga miljöer är 700 mm breda och 400 mm höga. Likt tidigare experiment (Jakobi, 1997a m.fl.) är startpositionen förbestämd, men robotens riktning slumpas (uniformt) i intervallet [-90°, 90°] (se även föregående sektion). Målet är en cirkel vars diameter är dubbelt så stor som robotens (dvs. 110 mm) och ligger i linje med startpositionen. Väggarnas startriktning slumpas i miljö 3 och 4 och de färdas alltid parallellt med de korta sidorväggarna (dvs. antingen i rikting 0° eller 180°). Dess hastighet ligger i intervallet [1, 8], där 1 motsvarar 1 mm/tidssteg (alt. 1 cm/sek).

5 Resultat

Detta kapitel beskriver och sammanfattar resultaten från experimenten. I sektion 5.1 presenteras en översikt av resultaten och i 5.2 och 5.3 följer djupare beskrivningar av hur de båda arkitekturerna löser de fyra miljöerna och några av de mer intressanta beteendena lyfts fram. Samtliga rörelsebanor och aktiveringsvärden presenteras i bilaga A.

5.1 Översikt

Resultaten av undersökningen visar att ESCN når avsevärt högre tillförlitlighet än SORN i de allra flesta fall (se tabell 5.1 och tabell 5.2). Detta är i linje med resultaten som tidigare har presenterats av Thieme (2002) och Ziemke (1999), där ESCN varit den mest tillförlitliga arkitekturen. Ingen kontrollarkitektur i experimenten lyckas emellertid lösa miljö 4, vilken anses som den svåraste av miljöerna i undersökningen. Detta är också anledningen till att kontrollarkitekturernas tillförlitligheter hålls tämligen låga.

Det kan observeras att ESCN är väldigt konsekvent i sitt beteende. Antingen klarar ESCN miljön med väldigt bra resultat (över 92 % tillförlitlighet i alla miljöer, undantaget miljö 4) eller så misslyckas den helt. Detta gäller för både f_1 (formel 4.1) och f_2 (formel 4.2), där ESCN når genomsnittlig tillförlitlighet på 72,3 % respektive 74,5 %. Att tillförlitligheten är något högre med f_2 kommer knappast som en överraskning, då f_2 uppmuntrar att agenten närmar sig målet. Det är snarare överraskande att det skiljer relativt lite (2,2 %) mellan f_1 och f_2.

SORN påverkas betydligt mer av fitnessfunktionens utformning än vad ESCN gör. Med f_1 når SORN en genomsnittlig tillförlitlighet på 48,9 %, vilket är långt ifrån värdet som ESCN har. Vidare lyckas inte SORN som tränats med f_1 lösa miljö 2 (experiment SORN-1-2). SORN når bättre tillförlitlighet när träningen skett med f_2 och visar på 56,6 %. Detta är en skillnad på 7,7 % mellan f_1 och f_2. De experiment där SORN tränats med f_2 är de enda som visar på tillförlitlighet som är något annat än svart eller vit. I experiment SORN-2-2 når agenten målet i ungefär hälften av fallen och i SORN-2-1 (den enklaste miljön) når agenten (uppseendeväckande nog) bara målet i ungefär åtta av tio fall.

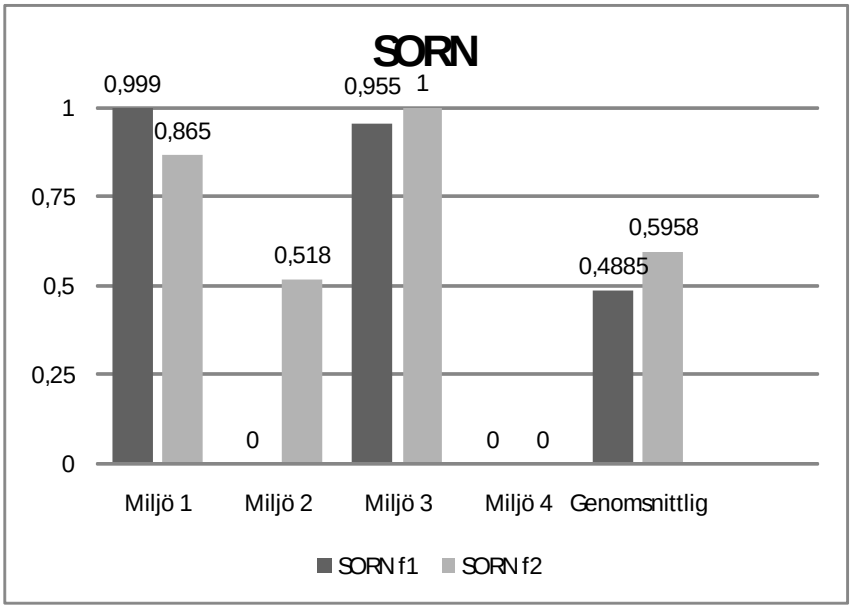
Tabell 5.1: Genomsnittlig tillförlitlighet för arkitekturerna som tränats med f_1.

Arkitektur	**Genomsnittlig tillförlitlighet**
Self-organized recurrent network (SORN)	48,9 %
Extended sequential cascaded network (ESCN)	72,6 %

Tabell 5.2: Genomsnittlig tillförlitlighet för arkitekturerna som tränats med f_2.

Arkitektur	**Genomsnittlig tillförlitlighet**
Self-organized recurrent network (SORN)	59,6 %
Extended sequential cascaded network (ESCN)	74,5 %

5.2 Self-organized recurrent network (SORN)

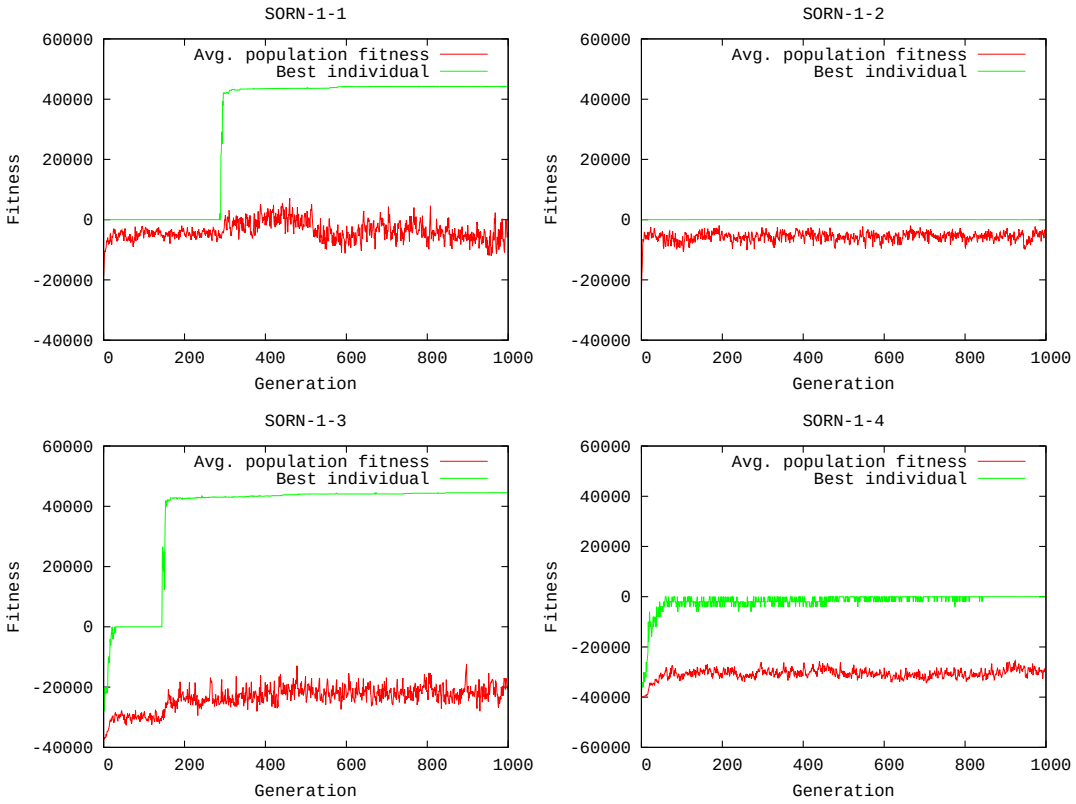


Figur 5.1: Tillförlitlighet för SORN i de olika miljöerna och för de två fitnessfunktionerna.

SORN visar på några intressanta skillnader mellan de två fitnessfunktionerna. Att miljö 1 exempelvis är svårare att lösa när nätverket tränats med f_2 (12,5 % lägre tillförlitlighet) är oväntat eftersom miljön anses som den enklaste. Samtidigt når den 100 % tillförlitlighet i miljö 3, vilken är den dynamiska motsvarigheten till miljö 1. Figur 5.1 visar tillförlitligheten för SORN i de olika miljöerna.

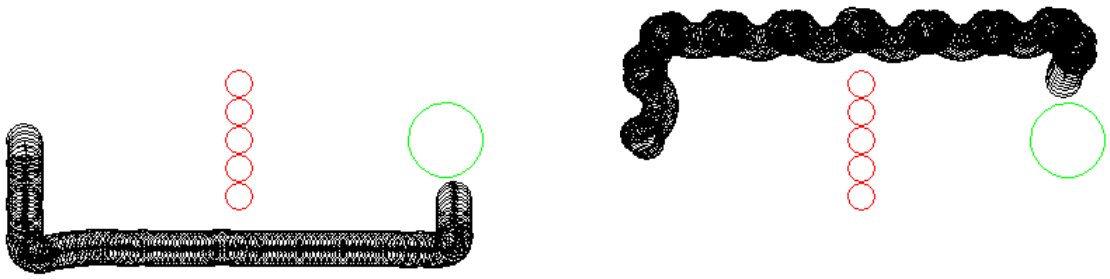
Beträffande träningen av SORN gäller det generellt att agenten först lär sig undvika hinder, vilket indikeras av att fitnessvärde 0 nås (se figur 5.2 och 5.3). Efter en viss tidsperiod utvecklar sedan agenten ännu ett beteende som låter agenten nå målet (vilket indikeras av fitness > 0). I alla experiment utom SORN-2-4 lär sig agenten ett kollisionsundvikande beteende, men långt ifrån alla agenter utvecklar ett beteende där målet faktiskt nås. I de fall när agenten lär sig att hitta målet är övergången från att undvika hinder till att även nå målet väldigt snabb (indikeras av en väldigt brant övergång i kurvan, t.ex. figur 5.2-1 som visar träningen i SORN-1-1).

