



Høgskolen i Oslo og Akershus

OPPGAVE NR.
EY-1503

TILGJENGELIGHET
Åpen

HOVEDPROSJEKT ELEKTRO

OPPGAVENS TITTEL Autonome Selvlærende Systemer	DATO 26.05.2015
	ANTALL SIDER / BILAG 99 / CD
FORFATTERE Atle Fjellang Sæther (s160905) Bendik Høye (s184492) Kenny Duy Luong (s188199) Ole Bernard Nygaard (s176452)	VEILEDER Evi Zouganeli P35-PS533 Tlf: 22 45 32 07
UTFØRT I SAMMARBEID MED Høgskolen i Oslo og Akershus	KONTAKTPERSON Evi Zouganeli

SAMMENDRAG

I dagens samfunn er selvlærende systemer et stadig mer aktuelt tema. Prosjektets formål har vært å belyse de viktigste metodene for selvlærende systemer, utvikle en selvlærende algoritme og en egnet demonstrator og simulator for denne. Systemet beskrevet i denne oppgaven er realisert med Q-læring ved både tabellmetoden og nevral nettverk.

3 STIKKORD

Selvlærende systemer

Q-learning

Reinforcement learning

Autonome selvlærende systemer

Bendik Høye, Atle F. Sæther, Kenny D. Luong, Ole B. Nygaard
26.05.2015

Forord

Denne rapporten beskriver de ulike prosessene som inngår i arbeidet med hovedprosjektet ved Høgskolen i Oslo og Akershus (HIOA), avdeling for ingeniørutdanning, våren 2015.

Rapporten omhandler utvikling av en selvlærende algoritme og en demonstrator i form av en robot som skal unngå statiske og dynamiske objekter i omgivelser som er i kontinuerlig endring. Oppgaven er gitt av HIOA. Rapporten tar for seg teorien bak de mest kjente og brukte selvlærende algoritmene, og diskuterer fordeler, ulemper og bruksområder for disse. Den inneholder også en beskrivelse av den tekniske løsningen for demonstratoren, og metoden som er brukt i dette prosjektet.

Rapporten er skrevet innen et tema som anses som nyteknisk og antas derfor å kunne brukes til videre forskning og/eller læring innen autonome selvlærende systemer.

Leseren forventes å ha grunnleggende kunnskap innen elektronikk og informasjonsteknologi.

Dette dokumentet er optimalisert for papir.

Vi ønsker å rette en stor takk til vår arbeidsgiver, Høgskolen i Oslo og Akershus, for muligheten til å gjennomføre prosjektet og for finansiell støtte. Vi vil også takke veileder Evi Zouganeli for et godt samarbeid, samt viktig og konstruktiv veiledning gjennom hele prosjektperioden.

Sammendrag

I dagens samfunn er selvlærende systemer et stadig mer aktuelt tema. Systemer som ikke eksplisitt er programmert til å utføre en konkret oppgave, men selv evner å tilpasse seg, kan være til stor nytte.

Prosjektets formål har vært å belyse de viktigste metodene for selvlærende systemer, utvikle en selvlærende algoritme og en egnet demonstrator for denne. Det gis i tillegg et innblikk i anvendelsesområder og state-of-the-art innen selvlærende systemer.

Systemet beskrevet i denne oppgaven er realisert med Q-læring ved både tabellmetoden og nevral nettverk. Q-læring er en læringsalgoritme basert på erfaring. Algoritmen går ut på at en agent utforsker sine omgivelser, der omgivelsene er representert ved et antall tilstander. Agenten eksperimenterer med omgivelsene ved å utføre en handling, for deretter å observere konsekvensen av denne handlingen. Konsekvensen gis i form av en positiv eller negativ belønning. Målet med metoden er å maksimere den akkumulerte belønningen over tid.

Det er i løpet av prosjektet utviklet både en simulator og en demonstrator. Demonstratoren består av en agent som befinner seg i et ukjent miljø med statiske og dynamiske hindringer. Over tid skal agenten lære seg hvilke handlinger som må gjøres for å navigere uten å kolliderer. Agenten oppfatter miljøet ved hjelp av fem sensorer montert i front. Simulatoren er utviklet i MATLAB med den hensikt å være mest mulig lik den fysiske modellen. Den fysiske modellen kan enten trenes opp fra grunn eller benytte seg av ferdig simulerte verdier. Programmetts brukergrensesnitt viser relevante data for systemet under simulering.

Simuleringsresultater viser at både nevral nettverk og tabellmetoden gjør den simulerte agenten i stand til å navigere i et ukjent miljø, mens den fysiske modellen kun håndterer statiske hindringer grunnet fysiske begrensninger og prosjektets tidsomfang.

Videre arbeid for prosjektet kan blant annet vurderes til å omfatte implementering av et større antall handlinger for agenten. Ved å introdusere hastighet og grader av retningsendring, antas det at agenten er i stand til å håndtere flere tilstander, herunder dynamiske hindringer.

Innholdsfortegnelse

Forord.....	5
Sammendrag	7
Innholdsfortegnelse	9
Figurliste	11
Tabelliste.....	12
Innledning	13
1.1 Problemstilling	13
2 Teori.....	15
2.1 Maskinl�ring	15
2.1.1 Veiledet l�ring (eng.: Supervised Learning).....	16
2.1.2 L�ring uten tilsyn (eng.: Unsupervised Learning)	16
2.1.3 Forsterkningsl�ring (eng.: Reinforcement Learning)	16
2.2 State-of-the-art.....	16
2.2.1 Autonom bil	16
2.2.2 S�ppelpost-filter	17
2.2.3 Virtuell organisme.....	17
2.2.4 Connected grid, selvl�rende overv�kning	17
2.2.5 Q-L�ring	18
2.2.6 Q-funksjonen.....	20
2.3 Utforskning eller utnytting	23
2.3.1 ϵ -gr�dig	24
2.3.2 Boltzmannfordeling.....	24
2.3.3 Kunstig nevral nettverk (eng.: Artificial Neural Network)	25
2.4 Kostnadsfunksjon	30
2.5 Tilbake-forplantnings algoritmen	30
2.6 Fuzzy Logikk.....	31
2.6.1 Virkem�te	32
3 Teknisk l�sning	35
3.1 Agenten.....	35
3.1.1 Oppbygning.....	36
3.1.2 Elektronikk.....	37
3.2 Arenaen.....	44
3.2.1 Dynamiske hindringer	45
4 Systeml�sning	47
4.1 Q-L�ring med tabellmetoden	49
4.2 Diskretisering	50
4.2.1 Tilstandsrommodell.....	50
4.2.2 Bel�nningsfunksjon.....	53
4.2.3 Utforskningsfunksjon	54
4.2.4 Implementering	54
4.3 Q-L�ring med nevral nettverk (Line�r funksjons approksimering)	57
4.3.1 Bel�nningsfunksjon.....	59
4.3.2 Foroverkoblet nevral nettverk (eng.: Feedforward)	60
4.3.3 Oppdatering av vektmatrisene	61
4.3.4 Virkem�te	65
5 Resultater	67
5.1 Prosedyre for simulering	67
5.2 Utforskningsfunksjon	68
5.2.1 Alfa og gamma.....	70
5.3 Nevral nettverk	72
5.3.1 Alfa.....	72
5.3.2 Gamma	73
5.3.3 Sammenligning	74
5.4 Simulering av system med dynamiske hindringer	77
6 Diskusjon	83
6.1 Sammenligning av demonstrator og simulering	83
6.1.1 Sensorer.....	83
6.2 Dynamiske hindringer	84

6.3	Metodevalg	85
6.3.1	Feillæring og tilfeldigheter	85
6.3.2	Utforskningsfunksjon og valg av parametere	85
6.3.3	Diskretisering	86
6.3.4	Prosjektets resultater	86
6.3.5	Videre arbeid	87
6.4	Uavhengig agent	87
7	Konklusjon	89
7.1	Videre arbeid	90
7.1.1	Ressurser	90
8	Litteraturliste	91
9	Vedlegg	95
9.1	A) Koblingsskjema	95
9.2	B) Flytskjema for tabellmetoden	97
9.3	C) Flytskjema for nevralt nettverk	98
9.4	D) Flytskjema for fysisk agent	99

Figurliste

Figur 2-1 - Illustrasjon av prinsippet for tilstand, handling og belønning	19
Figur 2-2 – Illustrasjon av agentens sekvens.....	19
Figur 2-3 – Illustrasjon av agentens erfaring.....	19
Figur 2-4 - Eksempel på tilstandsrom	21
Figur 2-5 - Tilstandstabell med tilhørende Q-verdier.....	22
Figur 2-6 - Illustrasjon av et naturlig nevron	26
Figur 2-7 - Illustrasjon av et kunstig nevron med inndata x , vektor w og utdata a	26
Figur 2-8 – Plott av aktiveringsfunksjonen $\tanh(x)$	27
Figur 2-9 - Illustrasjon av et kunstig nevral nettverk	28
Figur 2-10 - Illustrasjon av gradient nedstigningsprinsippet.....	31
Figur 2-11 - Medlemsskapsfunksjon.....	32
Figur 2-12 - Fuzzy logikk	32
Figur 3-1 – WLtoys modell A969 med karosseri.....	35
Figur 3-2 - WLtoys modell A969 uten karosseri	35
Figur 3-3 - Agent 1 med sensorrigg og topplate.....	35
Figur 3-4 – Målt svingradius for agent versjon 1	36
Figur 3-5 - Bøying av festebrakett for motorene.....	36
Figur 3-6 - Motorbrakett med begge motorene montert.....	36
Figur 3-7 - Motorbrakett festet til platen.....	37
Figur 3-8 - Ferdig agent versjon 2.....	37
Figur 3-9 - Oversikt over agentens elektronikk.....	37
Figur 3-10 - Testmiljø for måling av sensorens detekteringsvinkel og avvik	39
Figur 3-11 - Testing av en sensor koblet til en Arduino UNO	39
Figur 3-12 - Målt detekteringsvinkel for HC-SR04	39
Figur 3-13 - Konsepttegning for HC-SR04.....	40
Figur 3-14 - Tidsdiagram for trigger/echo for HC-SR04	40
Figur 3-15 - Koblingsskjema for Arduino UNO R3	41
Figur 3-16 - Step-up converter (justerskrue øverst)	42
Figur 3-17 - Koblingsskjema for agentens regulatortrinn	42
Figur 3-18 - Ferdig 5V spenningsregulator.....	42
Figur 3-19 - Koblingsskjema for motordriver	43
Figur 3-20 – Motordriver L298N Dual H-bro	43
Figur 3-21 – Bilde av arenaen.....	44
Figur 3-22 - System for dynamiske hindringer	45
Figur 3-23 - Bilde av motoraksling mot gummi hjul	46
Figur 3-24 - Oversikt over elektronikk for systemet for roterende hindringer	46
Figur 4-1 - Illustrasjon av agenten me synsfeltet for de fem sensorene	47
Figur 4-2 – Agentens rotasjon og rotasjonsendring	48
Figur 4-3 - Illustrasjon av prinsippet Q-læring med tabellmetoden	49
Figur 4-4 - Illustrasjon av agentens synsfelt	50
Figur 4-5 - Oppdeling av sektorer.....	51
Figur 4-6 – Skjermdump av brukergrensesnitt for simulering	56
Figur 4-7 - Skjermdump fra simulering av dynamiske hindringer	56
Figur 4-8 - Kommunikasjon mellom Arduino og MATLAB	57
Figur 4-9 - Agentens nevrale nettverk.....	58
Figur 4-10 - Konseptskisse for agentens læringsprosess med NN	59
Figur 4-11 - Gradient nedstigning for kvadratisk funksjon.....	64
Figur 5-1 – Sammeligning av steg for utforskningsfunksjonen med Q-tabell	69
Figur 5-2 – Sammenligning av belønning for utforskningsfunksjonene med Q-tabell	69
Figur 5-3 – Simulering av alfaverdier for tabellmetoden	70
Figur 5-4 – Simulering av gammaverdier for tabellmetoden	71
Figur 5-5 - Simulering av alfaverdier for nevral nettverk (test 1)	72
Figur 5-6 – Simulering av alfaverdier for nevral nettverk (test 2)	72
Figur 5-7 - Total belønning Q for ulike verdier av Γ for Neural Network.....	73
Figur 5-8 – Simulering for sammenligning av tabellmetoden og nevral nettverk.....	75
Figur 5-9 - Simulering av samlet belønning for tabellmetoden.....	75
Figur 5-10 - Simulering av samlet belønning for nevral nettverk	76

Figur 5-11 – Simulering av kostnadsfunksjon for nevralt nettverk	76
Figur 5-12 - Simulering av antall steg for tabellmetoden.....	77
Figur 5-13 - Simulering av belønning for tabellmetoden	78
Figur 5-14 – Simulering av antall steg for nevralt nettverk	78
Figur 5-15 – Simulering av belønning for nevralt nettverk.....	79
Figur 5-16 – Simulering av kostnadsfunksjon for nevralt nettverk.....	79
Figur 5-17 - Simulering av steg for system med dynamiske hindringer for nevralt nettverk (4000 forsøk)	80
Figur 5-18 - Simulering av belønning for system med dynamiske hindringer for NN (4000 forsøk)	80
Figur 5-19 - Simulering av kostnadsfunksjon for system med dynamiske hindringer for NN (4000 forsøk)	81
Figur 9-1 - Koblingsskjema for øvrig elektronikk (Del 1)	95
Figur 9-2 - Koblingsskjema for spenningsregulator og sensorer (Del 2)	96
Figur 9-3 - Flytskjema for tabellmetoden.....	97
Figur 9-4 - Flytskjema for nevralt nettverk	98
Figur 9-5 - Flytskjema for fysisk agent	99

Tabelliste

Tabell 1 - Sekvens av erfaringer	22
Tabell 2 - Sammenligning av teoretiske data for IR-sensorer og ultrasoniske sensorer.....	38
Tabell 3 - Målt avvik og detekteringsvinkel for HC-SR04	40
Tabell 4 - Pinnekonfigurasjon for Arduino UNO R3	41
Tabell 5 - Logisk tabell for motordriver L298N Dual H-bro for Motor 1	43
Tabell 6 - Logisk tabell for motordriver L298N Dual H-bro for Motor 2.....	43
Tabell 7 - Utsnitt av tilstandstabell for statiske hindringer	52
Tabell 8 - Oversikt over diagrammer i resultater	67
Tabell 9 – Resultat fra simulering av alfaverdier for tabellmetoden	70
Tabell 10 – Resultater fra simulering av gammaverdier for tabellmetoden	71
Tabell 11 – Resultat fra simulering av alfaverdier for nevralt nettverk	73
Tabell 12 – Resultat fra simulering av gammaverdier for nevralt nettverk	74
Tabell 13 – Resultat fra simulering av sammenligning av tabellmetoden og nevralt nettverk.....	74
Tabell 14 - Oversikt over totalverdier for dynamisk simulering	77

Innledning

Selvlærende systemer har alltid vært en viktig del av kunstig intelligens. Et selvlærende system kjennetegnes ved at det ikke programmeres til å utføre en eksplisitt oppgave, men basert på empirisk erfaring, kjenner igjen mønstre, tilpasser seg ulike omgivelser og optimaliserer effektiviteten og nøyaktigheten i et system.

Innen maskinlæring finnes mange ulike selvlærende metoder, og denne rapporten presenterer de mest kjente metodene innenfor dette feltet.

Autonome selvlærende systemer blir stadig mer aktuelt fordi systemet evner å tilpasse seg delvis eller helt ukjente situasjoner. Det lærer av erfaring og trenger mindre informasjon ved oppstart ettersom det tilegner seg informasjon underveis. Innen autonome selvlærende systemer og selvkjørende roboter er det å unngå hindringer en sentral oppgave. Denne rapporten tar for seg en demonstrator for et slikt system, realisert med Q-læring som presenteres senere i rapporten, og gir en beskrivelse av algoritmen og resultatene.

Rapportens resultater er basert på simuleringer, og viser testresultater for parametere, samt en sammenligning av Q-læring, implementert med nevral nettverk og tabellmetoden. I rapportens diskusjon drøftes metodevalg og parametere, arbeidets betydning og videre arbeid med blant annet å konstruere en uavhengig agent og å gjøre den i stand til å håndtere dynamiske hindringer.

1.1 Problemstilling

Oppgaven har bestått av å gjennomføre en studie av metoder for autonome selvlærende systemer og å lage en demonstrator for et slikt system.

Oppdragsgivers ønske er at det gjøres et litteraturstudie av de viktigste metodene for selvlærende systemer i dag, samt anvendelseseksempler for disse. Det skal også utvikles en metode for å realisere et selvlærende system som skal demonstreres i en egnet demonstrator, og det skal utarbeides en rapport som presenterer arbeidet, evaluerer resultatene, kommer med forslag til forbedringer og drøfter videre arbeid.

Litteraturstudiet er begrenset til å omhandle fagfeltet maskinlæring, og vil ikke ta for seg andre selvlærende metoder.

Arbeidet skal kunne kvalitetssikres ved at det demonstreres, samt dokumenteres ved statistikk og figurer.

2 Teori

Før man ser på oppbygningen av et selvlærende system, kan man med fordel se på hva begrepet læring er. Læring er ofte definert som en varig endring i oppførsel som et resultat av erfaring (St. Olavs Hospital, udatert). Egenskapen organismer har som er definert som læring er en av grunnpilarene i det som kalles intelligens som, blant annet, defineres som en evne til å tilegne og anvende kunnskap og ferdigheter. Mennesker og dyr anses som intelligente, blant annet, basert på deres evne til å lære av erfaring.

2.1 Maskinlæring

Dette underkapittelet er basert på teorien i S. Marsland, 2009.

Maskinlæring (eng.: machine learning) er en form for kunstig intelligens (eng.: artificial intelligence) som fokuserer på utvikling av selvlærende algoritmer.

Selvlærende systemer tar i de fleste tilfellerfor seg deler av naturlig intelligens, herav hukommelse, tilpasning og generalisering. I motsetning til tradisjonelle ikke-lærende systemer gjør metoden det mulig å konstruere et system som er i stand til å ekspandere, tilpasse seg ny informasjon og lære seg en gitt oppgave uten å være spesifikt programmert for dette. For maskinlæring kalles dette systemet en agent. Ved å benytte seg av hukommelse kan en agent kjenne igjen forrige gang den var i tilsvarende situasjon og hvilken handling den gjorde. Basert på utfallet fra forrige gang, kan den dersom det var riktig, velge å gjenta handlingen, eller prøve noe nytt. Ved å generalisere kan agenten kjenne igjen likheter i ulike situasjoner, og dermed benytte erfaring fra en situasjon og anvende denne erfaringen i en annen.

Målet med metoden er å få en agent til å modifisere eller tilpasse sine handlinger for å gjøre dem så nøyaktige som mulig, der nøyaktigheten er målt i hvor godt valgt handling, reflekterer korrekt handling.

For å kunne realisere dette konseptet benytter maskinlæring seg av prinsipper fra statistikk, matematikk, fysikk, nevrologi og biologi.

Når man snakker om maskinlæring og selvlærende systemer er det hovedsakelig algoritmer som er hovedproduktet. Selve prosessen i disse algoritmene kan sammenliknes med datamining. Datamining er en prosess som analyserer data fra ulike perspektiver og sammenfatter det til nyttig informasjon. Begge metodene går igjennom data for å finne mønstre, men i stedet for å pakke ut data for menneskelig tolkning blir informasjonen brukt til å forbedre agentens forståelse. For at agenten skal kunne lære, må den vite hvordan den skal forbedre seg, og om den blir bedre eller ikke. Det finnes

flere metoder for å løse dette, som igjen gir flere hovedkategorier innen maskinlæring: Veiledet læring, læring uten tilsyn og forsterkningslæring.

2.1.1 Veiledet læring (eng.: Supervised Learning)

En agent blir gitt et treningssett med for eksempel bilder av et ansikt og bilder uten ansikt. Agenten blir så forberedt gjennom en treningsprosess hvor den gir en prognose om hva bildet er av. Hver gang prognosen ikke er riktig, blir agenten korrigert. Denne prosessen fortsetter inntil modellen oppnår et ønsket nivå av nøyaktighet. Ettersom algoritmen ikke har en bestemt definisjon av hva som er et ansikt og ikke, må den dermed lære dette ved hjelp av eksempler. En god algoritme vil til slutt være i stand til å anslå om et bilde er av et ansikt eller ikke. Læringsmetodene forklares best ved eksempler:

2.1.2 Læring uten tilsyn (eng.: Unsupervised Learning)

Ved læring uten tilsyn blir ikke informasjon merket. Det vil si at systemet ikke blir fortalt hva som er bilde av et ansikt, og hva som ikke er. Som følge av dette finnes det ingen korrigering eller belønning for å indikere en potensiell løsning, men algoritmen forsøker å identifisere likheten mellom bildene, for så å kategorisere dem og dele dem inn i grupper.

2.1.3 Forsterkningslæring (eng.: Reinforcement Learning)

Ved forsterkningslæring blir algoritmen fortalt når svaret den avgir er feil, men får ingen forslag til hvordan den skal korrigere dette. Den må selv utforske og prøve ut forskjellige løsninger til den finner ut hvordan den får riktige svar. Dette er en slags mellomting av veiledet læring og læring uten tilsyn. Eksempler kan være å lære å spille et brettspill eller en robot som skal lære å gå. Hver gang en agent utfører en handling får den en belønning eller en straff, basert på hvor ønskelig resultatet av handlingen er. For eksempel, når en agent trenes til å spille et brettspill får den en positiv belønning ved seier, og en negativ belønning ved tap. Alle andre tilfeller gir ingen belønning.

2.2 State-of-the-art

For å få en bedre forståelse av hva maskinlæring er og hva det er i stand til ble det gjort et litteratursøk for å kartlegge state-of-the-art innen kunstig intelligens. Det kan med fordel ses på anvendelser som har adoptert konseptet for å gi et inntrykk av hvor utbredt maskinlæring er, og hva det er i stand til å utrette i dag.