Agenterna från SORN-1-1 och SORN-2-1 navigerar genom miljö 1 på helt olika sätt (se figur 5.4). SORN-1-1 använder sig av en mer direkt approach där väggen följs och hindret i mitten undviks helt. SORN-2-1 använder en helt annorlunda approach som i det närmaste kan liknas vid en studsande rörelse mot miljöns övre vägg. Figur 5.5 visar aktiveringsvärdena för det studsande be-



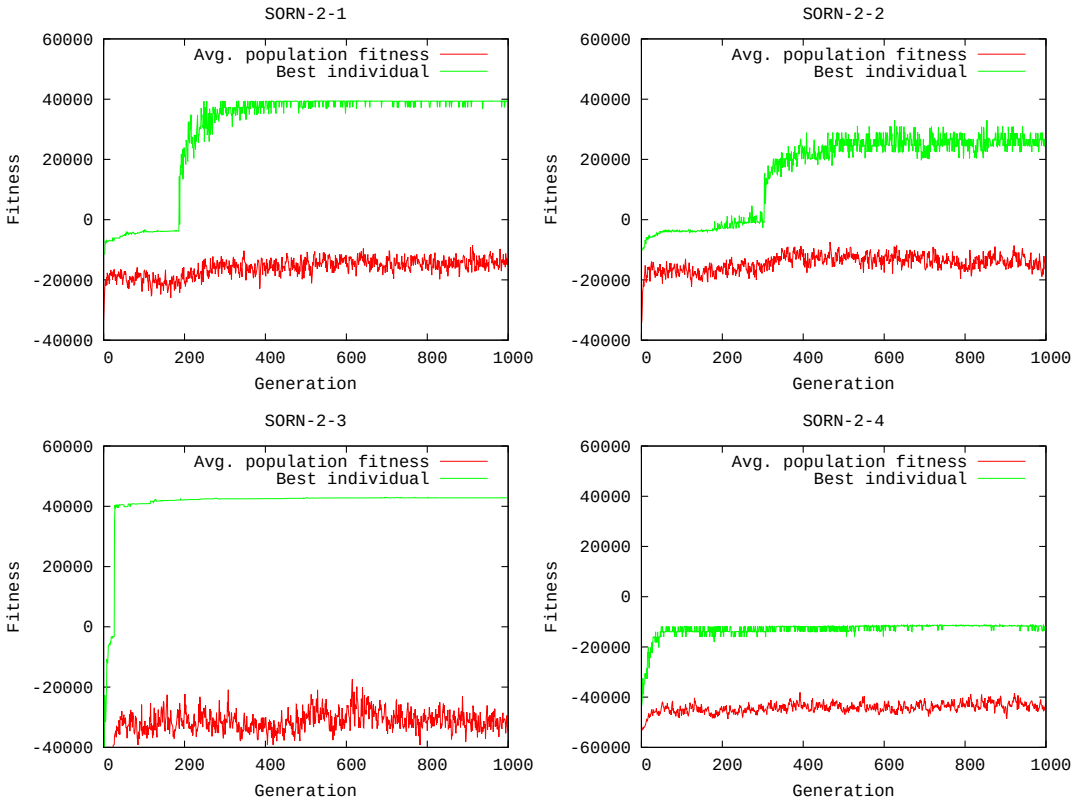
Figur 5.2: Fitnesskruvor för träningen av SORN enligt f_1.

teendet i SORN-2-1 (notera att motorvärde på 0,5 betyder att motorn inte rör sig). Båda agenterna följer emellertid väggen. Detta beteende är inte oväntat, eftersom agenterna, bildligt uttryckt, är blinda när de åker för långt från hinder och väggar.

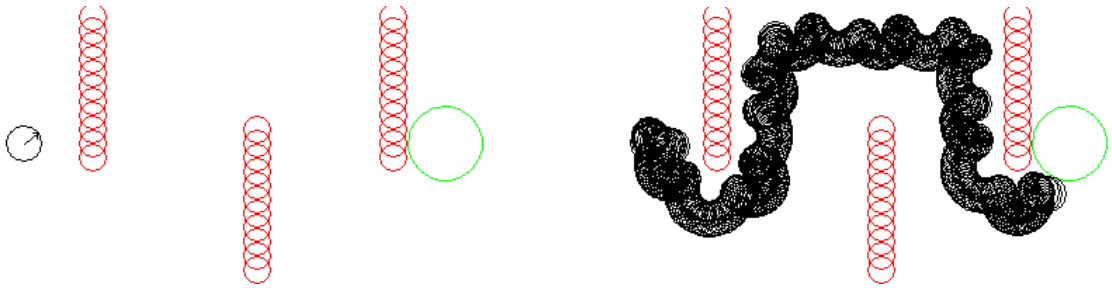


Figur 5.4: Agent från experiment SORN-1-1 (vänster) och SORN-2-1 (höger).

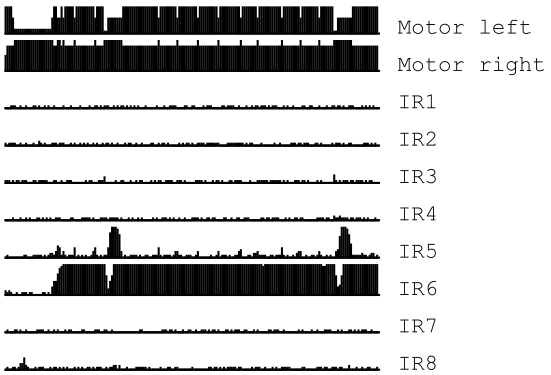
Agenten från SORN-2-2 (till höger i figur 5.6) som navigerar genom miljö 2 använder också ett tillvägagångssätt som är mycket likt det som används av agenten i SORN-2-1. Figur 5.7 visar aktiveringsvärden för SORN-2-2. Agenten från experiment SORN-1-2 lyckas inte nå målet utan har utvecklat ett beteende där den undviker hindren genom att stå helt stilla. Den stannar vid startpositionen utan att röra sig överhuvudtaget (till vänster i figur 5.6).



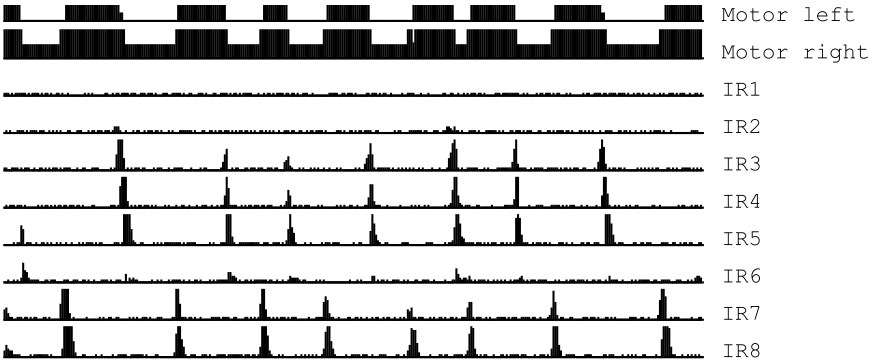
Figur 5.3: Fitnesskurvor för träningen av SORN enligt f_2.



Figur 5.6: Agent från experiment SORN-1-2 (vänster) och SORN-2-2 (höger).

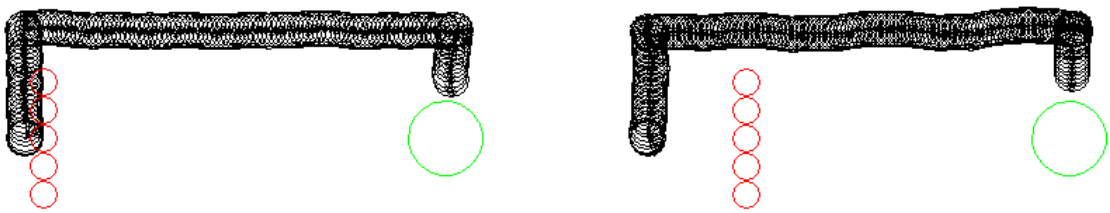


Figur 5.5: Aktiveringsvärden för SORN-2-1.

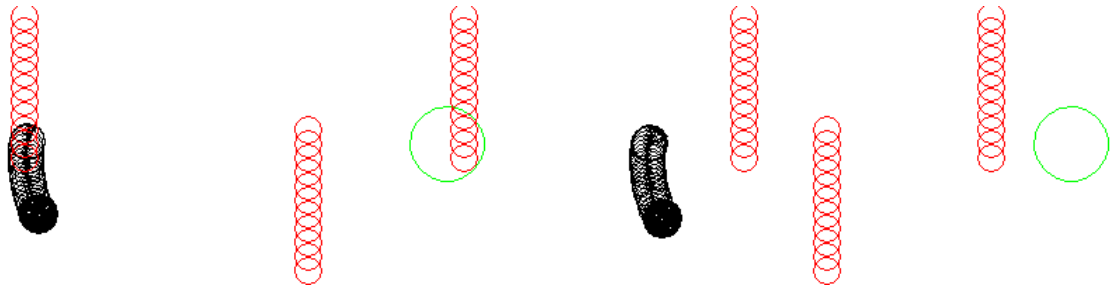


Figur 5.7: Aktiveringsvärden för SORN-2-2.

Agenterna i experiment SORN-1-3 och SORN-2-3 lyckas båda nå målen med optimal eller näst intill optimal tillförlitlighet. Intressant i de båda experimenten är att agenternas beteenden är snarlika. Figur 5.8 visar rörelsebanor över hur agenterna navigerar genom miljön. Agenterna från de experiment som äger rum i miljö 4 (SORN-1-4 och SORN-2-4) lyckas dock inte nå målet. De undviker de rörliga hindren genom att åka till ett område i det nedre vänstra hörnet av miljön där inget rörligt hinder kan nå dem. De stannar sedan på denna plats utan att finna en väg genom miljön. Detta illustreras i figur 5.9.

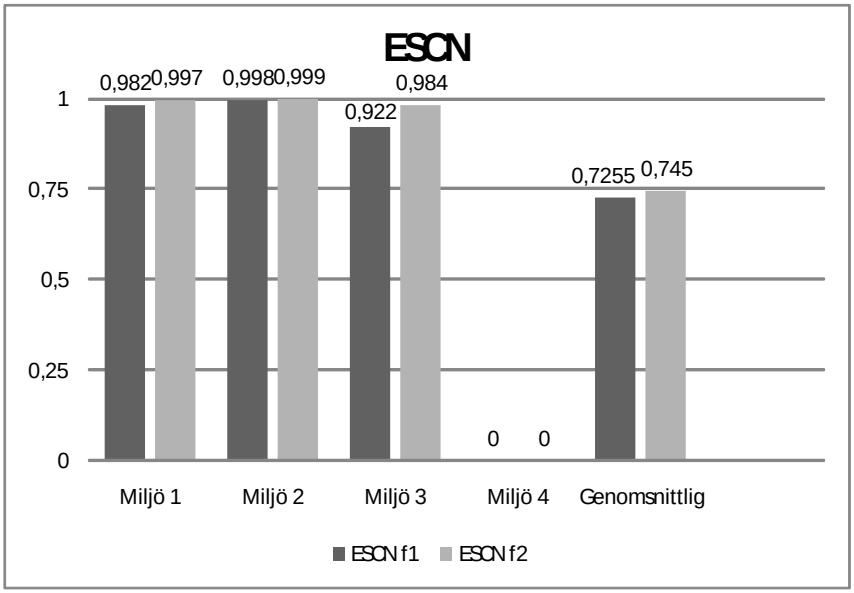


Figur 5.8: Agent från experiment SORN-1-3 (vänster) och SORN-2-3 (höger).