2.2.1 Autonom bil

STANLEY er en Volkswagen Touareg utstyrt med kamera, radar og avstandsmåling ved hjelp av laser for å oppfatte omgivelsene, og programvare for å styre retning, bremsing og akselerasjon. Bilen ble utviklet for høyhastighets ørkenkjøring uten menneskelig påvirkning, der robotens programvare

hovedsakelig var basert på maskinlæring. STANLEY vant "The Grand Challenge" arrangert av Defence Advanced Research Projects Agency (DARPA) i 2005 for å sette i gang innovativ forskning av ubemannet navigasjon med kjøretøy (Thrun et al, 2006)(Russel & Norvig, 2010)

2.2.2 Søppelpost-filter

Ved hjelp av et søppelpost-filter blir milliarder av e-poster klassifisert som søppelpost hver dag. Dette gjør at mottakeren sparer tid ved å slippe å slette mange uønskede e-poster. Ettersom ulike taktikker for å komme gjennom spam-filtre stadig blir mer sofistikerte, ville ikke dette vært mulig uten selvlærende algoritmer som kan kjenne igjen mønstre i søppelpostens innhold og mottakerens preferanser, og tilpasse seg informasjonen fortløpende. (Russel & Norvig, 2010)

2.2.3 Virtuell organisme

OpenWorm Project (OWP) er et prosjekt som er dedikert til å lage verdens første virtuelle organisme i en datamaskin. Organismen det jobbes med er en nematode kalt *Caenorhabditis elegans* (*C. elegans*). *C. elegans* har en veldig enkel hjerne som opererer med 302 nevroner, sammenliknet med menneskehjernen som har mellom 80-100 milliarder. Takket være denne enkelheten har man generelt en god forståelse av hvordan disse nevronene er koblet sammen og kommuniserer med hverandre. Denne kunnskapen har gjort det mulig å lage et dataprogram som simulerer hjernen til ormen. OWP har tatt denne kunnskapen et steg lengre og lastet opp programmet til en LEGO-robot. Roboten har en rekke sensorer som representerer ormens sanseinntrykk som gir den informasjon om sine omgivelser. Denne implementeringen viste at programmet var i stand til å omdirigere roboten på samme måte som ormen ville gjort for å unngå den samme hindringen, helt uten forhåndsprogrammering. Dette er foreløpig ny forskning, men videre i forskningen vil roboten være i stand til å utføre flere oppgaver og bedre kopiere ormens evne til å bevege seg og finne mat. Målet er å bygge en robot som er en komplett kopi av ormen, helt ned til syntetiske muskler i de faktiske lokasjonene i ormens kropp. (Shadbolt, 2015)

2.2.4 Connected grid, selvlærende overvåkning

Connected Grid er en løsning som er under utvikling av det norske selskapet eSmart. Løsningen overvåker strømmettet og gir informasjon om strømforbruket til forbrukerne. Systemet gjør det mulig å måle og analysere alt på en ny måte som innebærer at strømleverandøren bedre kan utnytte sine ressurser. Connected grid gir full oversikt over hvor mye strøm som brukes når, for intelligent å kunne tilby akkurat så mye som er nødvendig inn i hjemmet. Dette muliggjøres ved "maskinlæring", som baserer seg på at jo mer informasjon systemets enheter får, jo smartere blir det. Motivasjonen bak dette er det faktum at strøm som produseres ikke kan lagres i stor skala, kun brukes fortløpende. Ved å

utnytte informasjonen om forbruket kan strøm-leverandørene flytte energien der det trengs, for eksempel om morgenen når de fleste dusjer, og på samme måte spare på ressursene når varmtvannsbeholderen ikke er i bruk.

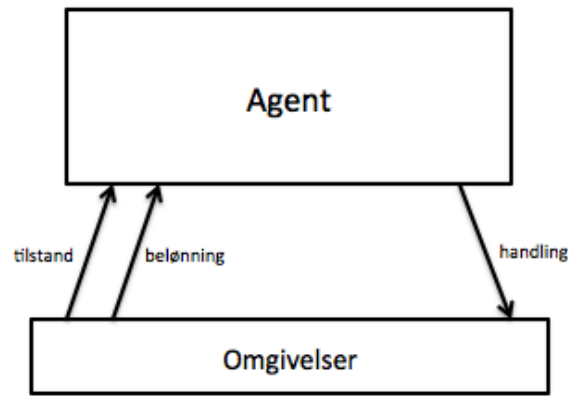
Det overordnede målet er at strømleverandører i større grad unngår å investere for mye eller for lite på feil tidspunkt, noe som kan være svært dyrt, både for forbruker og leverandør.

En del av Connected grid er Connected Drone, som er spesialdesignet for å inspisere strømmettet ved hjelp av høyoppløste kameraer, sensorer for navigasjon og termokamera som oppdager varme. Dronen brukes til å finne feil i elektrisk utstyr, og melde ifra om råtne trær som står i fare for å falle over strømsolper. Dronen er altså smart nok til å skille mellom råtne og sunne trær. Dronen er også selvlærende, og høster informasjon hver gang den er ute på oppdrag. Jo mer data den samler inn, jo smartere blir dronen. (Bie, 2015)

2.2.5 Q-Learning

Q-læring (eng.: Q-learning) er omtalt som en av de best kjente og mest brukte algoritmene innen forsterkningslæring, og baserer seg på å lære av erfaring. Metoden ble først introdusert av C. J. Watkins i 1989 og er en primitiv form for læring, men er allikevel et godt grunnlag for mer sofistikerte systemer (Chen, 2006).

Algoritmen går ut på at en agent utforsker sine omgivelser, der omgivelsene er representert av et antall tilstander (eng.: states). Agenten eksperimenterer med omgivelsene ved å utføre handlinger den er i stand til å utføre, for deretter å observere konsekvensen av denne handlingen. Konsekvens er i dette tilfellet i form av en belønning (eng.: reward). Belønningen er en vektet sum av forventningsverdien av alle fremtidige stegs belønninger. Hensikten med denne aktiviteten er å maksimere den akkumulerte belønningen over tid ved å velge den handlingen i en spesifikk tilstand som gir høyest belønning. Handlingen som gir den høyeste langsiktige belønningen er å anse som den optimale handlingen. Når agenten har utført alle handlinger i alle tilstander nok ganger vil den på denne måten lære seg et optimalt handlingsmønster, som er det overordnede målet (Marsland, 2009).



Figur 2-1 - Illustrasjon av prinsippet for tilstand, handling og belønning.

Figur 2-1 er hentet fra bok "Machine Learning" av Tom Mitchell, s. 368. Copyright 1997, McGraw-Hill Science/Engineering/Math. Oversatt til norsk.

Aktiviteten nevnt ovenfor kan representeres som en sekvens av tilstand-handling-belønning:

$$s_0 \xrightarrow{a_0} \{s_1, r_1\} \xrightarrow{a_1} \{s_2, r_2\} \xrightarrow{a_2} \{s_3, r_3\} \dots \xrightarrow{a_n} \{s_{n+1}, r_{n+1}\}$$

Figur 2-2 – Illustrasjon av agentens sekvens

Dette betyr at agenten var i tilstand s_0 , utførte handling a_0 som resulterte i at den mottok belønning r_1 og endte opp i tilstand s_1 . Videre utførte den handling a_1 , mottok belønning r_2 , og endte opp i tilstand s_2 , og så videre.

Denne sekvensen er bygd opp av erfaringer der erfaring er gitt som

$$s \xrightarrow{a} \{s', r\}$$

Figur 2-3 – Illustrasjon av agentens erfaring

der

s = agentens nåværende tilstand

a = valgt handling i tilstand s

s' = neste tilstand

r = belønningen agenten får fra handlingen

Erfaringen forteller at agenten var i tilstand s , utførte handling a , mottok belønning r , og endte opp i tilstand s' , og representert ved $\langle s, a, s', r \rangle$. (Poole, D. & Macworth, A. 2010)

2.2.6 Q-funksjonen

For at agenten skal kunne dra lærdom av de omtalte sekvensene og dermed kalle det en erfaring, har den en tabell kalt Q-tabell (eng.: Q-table) som fungerer som dens hukommelse. Samtlige datapunkter lagret i denne tabellen kalles Q-verdier (eng.: Q-values) og representerer hvor ønskelig det er å utføre en spesifikk handling i en spesifikk tilstand (Teknomo, 2005)

Én erfaring tilfører ett datapunkt $Q(s,a)$ i tabellen som representerer agentens nåværende estimat av den optimale Q-verdien. Det er denne informasjonen agenten benytter for å lære seg et optimalt handlingsmønster. Størrelsen på tabellen avhenger av hvor mange tilstander og handlinger som inngår i problemet man prøver å løse, der antall tilstander gir antallet rader, mens antall handlinger gir antallet kolonner. Dersom man for eksempel har 20 tilstander og 3 handlinger vil man få en tabell på 20x3 (Poole, D. & Macworth, A. 2010).

Før agenten begynner å eksperimentere, vet den ingenting annet hvilke handlinger den er i stand til å utføre, og Q-tabellen er følgelig tom. Det vil si alle Q-verdier er lik 0.

Nøkkelen til metoden beskrevet over er å oppdatere Q-verdien som tilføres agenten når den utfører en handling i en gitt tilstand i det aktuelle datapunktet når den får en erfaring.

Denne verdien er gitt ved Q-funksjonen

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

eller mer oversiktlig:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a')] \quad (2)$$

der

$$\begin{aligned} \alpha &: \text{Læringsrate} = [0,1] \\ \gamma &: \text{Diskonteringsfaktor} = [0,1] \\ r &: \text{Belønning} \end{aligned}$$

Læringsraten dikterer hvor mye av tidligere tilegnet lærdom som skal erstattes med ny informasjon. Ved $\alpha = 1$ erstattes tidligere verdier med ny informasjon, mens $\alpha = 0$ tilsvarer ingen oppdatering. Med andre ord vil agenten ved $\alpha = 1$ anta at den siste belønningen og resulterende tilstand er representative for fremtidige verdier.

Diskonteringsfaktoren indikerer vekten av alle fremtidige stegs belønninger. Dersom $\gamma = 0$ vil agenten kun vurdere nåværende belønninger, mens den vil tenke mer langsiktig og strebe etter høyere fremtidige belønninger etterhvert som γ nærmer seg 1.

Belønningen blir definert når man lager programmet i form av belønningsfunksjoner, og kan være positiv eller negativ. Hva denne belønningsverdien blir definert som er ikke kritisk. Det er derimot viktig at det er et tydelig skille mellom belønningen for gode handlinger og belønningen for dårlige handlinger (Teknomo, 2005).

Når algoritmen blir tilført en ny erfaring estimeres en ny Q-verdi, og Q-tabellens gamle verdi for siste erfaring blir oppdatert med den nye.

Man kan med fordel illustrere ved et eksempel hvordan tilstandstabellen blir oppdatert:

s_1	s_3	s_5 KATT
s_2	s_4	s_6 OST

Figur 2-4 - Eksempel på tilstandsrom

Anta at agenten er en mus som blir plassert i et rom delt inn i seks tilstander, se figur 2-4. I tilstand s_6 ligger det en ostebit, mens i tilstand s_5 befinner det seg en katt. Dersom agenten finner ostebiten mottar den en belønning på 10, mens den mottar en belønning på -10 dersom den beveger seg til tilstanden hvor katten befinner seg. Disse to tilstandene kalles terminaltilstander. Agenten utforsker omgivelsene helt til en av terminaltilstandene er oppnådd, før en ny iterasjon (forsøk) startes.

Agenten kan utføre fire handlinger: opp, ned, høyre, venstre. Dersom den beveger seg i en retning der det finnes en vegg får den en belønning på -1, og den forblir i samme tilstand. Alle andre tilstander har en verdi lik 0 (Poole, D. & Macworth, A. 2010). Dette implementeres i algoritmen ved hjelp av belønningsfunksjoner:

$$r_1 = \begin{cases} +10 & \text{dersom agenten finner osten} \\ -10 & \text{dersom agenten blir spist av katten} \\ -1 & \text{dersom agenten går i en vegg} \end{cases} \quad (3)$$

Agenten har ingen informasjon om sine omgivelser annet enn at det finnes seks tilstander og fire handlinger, hvilken tilstand den til enhver tid befinner seg i og eventuelle belønninger den får etterhvert som handlinger blir utført. Den vet heller ikke hva en belønning eller straff er. Q-tabellen er illustrert i tabell 1.

Ved å anta følgende sekvenser av erfaringer, $\langle s, a, r, s' \rangle$ kan man illustrere hvordan en sekvens oppdaterer Q-tabellen:

$$s_1, ned, 0, s_2 \rightarrow s_2, venstre, -1, s_2 \rightarrow s_2, høyre, 0, s_4 \rightarrow s_4, opp, 0, s_3 \rightarrow s_3, høyre, -10, s_5$$

For illustrasjonens skyld er læringsraten α og diskonteringsfaktoren γ satt lik 1 og alle Q-verdier har en startverdi lik 0.

Tabell 1 - Sekvens av erfaringer

s	s ₁	s ₂	s ₂	s ₄	s ₃
a	ned	venstre	høyre	opp	høyre
r	0	-1	0	0	-10
s'	s ₂	s ₂	s ₄	s ₃	s ₅
Ny Q(s,a)	Q[s ₁ ,ned]=0	Q[s ₂ ,ven]=-1	Q[s ₂ ,høy]=0	Q[s ₄ ,opp]=0	Q[s ₃ ,høy]=-10

Etter denne sekvensen av erfaringer vil Q-tabellen med oppdaterte verdier se slik ut:

S={s ₁ ,s ₂ ,s ₃ ,s ₄ ,s ₅ ,s ₆ }	s ₆	0	0	0	0
	s ₅	0	0	0	0
	s ₄	0	0	0	0
	s ₃	0	0	0	-10
	s ₂	0	0	-1	0
	s ₁	0	0	0	0
		opp	ned	venstre	høyre
		A={opp,ned,venstre,høyre}			

Figur 2-5 - Tilstandstabell med tilhørende Q-verdier

Hver erfaring vil oppdatere tilstandstabellens verdi for den utførte kombinasjonen av tilstand og handling, og Q-verdiene vil over tid konvergere til optimale verdier. Jo flere forsøk agenten gjennomfører, jo bedre blir estimatene i Q-tabellen. Agenten vil i teorien til slutt oppnå en optimalisert Q-tabell, og vil være i stand til å velge den korteste veien til osten hver gang, uavhengig av start-tilstand.

2.3 Utforsking eller utnytting

En av utfordringene med Q-læringsalgoritmen er at den ikke direkte forteller hva agenten skal gjøre, men heller fungerer som et verktøy som viser agenten hva den optimale handlingen er i en gitt tilstand. Dette er en utfordring fordi agenten må utforske miljøet nok ganger for å kunne bygge et solid grunnlag som gir et godt estimat av Q-verdiene. For å løse dette implementeres det en utforskningsfunksjon som bestemmer den strategi agenten skal bruke for å velge handlinger, enten utforske (eng.: explore) eller utnytte (eng.: exploit) (Poole, D. & Macworth, A. 2010).

- Utforskning: Agenten er dristig og velger ikke nødvendigvis den beste handlingen, med hensikt om å etablere et bedre estimat av Q-verdiene.

- Utnytte: Agenten utnytter erfaringen den allerede har bygd opp, og velger den optimale handlingen for tilstanden den er i. Altså den handlingen som gir høyest Q-verdi. Man sier at agenten er grådig

Hensikten med denne funksjonen er å fastslå et forhold mellom de to strategiene. Det er viktig å utforske nok slik at agenten bygger et solid grunnlag av Q-verdiene, men det er også viktig å utnytte den allerede tilegnede kunnskapen for å sikre høyest mulig belønning.

Det finnes en rekke måter å realisere dette på, men to av de mest populære metodene er ϵ -grådig funksjonen (eng.: ϵ -greedy) og Boltzmann fordeling, populært kalt "softmax":

2.3.1 ϵ -grådig

ϵ -grådig er en strategi som baserer seg på å alltid velge den optimale handlingen bortsett fra ϵ av gangene og velge en tilfeldig handling ϵ av gangene.

$$a = \begin{cases} a^* \\ a_r \end{cases} \quad (4)$$

der

a^* er optimal handling med sannsynlighet $1 - \epsilon$

a_r er tilfeldig handling med sannsynlighet ϵ

hvor

$$0 < \epsilon < 1 \quad (5)$$

Det er mulig å endre ϵ over tid. Dette gjør det mulig å få agenten til å velge en mer tilfeldig strategi i begynnelsen av læringsprosessen, for deretter å bli mer grådig ettersom den tilegner seg mer kunnskap.

En av utfordringene med ϵ -grådig er at den anser alle handlinger bortsett fra den optimale som likeverdige. Dersom det i tillegg til en optimal handling finnes to gode handlinger samt ytterligere handlinger som ser mindre lovende ut, vil det for agenten være fornuftig å velge en av de to bedre i stedet for å bruke tid på å utforske de dårlige handlingene. Ved bruk av ϵ -grådig er det like stor sannsynlighet for å utforske en dårlig handling som å utforske en av de to bedre. En måte å løse dette på er å bruke Boltzmannfordeling (Poole, D. & Macworth, A. 2010).

2.3.2 Boltzmannfordeling

Denne strategien er kjent som "softmax" handlingsvalg og går ut på å velge en handling med en sannsynlighet som avhenger av Q-verdien. Sannsynligheten for å velge handling a i tilstand s er

proporsjonal med $e^{\frac{Q(s,a)}{T}}$, som vil si at agenten i tilstand s velger handling a med sannsynlighet

$$\frac{e^{\frac{Q(s,a)}{T}}}{\sum_{\text{alle } a} e^{\frac{Q(s,a)}{T}}} \quad (6)$$

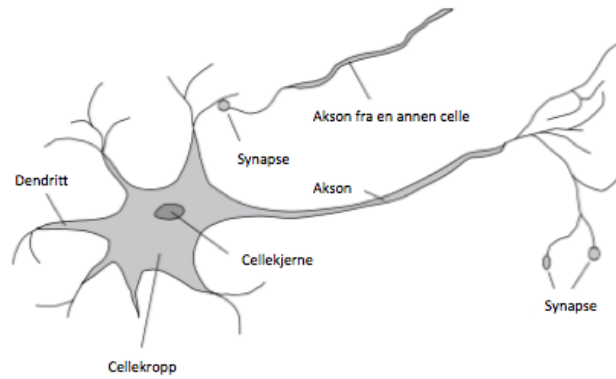
Parameteren T spesifiserer hvor tilfeldig verdier skal bli valgt. Når T har høy verdi blir handlinger valgt i tilnærmet likt omfang. Etterhvert som T reduseres er det mer sannsynlig at handlingene med høyest Q -verdi blir valgt, mens den beste handlingen alltid blir valgt når $T \rightarrow 0$ (Restelli, 2015).

En av fordelene med Q -læringsalgoritmen er at den er utforskingsufølsom (eng.: exploration insensitive). Det vil si at Q -verdiene vil konvergere til optimale verdier uavhengig av agentens oppførsel så lenge informasjon blir samlet inn. Dette er en av årsakene til at Q -læring er en av de mest anvendte og mest effektive metodene innen forsterkningslæring. Det er derimot en forutsetning at alle kombinasjoner av tilstand/handling blir prøvd ut mange nok ganger, noe som i de fleste tilfeller gjør at utforsking- og utnyttelsesaspektet må adresseres da det ikke alltid er mulig for en agent å eksperimentere uendelig antall ganger (P. Kaelbling & Littman, 1996).

2.3.3 Kunstig nevral nettverk (eng.: Artificial Neural Network)

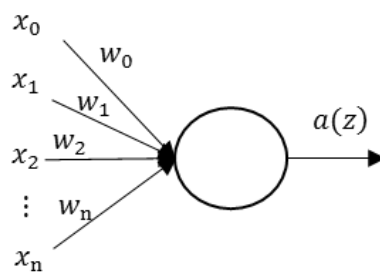
Nevrale nettverk (NN) er et konsept som er utviklet med inspirasjon fra biologien, mer spesifikt hvordan hjernen i mennesker og dyr fungerer. Ettersom læringen foregår i hjernen, er tanken at man kan komme langt ved å forsøke å kopiere den. For å forstå helheten i nevrale nettverk kan man med fordel se på en forenklet forklaring av hvordan hjernen faktisk fungerer.

Menneskets hjerne består av cirka 100 milliarder nerveceller kalt nevroner. Hvert nevron er koblet sammen med tusenvis av andre nevroner og kommuniserer via elektrokjemiske signaler. Signaler som kommer inn i et nevron er mottatt via koblinger kalt synapser, som igjen er lokalisert i enden av grener kalt dendritter. Se figur 2-6. Et nevron mottar kontinuerlig signaler gjennom disse inndata-terminalene. Nevronet tar deretter alle signalene den mottar og summerer dem. Dersom denne summen er større enn en gitt terskelverdi, sender nevronet et signal gjennom en utdata-terminal kalt akson, som er koblet til ett eller flere andre nevrons dendritter. Nevrale nettverk er en forenklet versjon av dette konseptet med et nettverk av kunstige nevroner (Russel & Norvig, 2010). Et kunstig nevron er illustrert ved figur 2-7. All teori beskrevet fra dette punktet er basert på Nielsen, 2015 og Ng et al., 2015.



Figur 2-6 - Illustrasjon av et naturlig nevron

Figur 2-6 er hentet fra bok "Artificial Intelligence A Modern approach" av Russel & Norvig, 2010, s. 11.
Copyright 2010 Pearson Education, Inc. Oversatt til norsk



Figur 2-7 - Illustrasjon av et kunstig nevron med inndata \vec{x} , vektor \vec{w} og utdata a

$$\text{Inndata : } \vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ Vekt : } \vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \text{Utdata : } a(z)$$

Et kunstig nevron(node) er en beregningsenhet med ett eller flere inndata-verdier $x_1, x_2, x_3, \dots, x_n$ representert som en vektor \vec{x} , der x_0 er bias. Bias har en konstant verdi, ofte lik 1. Dersom en bias ikke er inkludert, kan det medføre en begrensning av læringen i det nevrale nettverket. For hver inndata x_1, x_2, \dots, x_n er det en tilhørende vekt w_1, w_2, \dots, w_n , representert ved \vec{w} . Etterhvert som en verdi entrer noden blir den multiplisert med sin vekt. Det er disse vektene som justeres ved trening av nettverket. Noden summerer alle disse input-verdiene og kan skrives som z .

$$z = w_1x_1 + w_2x_2 + w_3x_3 \dots + w_nx_n = \vec{w} \cdot \vec{x} \quad (7)$$

z er produktet av vekten \vec{w} og inndata \vec{x} .

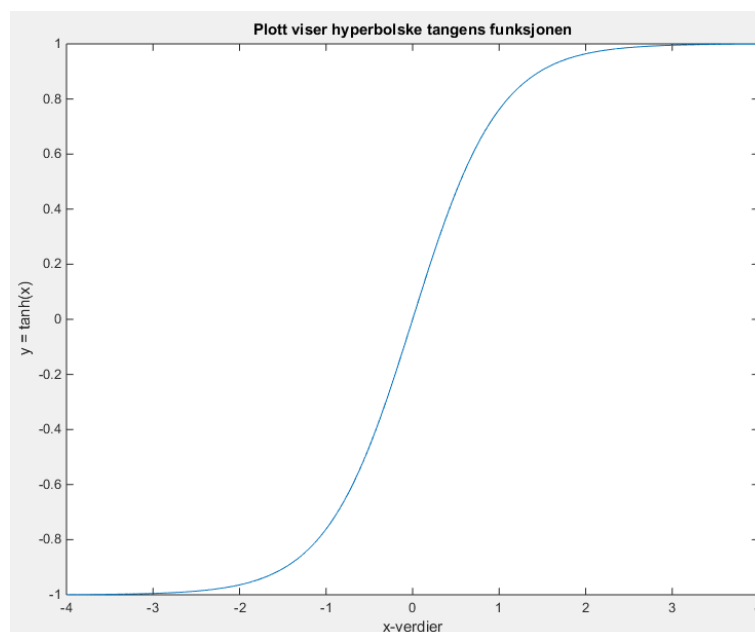
Utdata-verdien $a(z) = g(\vec{w} \cdot \vec{x})$, der g er en aktiveringsfunksjon gitt ved den hyperbolske tangens funksjonen $g(z)$.

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (8)$$

Utdata er dermed definert som $a(z)$.

$$a(z) = g(\vec{w} \cdot \vec{x}) = \tanh(\vec{w} \cdot \vec{x}) \quad (9)$$

Hyperbolske tangens funksjonen har følgende plot, her illustrert ved hjelp av MATLAB i figur 2-8.

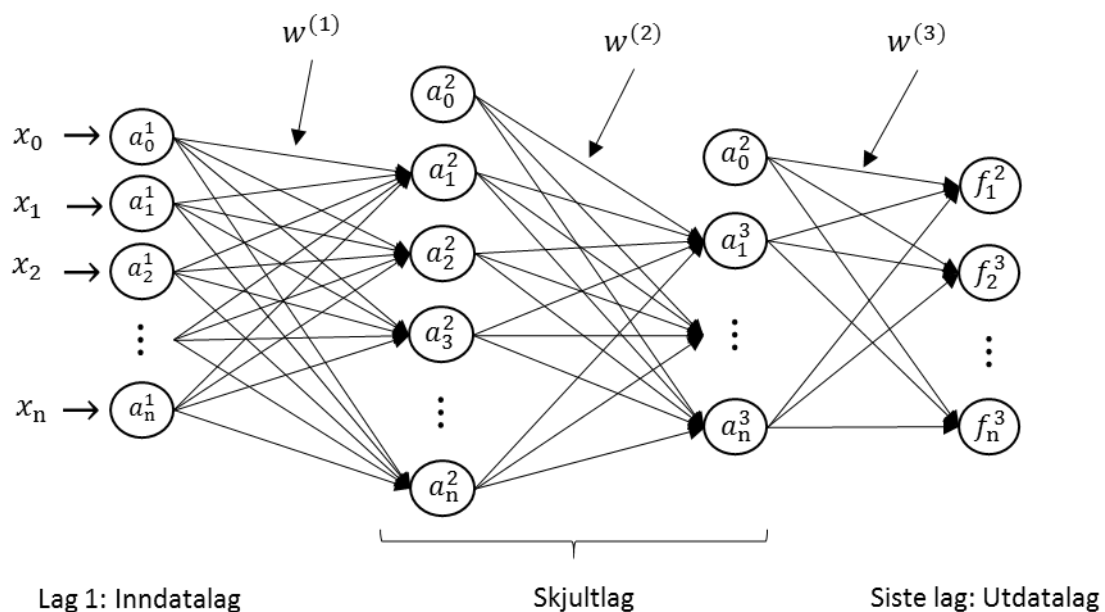


Figur 2-8 – Plott av aktiveringsfunksjonen $\tanh(x)$

Aktiveringsfunksjonen $\tanh(z)$ begrenser utdataene til å ligge i intervallet $[-1,1]$.

Med aktivering mener man den verdien z som blir beregnet i noden ved hjelp av aktiveringsfunksjonen og sendt videre som utdata. Denne verdien er i seg selv lite nyttig. Det er først når noder blir koblet sammen til et kunstig nevralt nettverk man kan dra nytte av et nevralt nettverks egenskaper.

Figur 2-9 er et eksempel på et kunstig nevralt nettverk. Nettverket består av flere lag, herunder inndatalag, skjulte lag og utdatalag. Bruken av begrepet skjult har ingen spesiell betydning annet enn at det hverken er et inndatalag eller et utdatalag, og det faktum at man ikke observerer verdiene representert i disse nodene. Antall noder i inndatalaget er lik antall inndata-verdier, mens antall noder i utdatalaget er lik antall utdata-verdier. Når det gjelder å avgjøre antallet skjulte lag/noder finnes det ingen fasit. Flere lag og noder vil derimot bidra til å skape et mer effektivt nettverk, men vil samtidig kreve ytterligere datakraft. (RSS2014: 07/16 09:00-10:00 Invited Talk: Andrew Ng (Stanford University): Deep Learning, 2014)



Figur 2-9 - Illustrasjon av et kunstig nevralt nettverk

Det finnes flere metoder for å sette opp et kunstig nevralt nettverk. En av de vanligste, som også er benyttet i dette prosjektet, kalles for et foroverkoblet nevralt nettverk (eng.: feedforward neural network), hvilket betyr at informasjonen alltid er matet fra inndata-siden og forsynt videre gjennom nettverket. Med andre ord benyttes utdata fra ett lag + bias som inndata i det neste.

Nodene i inndata-laget i nettverket er definert som \vec{x} og består av nettverkets inndata-verdier. Disse verdiene fungerer som inndata-verdier for det neste laget i nettverket, da gitt som \vec{a}^1 (se ligning (10)).

$$\vec{a}^1 = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (10)$$

x_0 er bias, og antallet noder er representert ved n.

Parameteren z fra ligning (8), er følgelig gitt som

$$\vec{z}^L = \begin{bmatrix} w_{1,0}^{(L-1)} a_0^{(L-1)} + w_{1,1}^{(L-1)} a_1^{(L-1)} + w_{1,2}^{(L-1)} a_2^{(L-1)} \dots + w_{1,n}^{(L-1)} a_n^{(L-1)} \\ w_{2,0}^{(L-1)} a_0^{(L-1)} + w_{2,1}^{(L-1)} a_1^{(L-1)} + w_{2,2}^{(L-1)} a_2^{(L-1)} \dots + w_{2,n}^{(L-1)} a_n^{(L-1)} \\ \vdots \\ w_{m,0}^{(L-1)} a_0^{(L-1)} + w_{m,1}^{(L-1)} a_1^{(L-1)} + w_{m,2}^{(L-1)} a_2^{(L-1)} \dots + w_{m,n}^{(L-1)} a_n^{(L-1)} \end{bmatrix} = w^{(L-1)} \cdot \vec{a}^{(L-1)} \quad (11)$$

Det bør presiseres at det ikke finnes noen \vec{z}^1 ettersom lag 1 er der inndata-verdiene \vec{x} mates inn i nettverket.

Parameteren m representerer antallet noder i lag L, mens n representerer antall noder + bias i lag L-1.

Vekten w^L er en matrise for styrken på koblingen mellom lag L og lag L+1, i motsetning til figur 2-7 hvor vekten var en vektor. Dimensjonen på vekmatrisen er $m \times n$, altså rader lik antall noder i lag L+1 og kolonner lik antall noder + bias i lag L, se figur 2-9.

Følgelig er utdatavektoren \vec{a}^L til et lag gitt som

$$\vec{a}^L = g(\vec{z}^L) = \begin{bmatrix} g(\vec{w}_1^{(L-1)} \cdot \vec{a}^{(L-1)}) \\ g(\vec{w}_2^{(L-1)} \cdot \vec{a}^{(L-1)}) \\ \vdots \\ g(\vec{w}_m^{(L-1)} \cdot \vec{a}^{(L-1)}) \end{bmatrix} \quad (12)$$

De estimerte verdiene i det siste laget er et resultat av beregningene gjort i de skjulte lagene, der utdata er lik $\vec{f} = g(z^{(\text{utdata} \text{ lag})}) = g(w^{(\text{siste skjulte lag})} \cdot \vec{a}^{(\text{siste skjulte lag})})$.

2.4 Kostnadsfunksjon

Som nevnt er det vektene w til koblingene mellom lagene som justeres for å trene nettverket. For å gjøre dette benyttes det en kostnadsfunksjon (eng.: cost function) definert ved:

$$C = \frac{1}{2} \|f - y\|^2 \quad (13)$$

Kostnadsfunksjonen gir avviket mellom nettverkets estimerte (ønskede) utdata-verdi f fra utdatalaget og den faktiske verdien y . Ettersom det er ønskelig at den faktiske verdien er så lik den estimerte verdien som mulig er målet følgelig at $C \approx 0$. Dette gjøres ved å beregne den partiellderivate $\frac{\partial C}{\partial w}$ av vektmatrisene ved å bruke tilbake-forplantnings algoritmen (eng.: backpropagation algorithm), for så å oppdatere vektmatrisene.

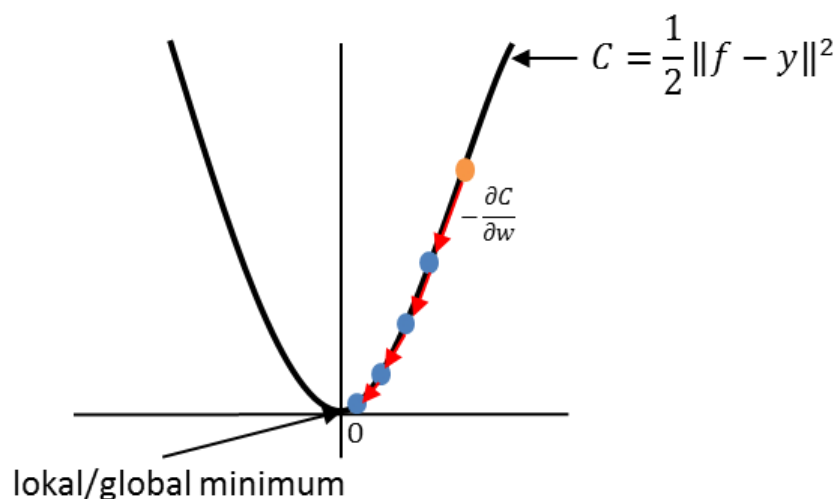
2.5 Tilbake-forplantnings algoritmen

For å finne de partiellderivate må man først finne avviket mellom noderes estimer og faktisk verdi i hver enkelt node i nettverket. Dette blir gjort med tilbake-forplantnings algoritmen.

Algoritmen begynner med å beregne avviket i det siste laget, og jobber seg bakover i nettverket for å beregne avviket for hver enkelt node i hvert enkelt lag (bortsett fra inndatalaget). Avviket er definert ved avviksvektor $\vec{\delta}^L$, og har størrelse lik antall noder+bias i laget L .

Etter man har funnet $\vec{\delta}^L$ for hvert enkelt lag brukes dem videre for å finne de partiellderivate av kostnadsfunksjonen. De partiellderivate brukes så til minimere kostnadsfunksjonen ved å finne et lokalt minimum ved gradient nedstigning.

Figur 8 viser prinsippet med gradient nedstigning på en enkel kvadratisk funksjon, hvor man starter fra et punkt på funksjonen og jobber seg ned til et lokalt minimum. De røde pilene på figuren viser hvert enkelt steg gjort av gradient nedstigningsalgoritmen. Ved å ta mange nok steg vil man til slutt nærme seg et lokalt minimum. Etter å ha beregnet de partiellderivate for hver vektmatrise med tilbake-forplantnings algoritmen brukes disse til å oppdatere vektmatrisene i det nevrale nettverket.



Figur 2-10 - Illustrasjon av gradient nedstigningsprinsippet

Fordi oppbygningen av et kunstig nevral nettverk varierer mye basert på problemet man forsøker å løse er fremgangsmåten nøyere forklart i kapittel 3.

2.6 Fuzzy Logikk

Teorien om fuzzy logikk er basert på Sunnevåg, K. J. (2007).

Fuzzy logikk er en beslutnings- eller styringsmodell basert på diffuse mengder, og inndata klassifisert som grader av medlemskap i en diffus mengde. En diffus mengde har ingen skarpe grenser, og er et sett med verdier som tilhører samme definisjon, hvor definisjonen er et diffust begrep gjerne beskrevet av et adjektiv. For eksempel: store poteter eller høye mennesker. En diffus mengde kalles også semantisk variabel eller sett.

Fuzzy logikk klassifiserer inndata etter hvor stort medlemskap inndataen har til et sett. Graden av medlemskap blir gradert mellom 0 og 1, og bestemt utifra en medlemskapsfunksjon.

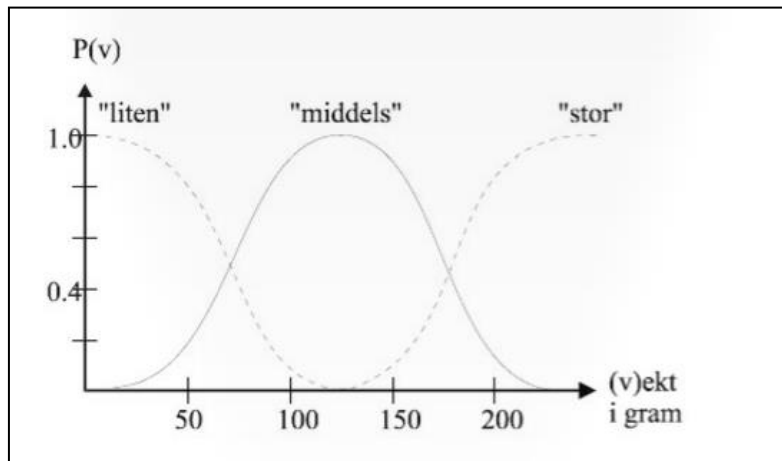
Medlemskapsfunksjonen bestemmes på bakgrunn av kunnskap om fagfeltet, og kan være ulineær.

Inndata kan ha medlemskap i flere ulike sett, men kan kun ha fullt medlemskap i ett sett.

Figur 2-11 viser et eksempel på en medlemskapsfunksjon med vekt på en potet som inndata.

Avhengig av vekten blir poteten gradert til de ulike diffuse mengdene: liten, middels og stor.

Eksempel på medlemskapsfunksjon

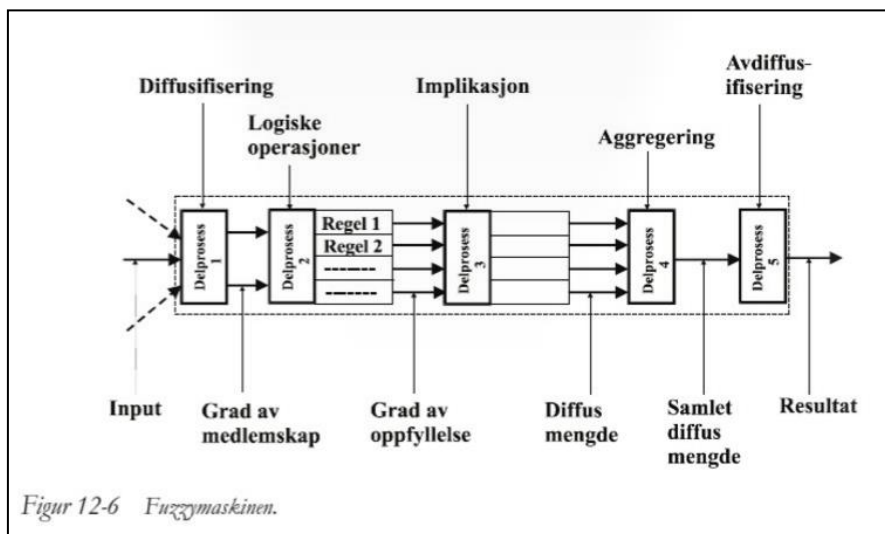


Figur 2-11 - Medlemskapsfunksjon

Figur 2-11 er hentet fra "Beslutninger på svakt informasjonsgrunnlag," av Sunnevåg, K. J., 2007, Concept rapport nr 17, s.245. Copyright 2007. Figuren er gjengitt uten tillatelse.

Fuzzy logikk metoden består i hovedsak av fem ledd som illustrert i figur 2-12.

2.6.1 Virkemåte



Figur 12-6 Fuzzymaskinen.

Figur 2-12 - Fuzzy logikk

Figur 2-12 er hentet fra "Beslutninger på svakt informasjonsgrunnlag," av Sunnevåg, K. J., 2007, Concept rapport nr 17, s.250. Copyright 2007. Figuren er gjengitt uten tillatelse.

Fuzzy logikk virkemåte

1. Diffusifisering: inndata graderes i henhold til de diffuserte mengdene, og de diffuserte mengdene sendes til delprosess 2.
2. Logiske regler: de diffuserte mengdene testes opp mot regler. Graden av oppfyllelsen fra hver regel sendes til delprosess 3. Reglene kan være implementert med {og, hvis, eller}, eller andre metoder som for eksempel en konsekvensmatrise. Reglene beskriver sammenhengen mellom de diffuserte inndataene og utdata.
3. Implikasjon: de mottatte data fra delprosess 2 testes opp mot regler som graderer innvirkningen på totalkonsekvensen for hver av de mottatte data. Graderingene blir så sendt til delprosess 4.
4. Aggregering: alle konsekvensmengdene blir sammenstillet til en aggregert mengde, som er unionen av mengdene fra delprosess 3.
5. Avdiffusering: Det mest representative element trekkes ut av den aggregerte mengde. Dette er som regel tyngdepunktet. Den verdien vil da være styringssignalet.

Fuzzy logikk metoden har en rekke fordeler ved at den evner å behandle upresise data og modellere ulineære sammenhenger mellom inndata og utdata. Prinsippet muliggjør en nøyaktig og virkelighetsnær klassifisering av data, ettersom data sjelden er kun sant eller usant.

3 Teknisk løsning

3.1 Agenten

Det ble utviklet to versjoner av agenten. Den første versjonen, som senere ble byttet ut, ble bygget med utgangspunkt i drivverk og mekanikk hentet fra «WLtoys» modell «A969» (figur 3-1 og figur 3-2). Den radiostyrte bilen i skala 1:18 ble levert kjøreklar med radio og batteri. Bilen målte 285x140x105mm (LxBxH) og kunne kjøre omtrent 10 minutter på fulladet batteri.



Figur 3-1 – WLtoys modell A969 med karosseri



Figur 3-2 - WLtoys modell A969 uten karosseri

Ettersom bilen skulle utstyres med en Xbee-modul og kommunisere via bluetooth, ble bilens opprinnelige mottaker fjernet. Styreservoene var tilpasset bilens mottaker og ble derfor byttet ut til fordel for en PWM-kontrollerbar servo. For å forbedre agentens kjøretid ble det originale batteriet (7,4V/1100mAh) erstattet med et batteri av typen “Fire Bull” på 7,4V/5200mAh, som teoretisk ville gi omtrent fem ganger så lang kjøretid.

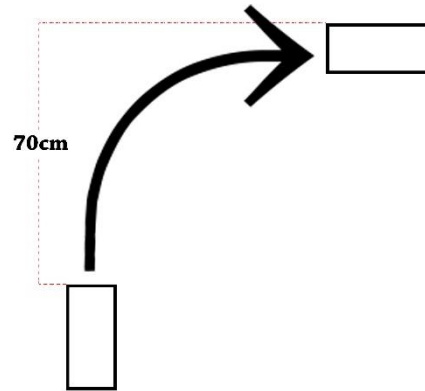
For å få plass til all elektronikken ble en plate av 1,5mm aluminium skåret ut og montert på toppen av bilen (figur 3-3). Motordriveren, mikrokontrollerkortet og Xbee-modulen ble plassert sammen med motoren og servoen på selve chassiset, mens batteriet, sensorene og spenningsregulatoren ble montert på den nye platen for enklere tilgang.

Etter nevnte endringer fikk agenten bedret batterivarighet, justerbar fart og riktig fysisk størrelse, men den viste seg å ikke ha god nok svingradius. Den hadde spesielt problemer i situasjoner der den måtte reagere i en retning, for deretter å bytte retning.



Figur 3-3 - Agent 1 med sensorrigg og topplate

Bilens svingradius ble testet før ombyggingen ble påbegynt, og det ble målt at bilen trengte 70 cm på å snu 90 grader mot venstre eller høyre (figur 3-4). Den trengte dermed et område på 140 cm i diameter på å snu 180 grader. Det ble derfor antatt at det var nødvendig med en rund arena med diameter på omtrent fire meter. Arenaen ble bygget, og det ble funnet ut at fire meter var nok til å kunne kjøre rundt uten hindere, men at det ble for lite når bilen måtte unngå objekter.



Figur 3-4 – Målt svingradius for agent versjon 1

Parallelt med programmeringen av den første agenten ble konstruksjon av en ny agent derfor påbegynt. Den nye agenten ble basert på en modell beskrevet i *Huang, Cao og Guo (2005)*, hvor agenten var utstyrt med to drivhjul og et støttehjul, og den endret retning ved å kjøre de to drivhjulene med ulik hastighet eller motsatt retning av hverandre.

3.1.1 Oppbygning

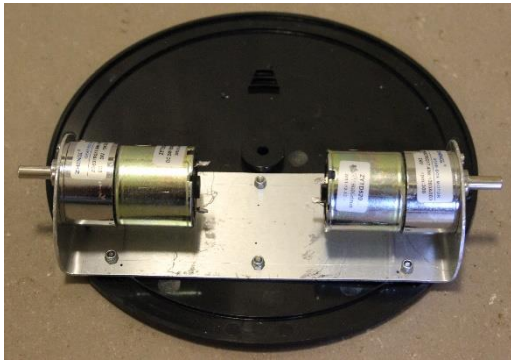
Agenten som til slutt ble brukt i demonstratoren er utformet med utgangspunkt i en plastplate med diameter på 21 centimeter. Festebraketten til motorene ble skåret ut av 1,5mm aluminium og bøyd til riktig fasong (figur 3-5). Deretter ble de to motorene montert (figur 3-6) før den ferdige braketten til slutt ble montert et lite stykke bak platens vippepunkt (figur 3-7). Deretter ble støttehjul, Arduino UNO, elektronisk fartsregulator, batteri, Xbee og de fem sensorene med tilhørende spenningsregulatortrinn som tidligere ble brukt på den første, flyttet over til den andre agenten (figur 3-8).