Figur 5.9: Agent från experiment SORN-1-4 (vänster) och SORN-2-4 (höger).

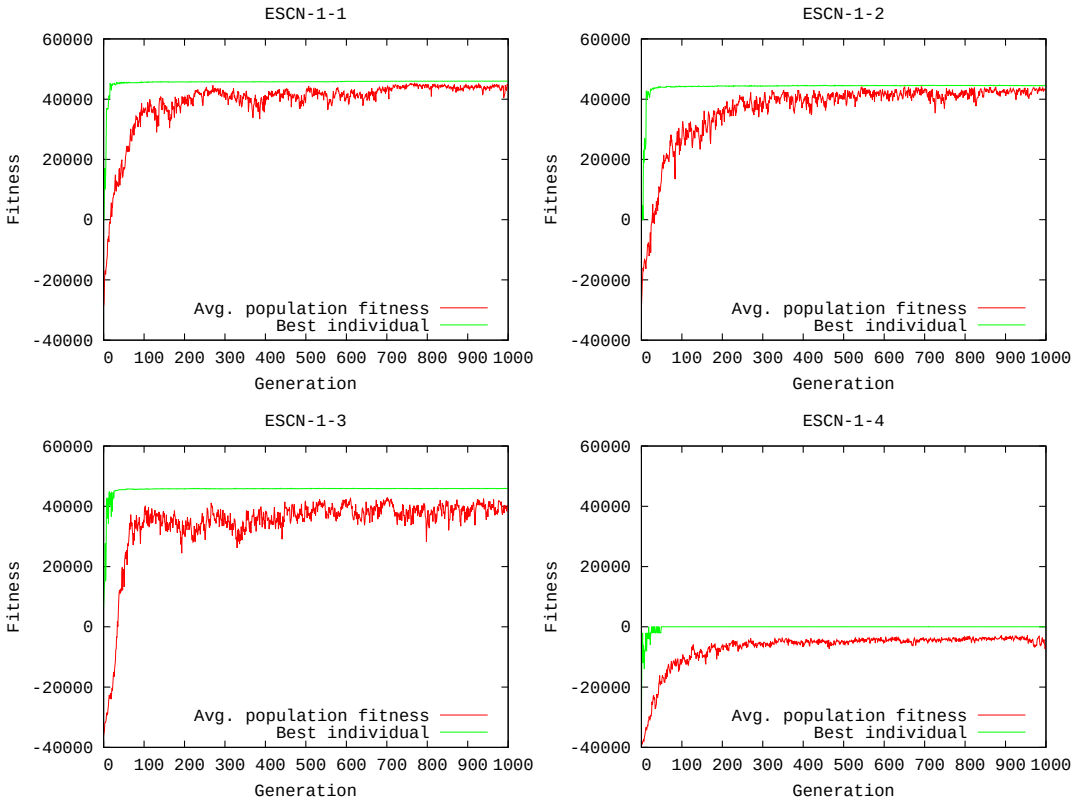
5.3 Extended sequential cascaded network (ESCN)



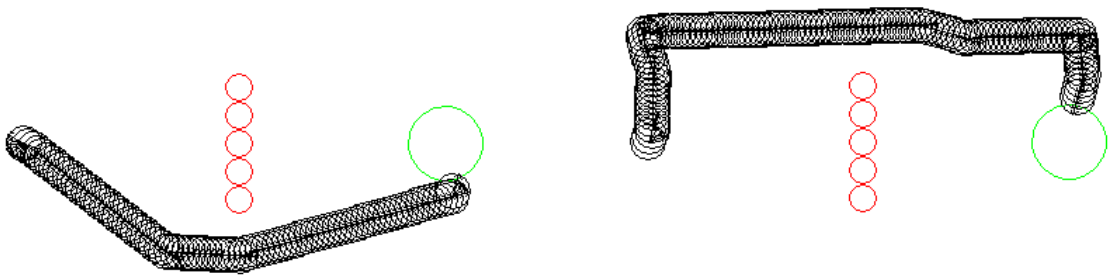
Figur 5.10: Tillförlitlighet för ESCN i de olika miljöerna och för de två fitnessfunktionerna.

Vad som kanske är mest uppseendeväckande är att ESCN når målet i 99,8 % respektive 99,9 % av alla försök i miljö 2, men inte lyckas nå målet en enda gång i miljö 4, trots att miljöerna har samma grundstruktur. Figur 5.10 visar tillförlitligheten hos ESCN i de olika experimenten. Tillförlitligheten hos ESCN är emellertid genomgående väldigt hög i samtliga lyckade experiment. Vad som vidare kan observeras är att arkitekturen endast når något högre tillförlitlighet med träning enligt f_2 än vad den gör när den tränats enligt f_1. Detta trots att f_1 inte uppmanar agenten att röra sig i riktning mot målet.

Under träning nådde ESCN väldigt snabbt god fitness i de miljöer som lyckats lösas, vilket kan observeras i figur 5.11 och figur 5.12. I experiment ESCN-1-4 lär sig agenten tidigt att undvika de rörliga väggarna, medan i ESCN-2-4 lyckas agenten inte lära sig ett hinderundvikande beteende, vilket skulle indikeras av fitnessvärde 0. Istället ligger fitnessvärdet under 0, vilket betyder att agenten ibland kolliderar.

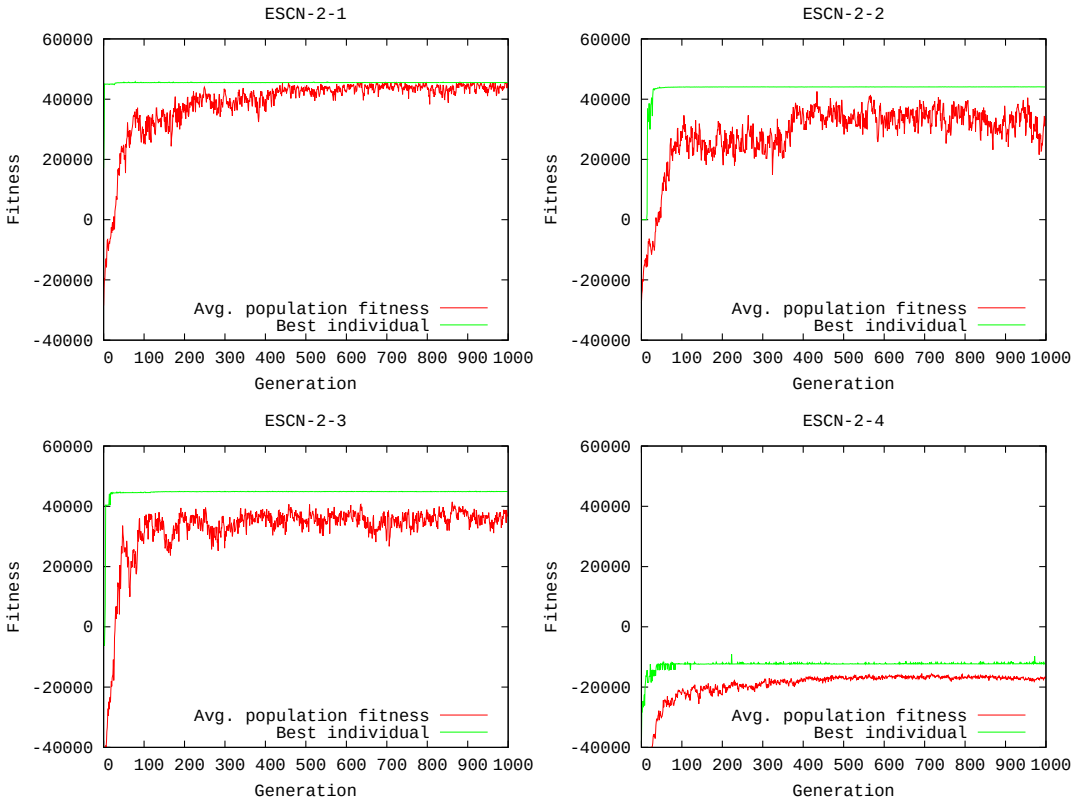


Figur 5.11: Fitnesskurvor för träningen av ESCN enligt f_1.

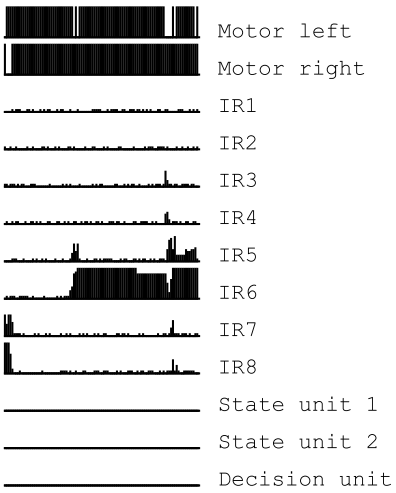


Figur 5.13: Agent från experiment ESCN-1-1 (vänster) och ESCN-2-1 (höger).

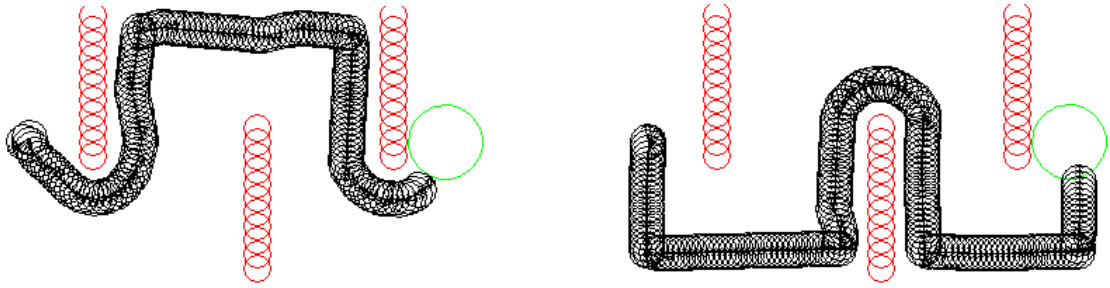
Agenterna från ESCN-experimenten visar på olika beteenden beroende på vilken fitnessfunktion som de har tränats med. Figur 5.13 visar rörelsebanor i miljö 1 för agenterna i experiment ESCN-1-1 och ESCN-2-1. Den första agenten svänger nedåt, för att undvika väggen. Efter att väggen passerats svänger den sedan i riktning mot målet. Detta innebär att agenten tappar kontakt med väggen, vilket gör agenten ganska unik bland undersökningens experiment. Figur 5.14 visar agentens aktiveringsvärden från körningen. Tidigt i epoken kan det observeras att agenten inte får någon input alls från dess sensorer. Agenten från ESCN-1-1 tar därför en kortare väg till målet än agenten från ESCN-2-1. Den andra agenten, vilken tränats enligt f_2, följer istället den övre väggen på ett sätt som liknar agenten från SORN-1-1.



Figur 5.12: Fitnesskurvor för träningen av ESCN enligt f_2.