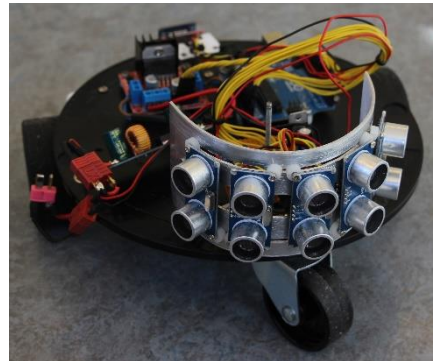
Figur 3-5 - Bøying av festebrakett for motorene



Figur 3-6 - Motorbrakett med begge motorene montert



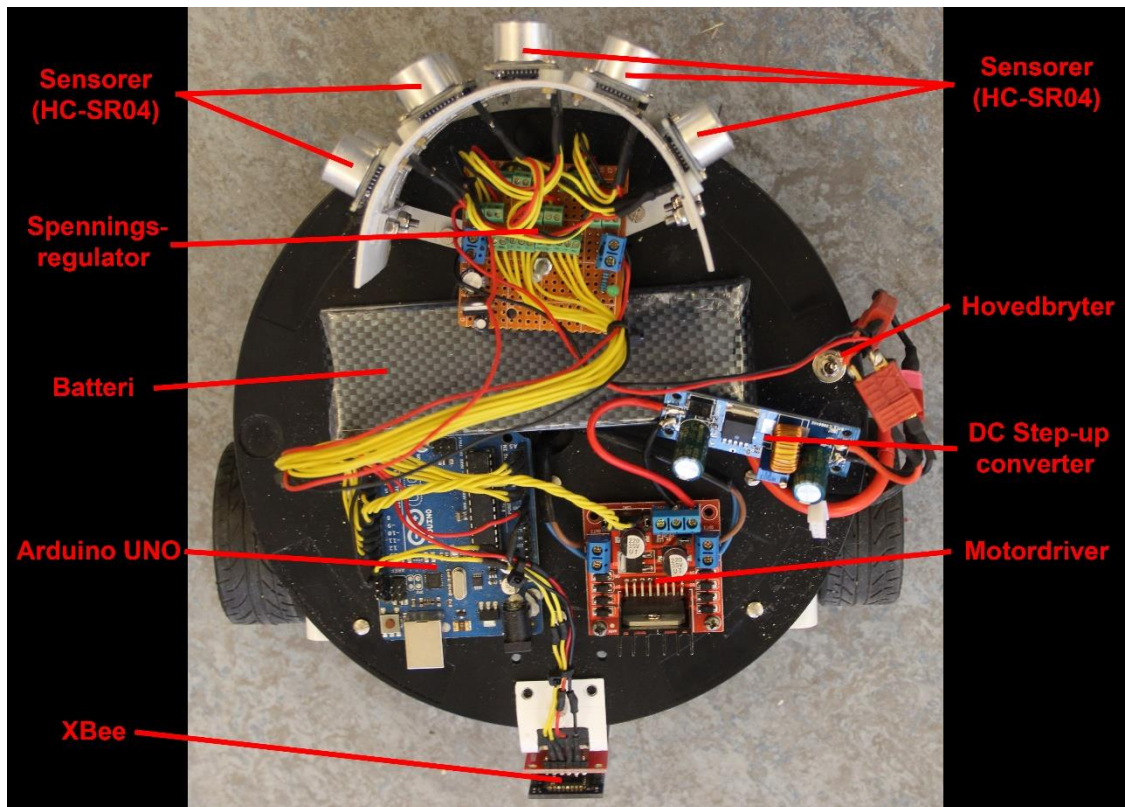
Figur 3-7 - Motorbrakett festet til platen



Figur 3-8 - Ferdig agent versjon 2

3.1.2 Elektronikk

De to agentene var utstyrt med samme elektronikk, med unntak av at første versjon brukte en servo for å styre, mens den andre endret retning ved hjelp av to motorer. Figur 3-9 viser en oversikt over agentens elektronikk. Fullstendig koblingsskjema for systemet vises i vedlegg A.




Figur 3-9 - Oversikt over agentens elektronikk

3.1.2.1 Sensorer

En del av konstruksjonen var å utstyre agenten med avstandssensorer som kunne gi agenten informasjon om omgivelsene. Avstandssensorer kommer i ulike typer for ulike formål og det var derfor viktig å finne ut hvilken type som best ville passe dette prosjektet. Det stod i hovedsak mellom IR-sensorer og ultrasoniske sensorer på grunn av sensorenes rekkevidde, pris og tilgjengelig

dokumentasjon . For å finne ut hvilken som var best egnet, ble spesifikasjonen for de to sensortypene sammenlignet (tabell 2).

Tabell 2 - Sammenligning av teoretiske data for IR-sensorer og ultrasoniske sensorer

Infrarød (IR) (lys) <i>Sharp GP2Y0A02YK0F</i>	Ultrasonisk (lyd) <i>HC-SR04</i>
	
<ul style="list-style-type: none"> - Lesbar avstand: 20cm – 150cm - Detekteringsvinkel: 10° - Driftsspenning: 4.5-5.5V DC - Utgangssignal: 0.3-2.8V - Strømforbruk: 33mA 	<ul style="list-style-type: none"> - Lesbar avstand: 2cm - 450cm - Detekteringsvinkel: 15° - Driftsspenning: 5V - Utgangssignal: 0-5V DC - Feilmargin: < 0.3cm - Strømforbruk: 15mA

Ut ifra sensorenes spesifikasjoner ble det klart at de ultrasoniske sensorene var best egnet for prosjektet, ettersom det ble antatt at det var nødvendig med avstandsmålinger over 150cm fra agenten.

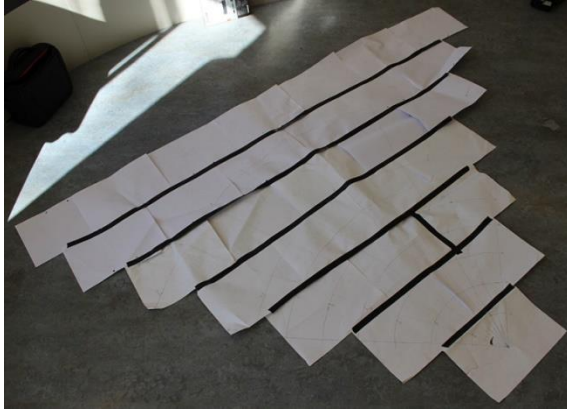
Mens IR-sensoren fra Sharp (GP2Y0A02YK0F) har en svært smal detekteringsvinkel og en lesbar avstand på mellom 20cm og 150cm, har HC-SR04 en lengre rekkevidde (2-400cm) og en noe større detekteringsvinkel. Artikler og bilder fra lignende prosjekter beskrev i tillegg at sistnevnte sensor kunne ha enda større detekteringsvinkel enn det som var oppgitt, og at denne typen sensorer vanligvis blir benyttet. For å unngå dødsoner, nærme agenten eller mellom sensorene, ble det valgt å bruke de ultrasoniske sensorene av typen HC-SR04.

3.1.2.2 Testing av sensorer

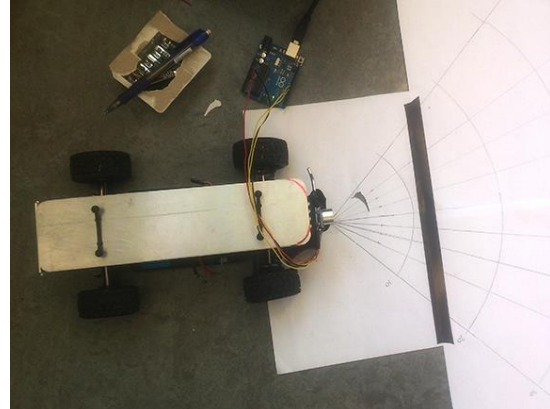
Det ble dermed kjøpt inn 10 ultrasoniske sensorer av typen HC-SR04 før det ble igangsatt testing av sensorene med spesielt fokus på lesbar avstand, feilmargin og detekteringsvinkel.

Testingen ble utført ved å tape sammen rekker av ark for å dekke en sektor på omkring 90° (figure 3-10). Denne sektoren ble igjen delt opp i mindre sektorer, hver på 10 grader, samt merket for hver tiende centimeter fra sektorens startpunkt. Deretter ble bilen påmontert en sensor 80mm fra bakken og plassert i dette punktet (figure 3-11). Et program som kontinuerlig leste og printet sensorens målinger i Arduinos seriell-monitor ble lagt inn på en Arduino UNO slik at det var mulig å observere resultatene.

Målingene ble utført ved at en pappeske ble ført inn langs sirkelbuen ved de ulike avstandene fra sektorens startpunkt. Punktet der sensoren først viste en stabil avstandsmåling til esken ble markert på arket sammen med eventuelle avvik.

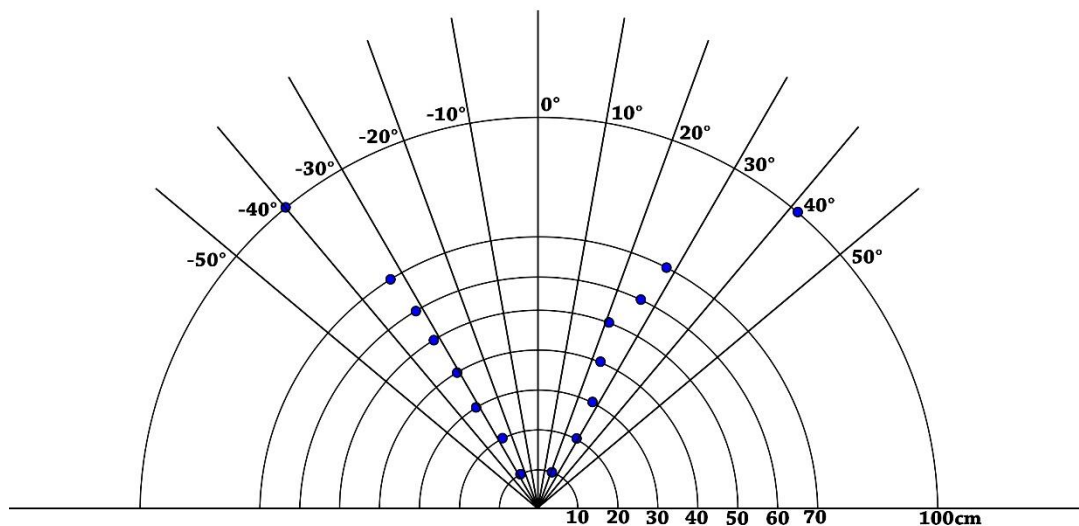


Figur 3-10 - Testmiljø for måling av sensorens detekteringsvinkel og avvik



Figur 3-11 - Testing av en sensor koblet til en Arduino UNO

Figur 3-12 viser resultatet av testen. Sensoren ga stabile målinger i en sektor på 50° - 60° mellom 10cm og 100cm fra sensoren. Opp til 70cm fra sensoren ble det avlest et avvik på inntil tre centimeter (ca 4%) ved hvert punkt. Ved 100cm fra sensoren økte derimot avviket til åtte centimeter (8%). For eksakte avleste verdier, se tabell 3. Testens resultater viste at sensoren hadde større detekteringsvinkel, men høyere avvik enn hva som var oppgitt.



Figur 3-12 - Målt detekteringsvinkel for HC-SR04

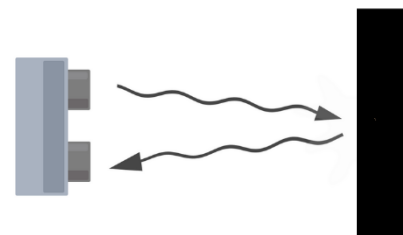
Tabell 3 - Målt avvik og detekteringsvinkel for HC-SR04

Avlest måling fra venstre[cm]	Detekteringsvinkel venstre [°]	Avstand [cm]	Detekteringsvinkel høyre [°]	Avlest måling fra høyre [cm]
10	-28	10	20	8-12
22	-28	20	28	21
33	-32	30	27,5	30-31
42-43	-30,5	40	23	40-42
52	-32	50	20,5	49-51
60	-32	60	26	59-60
71-72	-33	70	28	71-72
98-104	-40	100	41	98-106

3.1.2.3 Virkemåte

Et menneskeøre kan høre frekvenser på 20Hz-20kHz.

Ultrasonisk lyd er derimot i det ikke hørbare området med kortere lydbølger (40kHz). En ultrasonisk sensor består av en høyttaler og en mikrofon. Høyttaleren sender ut en lydbølge, mens mikrofonen lytter etter ekkoet av denne lyden (figur 3-13).



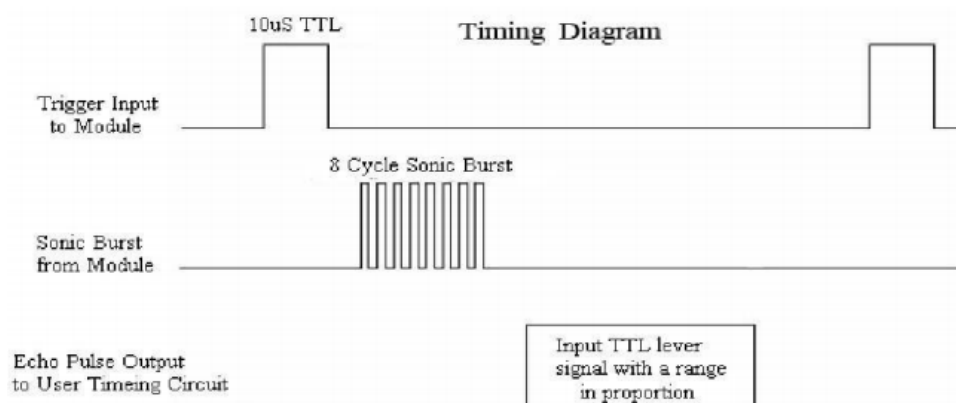
Figur 3-13 - Konsepttegning for HC-SR04

Basert på tiden fra lyden sendes, reflekteres og mottas, kan en avstand beregnes, ettersom man vet at lyden i luft beveger seg

340 meter i sekundet. For å få avstanden mellom sensor og objekt må tiden deles på to, ettersom lyden må bevege seg både frem og tilbake. Formelen blir dermed:

$$Avstand = \frac{Målt\ tid * Lydens\ hastighet}{2} \quad (14)$$

Tidsdiagrammet for HC-SR04 er vist nedenfor (figur 3-14). For å starte en måling må det sendes en 10us lang puls til triggerinngangen på sensoren. Denne gjør at modulen sender ut åtte sykluser av ultralyd ved 40kHz og samtidig setter echoutgangen høy. Når ekkoet av de åtte syklusene mottas, settes echoutgangen igjen lav. På denne måten blir ekkoets pulsbredde proporsjonal med avstanden.

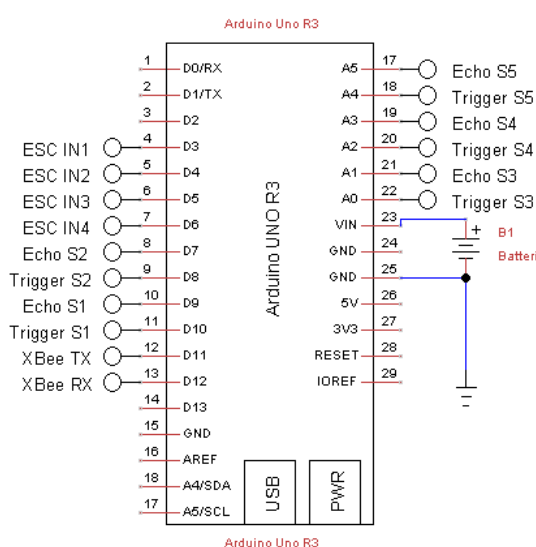


Figur 3-14 - Tidsdiagram for trigger/echo for HC-SR04

3.1.2.4 Mikrokontroller

Ettersom prosjektets mål var å utvikle en selvlærende algoritme og en studie rundt autonome systemer, ble det fokusert på å holde elektronikken så enkel som mulig. Prosjektet har benyttet seg av Arduino, som er en svært utprøvd og godt dokumentert plattform for prototyping av elektronikk. Deres modell Arduino UNO R3 er et mikrokontrollerkort med 14 digitale inn- og utganger i tillegg til seks analoge innganger. På mikrokontrolleren ligger det en del ferdige biblioteker som forenkler programmeringen for tilkobling av for eksempel sensorene og bluetoothmodulen benyttet i dette prosjektet.

Tabell 4 viser pinnekonfigurasjon for mikrokontrollerkortet benyttet i dette prosjektet. Figur 3-15 er ment som en illustrasjon av dataene fra samme tabell.



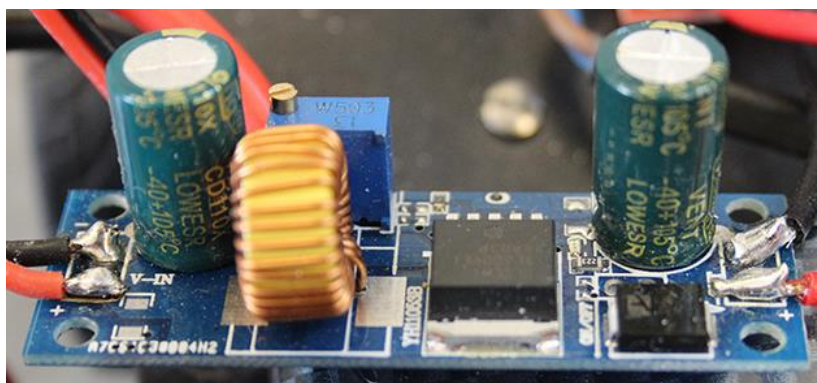
Figur 3-15 - Koblingsskjema for Arduino UNO R3

Pin	Funksjon
D4	ESC IN1
D5	ESC IN2
D6	ESC IN3
D7	ESC IN4
D8	Echo S2
D9	Trigger S2
D10	Echo S1
D11	Trigger S1
D12	XBee TX
D13	XBee RX
A5	Echo S5
A4	Trigger S5
A3	Echo S4
A2	Trigger S4
A1	Echo S3
A0	Trigger S3

Tabell 4 - Pinnekonfigurasjon for Arduino UNO R3

3.1.2.5 Elektroniske komponenter

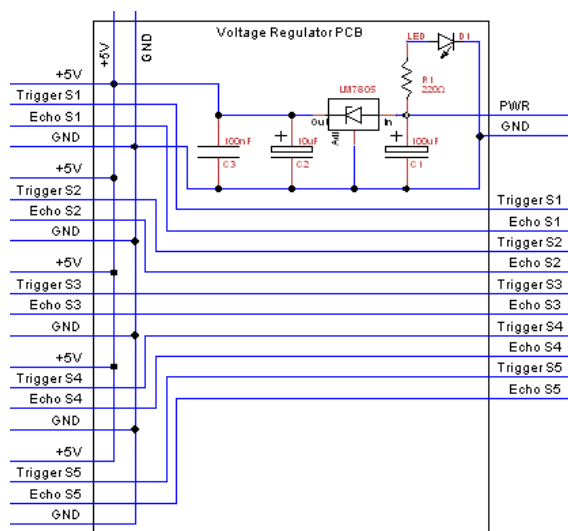
Agentens motorer av typen Zheng ZGB37RG17 var laget for 24 volt. Med et batteri på 7,4V var det derfor nødvendig med en DC step-up converter (figur 3-16) for å gi motorene korrekt spenning. En step-up converter kan ta en lav innspenning (4,5-32V) for så og avgi en høyere utspenning (5-42V). Utspenningen justeres ved å endre en variabel motstand. Ved hjelp av denne justeringsmuligheten kunne agentens maksfart både endres analogt og gjennom programmering. Det ble valgt en spenning på ca 9V, som ga agenten en hastighet på ca 3 km/t.



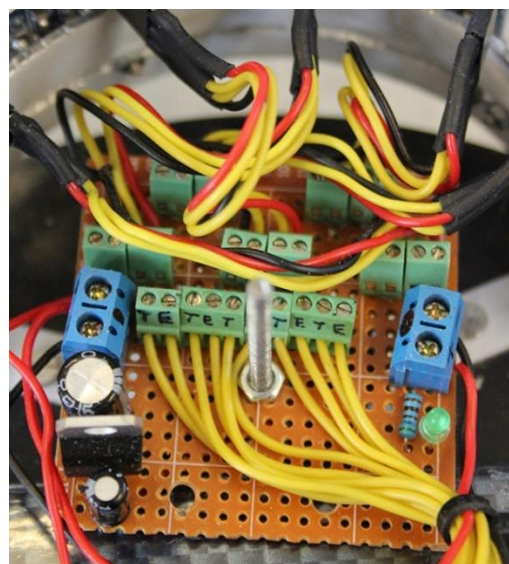
Figur 3-16 - Step-up converter (justerskrue øverst)

Arduino UNO har kun et visst antall inn- og utganger i tillegg til en strømbegrensning på 20mA per utgang. Hver sensor kobles til via fire ledninger (GND, +5V, trigger, echo). Med fem sensorer ble dette alene 20 inn- og utganger. For å redusere antall ledninger til mikrokontrolleren ble det derfor bestemt å holde GND og +5V utenom Arduino-kortet på et eget printkort ved å justere batterispenningen ned til 5V.

Regulatortrinn (koblingsskjema: figur 3-17, PCB: figur 3-18) består av en fem-volts spenningsregulator (LM7805), tre kondensatorer (100uF, 10uF, 100nF) og en LED for å indikere om kortet er spenningssatt. Trigger og echo er inkludert på kortet for å ha skrueterminaler for alle tilkoblede ledninger og forenkle eventuelle utskiftninger av defekte sensorer.



Figur 3-17 - Koblingsskjema for agentens regulatortrinn



Figur 3-18 - Ferdig 5V spenningsregulator

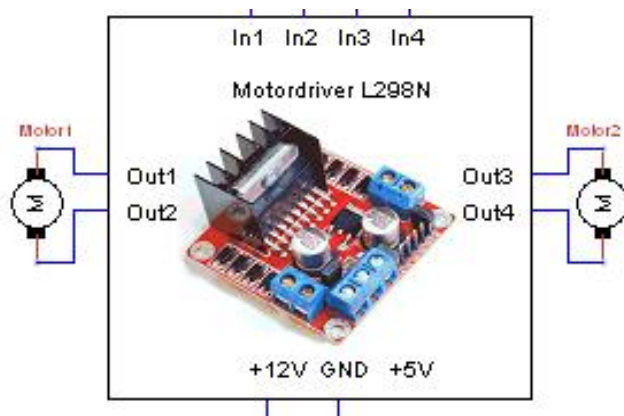
Motordriveren (figur 3-20) som ble valgt er basert på en dobbel h-bro (eng: h-bridge) av typen L298N. Motordriveren har fire utganger til de to motorene (1, 2, 3, 4), tre innganger for forsyningsspenning (12V, GND, 5V), og seks innganger som brukes til å angi retning på motorene. To av disse inngangene er Enable A og Enable B. Disse settes høy (+5V) for å slå på hver av de to motorutgangene. De siste fire inngangene er IN1, IN2, IN3, IN4. Logisk tabell for motordriveren både for motor 1 og motor 2 er vist i tabell 5 og tabell 6. IN1 og IN2 kontrollerer retningen på motor 1, mens IN3 og IN4 kontrollerer på samme måte motor 2. Figur 3-19 viser koblingsskjema for kortet.

Tabell 5 - Logisk tabell for motordriver L298N Dual H-bro for Motor 1

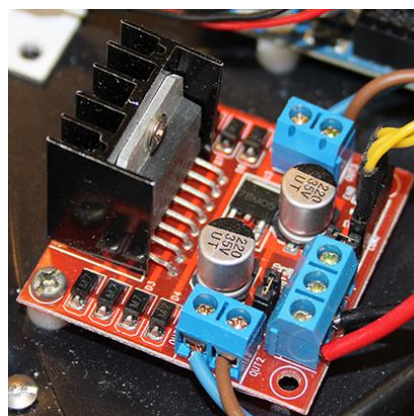
INPUT M1			OUTPUT (1,2)
ENABLE A	IN1	IN2	M1
1	0	0	Av
1	0	1	Forover
1	1	0	Revers
1	1	1	Fare!
0	X	X	Motorbrems

Tabell 6 - Logisk tabell for motordriver L298N Dual H-bro for Motor 2

INPUT M2			OUTPUT (3,4)
ENABLE B	IN3	IN4	M2
1	0	0	Av
1	0	1	Forover
1	1	0	Revers
1	1	1	Fare!
0	X	X	Motorbrems



Figur 3-19 - Koblingsskjema for motordriver



Figur 3-20 – Motordriver L298N Dual H-bro

3.2 Arenaen



Figur 3-21 – Bilde av arenaen

Arenaveggene (figur 3-21) er bygget i 1mm permpapp. Pappen ble levert i syv ark på 70x100cm som ble delt langsgående, for deretter å bli stiftet sammen. De 14 arkene ble stiftet sammen alle steder med unntak av ett for å kunne variere arenaens størrelse. Maksimal størrelse er en omkrets på ca 14 meter, eller en diameter på omkring 4 meter. Arenaens størrelse varieres ved å endre graden av overlapp i overgangen der pappen ikke er stiftet sammen.

3.2.1 Dynamiske hindringer

For å vise bilens evne til å tilpasse seg omgivelser i endring måtte det bygges et system hvor antall hindringer kunne endres og hvor hastighet og mønster på disse, kunne endres. For å begrense kompleksiteten av systemet ble det tatt utgangspunkt i et roterende hinder. Hinderet skulle rotere om en saktegående aksel med 1.5 meter lange firkanttrør i aluminium montert slik at hindringene kunne henge ned fra disse. Kravene var at vinkelen mellom armene og avstanden fra aksel til hinder kunne varieres når systemet stod stille. Dette for å kunne introdusere bilen for nye mønstre.

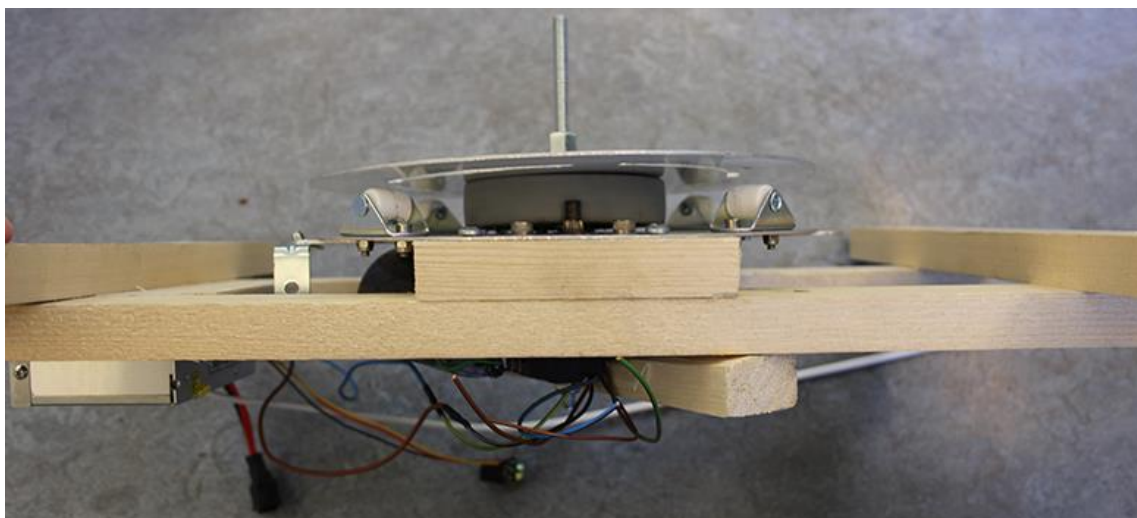


Figur 3-22 - System for dynamiske hindringer

Hinderets hastighet var viktig. Systemets rotasjonshastighet ble antatt å måtte kunne varieres med en hastighet fra 0 til omkring 30 r/min enten manuelt via et potensiometer, eller en forhåndsprogramert sekvens via en mikrokontroller. Under testing ble det funnet ut at spenningen motoren måtte ha før den begynte å rotere endret seg utifra vekten på det antallet hindere systemet skulle rotere, og at det derfor var enklere å utføre start og hastighetsregulering manuelt via et potensiometer fremfor et hardkodet program.

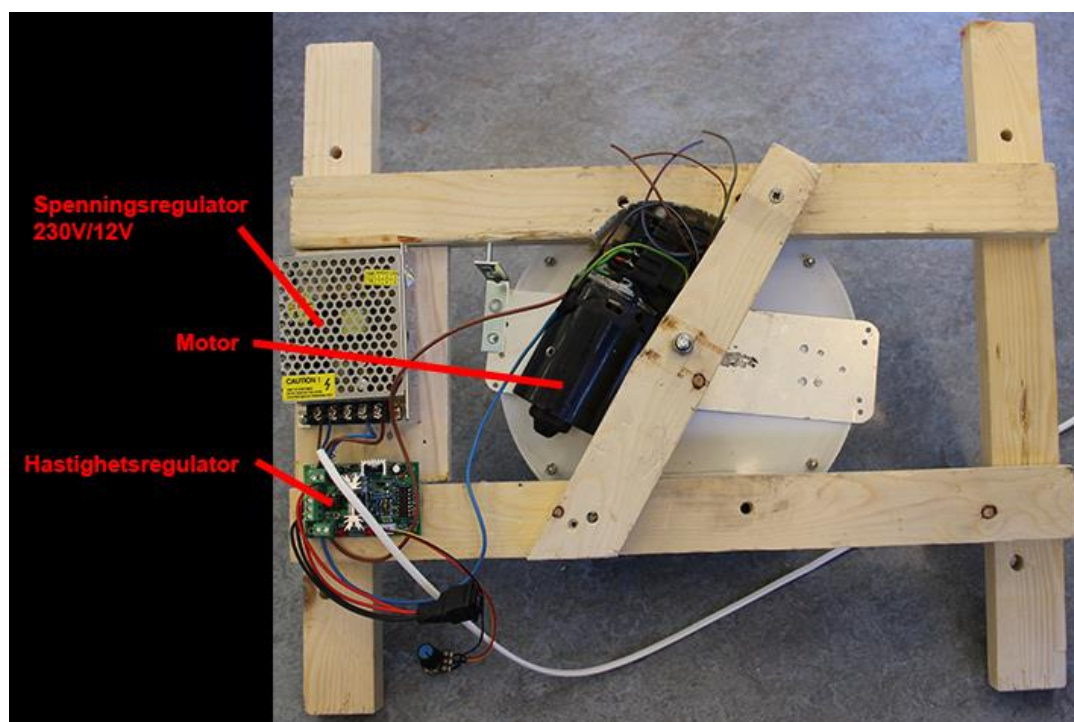
En DC-motor mister moment ved lavt turtall. En girmotor var derfor nødvendig for å rotere alle hindrene i lav hastighet. En girmotor er en motor med et gir eller utveksling koblet mellom motorens aksling og den utgående akslingen. Dette gjør at den utgående akslingen holder lavere hastighet med et høyere moment.

En vindusviskermotor med oppgitt rotasjonshastighet på 300 r/min ble anskaffet. Det var behov for svært lav rotasjonshastighet og ettersom laveste regulerbare hastighet for motoren ga for lavt moment, ble det derfor laget en utveksling på 1/12.5 for å redusere hastigheten og øke momentet. Utvekslingen består av et gummi hjul med diameter på 10cm lagt inntil den 8mm tykke akslingen på motoren. De roterende armene ble deretter koblet til gummi hjulets aksling som nå roterte med 1/12.5 av motorakslingens hastighet. I figur 3-23 vises et bilde av den ferdige utvekslingen.



Figur 3-23 - Bilde av motoraksling mot gummi hjul

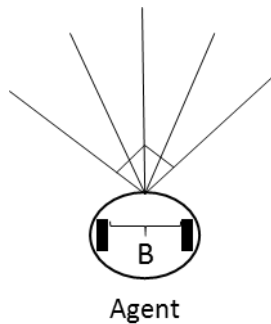
I figur 3-24 vises baksiden av systemet for roterende hindringer. Systemet består av en motor, en hastighetsregulator og en spenningsregulator.



Figur 3-24 - Oversikt over elektronikk for systemet for roterende hindringer

4 Systemløsning

Dette kapitlet tar for seg løsningen av det spesifikke systemet og beskriver implementering, metode og fysikk bak demonstrator og simulering. Programmeringen er realisert i MATLAB (v2014b) og Arduino (v1.5.6-r2). I MATLAB er det brukt en funksjons fil, SETPROD for kryssmultiplikasjon av vektorer og matriser (Ullah, 2006). Disse versjonene legges til grunn i videre beskrivelse av systemløsningen.



Figur 4-1 - Illustrasjon av agenten med synsfeltet for de fem sensorene

fem sensorer for agentens tilstander, $s \in S$

$a \in A, a = \{\text{venstre, rett frem, høyre}\}$

Agenten (figur 4-1) har fem ultrasoniske sensorer som til sammen utgjør tilstandsrommet S i tillegg til tre handlinger for handlingslisten A . Vinkelen mellom de ytterste sensorene er på 90 grader.

Konstruksjon av demonstrator er utført ved å lage en simulering av en agent i MATLAB, samt en fysisk modell av agenten. Simuleringen er ment som et verktøy for å finne optimale verdier og parametere for, og samtidig få en innsikt i hvordan læringsalgoritmen fungerer. Det er også mulig å bruke de simulerte verdiene for den fysiske modellen, slik at man får en ferdig trent agent. Agentens prestasjoner avhenger av simuleringsverdiene, og i hvilken grad disse er tilpasset den fysiske modellens miljø.

Ligninger som brukes i simuleringsalgoritmen

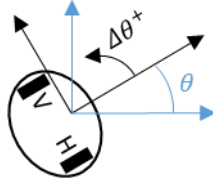
Agentens hastighet:

$$v = \frac{1}{2} \cdot R \cdot (\omega_h + \omega_v) \left[\frac{m}{s} \right] \quad (12)$$

Hvor R er hjulets radius.

ω_h : Høyrehjulets rotasjonshastighet [rad/s]

ω_v : Venstrehjulets rotasjonshastighet [rad/s]



Figur 4-2 – Agentens rotasjon og rotasjonsendring

$$\Delta\theta = \frac{1}{2 \cdot B} (\omega_h - \omega_v) \left[\frac{rad}{s} \right] \quad (15)$$

Hvor B er avstanden mellom venstre og høyre hjul og Θ er den totale rotasjon om agentens senterakse.

Når agenten kjører rett frem roterer begge hjulene med samme rotasjonshastighet. Når agenten skal endre retning stoppes det ene hjulet mens det andre roterer. Vi får dermed at:

- Rett frem: $\omega_h = \omega_v$
- Høyre: $\omega_h > 0$ og $\omega_v = 0$
- Venstre: $\omega_h = 0$ og $\omega_v > 0$

Endring i x-posisjon: $x = x + v \cdot \cos(\theta + \Delta\theta)$

Endring i y-posisjon: $y = y + v \cdot \sin(\theta + \Delta\theta)$

Totale posisjons endring: $P = [x, y] + v \cdot [\cos(\theta + \Delta\theta), \sin(\theta + \Delta\theta)]$

Agentens fysikk er inspirert av (Hatem & Foudil, 2009)

Egenskaper for simulert agent, skalert 1:10 [cm]:

- Agentens radius, Radius = 1.0
- Bredden mellom hjulene, $B = 2 \cdot \text{Radius} = 2.0$
- Radius på hvert hjul, $R = 0.325$
- Maksimal lengde på sensorene = 10.0
- Kollisjon ved sensorverdier mindre enn 1.5
- Rotasjonshastigheten på hjulene når agenten kjører rett frem: $\omega_h = \omega_v = 0.42 \text{ rad/s}$
- Rotasjonshastigheten på hjulene når agenten svinger: $\omega_h = 0,15 \text{ rad/s}$, eller $\omega_v = 0,15 \text{ rad/s}$

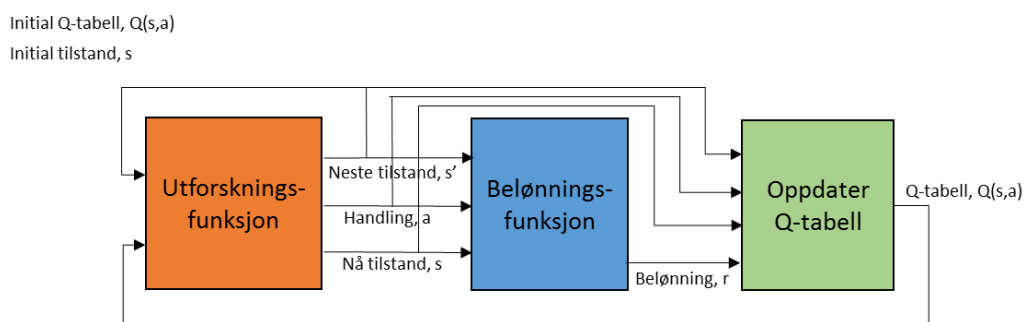
Det er implementert to metoder av Q-læringsalgoritmen. Første metode er basert på at Q-verdiene lagres i en oppslagstabell (Q-tabell), mens den andre metoden går ut på å approksimere en lineær Q-funksjon for systemet (nevralt nettverk).

4.1 Q-Læring med tabellmetoden

Denne metoden er basert på teorien beskrevet i kapittel 2.2.1. Metoden benytter en tabell som oppdateres etter hvert som agenten utforsker sine omgivelser. Ved oppstart er tabellen tom, og agenten vet ingenting annet enn hvilken tilstand den er i og hvilke handlinger den kan utføre. Ligning (1) brukes for oppdatering av Q-verdiene i denne tabellen.

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)] \quad (1)$$

Basert på simuleringresultater brukes $\alpha = 0.5$ og $\gamma = 0.9$ (Se simuleringer av alfa og gamma, kap. 5.2.1). Belønningsverdien r (fra eng: reward) er den umiddelbare verdien agenten får etter en utført handling. Disse verdiene lagres i tabellen slik at agenten neste gang den oppnår samme tilstand kan velge å bruke handlingen fra forrige iterasjon, dersom den ga en god belønning, eller å utføre en ny handling. Ettersom agenten over tid utforsker omgivelsen flere ganger, vil tabellens Q-verdier konvergerer mot en optimal løsning.



Figur 4-3 - Illustrasjon av prinsippet Q-læring med tabellmetoden

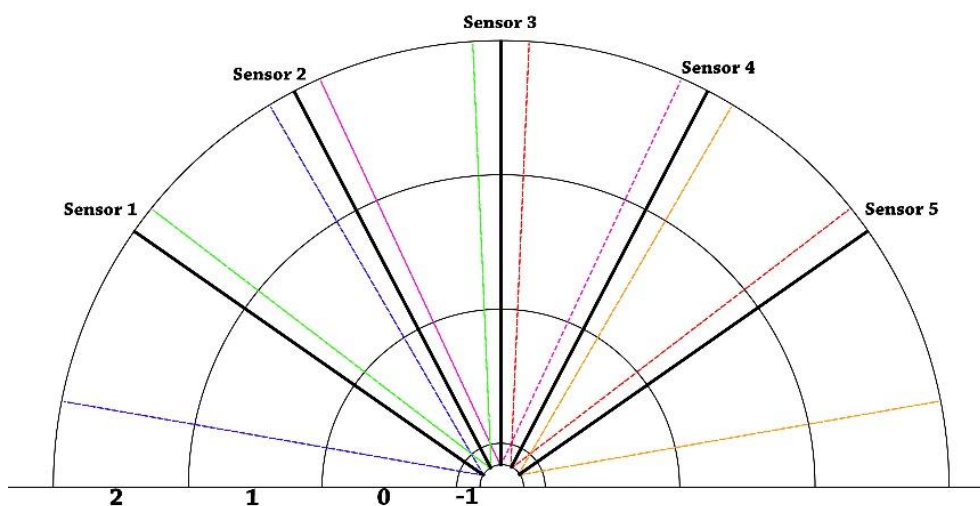
Figur 4-3 viser agentens læringsprosess ved hver nye tilstand. Prinsippet går ut på at agenten starter i en initial tilstand s , utfører en handling a og ender opp i en ny tilstand s' . Utforskningsfunksjonen bestemmer hvordan agenten utforsker omgivelsene ved å bestemme i hvilken grad agenten skal bruke gode tidligere erfarte handlinger eller å prøve en tilfeldig handling i håp om et bedre resultat. Belønningsfunksjonen bestemmer belønningen agenten skal få ut ifra de handlingene den gjør i omgivelsene. Til slutt oppdateres tilstandstabellen og prosessen gjentas, men nå med $s = s'$. Når agenten har nådd en terminaltilstand starter sekvensen på nytt med initialtilstand s . På dette stadiet er verdier fra tidligere sekvenser lagret i Q-tabellen. Hver sekvens defineres av tilstander og handlinger mellom hver terminaltilstand, og betegnes som et forsøk (eng.: trial).

De ultrasoniske sensorene er begrenset, gjennom diskretisering, til heltallsverdier mellom 1 cm og 120 cm. Dersom man skal lage en oppslagstabell for hele spekteret vil tabellen bestå av 120^5 forskjellige kombinasjoner, og ville med tre mulige handlinger endt med en oppslagstabell på $120^5 \cdot 3$ antall plasser. Dette er en mindre god løsning ettersom tabellen ville blitt veldig stor (74 milliarder plasser). En stor oppslagstabell fører til at agenten bruker mye lengere tid på å lære seg alle kombinasjonene av tilstander, og handlinger for disse. En slik løsning blir i tillegg altfor stor for å kunne implementeres på en vanlig datamaskin. Denne utfordringen løses ved å diskretisere, eller dele opp, det kontinuerlige sensorområdet i mindre og mer håndterbare tilstandsrom.

4.2 Diskretisering

Diskretiseringen for simuleringen og den fysiske agenten er noe ulik, da simuleringen bruker en utvidet tilstandsrom-modell for å behandle dynamiske hindringer. Den utvidede modellen er beskrevet i kap. 4.2.1.1 hvor en sammenligning av diskretisering for den fysiske modellen og simuleringen blir presentert.

Agenten oppfatter miljøet rundt seg ved hjelp av sensorer. Flere sensorer vil gi høyere grad av nøyaktighet, men vil også medføre et større behov for prosessering. Agenten er i dette tilfellet utstyrt med fem sensorer, hvor hver har et synsfelt på ca 60 grader. Sensorene er montert slik at de overlappes med ca 15 grader, og agenten har dermed et samlet synsfelt på ca 150°, illustrert i figur 4-4.



Figur 4-4 - Illustrasjon av agentens synsfelt

4.2.1 Tilstandsrommodell

Alle sensorene kan måle avstander opp til 400 cm. For å unngå en stor tilstandstabell og for å effektivisere læringsprosessen, er sensorverdiene diskretisert ned til 144 tilstander. For prosjektet beskrevet i denne rapporten forenkler dette også belønningsfunksjonen.

Hver sensor er delt inn i fire soner basert på avstand i centimeter. Oppdelingen er vist i figur 4.4.

$$\text{Sone} \begin{cases} < 10, \text{ sone} - 1 \\ 10 - 39, \text{ sone } 0 \\ 40 - 69, \text{ sone } 1 \\ > 70, \text{ sone } 2 \end{cases} \quad (16)$$

En tilstand består av fire parametere. To for venstre side og to for høyre side av agentens synsfelt. En tilstand er en kombinasjon av avstand til nærmeste hinder og i hvilken sektor hinderet befinner seg i. En sektor beskriver i hvilken vinkel et objekt befinner seg, relativt til agenten.

De fire parameterene defineres ved:

k1: sone venstre side

k2: sone høyre side

k3: sektor venstre side

k4: sektor høyre side

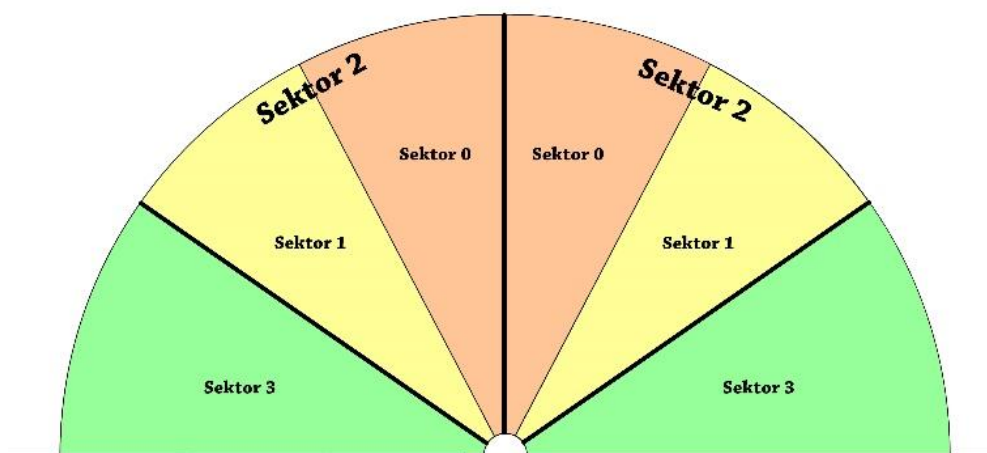
k5: observasjon av dynamisk eller statisk hindring venstre side

k6: observasjon av dynamisk eller statisk hindring høyre side

$$S (\text{tilstandsrommodell}) = k1 \times k2 \times k3 \times k4 (\times k5 \times k6)$$

Agenten kan oppfatte to hindringer samtidig, men ikke på samme side. Ved to hindringer på samme side vil algoritmen prioritere det hinderet som befinner seg i sonen med lavest nummerering.