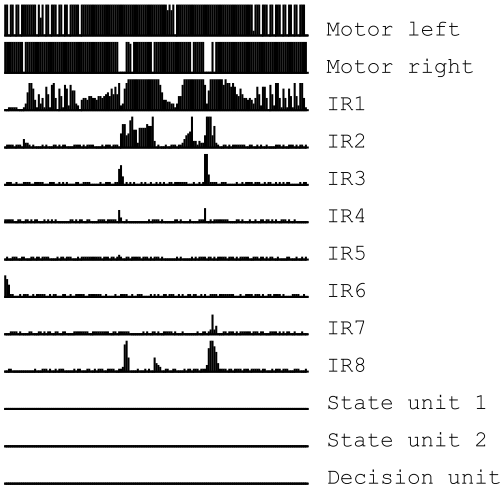


Figur 5.14: Aktiveringsvärden för ESCN-1-1.

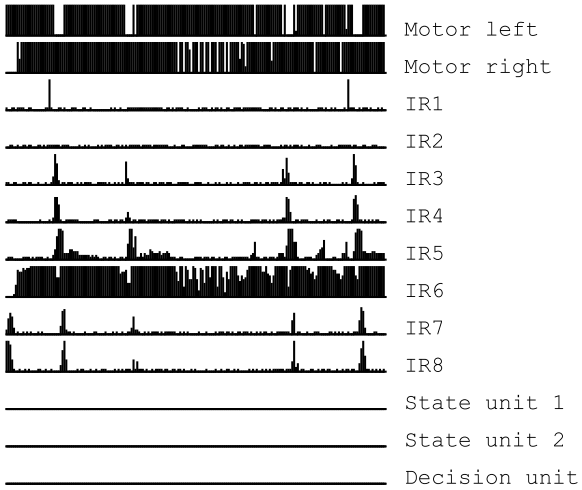


Figur 5.15: Agent från experiment ESCN-1-2 (vänster) och ESCN-2-2 (höger).

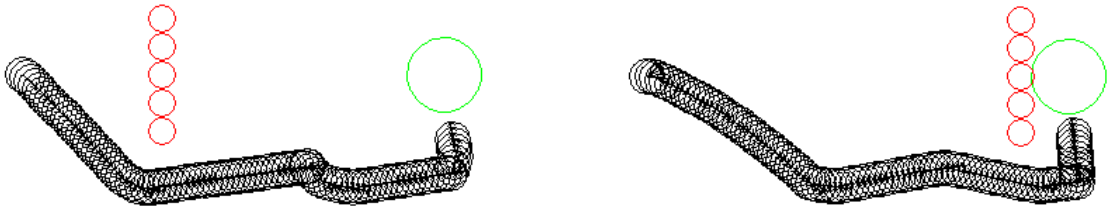
Rörelsebanor från experiment ESCN-1-2 och ESCN-2-2 visas i figur 5.15. Agenten som tränats med f_1 väljer att svänga runt det första hindret för att sedan färdas längs norra väggen fram till målet. Agenten som tränats med f_2 väljer emellertid att undvika det första hindret för att sedan färdas längs nedre väggen och därför endast stöta på ett hinder. Figur 5.16 och 5.17 visar aktiveringsvärden för respektive experiment. Det bör tilläggas att ett hjul på roboten står still när kraften är 0.5 och 0.0 motsvarar därför full fart bakåt. Båda agenterna färdas i maxfart ändå tills de skall svänga, då en av motorerna sätts på full back.



Figur 5.16: Aktiveringsvärden för ESCN-1-2.

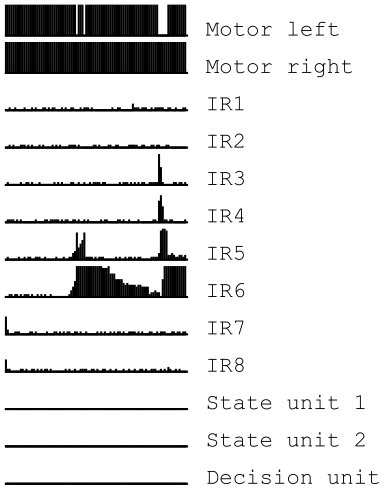


Figur 5.17: Aktiveringsvärden för ESCN-2-2.



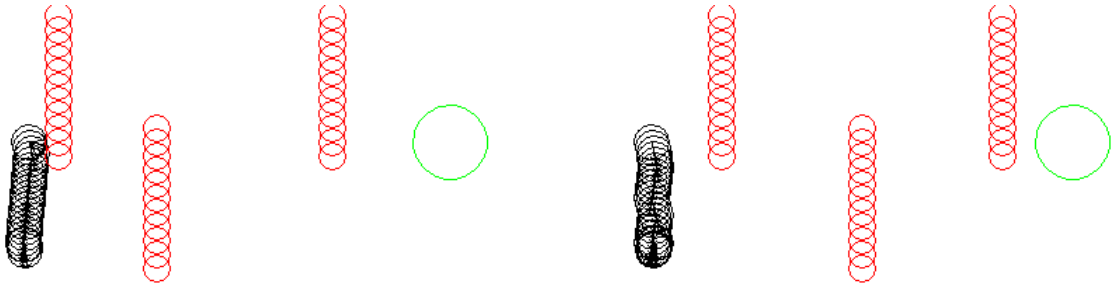
Figur 5.18: Agent från experiment ESCN-1-3 (vänster) och ESCN-2-3 (höger).

Även om ESCN visat på varierande beteenden i tidigare miljöer är dess försök att lösa miljö 3 snarlika. Figur 5.18 illustrerar detta och figur 5.19 visar aktiveringsvärdena för ESCN-1-3. När agenten når högre hörnet svänger den vänster, vilket indikeras av att *Motor left* backar samtidigt som *Motor right* fortsätter framåt.



Figur 5.19: Aktiveringsvärden för ESCN-1-3.

ESCN lyckades, likt SORN, inte lösa miljö 4, vilket resulterar i ett beteende där agenten söker sig till miljöns nedre vänstra hörn för att på så vis undvika de rörliga väggarna.



Figur 5.20: Agent från experiment ESCN-1-4 (vänster) och ESCN-2-4 (höger).

5.4 Analys

Resultaten visar att ESCN är den bättre av de två arkitekturerna när navigeringsproblemet skall angripas. Inte bara når ESCN högre tillförlitlighet men arkitekturen visar också att den är mindre beroende av vilken typ av fitnessfunktion som används. SORN är mer ojämn i sina beteenden och valet av fitnessfunktion spelar större roll.

I fitnessfunktion f_2 används det euklidiska avståndet från robot till mål för att uppmuntra roboten att röra sig närmare målet. Detta skulle dock kunna leda till att roboten fastnar på vissa platser i miljöerna. Roboten skulle kunna fastna på en plats som är nära till målet via fågelvägen men där en vägg hindrar roboten från att ta sig ända fram till målet. Platsen blir som ett lokalt optima som kan vara svårt för roboten att ta sig ur. Fördelen är dock att det euklidiska avståndet från robot till mål är enkelt och billigt att beräkna (speciellt i en simulatormiljö).

De två fitnessfunktionerna ger i experimenten en något oklar indikation av vilken typ som lämpar sig bäst för navigeringsproblemet. För ESCN ger f_2 endast marginellt bättre resultat än f_1. Med

detta i åtanke kan f_1 vara att föredra, eftersom den specificerar mindre om agentens önskade beteende och låter därmed agenten själv bestämma hur den skall angripa uppgiften. Det är emellertid möjligt att designa en fitnessfunktion som tar in i beräkning ännu fler parametrar än f_2. Det bör därför noteras att f_2 inte är väldigt specificerande rörande agentens beteende som först kanske antytts. Att designa en ännu mer specificerande fitnessfunktion skulle dock vara intressant för att undersöka agenternas beteenden, men som Risi et al (2010) menar gör detta målet mer ambitiöst, vilket medför en större sannolikhet att fastna i ett lokalt optima.

För SORN varierar resultaten kraftigt beroende på vilken fitnessfunktion som används i vilken miljö. Detta ger upphov till några uppseendeväckande (och potentiellt förvirrande) iakttagelser. Att agenten från SORN-2-1 bara når målet i drygt åtta av tio fall är överraskande dåligt. Detta experiment, vilket har minimalt med hinder, skulle först kunna misstänkas ge väldigt god tillförlitlighet. Men resultaten vittnar om motsatsen.

SORN-1-1 ger dock betydligt bättre (nära optimal) tillförlitlighet trots att dess fitnessfunktion inte direkt uppmuntrar rörelse mot målet. När detta jämförs med experimenten i miljö 2 där SORN-1-2 ger 0 % tillförlitlighet och SORN-2-2 ger 51,8 % tillförlitlighet blir det hela något förvirrande. Vidare ger också f_2 optimal tillförlitlighet i miljö 3, vilken är den dynamiska varianten på miljö 1. För SORN tyder detta på att det är bättre att inte uppmuntra till specifika beteenden i enkla miljöer, medan det underlättar för agenten att bli knuffad i rätt riktning i de svårare.

Värt att uppmärksamma är det ”studsande” beteende som utvecklades hos agenterna i både SORN-2-1 och SORN-2-2. Trots att agenterna tränats enligt f_2 tar de en minst sagt suboptimal väg till målet. Det har observerats att tillförlitligheten hos SORN-2-1 är lägre än hos SORN-1-1 på grund av att den studsande rörelsen leder till att agenten helt enkelt inte hinner fram innan tiden för en körning tar slut.

Styrkan hos ESCN ligger i att nätverket kan använda feedback av andra ordningen för att förändra vikterna (den sensomotoriska mappningen) under körning. Arkitekturen väljer emellertid aldrig att använda denna funktion (se bilaga A för samtliga aktiveringsvärden). Detta indikeras av att beslutsneuronen aldrig aktiveras och en ommappning av vikterna tillåts därför aldrig. Detta medför emellertid att ESCN-arkitekturen är helt reaktiv, men presterar trots detta bättre än den rekurrenta arkitekturen SORN. I tidigare experiment med ESCN (Ahlén, 2003; Thieme, 2002; Ziemke, 1999, 2000) har en delmängd av tillståndsneuronerna kontinuerligt givit output trots att beslutsneuronen varit inaktiv. Endast när tillräckligt många tillståndsneuroner aktiverats har också beslutsneuronen aktiverats. Eftersom tillståndsneuronerna i experimenten här alla har varit inaktiva indikeras det att ett helt reaktivt beteende har tränats fram.

Navigeringsproblemet är tveklöst ett svårt problem och det försvåras ytterligare i dynamiska miljöer, vilket indikeras av att ingen arkitektur löser miljö 4. De övriga tre miljöerna löses emellertid väldigt tillförlitligt av ESCN, men inte lika tillförlitligt av SORN. Resultaten visar på att en avancerad arkitektur inte nödvändigtvis alltid är att föredra vid navigering då ESCN antar formen av ett enkelt feedforward-nätverk.