Oppdelingen av sektorene er illustrert i figur 4.5.



Figur 4-5 - Oppdeling av sektorer

Sektorene har fire ulike verdier, 0, 1, 2 og 3. Sensorene oppfatter et objekt når soneverdien er mindre enn 2. Sektorene er symmetrisk fordelt, slik at agentens høyre synsfelt diskretiseres på samme måte som for venstre. Sensor 1, 2 og 3 på venstre side tilsvarer sensor 3, 4 og 5 for høyre side.

4.2.1.1 Diskretisering for system med dynamiske hindringer

I simuleringen for et system med dynamiske hindringer er tilstandsrommodellen utvidet med to ekstra parametere, k5 og k6. k5 angir med verdien 1 eller 0 om hinderet som er oppdaget på venstre side er dynamiske, mens k6 gjør det samme for høyre side. Tallet 1 angir et dynamiske hinder, mens 0 angir statiske. Tilstandstabellen er med dette utvidet til 576 tilstander. Et objekt blir klassifisert som et dynamisk hinder ved at agenten først, mens den står stille, utfører to målinger rett etter hverandre. Deretter sammenlignes de to målingene. Om hindringen beveger seg bort fra agenten vil første måling trukket fra andre måling bli et positivt tall, og klassifisert som et statisk hinder. Hvis samme regneoperasjon derimot gir et negativt tall, klassifiseres hindringen som et dynamisk hindring og agenten må reagere deretter.

4.2.1.2 Tilstandstabellen

Tilstandstabellen er bygget opp som en tabell hvor alle kombinasjoner av tilstander og handlinger er representert som et tabellpunkt. Tilstandstabellen for den fysiske modellen har 144 rader, og tre kolonner som presenterer hver av de mulige handlingene (venstre, høyre, rett frem). Et utsnitt av tilstandstabellen vises i tabell 7.

Tabell 7 - Utsnitt av tilstandstabell for statiske hindringer

Tilstand: k1x k2 x k3 x k4	a = 1	a = 2	a = 3
0 0 0 0	-15,37429161	-10,52491053	-0,612616775
0 0 0 1	0	0	0
0 0 0 2	-15,97028234	-25,40510262	-0,257283015
0 0 0 3	0	0	-0,133666927
0 0 1 0	0,00116178	-0,028690741	-0,443826888
0 0 1 1	0,422954034	-0,496646809	-0,303721668
⋮			
2 2 3 0	0,003119721	-0,269173241	-0,368630839
2 2 3 1	0,039068136	-0,33971912	-10,03379701
2 2 3 2	-0,002509818	-0,273656989	-0,346646134
2 2 3 3	0,693213719	-10,37765615	-10,16193666

Som vist i tabell 7 vil det være noen tilstander som ikke er oppdatert etter at agenten har gjennomført en lengere simulering. I dette tilfellet er det kjørt 500 forsøk med 400 steg i hver. Tilstandene som fremdeles er 0 er tilstander den simulerte agenten ikke vil kunne nå, slik den er designet. Ved k1x k2 x

$k_3 \times k_4 = 0 \ 0 \ 0 \ 1$, vises en slik tilstand. Dette er en tilstand hvor hinderet befinner seg innerst i høyre sektor ($k_2 = 0$, $k_4 = 1$, se presentasjon av de fire parameterene under figur 4-3). En grunn til at Q-verdiene for denne tilstanden ikke er endret, kan være at agenten har unngått hindringene før nærmeste sone er besøkt.

4.2.2 Belønningsfunksjon

Belønningsfunksjonen bestemmer agentens umiddelbare belønning etter hver erfaring. Ved implementering av belønningsfunksjonen skilles det mellom gode handlinger som gir en positiv belønningsverdi, og dårlige handlinger som gir en negativ belønningsverdi.

Belønningsfunksjonen som brukes for Q-læring med tabellmetoden er inspirert av (Strauss & Sahin, 2008) og vist under ligning (17-20):

$$r_1 = \begin{cases} 0.2, & \text{rett frem} \\ -0.1, & \text{venstre} \\ -0.1, & \text{høyre} \end{cases} \quad (17)$$

$$r_2 = \begin{cases} 0.2, & \text{total endringen av sensorverdiene er } \geq 0 \\ -0.2, & \text{total endringen av sensorverdiene er } < 0 \end{cases} \quad (18)$$

$$r_3 = \begin{cases} 0 \\ -0.8, & \text{ulik sving rett etter hverandre} \end{cases} \quad (19)$$

$$r = \begin{cases} r_1 + r_2 + r_3, & \text{dersom det er ingen kræs} \\ -100, & \text{kollisjon med en hindring} \end{cases} \quad (20)$$

Fra belønningsfunksjonen defineres r_1 til å gi positiv belønningsverdi når agenten kjører rett frem, mens det for sving gis en liten negativ belønningsverdi. Den negative verdien gis for å prioritere kjøring rett frem.

Parameteren r_2 gir positiv belønning når den totale endringen i sensorverdiene er positiv. Det vil si at agenten beveger seg bort fra en hindring. Dersom agenten beveger seg mot en hindring blir den totale endringen av sensorverdiene negativ, og agenten får negativ belønning.

Funksjonen r_3 gir agenten negativ belønningsverdi dersom den først svinger til høyre og deretter til venstre, eller først til venstre og deretter til høyre etter hverandre. Denne funksjonene gjør at agenten unngår en ubesluttos sekvens, hvor den bare svinger fram og tilbake.

Funksjonen r gir agenten en samlet belønningsverdi lik summen av r_1 , r_2 og r_3 hvis den ikke kolliderer, mens ved kollisjon får den en negativ belønningsverdi på -100.

4.2.3 Utforskningsfunksjon

Utforskningsfunksjonen avgjør agentens strategi for valg av handlinger. Den kan enten utforske tilstandsrommet eller utnytte erfaringene den allerede har. Funksjonen sørger for en passende balanse mellom disse strategiene (se kap. 2.3).

Simuleringene for utforskningsmetodene viser at softmax gir best resultater i den simulerte modellen, og brukes dermed i simuleringen. Parameteren T er satt lik 24, og reduseres med 5% av sin totalverdi for hvert enkelt forsøk. I den fysiske modellen brukes epsilon fordi den gir bedre oversikt over hvor stor tilfeldighet det er for at agenten tar en vilkårlig handling, og epsilon er satt til 5% sjanse for å gjøre en tilfeldig handling.

4.2.4 Implementering

Implementeringen av Q-læring med tabell metoden er basert på samme algoritme og belønningsfunksjon i simuleringen og for den fysiske modellen. Forskjellen mellom simuleringen og fysiske agenten er utforskningsfunksjonen og en ekstra utvidelse av tilstandsrommet i simuleringen (se diskretiserings kapittelet). I simuleringen har man valget mellom statiske og dynamiske hindringer, og for å tilpasse agenten til dynamiske hindringer utvides Q-tabellen.

4.2.4.1 Q-Læringsalgoritmen med tabellmetoden

Ved oppstart:

- Initialiser tilstandsrommet S
- Initialiser handlingslisten A
- Initialiser Q-tabellen $Q(s, a)$ (tom tabell) ut ifra tilstandsrommet S og handlingslisten A

For hvert enkelt forsøk:

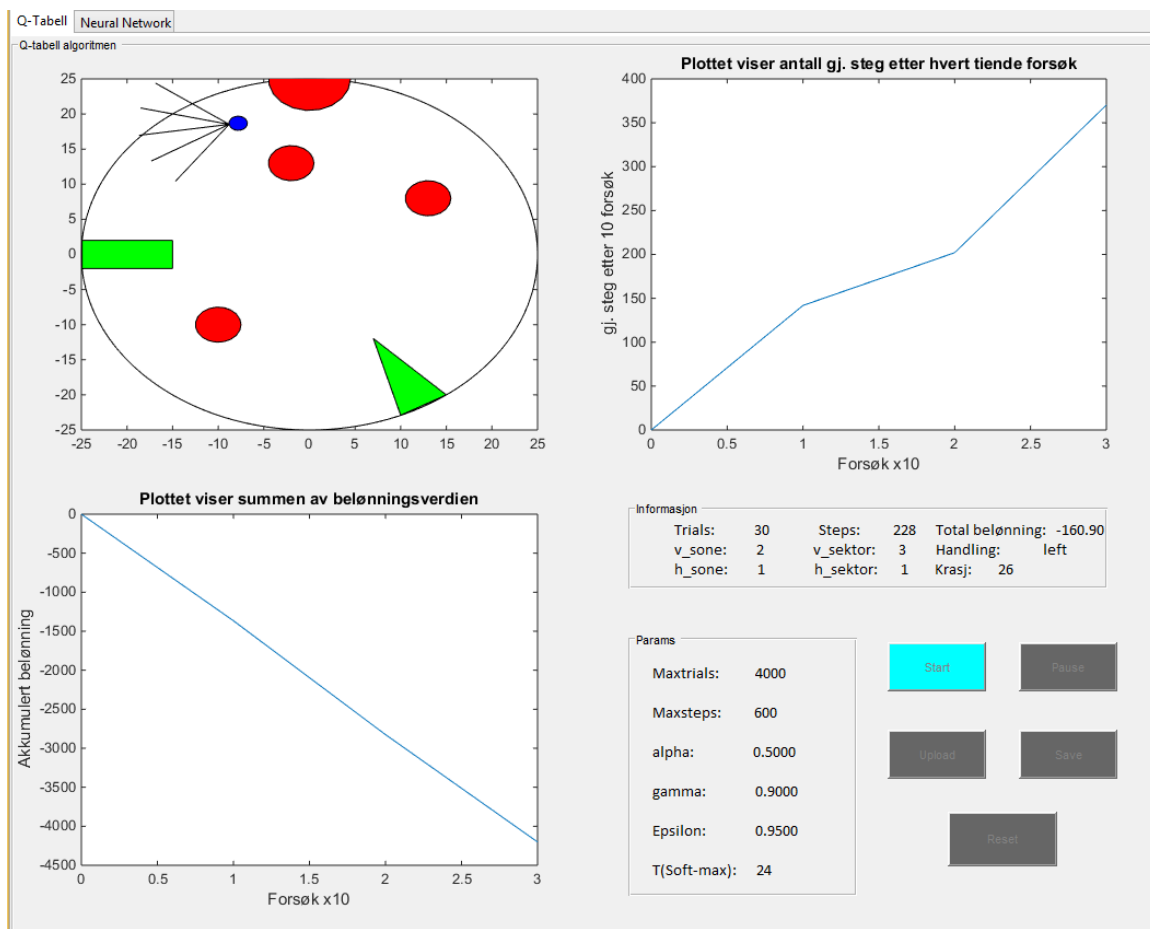
- i) Observer initialsensorverdier
- ii) Diskretiser sensorverdiene og sett det lik s (Initial-tilstand)
- iii) Velg en handling a ut ifra utforskningsfunksjonen og utfør handlingen
- iv) Observer agentens sensorverdier etter handlingen
- v) Diskretiser sensorverdiene og sett det lik s' (Neste-tilstand)
- vi) Gi agenten belønningen r ut ifra belønningsfunksjonen
- vii) Oppdater Q-tabellen
- viii) Sett $s = s'$ (Nå-tilstand)
- ix) Behold siste handling, $prev_a = a$ (Brukes i belønningsfunksjonen)
- x) Gjenta operasjonen fra punktet iii) (et/en nytt/ny skritt/erfaring) dersom agenten ikke har oppnådd sin terminaltilstand (kollisjon eller maksimalt antall skritt). Start et nytt forsøk dersom agenten har nådd sin terminal tilstand (Start på nytt fra i).

4.2.4.2 Simulert modell

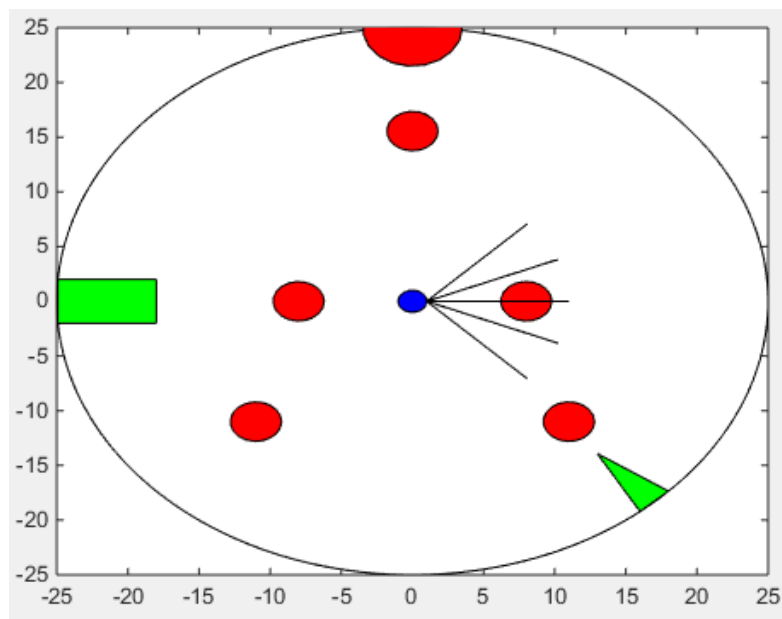
I simuleringen er det laget et brukergrensesnitt som viser brukeren simuleringen av agenten, to ulike grafer, knapper for ulike funksjoner samt et informasjonspanel. Informasjonspanelet viser brukeren relevante verdier fra simuleringen. Den ene grafen viser et gjennomsnitt av antall steg for hvert tiende forsøk, mens den andre viser agentens akkumulerte belønning. Ett steg i programmet tilsvarer en erfart situasjon, enten den er erfart fra før eller er ny (se figur 4-3). Et nytt forsøk startes for hver gang agenten har beveget seg maksimalt antall steg for inneværende forsøk, eller har kollidert med en hindring. Maksimalt antall steg kan endres, men er normalt 400-600. Dimensjonene i simuleringen er skalert 1:10 [cm].

Figur 4-6 viser brukergrensesnittet MATLAB for simuleringen. Simuleringen kan brukes som et hjelpemiddel for å se på agentens læringsprosess, samt brukes for å finne de parameterene som best passer systemet. En av ulempene med den fysiske modellen er at agenten må flyttes til en ikke-kolliderende tilstand hver gang den kolliderer, mens dette gjøres automatisk i simuleringen. Punktet (0, 0) i senter av arenaen i simuleringen er satt som initialtilstanden s , og vil være agentens startpunkt for hvert nye forsøk (figur 4-6).

Algoritmen for dynamiske hindringer fungerer i simuleringen på samme måte som for de statiske. De har samme belønningsfunksjon og utforskningsfunksjon, men tilstandstabellen for dynamiske hindringer er utvidet med $k5$ og $k6$ (se kap. 4.2.1.1). En hindring blir klassifisert som dynamisk dersom en av agentens sensorer gir en negativ endringsverdi. I simuleringen beregnes endringsverdien ved at agenten står stille i en iterasjon, utfører to målinger, før neste iterasjon.



Figur 4-6 – Skjermdump av brukergrensesnitt for simulering

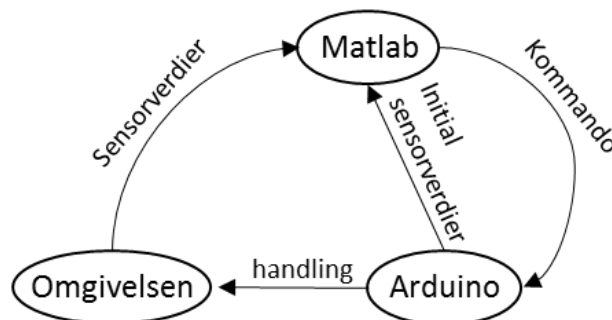


Figur 4-7 - Skjermdump fra simulering av dynamiske hindringer

De dynamiske hindringene i simuleringen består av fem objekter, illustrert ved røde sirkler i figur 4-7, som beveger seg i en sirkulær bevegelse. De to innerste hindringene beveger seg i én retning, mens de tre ytterste beveger seg i motsatt retning.

4.2.4.3 Fysisk modell

Implementering av den fysiske agenten er gjort med MATLAB og Arduino. Ideen er å bruke MATLAB til å gjøre beregninger og Arduino til å utføre kommandoene sendt fra MATLAB. Kommunikasjonen skjer via bluetooth.



Figur 4-8 - Kommunikasjon mellom Arduino og MATLAB

Figur 4-8 illustrerer kommunikasjonen mellom MATLAB og Arduino. MATLAB sender kommandoene til Arduino som utfører disse. Selve læringsprosessen er helt lik som i figur 4-3. De ultrasoniske sensorene på den fysiske agenten er begrenset til heltallsverdier fra 1 cm til 120 cm i koden på agentens mikrokontroller. Avstand mindre enn 8 cm er definert som kollisjon. Initialtilstanden i den fysiske agenten er en vilkårlig posisjon i omgivelsene. Det eneste kravet er at initialtilstanden ikke er en terminaltilstand (kollisjon).

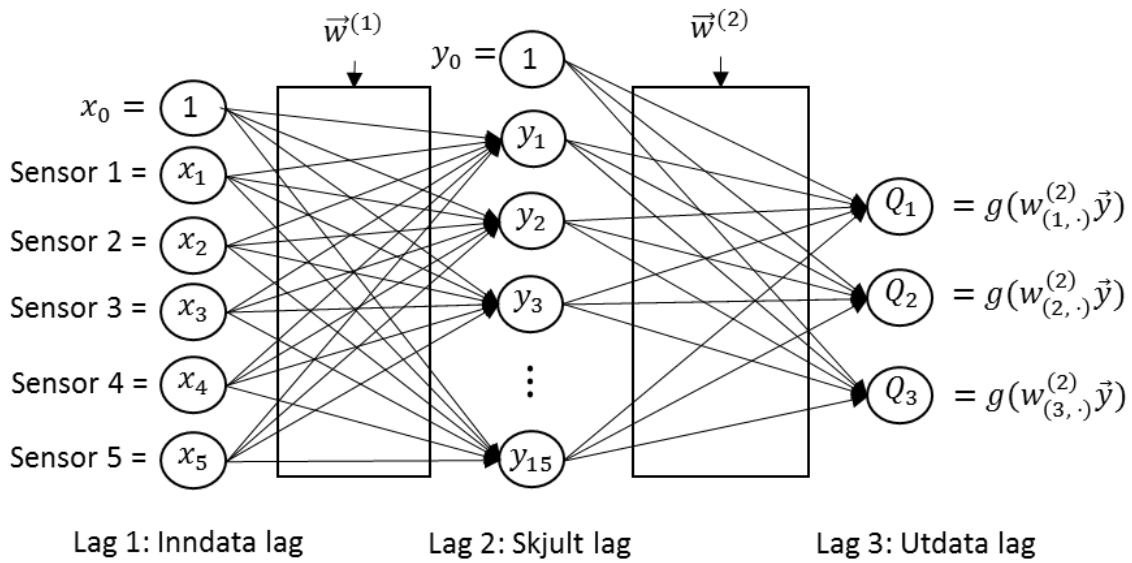
Kommandoene

- Kommando1: Be om sensorverdier
- Kommando2: Be agenten kjøre rett frem
- Kommando3: Be agenten svinge høyre
- Kommando4: Be agenten svinge venstre
- Kommando5: Be agenten stoppe, sensorverdier < 8cm (kollisjon)

4.3 Q-Læring med nevral nettverk (Lineær funksjons approksimering)

Q-Læring med nevral nettverk er basert på approksimasjon av en eller flere lineære Q-funksjoner. I motsetning til Q-læring med tabell, lagrer denne metoden Q-funksjonene i et nevral nettverk. Ved å bruke denne metoden er ikke diskretisering nødvendig, og hele det kontinuerlige sensorspekteret kan brukes. Virkemåten bak et kunstig nevron er beskrevet i kapittel 2.3.3.

Nevralt Nettverk:



Figur 4-9 - Agentens nevralt nettverk

Figuren over viser det nevralt nettverket slik det er implementert i dette systemet. I nettverket kobles alle nevronene inkludert biasen fra et lag videre til hvert enkelt nevron i neste lag. Det nevralt nettverket har tre lag. Lag en, inndata laget, har fem nevroner (et for hver sensor), lag to har femten nevroner, og lag tre har tre nevroner etter antall handlinger i handlingslisten. Lag 1 er starten på nettverket, og sensorverdiene brukes som inndata, og disse må ligge i intervallet $[-1, 1]$. Vektmatrisen $w^{(1)}$ brukes til å vekte koblingen mellom lag 1 og lag 2, mens $w^{(2)}$ for lag 2 og lag 3 (Se teori).

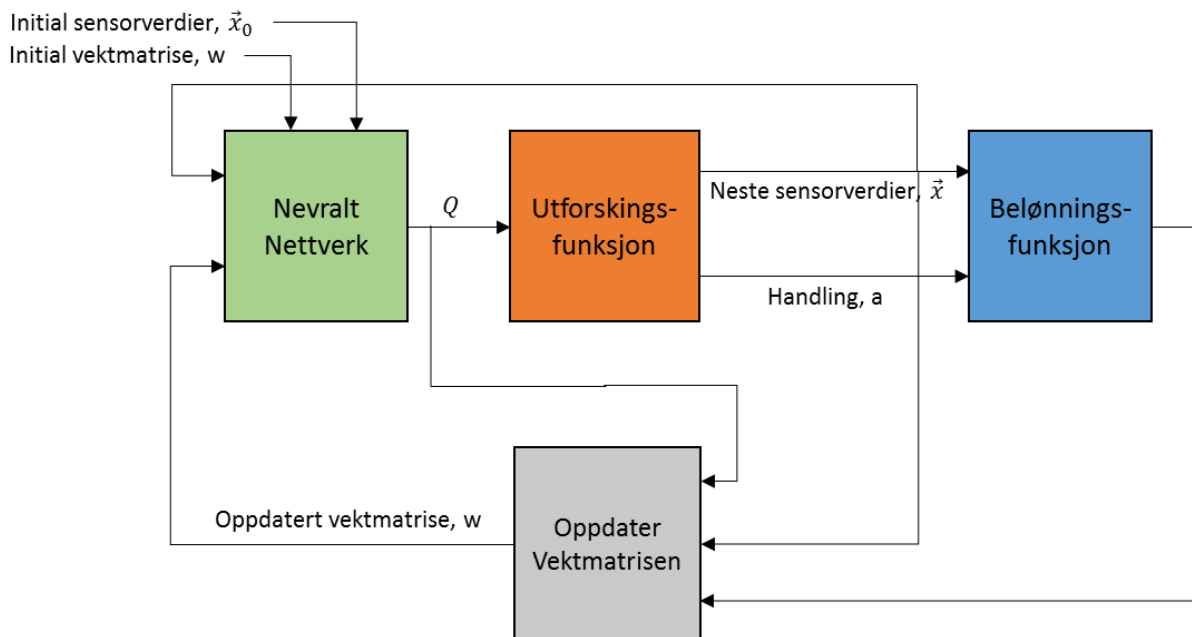
Matrisene har en dimensjon på $(\text{antall nevroner i lag } L) \times (\text{antall nevroner i lag } (L-1) + \text{bias})$.

$w^{(1)}$ har en dimensjon på 15×6 og $w^{(2)}$ dimensjon på 3×16 . I utdata laget er det tre nevroner, og hvert enkelt nevron tilsvarer en Q-funksjon. Antall nevroner i utdata laget er lik antall handlinger til agenten, og hvert enkelt nevron estimerer en Q-funksjon med hensyn på en handling i agentens handlingsliste Ligning (21).

$$\vec{Q}(s,a) = \begin{bmatrix} Q_1 = Q(s, \text{rett frem}) \\ Q_2 = Q(s, \text{høyre}) \\ Q_3 = Q(s, \text{venstre}) \end{bmatrix} \quad (21)$$

Disse Q-funksjonene estimerer en Q-verdi som brukes videre til å bestemme de optimale handlingene for agenten, hvor den optimale handlingen for hver tilstand tilsvarer den største Q-funksjonen. Den optimale handlingen for hver tilstand er derfor definert som $a = \max(\vec{Q}(s,a))$. Det er umulig å si hvilken Q-funksjon(er) som passer til systemet, siden det nevralt nettverket estimerer Q-funksjonen ut

ifra agentens erfaringer. For at agenten skal oppnå høyest total belønning, må den derfor utforske miljøet lenge nok for at nettverket skal konvergere til en optimal funksjon.



Figur 4-10 - Konseptskisse for agentens læringsprosess med NN

Figur 4-10 viser læringsprosessen til agenten. Metoden trenger initialverdier, og disse gis ved sensorverdier og nettverkets vektmatriser. Vektmatrisene initialiseres med tilfeldige verdier mellom 0 og 1, men ikke null. Dersom vektmatrisene initialiseres med en verdi lik 0 får alle nevronene lik verdi, noe som gjør at det nevrale nettverket ikke blir brukbart. Figur 4-10 viser at algoritmen bruker agentens sensorverdier, i motsetning til tabellmetoden som bruker en diskretisering av verdiene. Utforskings-funksjonen som brukes er den samme som for tabellmetoden, men belønningsfunksjonen er ulik. Hensikten med denne metoden er å finne optimale Q-funksjoner ved å oppdatere vektmatrisene etter hvert forsøk.

4.3.1 Belønningsfunksjon

Ettersom agentens tilstandsrom ved nevral nettverk er kontinuerlig er det ikke nødvendig med en like utdypende belønningsfunksjon som for tabellmetoden. Belønningsfunksjonen er definert ved at agenten får positiv belønning dersom den kjører rett frem, og negativ belønning ved sving og kollisjon. Belønningsverdien r må ligge i intervallet $[-1,1]$, og funksjonen slik den er definert for systemet er vist i ligning (22).

$$r = \begin{cases} 0.01, \text{ rettfrem} \\ -0.01, \text{ venstre} \\ -0.01, \text{ høyre} \\ -0.9, \text{ kollisjon} \end{cases} \quad (22)$$

Belønningsfunksjonen r er inspirert av (Huang, Cao & Guo, 2005).

4.3.2 Foroverkoblet nevral nettverk (eng.: Feedforward)

Som beskrevet i kapittel 2 Teori er nevral nettverk i dette prosjektet et foroverkoblet nevral nettverk.

Nevronene aktiveres ved bruk av den hyperbolske tangens funksjonen gitt i ligning (23).

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (23)$$

De aktiverte verdiene blir utdata-verdier i dette laget, og brukes videre som inndata for neste lag. De tre nevronene i utdata laget tilsvarende Q-funksjonene når all data er matet frem i nettverket.

$$w^{(2)} = \begin{bmatrix} w_{1,0}^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} & \dots & w_{1,15}^{(2)} \\ w_{2,0}^{(2)} & w_{2,1}^{(2)} & w_{2,2}^{(2)} & \dots & w_{2,15}^{(2)} \\ w_{3,0}^{(2)} & w_{3,2}^{(2)} & w_{3,2}^{(2)} & \dots & w_{3,15}^{(2)} \end{bmatrix} \quad (24)$$

$$\vec{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{15} \end{bmatrix} \quad (25)$$

Vektmatrisen w har m antall rader lik antall nevroner, og kolonner n lik antall inndata (ligning (24)).

Matrisen $w^{(2)}$ og vektoren \vec{y} multipliseres med hverandre, og produktet brukes i

aktiveringsfunksjonen som ble beskrevet tidligere i kapittelet. Elementene i både vektmatrisen og inndatavektoren er reelle tall, og verdien for inndataene ligger i intervallet $[-1, 1]$. Ligning (26-28) viser Q-funksjonene for de tre handlingene.

$$Q_1 = Q(s, \text{rett frem}) = g(\vec{w}_{(1,0)}^{(2)} \cdot y_0 + \vec{w}_{(1,1)}^{(2)} \cdot y_1 + \vec{w}_{(1,2)}^{(2)} \cdot y_2 \dots + \vec{w}_{(1,15)}^{(2)} \cdot y_{15}) \quad (26)$$

$$Q_2 = Q(s, \text{høyre}) = g(\vec{w}_{(2,0)}^{(2)} \cdot y_0 + \vec{w}_{(2,1)}^{(2)} \cdot y_1 + \vec{w}_{(2,2)}^{(2)} \cdot y_2 \dots + \vec{w}_{(2,15)}^{(2)} \cdot y_{15}) \quad (27)$$

$$Q_3 = Q(s, \text{venstre}) = g(\vec{w}_{(3,0)}^{(2)} \cdot y_0 + \vec{w}_{(3,1)}^{(2)} \cdot y_1 + \vec{w}_{(3,2)}^{(2)} \cdot y_2 \dots + \vec{w}_{(3,15)}^{(2)} \cdot y_{15}) \quad (28)$$

Beregningene i figurene over kan forkortes, og vises i ligning (27) på vektorform.

$$\vec{Q} = g(\mathbf{w}^{(2)} \cdot \vec{y}) \Rightarrow \vec{Q} = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} \quad (29)$$

Aktivering av det skjulte laget skjer på samme måte som for utdata laget, og inndataen til dette laget er sensorverdiene.

4.3.3 Oppdatering av vektmatrisene

For å lære opp nettverket brukes tilbake-forplantingsalgoritmen (eng.: backpropagation algorithm), og gradient nedstigning (eng.: gradient descent). Målet med tilbake-forplantingsalgoritmen er å finne funksjonens partiell deriverte, som brukes til å oppdatere vektmatrisen w . I korte trekk går læringsprosessen ut på å minimalisere avviket mellom Q -funksjonene og den estimerte Q -verdien det nevrale nettverket ønsker å oppnå, Q_{est} . Dette avviket minimaliseres ved å finne optimale verdier for vektmatrisene. Kostnadsfunksjonen C er et mål på avviket mellom Q -funksjonen og Q_{est} .

$$C = \frac{1}{2}(\text{error})^2 = \frac{1}{2}(Q_{\text{est}} - Q(s, a))^2 \quad (30)$$

$$Q_{\text{est}} = r + \gamma \max_a Q(s', a) \quad (31)$$

Ligning (31) er basert på (Huang et al., 2005). Q_{est} er en reell verdi og et resultat av agentens erfaring (ligning (29)), og brukes videre til å trene opp det nevrale nettverket. Parameteren gamma har lik betydning som for tabellmetoden, og er satt lik 0.9 etter simulering i testresultatene (kap. 5.3.2). Det fins flere måter å definere en kostandsfunksjon på, men funksjonen i ligning (28) er matematisk enkel og gjør implementeringen av funksjonen enkel.

Tilbakeforplantingsalgoritmen:

Avvikene i hvert enkelt nevron brukes til å beregne de partiell deriverte verdiene. Avviket kalles for $\vec{\delta}^L$.

Parameteren L indikerer hvilket lag avviket er for. Avviksvektoren har lik dimensjon som antall nevroner + bias for laget L. Algoritmen starter i det siste laget og jobber seg bakover i nettverket. Formel 1-10 viser likningen for avviksberegning i utdatalaget.

$$\vec{\delta}^3(a) = (Q_{est} - \vec{Q}(s, a)) \square \vec{I}(a) \quad (32)$$

Vektoren $\vec{Q}(s, a)$ er Q-verdiene fra det nevrale nettverket. Ved beregningen av avviket tas det kun hensyn på Q-verdien som samsvarer med den siste utførte handlingen. I er en identitetsmatrise, hvor $\vec{I}(a)$ henviser til hver kolonne for de ulike handlingene i identitetsmatrisen.

$$I = [\vec{I}(\text{rettfrem}), \vec{I}(\text{høyre}), \vec{I}(\text{venstre})] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (33)$$

Formel 12-1 viser dersom a = høyre blir avviksvektoren i utdatalaget

$$\vec{\delta}^3(\text{høyre}) = \begin{bmatrix} 0 \\ Q_{est} - Q_2 \\ 0 \end{bmatrix} \quad (34)$$

Årsaken til at avviksvektoren multipliseres med enhetsvektoren i lag 3 er for å unngå feilestimeringer av de andre Q-funksjonene. Etter beregning av avviksvektoren $\vec{\delta}^3$, sendes den tilbake til forrige lag 2, for å finne avviksvektoren $\vec{\delta}^2$ i det laget. Beregning for avviksvektoren i lag 2 vises i ligning (35).

$$\vec{\delta}^2 = ((w^{(2)})^T \cdot \vec{\delta}^3) \square g' \left(\begin{bmatrix} \text{bias} \\ (w^{(1)} \cdot \vec{x}) \end{bmatrix} \right) \quad (35)$$

$(w^{(2)})^T$ er den transponerte vektmatrisen for vektene mellom lag 2 og lag 3 (se figur 4-9). Den transponerte matrisen multipliseres med avviksvektoren fra lag 3. Matrisen $w^{(2)}$ har en dimensjon på 3x16, og $(w^{(2)})^T$ får derfor en dimensjon på 16x3. Avviksvektoren $\vec{\delta}^3$ har dimensjonen 3x1. Produktet

mellom vektmatrisen og avviksvektoren $\vec{\delta}^3$ blir en vektor med dimensjon 16x1 (lik dimensjon som \vec{y}). Vektoren blir videre elementvis multiplisert med den deriverte av aktiveringsfunksjonen. Ligning(36) viser derivasjonen av g.

$$g'(x) = \frac{d}{dx}[\tanh(x)] = \frac{4 \cos h^2(x)}{(\cosh(2x) + 1)^2} \quad (36)$$

Matrisen $w^{(1)}$ er vektmatrisen for koblingen mellom lag 1 og lag 2, og har en dimensjon på 15x6. Vektoren \vec{x} er inndataene + x_0 (bias) i lag 1. Dimensjonen på vektoren \vec{x} er 6x1. Multiplikasjon av matrisen $w^{(1)}$ og vektoren \vec{x} gir en vektor med dimensjon 15x1, som ikke er lik dimensjonen i uttrykket $(w^{(2)})^T \cdot \vec{\delta}^3$. Dette løses ved å legge til bias = 1, og vektorene får like dimensjoner.

Lag 1 får inn sensorverdier som inndata. Fordi sensorverdiene ikke er estimerte verdier fra nevroner, finnes det ingen avviksvektor $\vec{\delta}^1$. Etter å ha funnet alle $\vec{\delta}$ -vektorene med tilbake-forplantingsalgoritmen, brukes avviksvektorene videre til å finne de partiellderivate verdiene med hensyn på vektmatrisene $w^{(1)}$ og $w^{(2)}$.

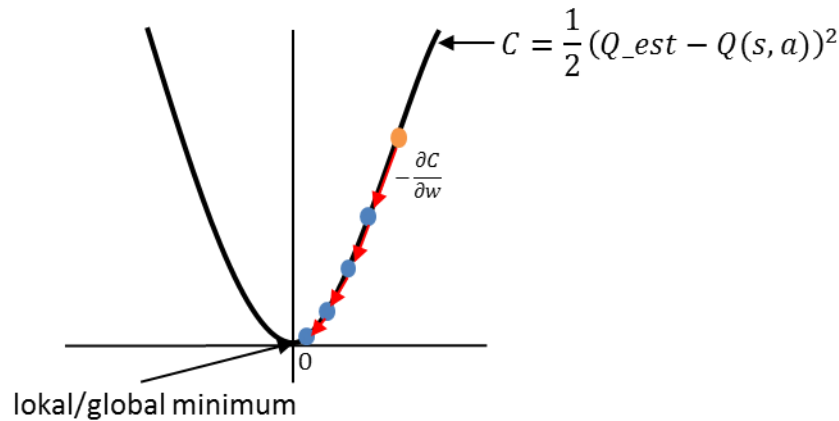
$$\frac{\partial C}{\partial w^{(1)}} = \vec{\delta}^2 \cdot (\vec{x})^T \quad (37)$$

$$\frac{\partial C}{\partial w^{(2)}} = \vec{\delta}^3 \cdot (\vec{y})^T \quad (38)$$

De partiellderivate verdiene brukes til å oppdatere vektmatrisene, og dette gjøres ved algoritmen gradient nedstigning. Tilbake-forplantningsalgoritmen er basert på (Ng et al., 2015).

Gradient nedstigning (eng.: Gradient descent):

Algoritmen gradient nedstigning brukes til å oppdatere vektmatrisene med de partiellderiverte verdiene. Algoritmens mål er å finne et lokalt minimum som gjør at kostnadsfunksjonen går mot null.



Figur 4-11 - Gradient nedstigning for kvadratisk funksjon

Figur 4-11 illustrerer en gradient nedstigning for en kostnadsfunksjon. De røde pilene i figuren viser hvert enkelt steg for gradient nedstignings algoritmen. Algoritmen nærmer seg et lokalt minimum ved mange nok steg. Etter å ha beregnet gradienten til kostnadsfunksjonen $\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} & \frac{\partial C}{\partial w^{(2)}} \end{bmatrix}$ med tilbake-forplantings algoritmen, brukes ∇C til å oppdatere vektmatrisen i det nevrale nettverket. Oppdateringen av vektmatrisene vises i ligning (39-40).

$$w^{(1)} = w^{(1)} - \alpha \frac{\partial C}{\partial w^{(1)}} \quad (39)$$

$$w^{(2)} = w^{(2)} - \alpha \frac{\partial C}{\partial w^{(2)}} \quad (40)$$

Alfa parameteren er læringsraten til algoritmen og bestemmer størrelsen på nedstigningen algoritmen skal flytte seg per steg. Resultatet i kapittel 5.3.1 viser at $\alpha = 0.03$ gir best resultat for dette systemet. Denne læringsraten er den samme som læringsraten i formel 1-1 for Q-læring metoden. Løsningen av gradient nedstignings algoritmen er basert på (2.5 - Gradient Descent - [Machine Learning] By Andrew Ng, 2014).

4.3.4 Virkemåte

Oppbyggingen av algoritmen er basert på figur 4-9, figur 4-10 og kapittel 4.3. Flytskjema for programmet ligger i vedlegg C.

4.3.4.1 Q-læring med nevral nettverk:

Virkemåte for nevral nettverk algoritmen gjennomgås i punktene under.

Før start:

- Oppgi størrelsen på det nevrale nettverket (5 nevroner i lag 1, 15 nevroner i lag 2, og 3 nevroner i lag 3).
- Initialiser vektmatrisene $w^{(1)}$ og $w^{(2)}$ med tilfeldige verdier mellom 0 og 1

For hvert enkelt forsøk:

- i) Observer agentens sensorverdier
- ii) Estimer Q-funksjonene med foroverkobling i agentens nevrale nettverk
- iii) Velg en handling a med utforskningsfunksjonen og utfør handlingen
- iv) Observer agentens sensorverdier etter utført handling
- v) Gi agenten belønningen r med belønningsfunksjonen
- vi) Estimer Q-funksjonene med foroverkobling
- vii) Estimer Q_{est}
- viii) Beregn partiell deriverte $\frac{\partial C}{\partial w^{(1)}}$ og $\frac{\partial C}{\partial w^{(2)}}$ med tilbake-forplantings algoritmen
- ix) Oppdater vektmatrisene w_1 og w_2 med gradient nedstignings algoritmen
- x) Gjenta algoritmen fra punkt iii). Dersom agenten ikke har nådd sin terminal tilstand (kollisjon eller maks steg) startes et nytt steg. Dersom terminaltilstand er nådd startes algoritmen på nytt fra punkt i).

Brukergrensesnittet for simuleringen av Q-læring med nevral nettverk og tabell metoden er helt lik.

Implementeringen er helt lik for systemene med statiske og dynamiske hindringer.

5 Resultater

5.1 Prosedyre for simulering

Alle testene i kapittelet er basert på simuleringer. Simuleringer brukes fordi det er tidsbesparende og enklere å se utvikling over tid for ulike metoder og parametre. Testene er gjort i en bestemt rekkefølge, ettersom ulike metoder og parametre avhenger av hverandre. Det vil i de tilfeller det er nødvendig for påfølgende testing trekkes slutninger om hvilke metoder eller parametre som fungerer best.

Rekkefølgen på simuleringene er kronologisk gitt i oversikten over diagrammer (tabell 8). Metodene er i tabellen angitt som Q (tabellmetoden) og NN (nevralt nettverk).

Tabell 8 - Oversikt over diagrammer i resultater

Oversikt over diagrammer				
Metode	Hva	Verdier	Forsøk	Steg
Annet	Epsilon og softmax	$\alpha = 0.1, \gamma = 0.7$	400	400
Q	Alfa	$\alpha = \{ 0.01, 0.1, 0.3, 0.5, 0.7, 0.9 \}$	500	400
Q	Gamma	$\gamma = \{ 0.01, 0.1, 0.3, 0.5, 0.7, 0.9 \}$	400	400
NN	Alfa	$\alpha = \{ 0.01, 0.1, 0.3, 0.5, 0.7 \}$	500	600
NN	Alfa	$\alpha = \{ 0.001, 0.01, 0.03, 0.05, 0.07, 0.09 \}$	500	600
NN	Gamma	$\gamma = \{ 0.01, 0.1, 0.3, 0.5, 0.7, 0.9 \}$	500	600
NN og Q	Sammenligning	Optimal α og γ	1000	600
Q	Dynamisk	Optimal α og γ	1400	600
NN	Dynamisk	Optimal α og γ	1400	600
NN	Dynamisk	Optimal α og γ	4000	600

Simulatoren er laget så lik som mulig den fysiske demonstratoren, men skiller seg fra den på følgende punkter:

1. Simuleringen er basert på sensorer som ser en rett linje, en gitt lengde og retning relativt til bilen. Sensorene i den fysiske modellen oppfatter objekter i et område, og evnen til å oppfatte et objekt avhenger også av den relative vinkelen mellom agenten og objektet. Det fører i noen tilfeller til feil målerverdier.
2. Simuleringen registrerer når agentens kropp krasjer med objekter, men dette gir ingen negativ belønning. Demonstratoren oppdager ikke denne type krasj siden den ikke har sensorer på sidene og bak.
3. Simuleringen hastighet bestemmes av prosessorkapasitet, og kjører så raskt at agenten kan måle to ganger rett etter hverandre uten at det oppfattes eller gir noen konsekvenser. To målinger brukes til å avgjøre om et objekt er dynamisk eller statisk. I den fysiske modellen er ikke det mulig på grunn av tidsforsinkelse i elektronikk og kommunikasjon, samt fysikk.

Resultatene viser prinsippet bak metoden, og verdiene som er presentert vil ikke nødvendigvis være de mest optimale verdiene. Deler av læringsprosessen er basert på tilfeldigheter, og resultatene vil variere noe for hver test.

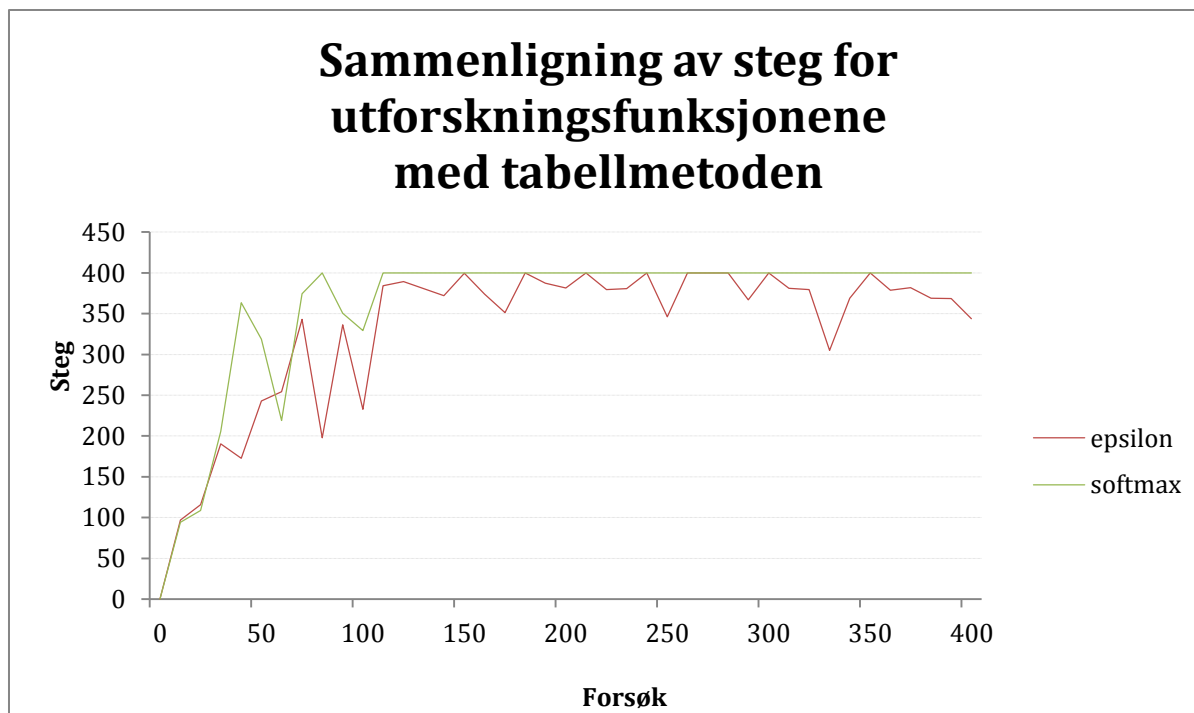
Ett forsøk består av et begrenset antall steg, og avbrytes ved oppnåelse av antallet steg eller kollisjon.