6 Slutsats och diskussion

Detta arbete har undersökt tillförlitligheten hos de två artificiella neurala nätverksarkitekturerna Extended sequential cascaded network (ESCN) och Self-organized recurrent network (SORN). De har använts för navigerande agenter (i form av en simulerad Khepera-robot) i fyra olika miljöer. Den självorganiserande arkitekturen SORN har implementerats som en modifikation till simulatorn YAKS. Arbetet har också undersökt två olika fitnessberäkningar för den genetiska algoritm som använts för att träna agenterna.

Den kanske främsta slutsatsen som kan dras kring arbetets resultat är att svåra problem inte alltid kräver avancerade arkitekturer. Det har visats att ESCN utvecklat ett helt reaktivt beteende och presterar trots det bättre än det rekurrenta nätverket SORN. Det är inte en omöjlighet att samma gäller för flera andra problem som ofta löses med artificiella neurala nätverk. Enkla nätverk har fördelen att färre parametrar sätts under den genetiska algoritmens kontroll, vilket medför att sökrymden minskar. Om ett nätverk har kapacitet att lösa ett visst problem hittas en bra nätverkskonfiguration sannolikt snabbare om sökrymden är mindre. Det är dock inte alltid säkert att en enkel arkitektur har denna kapacitet, vilket är något som måste övervägas vid val av nätverksarkitektur.

På grund av de reaktiva arkitekturernas enkelhet finns det anledning att misstänka att antalet beteenden som ett reaktivt nätverk kan lära sig är mycket begränsat. Denna misstanke bekräftas av bl.a. Graham et al (2005) som inte lyckas träna ett feedforward-nätverk till ett hinderundvikande beteende och ett målföljande beteende samtidigt. Om den reaktiva arkitekturens skalbarhet är dålig i jämförelse med mer avancerade arkitektur finns det istället anledning att överväga användning av en mer avancerad arkitektur.

Exempelvis lyckas ingen arkitektur i dessa experiment lösa miljö 4. Möjligtvis skulle det vara mer lyckat att använda en inkrementell approach till träning. Roboten utsätts i det första steget förslagsvis för ett scenario i vilket den behöver *följa en rörlig vägg* utan att krocka med den. På så vis lär sig roboten att först hantera rörliga väggar. I nästa steg tränas roboten för att *nå målet* utan att krocka med den rörliga väggarna. Men eftersom roboten redan har tränats för ett väggföljande beteende kan roboten fokusera på att nå målet istället för att samtidigt lära sig ytterligare ett beteende.

Begrunda även den mer generella approachen, där en robot tränas för varje beteende i isolation, för att sedan tillämpa samtliga beteenden den tidigare lärt sig för att lösa problemet i sin helhet. Problemet med detta är att den som designar roboten och dess träning explicit måste välja vilka delproblem roboten skall utsättas för. Det har tidigare argumenterats för att mänskliga beslut som görs i samband med robotdesign inte alltid stämmer, vilket gör approachen problematisk. Trots detta kan det vara värt att överväga en inkrementell approach, då roboten i slutändan kan ha större chans att lyckas lösa problemet, givet att roboten faktiskt tränas för *rätt* delproblem.

Eftersom träningen sker i en simulatormiljö är det ett trivialt problem att använda så kallade externa fitnessfunktioner. Det vill säga fitnessfunktioner som förlitar sig på extern data istället för data som den har tillgång till själv. Om träningen skulle skett i verkligheten skulle det krävas mycket

kringutrustning för att avgöra något så enkelt som om roboten når målet, eller hur långt roboten är från målet vid slutet av en epok (Nolfi & Floreano, 2000). Vidare beskriver också Nolfi & Floreano (2000) flera experiment där träningen av en robot sker i verkligheten istället för i simulering. På grund av detta bör externa fitnessfunktioner inte använda orealistiska mätinstrument då försök att replikera experiment i verkligheten kan omöjliggöras.

Resultaten från undersökningen vittnar också om att miljöernas svårighetsgrad inte nödvändigtvis stämmer med de antaganden som gjordes då de designades. Miljö 2 framstår, att döma av resultaten, som svårare än miljö 3. Detta trots att miljö 3 är dynamisk. Den ensamma dynamiska väggen är enkel att undvika då det alltid är möjligt att åka över eller under den. Detta innebär i praktiken att miljö 3 inte är mycket svårare än vad miljö 1 är.

Miljö 4 som var undersökningens svåraste miljö visade sig vara betydligt svårare än förväntat. Att ingen agent lyckades nå målet är en bidragande faktor till varför de genomsnittliga tillförlitlighetsvärdena är förhållandevis låga. Försöket att designa miljöer enligt idén att de skall vara av ökande komplexitet har bestämt misslyckats. Detta är ett fall som tyder på ”den mänskliga faktorn” har spelat in och antaganden som görs nödvändigtvis inte stämmer. Det bekräftar återigen att mänskliga antaganden kring robotdesign inte nödvändigtvis stämmer och att det är svårt för människan att bedöma hur robotsystem skall designas (som t.ex. även Nolfi & Floreano (2000) uppmärksammar).

Komplexitetsgapet mellan de två dynamiska miljöerna i experimenten är väldigt stort. Detta medför, eftersom miljö 4 inte löses, att arkitekturernas tillförlitlighet i dynamiska miljöer inte direkt undersöks. Miljö 3 är i praktiken likvärdig miljö 1 i svårighetsgrad då den alltid kan undvikas med ett väggföljande beteende. Det kan vara på grund av det stora komplexitetsgapet som ESCN inte använder andragrads-feedback; miljöerna är antingen för enkla eller för svåra för att andragrads-feedback skall vara användbart. Fler dynamiska miljöer med mer balanserad komplexitet skulle kanske kunna få ESCN att använda andragrads-feedback.

Arbetet har ytterligare några punkter som är värda att belysa. Till att börja med har experimenten endast upprepats en gång, vilket är en klar brist, men var tvunget på grund av tidsbrist. Slumpen spelar stor roll i genetiska algoritmer och arbetet skulle gynnas av att upprepa experimenten flera gånger. På så sätt kan ett mer tillförlitligt resultat kan erhållas.

Vidare kan det diskuteras huruvida parametervalen för den genetiska algoritmen är lämpliga. Valen baseras på tidigare arbeten som har visat på goda resultat med dessa parametrar, men dessa arbeten är flera år gamla. Det är tänkbart att dessa parametrar har varit anpassade för dåtidens datorer (det framgår dock inte i dessa arbeten att så är fallet). Om parametrarna skulle anpassats för dagens datorer skulle resultaten kanske se annorlunda ut. Trots parametervalen tog träningen 24 timmar och en ändring av parametrarna, kombinerat med upprepning av experiment, skulle öka träningstiden avsevärt. Det finns trots allt värde i att begränsa inlärningstiden. En viktig poäng med inlärning är att det skall gå snabbare att lära roboten beteenden än vad det tar att programmera dem för hand.

6.1 Framtida arbete

Undersökningen som gjorts här är ganska begränsad vad det gäller antalet arkitekturer som undersökts. Det har också argumenterats för att komplexiteten mellan de olika miljöerna varit ojämn. En

naturlig fortsättning på detta arbete skulle därför vara att genomföra ytterligare experiment med fler arkitekturer och miljöer för att undersöka navigeringsproblemet.

Intressanta arkitekturer skulle bland annat vara flerlagers feedforward-nätverk och enkla rekurrenta nätverk. Eftersom ESCN, i form av ett enlagers feedforward-nätverk, visar på bra tillförlitlighet skulle det vara intressant att undersöka hur ett feedforward-nätverk med flera lager beter sig. Enkla rekurrenta nätverk har använts i flera tidigare undersökningar för robotar, varför det kan vara motiverat att undersöka dess tillförlitlighet för navigeringsproblemet.

Det skulle också vara intressant att undersöka arkitekturen som Bergfeldt & Linåker (2002) presenterar. Det är en multilager-arkitektur som består av två lager (kallade echelons) och den påminner till viss del om ESCN. Nätverket använder en Adaptive Resource Allocating Vector Quantizer (ARAVQ) för att klassificera input som beteenden, vilka lagras i den övre nivån. Det undre lagret mappar input till output likt ett perceptronnät och matar output kontinuerligt till omgivningen. Det övre lagret arbetar asynkront och har möjlighet att gå in och modifiera det underliggande lagret, för att på så sätt åstadkomma olika beteenden. Det övre lagret använder en gating-nod (samma princip som för beslutsneuronen hos ESCN) och kan därför vara restriktiv med när ett beteende skall ändras.

Ytterligare miljöer som undersöks bör vara av mer varierande komplexitet än de miljöer som används i denna undersökning. En vidareutveckling skulle kunna vara att ta fram fler dynamiska miljöer som minskar komplexitetsgapet mellan miljö 3 och miljö 4 i denna undersökning. På så sätt fokuserar arbetet mer på hur väl dynamiska miljöer löses, vilket är betydligt mer intressant än statiska miljöer (eftersom miljöer sällan är statiska i verkligheten).

För svårare miljöer kan det också undersökas huruvida det är möjligt att träna en robot inkrementellt, dvs. för ett beteende åt gången. Det tvingar dock den som designar miljöerna att göra antaganden om vilka delar i miljön som är svåra. Till exempel i miljö 4 kan det antas att roboten först måste lära sig ett väggföljande beteende för de rörliga väggarna innan den kan lyckas nå målet. När roboten försöker lösa hela uppgiften på en gång verkar det som att det blir för många faktorer som spelar in och tiden räcker inte till för att lära sig de beteenden som krävs.

I detta arbete används det euklidiska avståndet i fitnessfunktionen f_2 för att uppmana agenten att röra sig mot målet i en större utsträckning. Denna uträkning är billig och använder det kortaste avståndet till målet som uppmuntran. I ett framtida arbete skulle andra typer av avståndsuträkningar användas. Exempelvis skulle den så kallade city-blockdistansen kunna användas (*eng. city block distance, manhattan distance, taxicab geometry*), vilken är ännu billigare att räkna ut än det euklidiska avståndet.

Referenser

Ahlén, N. (2003) *En situerad ansats för utvecklingen av en räknande robot*. Kandidatuppsats. Institutionen för datavetenskap, Högskolan i Skövde, Sverige.

Aktius, M. (2007) *Historical background: the road to embodied AI* (PowerPoint slides). Tillgängligt på Internet: http://www2.his.se/icea/aktius/teaching/AR2007/HistoryAISlides20071108_toWeb.pdf [Hämtad 11.04.18].

Alnajjar, F., Zin, I. & Murase, K. (2009) A hierarchical autonomous robot controller for learning and memory: adaptation in a dynamic environment. *Adaptive Behavior, 17*, 179-196.

Bergfeldt, N. & Linåker, F. (2002) Self-organized modulation of a neural robot controller. *Proceedings of the 2002 International Joint Conference on Neural Networks* (s. 495-500), 12-17 May, 2002, Honolulu, HI, USA.

Búason, G. (2002) *Competitive co-evolution of sensory-motor systems*. Master's dissertation. Department of Computer Science, University of Skövde, Sweden.

Búason, G. & Ziemke, T. (2003) Competitive co-evolution of predator and prey sensory-motor systems. I: S. Cagnoni et al (red:er.), *LNCS 2611* (s. 605–615). Berlin: Springer-Verlag.

Brooks, R. A. (1986) A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation, 2*, 14-23.

Brooks, R. A (1990) Elephants don't play chess. *Robotics and Autonomous Systems, 6*, 3-15.

Brooks, R. A. (1991) Intelligence without representation. *Artificial Intelligence, 47*, 139-159.

Brooks, R. A. (1992) Artificial life and real robots. I: Varela & Bourgine (red:er), *Proceedings of the First European conference on Artificial Life, 3-10*, MIT Press.

Buckland, M. (2002) *AI techniques for game programming*. Cincinnati, OH: Premier Press.

Buckland, M (2005) *Programming game AI by example*. Sudbury, MA: Wordware Publishing, Inc.

Cormen, T., Leiserson, C. E. & Rivest, R. L. (1990) *Introduction to algorithms* (first edition). Cambridge, MA: The MIT Press & McGraw-Hill.

Dahlbom, A. (2011) *Petri nets for situation recognition*. Doctoral dissertation. Örebro University, Sweden.

Davis, L. (1991) *Handbook of genetic algorithms*. New York, USA: International Thomson Computer Press

DeLoura, M. (2000) *Game programming gems*. Hingham, MA: Charles River Media.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269-271.

Floreano, D. & Mattiussi, C. (2008) *Bio-inspired artificial intelligence: theories, methods and technologies*. Cambridge, MA: The MIT Press.

Floreano, D. & Modada, F. (1994) Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. I: Cliff, Husbands, Meyer, and Wilson (red:er), *Proceedings of the Third International Conference on the Simulation of Adaptive Behavior 3* (s. 421–431).

Graham, R., McCabe. H. & Sheridan, S. (2003) Pathfinding in computer games. *ITB Journal, 8*, 56-80.

Graham, R., McCabe. H. & Sheridan, S. (2005) Realistic agent movement in dynamic game enviroments. *Changing Views - Worlds in Play* (s. 249-259). Proceedings of DIGRA, June, 2005, Vancouver, Canada.

Graham, R. (2006) *Real-time agent navigation with neural networks for computer games*. Master's dissertation. Institute of Technology Blanchardstown, Ireland.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*, 100-107.

Jakobi, N., Husbands, P. & Harvey, I. (1995) Noise and the reality gap: the use of simulation in evolutionary robotics. I: Morán, Moreno, Merelo & Chacón (red:er), *Advances in Artificial Life Third European Conference on Artificial Life*, 4-6 June, 1995, Granada, Spain.

Jakobi, N. (1996) Encoding scheme issues for open-ended artificial evolution. I: Vogit, Ebeling, Rechenberg & Schwefel (red:er), *PPSN IV Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*. Berlin: Springer-Verlag.

Jakobi, N. (1997a) Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior, 6*, 325- 368.

Jakobi, N. (1997b) Half-baked, ad hoc and noisy: minimal simulations for evolutionary robotics. I: Husbands & Harvey (red:er), *Fourth European Conference on Artificial Life*, 348-357, MIT Press.

Jakobi, N. (1998a) *Minimal simulations for evolutionary robotics*. Doctoral dissertation. University of Sussex, England.

Jakobi, N. (1998b) Running across the reality gap: octopod locomotion evolved in a minimal simulation, I: Husbands & Meyer (red:er), *Proceedings of the First European Workshop on Evolutionary Robotics* (s. 39-58), 16-17 April, 1998, France.

Jakobi, N. & Quinn, M. (1998) Some problems (and a few solutions) for open-ended evolutionary robotics. I: Husbands & Meyer (red:er), *Proceedings of the First European Workshop on Evolutionary Robotics* (s. 108-122), 16-17 April, 1998, France.

Korf, E. R. (1998) Artificial intelligence search algorithms. I: M.J. Atallah (red.), *CRC Handbook of Algorithms and Theory of Computation* (s. 36-1 till 36-20). Florida: CRC Press.

Meeden, L. A. (1994) *Towards planning: incremental investigations into adaptive robot control*.

Doctoral dissertation. Department of Computer Science, Indiana University, United States.

Mitchell, M. (1996) *An introduction to genetic algorithms*. Cambridge, MA: The MIT Press.

Nolfi, S. (1997) Using emergent modularity to develop control systems for mobile robots. *Adaptive Behavior, 5*, 343-363.

Nofi, S. & Floreano, D. (2000) *Evolutionary robotics: the biology, intelligence and technology of self-organizing machines*. Cambridge, MA: The MIT Press.

Parker, G. B. & Zhiyi, L. (2003) Evolving neural networks for hexapod leg controllers. Intelligent Robots and Systems vol. 2 (s. 1376-1381). *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 27-31 October, 2003.

Risi, S., Hughes, C., E. & Stanley, K., O. (2010) Evolving plastic neural networks with novelty search. *Adaptive Behavior, 18*, 470-491.

Rosen, H. K. (2007) *Discrete mathematics and its applications sixth edition*. New York, USA: McGraw-Hill.

Russel, S. & Norvig, N. (2010) *Artificial intelligence a modern approach third edition*. New Jersey: Prentice Hall.

Sims, K. (1994) Evolving virtual creatures. *Computer Graphics, Annual Conference Series* (s.15-22). SIGGRAPH '94 Proceedings, July 1994.

Stentz, A. (1994) Optimal and efficient path planning for partially-known environments. Proceedings of the International Conference on Robotics and Automation (s. 3310–3317). *1994 IEEE International Conference on Robotics and Automation*, 08-13 May, 1994, San Diego, CA, USA

Thieme, M. (2002) *Intelligence without hesitation*. Master's dissertation. Department of Computer Science, University of Skövde, Sweden.

Tomlinson. L. S. (2004) The long and short of steering in computer games. *International Journal of Simulation, 5*, 33-46.

Wahde, M. (2007) *Evolutionary robotics the use of artificial evolution in robotics* (Technical Report TR-BBR-2007-001). Department of Applied Mechanics, Chalmers University of Technology, Sweden.

Ziemke, T. (1999) Remembering how to behave: recurrent neural networks for adaptive robot behavior. I: Medsker & Jain (red:er.), *Recurrent Neural Networks: Design and Applications* (s. 341-376). Boca Raton: CRC Press.

Ziemke, T. (2000) *Situated neuro-robotics and interactive cognition*. Doctoral dissertation. Department of Computer Science, University of Sheffield, England.

Ziemke, T. (2001) Are robots embodied?. I: Balkenius, Zlatev, Dautenhahn, Kozima, Breazeal (red:er.), *Proceedings of the First International Workshop on Epigenetic Robotics - Modeling Cognitive Development in Robotic Systems* (s. 75-83). Lund University Cognitive Studies, vol. 85. Lund, Sweden.

Ziemke, T. & Thieme, M. (2002) Neuromodulation of reactive sensorimotor mappings as a short-term memory mechanism in delayed response tasks. *Adaptive Behavior, 10*, 185-199.

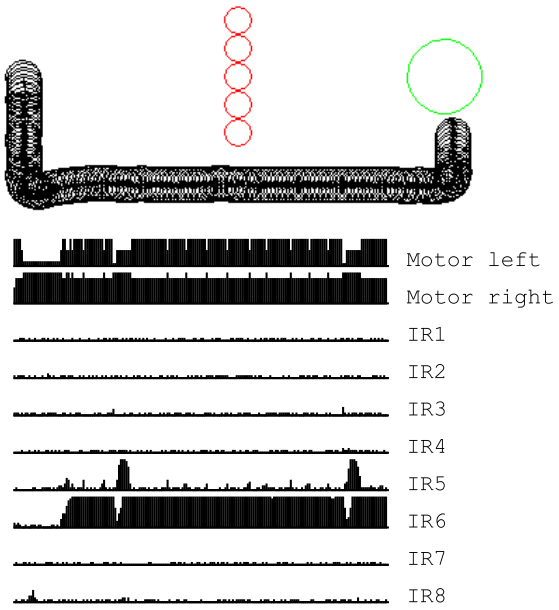
Bilaga A

Rörelsebanor och aktiveringsvärden

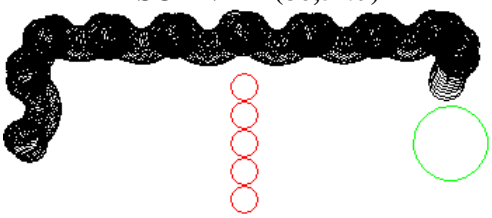
Denna bilaga illusterar representativa rörelsebanor och aktiveringsvärden för samtliga experiment i denna undersökning. En motor ger full effekt framåt vid 1,0 och full effekt bakåt vid 0,0. En motor står still vid 0,5. Värdet inom parentes beskriver antalet lyckade försök för agenten att lösa experimentets miljö.

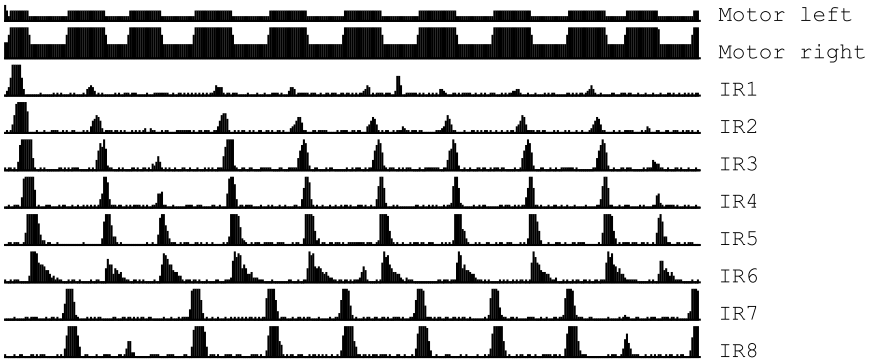
A.1 Self-organized recurrent network

SORN-1-1 (99,9 %)

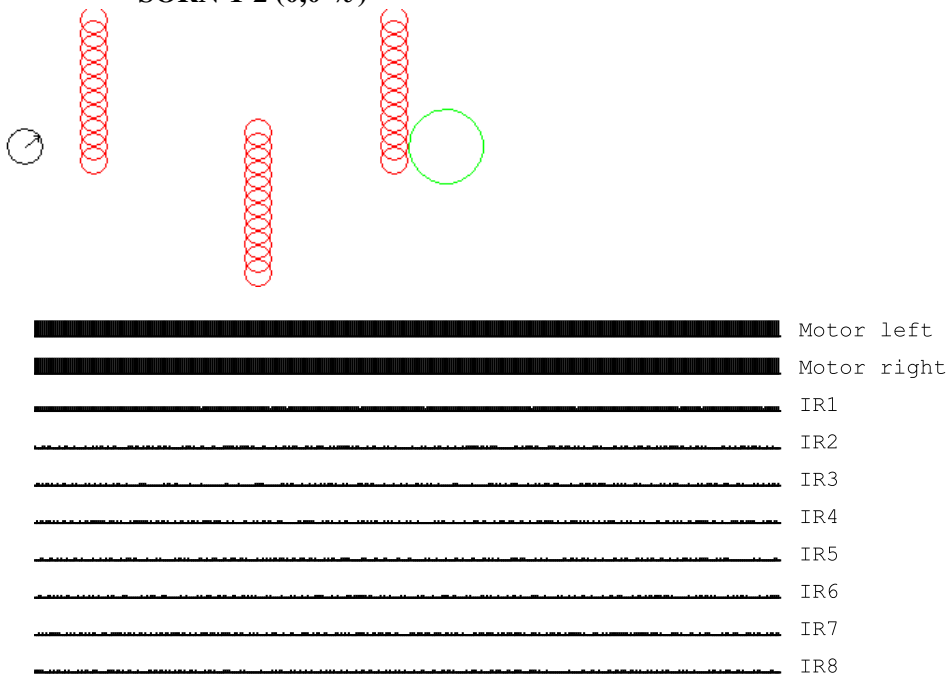


SORN-2-1 (86,5 %)

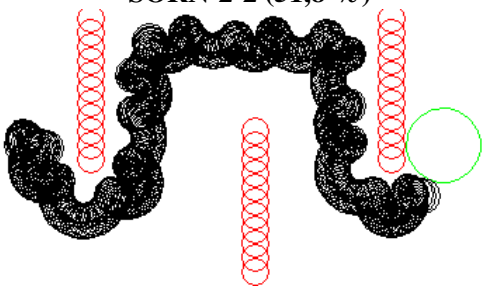


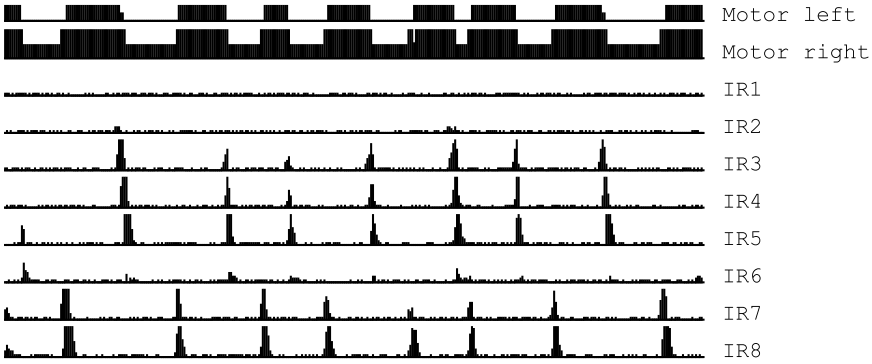


SORN-1-2 (0,0 %)

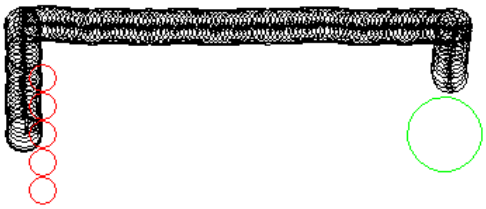


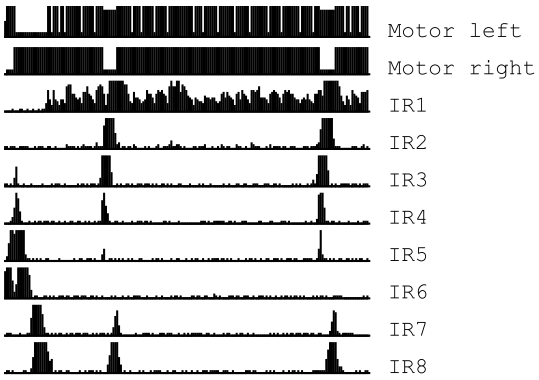
SORN-2-2 (51,8 %)



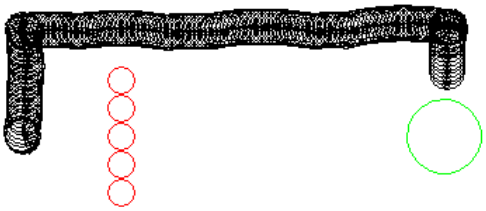


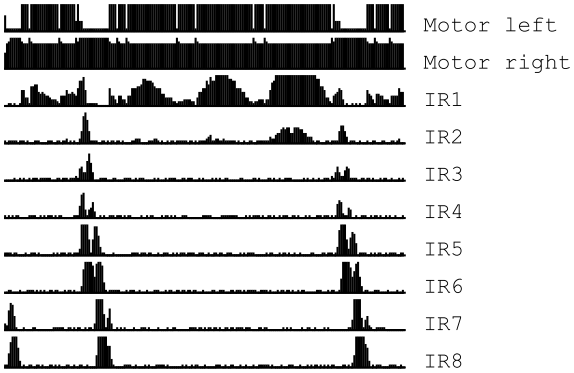
SORN-1-3 (95,5 %)



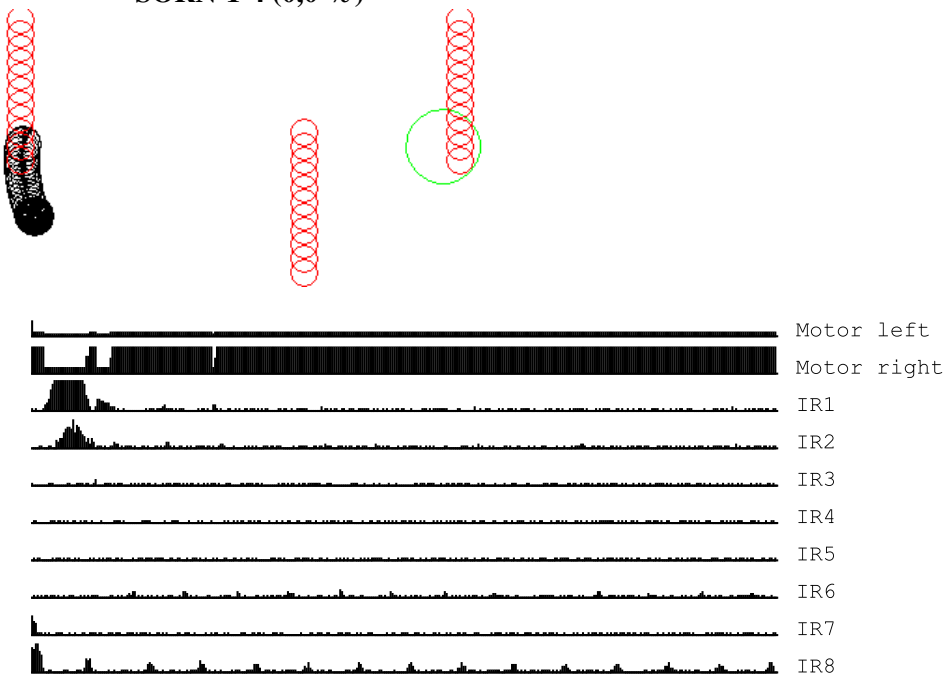


SORN-2-3 (100,0 %)

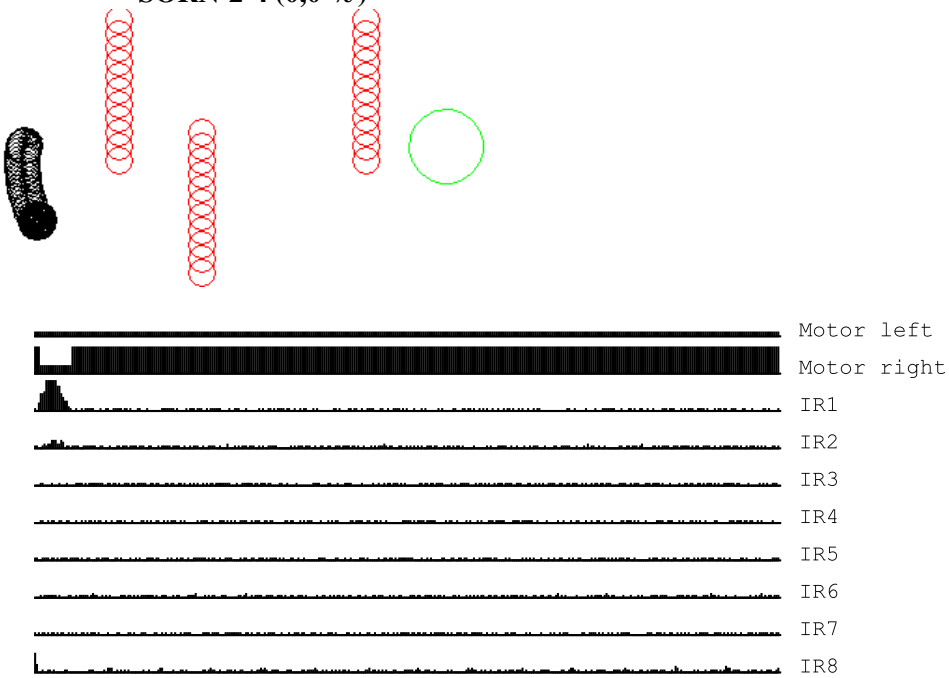




SORN-1-4 (0,0 %)

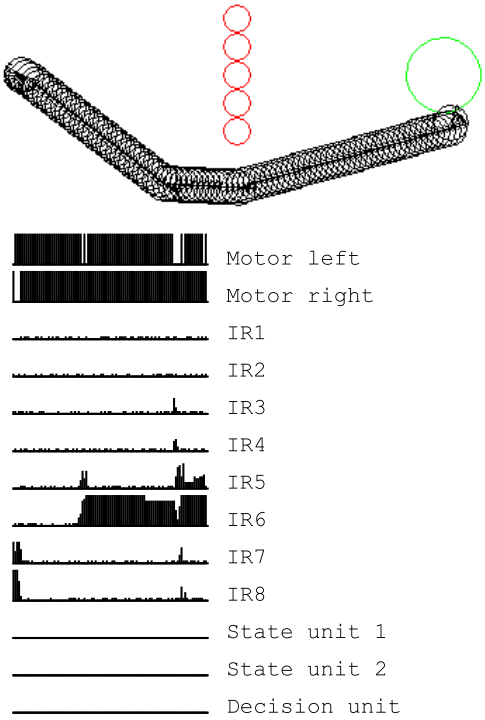


SORN-2-4 (0,0 %)

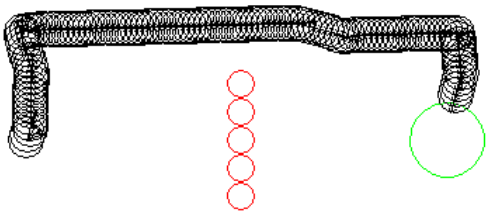


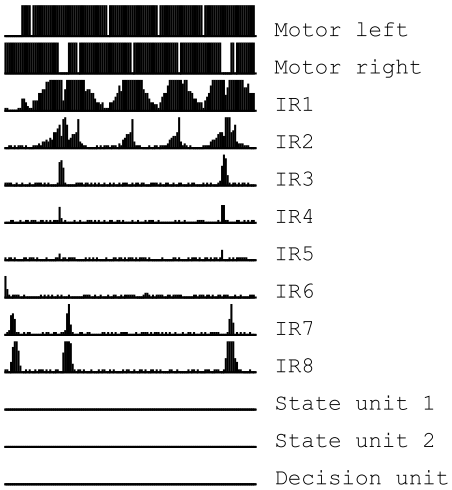
A.2 Extended Sequential cascaded network

ESCN-1-1 (92,2 %)

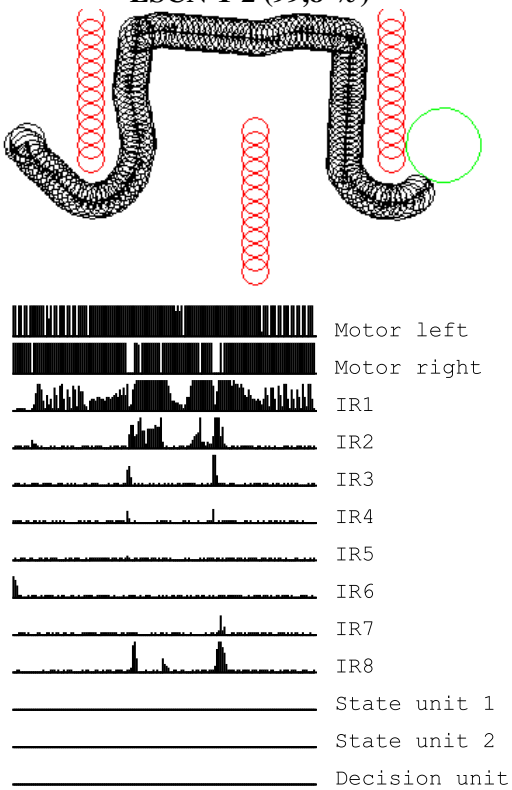


ESCN-2-1 (99,7 %)

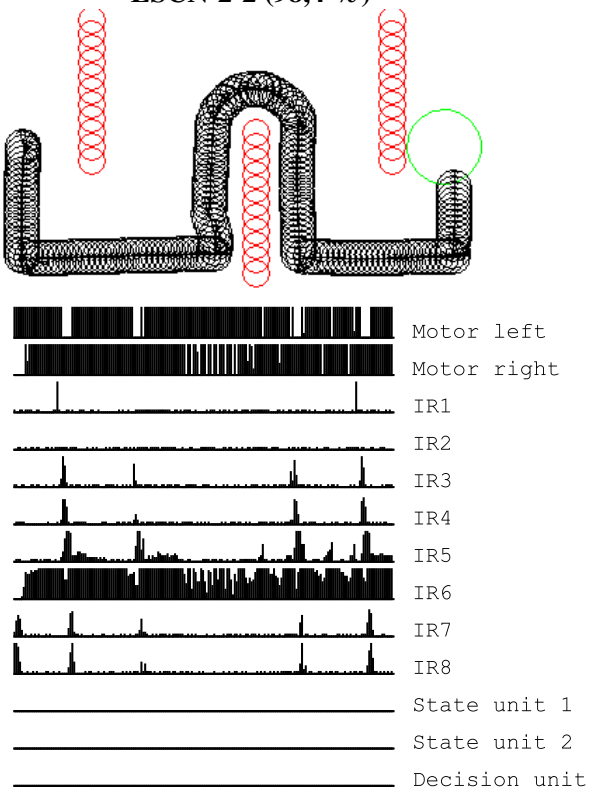



Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

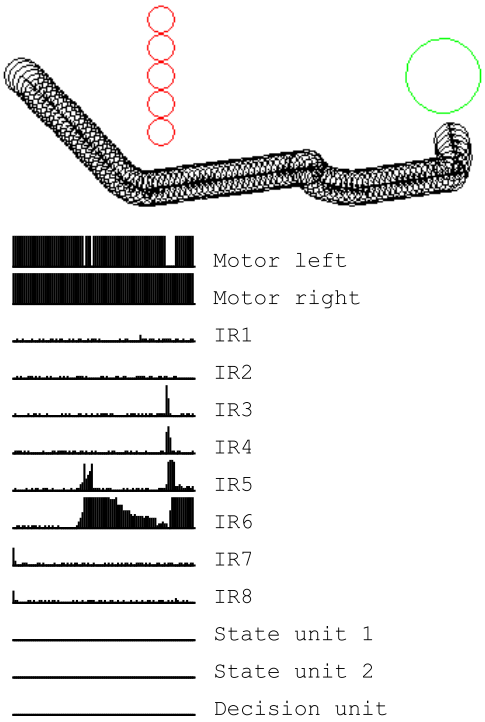
ESCN-1-2 (99,8 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

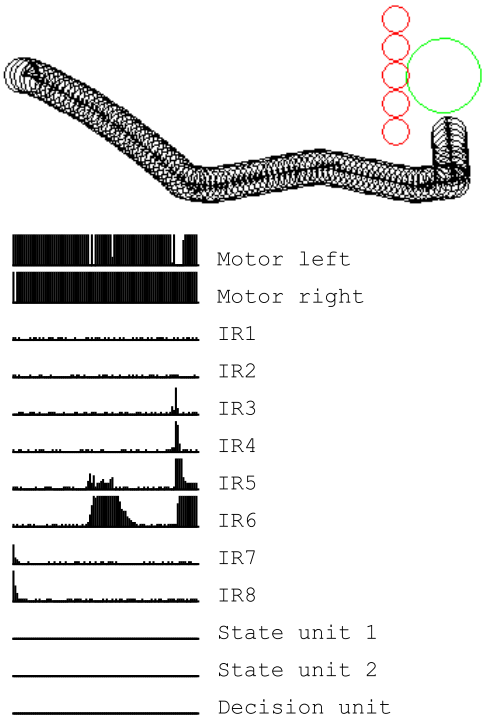
ESCN-2-2 (98,4 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

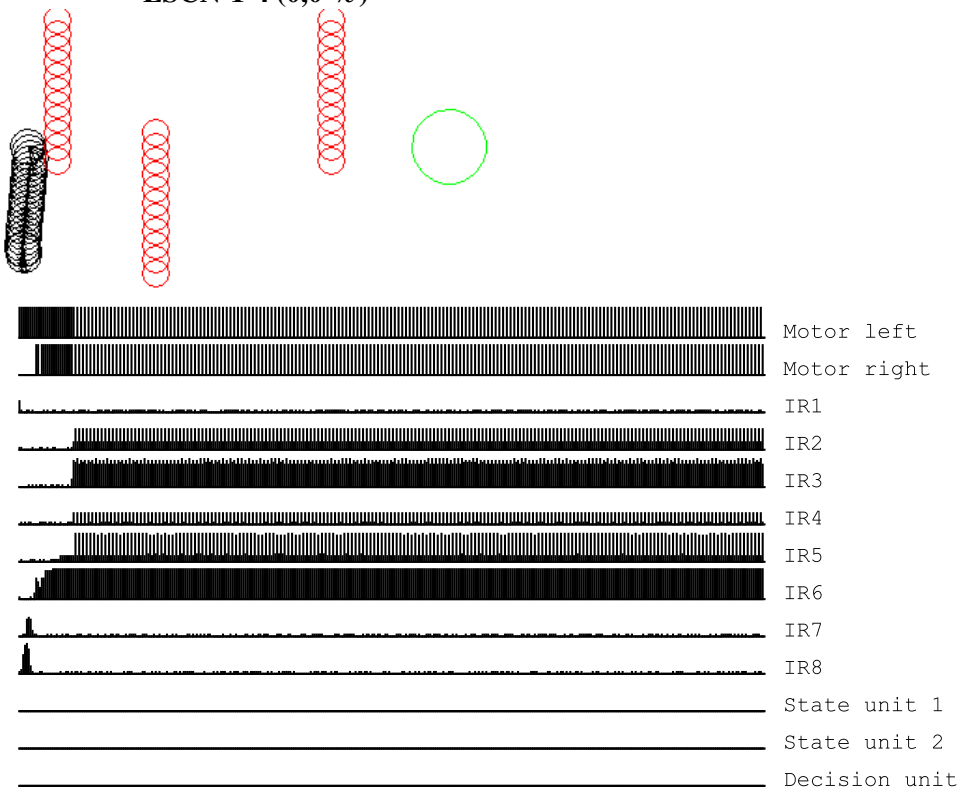
ESCN-1-3 (92,2 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

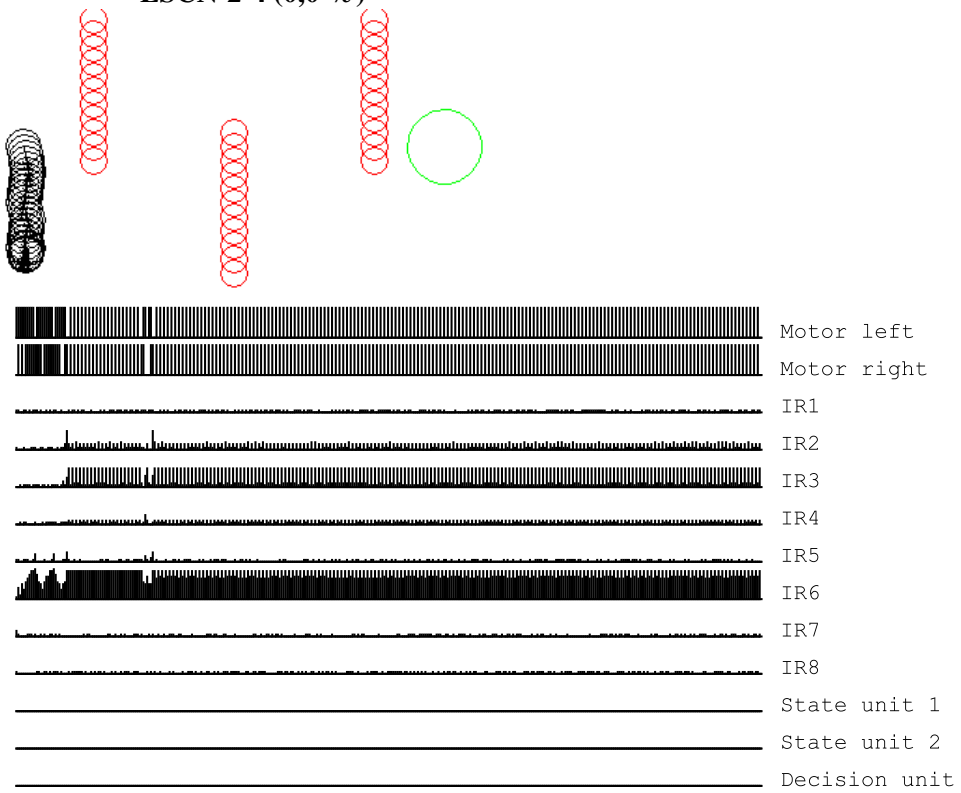
ESCN-2-3 (98,4 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

ESCN-1-4 (0,0 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit

ESCN-2-4 (0,0 %)


Motor left
Motor right
IR1
IR2
IR3
IR4
IR5
IR6
IR7
IR8
State unit 1
State unit 2
Decision unit