5.2 Utforskningsfunksjon

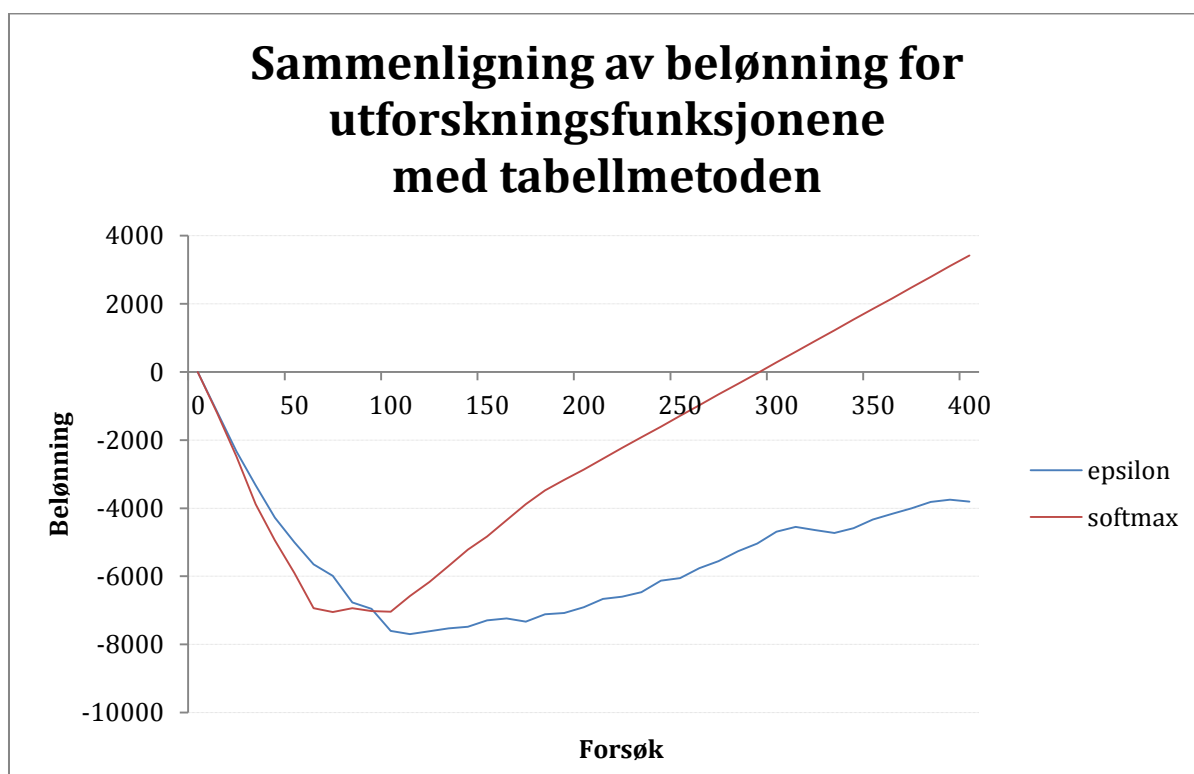
Grafene i figur 5-1 sammenligner de to utforskningsfunksjonene epsilon og softmax. Verdien for gamma er satt til 0.7, ut ifra et ønske om at agenten skal være langsiktig. Alfa er satt etter prøving og feiling til 0.1. Begge metodene er implementert slik at de gradvis synker sannsynligheten for tilfeldige handlinger for hvert forsøk. Softmax er implementert ved $T = 24$, og synker med 5% for hvert forsøk ned til $T = 0.01$. Epsilon er implementert ved $e = 0.95$ og synker med 2% for hvert forsøk til $e = 0.05$. Begge metodene når denne verdien etter 150 forsøk.

Testen simulerer 400 forsøk med 400 steg for hver metode, men stegdiagrammet viser punkter som er et gjennomsnitt av hvert tiende forsøk og akkumulert belønning i belønningsdiagrammet. Dette gjøres for å jevne ut grafene og vise en trend fremfor de konkrete punktene. Resultatene for epsilon og softmax vises i figur 5-1 og figur 5-2.

Totalbelønning og antall steg per forsøk er simulert for både epsilon og softmax. Figur 5-1 viser at softmax konvergerer raskere mot en løsning. Når en mulig løsning er funnet fortsetter funksjonen å bruke denne. Fordi epsilon etter utforskning fremdeles velger 5% tilfeldige handlinger vil agenten heller ikke stabilisere seg på samme måte som softmax. Softmax har stabilt 400 steg per forsøk etter å ha konvergert til en løsning og har større stigningstall på grafen for akkumulert belønning enn epsilon (figur 5-2). Dette viser at softmax har funnet en bedre løsning. Figur 5-2 viser at systemet er stabilt når grafen for belønning vokser jevnt. Softmax antas å ligge til grunn i de videre resultatene.



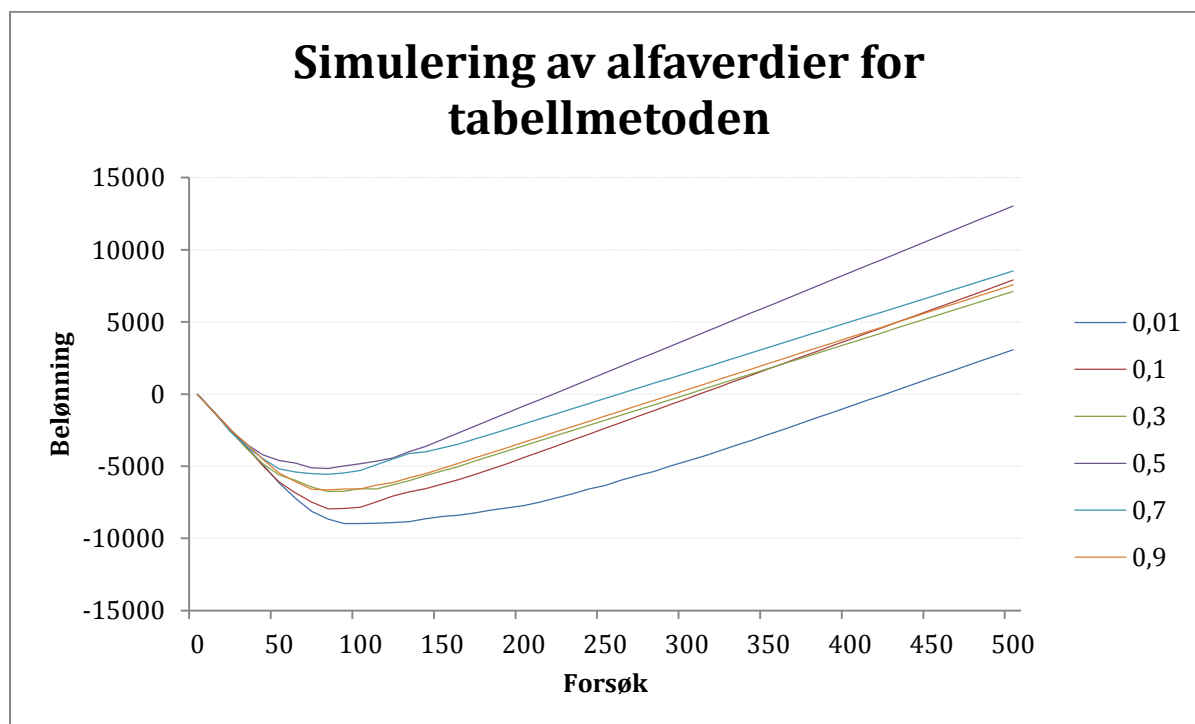
Figur 5-1 – Sammenligning av steg for utforskningsfunksjonen med tabellmetoden



Figur 5-2 – Sammenligning av belønning for utforskningsfunksjonene med tabellmetoden

5.2.1 Alfa og gamma

Det er ønskelig at agenten er langsiktig ettersom den ikke har noen slutterminal og systemet skal simuleres over lang tid. Derfor er gamma fastsatt til 0.9, og simuleringer for syv ulike verdier av alfa er vist i figur 5-3. Simuleringen ble gjort over 400 steg og 500 forsøk for hver alfaverdi. Grafene i figur 5-3 viser et gjennomsnitt av tre tester for å jevne ut tilfeldigheter i målingene og vise en trend fremfor enkeltmålinger. Alfa- og gammaparametrene påvirker hverandre, men det er for dette prosjektet satt en statisk gamma for å ha et system som prioriterer handlinger lengere frem i tid.



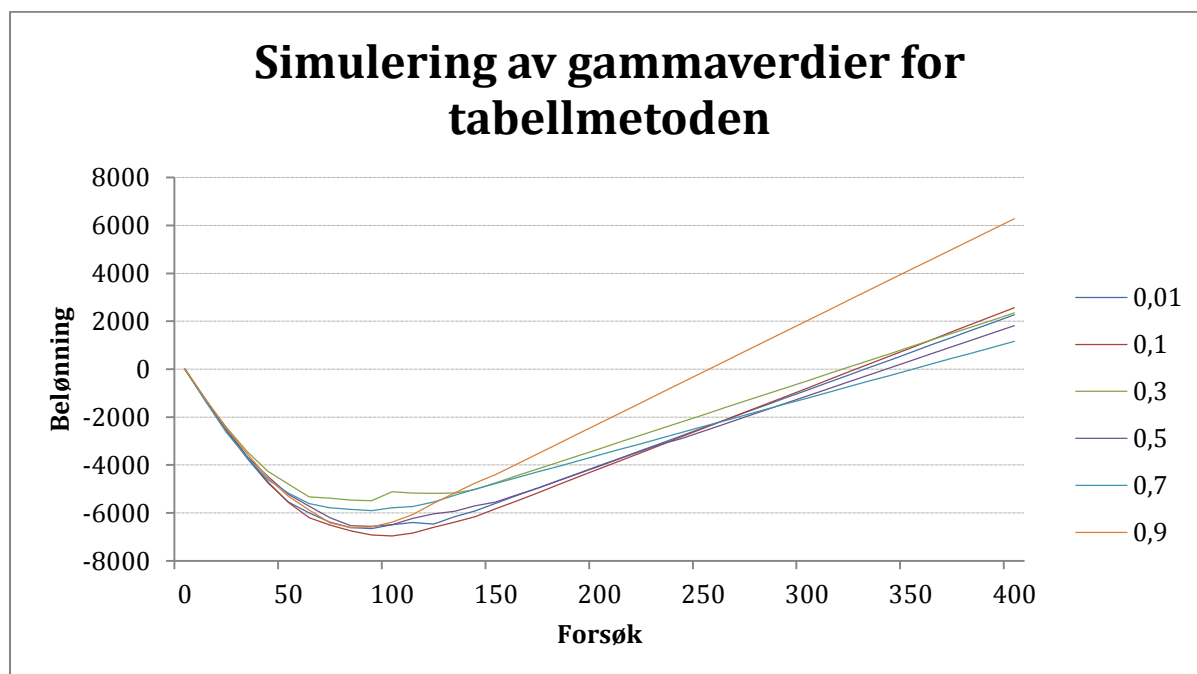
Figur 5-3 – Simulering av alfaverdier for tabellmetoden

Tabell 9 – Resultat fra simulering av alfaverdier for tabellmetoden

Alfaverdier	Totalt antall krasj	Totalt antall steg	Total belønning
0,01	84,3	178891,7	3076,5
0,1	59,0	185209,3	7899,1
0,3	47,0	189367,0	7113,1
0,5	40,0	191102,0	13039,5
0,7	40,0	190499,3	8527,6
0,9	38,7	190713,7	7568,5

Tabell 9 viser totalt antall krasj, totalt antall steg og total belønningsverdi basert på gjennomsnittet av tre tester. Basert på tabellen og figur 5-3 er 0.5 den alfaverdien som gir best belønning og flest antall steg. For videre testing med tabellmetoden brukes alfa 0.5.

Gammaverdier ble simulert etter lik fremgangsmåte som for alfa, men med alfa satt til 0.5. Figur 5-4 viser samlet belønning for de ulike gammaverdiene. Grafene i figur 5-4 viser gjennomsnittet av tre målinger for hver gammaverdi.



Figur 5-4 – Simulering av gammaverdier for tabellmetoden

Tabell 10 – Resultater fra simulering av gammaverdier for tabellmetoden

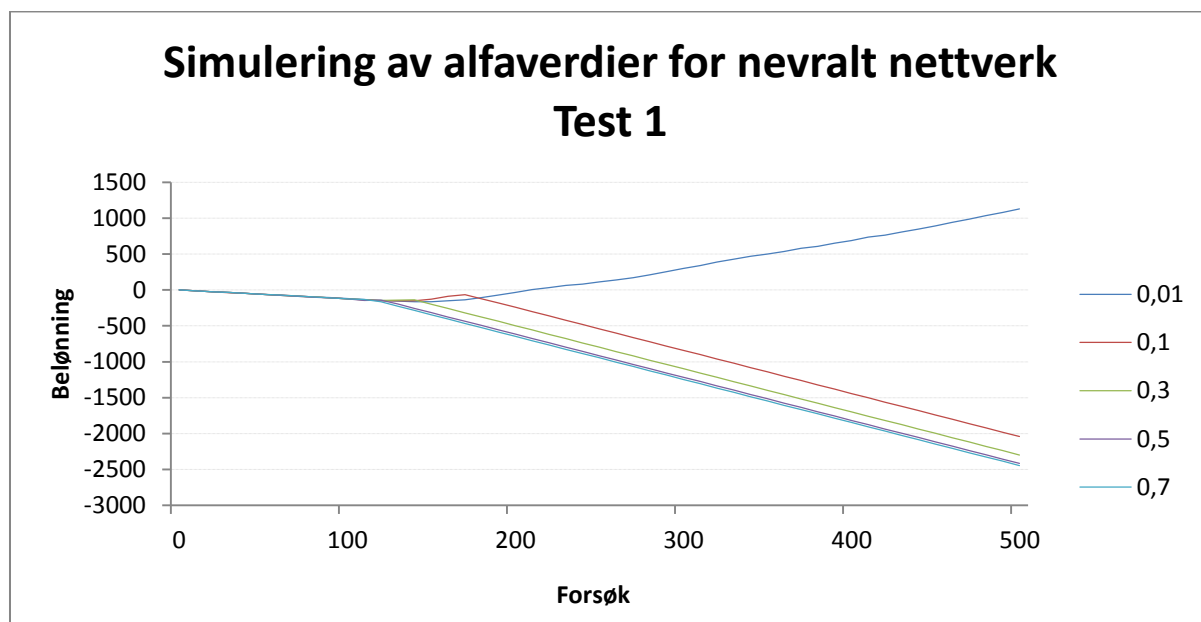
Gammaverdier	Totalt antall krasj	Totalt antall steg	Total belønning
0,01	52,3	161982,7	3343,6
0,1	47,0	162867,0	3671,8
0,3	43,7	164160,0	3442,7
0,5	41,3	164503,7	3057,4
0,7	36,0	165349,7	2248,0
0,9	29,7	167224,0	7425,1

Tabell 10 viser en oversikt over totalt antall krasj, totalt antall steg og total belønningsverdi for de ulike gammaverdiene i simuleringen. I dette systemet er gamma 0.9 den verdien som gir høyest samlet belønning, færrest antall krasj og totalt flest antall steg. Grafen for gamma 0.3 har et høyere minimumspunkt i belønningsverdien enn gamma 0.9, men et lavere stigningstall. Som grunnlag i videre testing brukes derfor gamma 0.9.

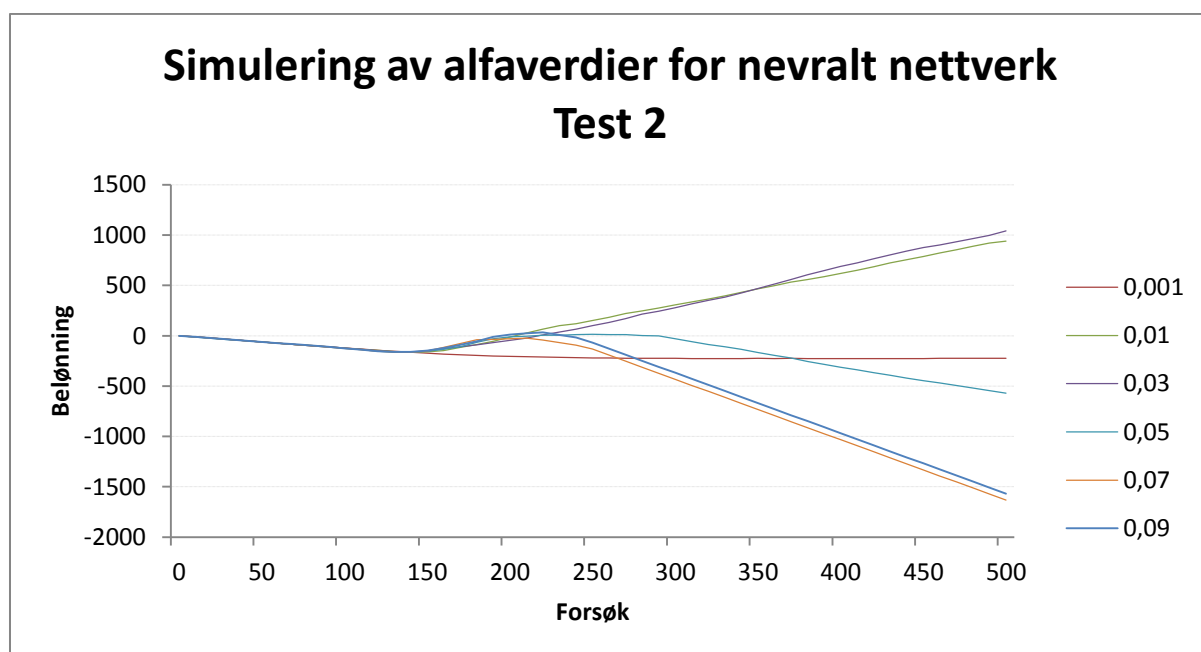
5.3 Nevralt nettverk

5.3.1 Alfa

Grafene i figur 5-5 viser totalbelønning for de ulike verdiene av alfa. Simuleringen er realisert ved bruk av nevralt nettverk. Gamma er satt til 0.9 av samme grunn som for Q-læring, og simuleringen er for 600 steg og 500 forsøk. Det er nødvendig å gjøre mer nøyaktig testing for å verifisere at resultatet ikke er 0. Et resultat på 0 betyr at systemet ikke tilegner seg ny kunnskap. Figur 5-6 viser simulering for alfaverdier i nærheten av den beste verdien fra test 1 som ble 0.01.



Figur 5-5 - Simulering av alfaverdier for nevralt nettverk (test 1)



Figur 5-6 – Simulering av alfaverdier for nevralt nettverk (test 2)

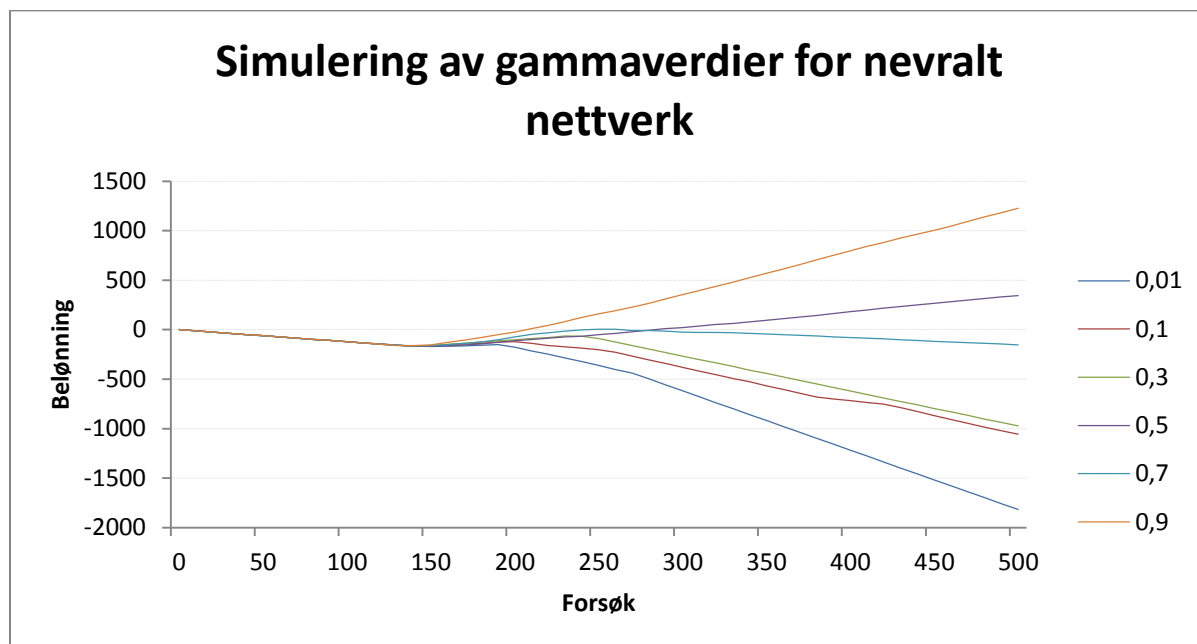
Tabell 11 – Resultat fra simulering av alfaverdier for nevralt nettverk

Alfaverdier	Totalt antall krasj	Totalt antall steg	Total belønning
0,01	496,3	53960,3	-222,8
0,1	309,3	177232,0	940,8
0,3	250,3	197879,7	1042,1
0,5	201,7	216721,3	-570,9
0,7	151,0	231607,0	-1632,9
0,9	148,0	233147,0	-1568,5

Tabell 11 viser en oversikt over totalt antall krasj, totalt antall steg og total belønningsverdi for de ulike alfaverdiene basert på gjennomsnittet av tre tester. Det er gjort tre målinger for å jevne ut grafene Ved simulering (figur 5-6) for det gitte antall steg og forsøk er 0.3 den verdien som gir størst total belønning. Det er ikke den verdien som gir færrest antall krasj eller flest steg, men for det praktiske formål er stigningen på belønningsverdien og belønningsverdien avgjørende for valg av verdi. Videre simulering for gammaverdier forutsetter en gitt alfaverdi. For videre testing brukes derfor alfa 0.3.

5.3.2 Gamma

Figur 5-7 viser en simulering for ulike gammaverdier. Testen er simulert med 600 steg og 500 forsøk. Tabell 12 viser en oversikt over totalt antall krasj, totalt antall steg og samlet belønning for de ulike gammaverdiene.



Figur 5-7 - Total belønning Q for ulike verdier av Gamma for Neural Network

Tabell 12 – Resultat fra simulering av gammaverdier for nevralt nettverk

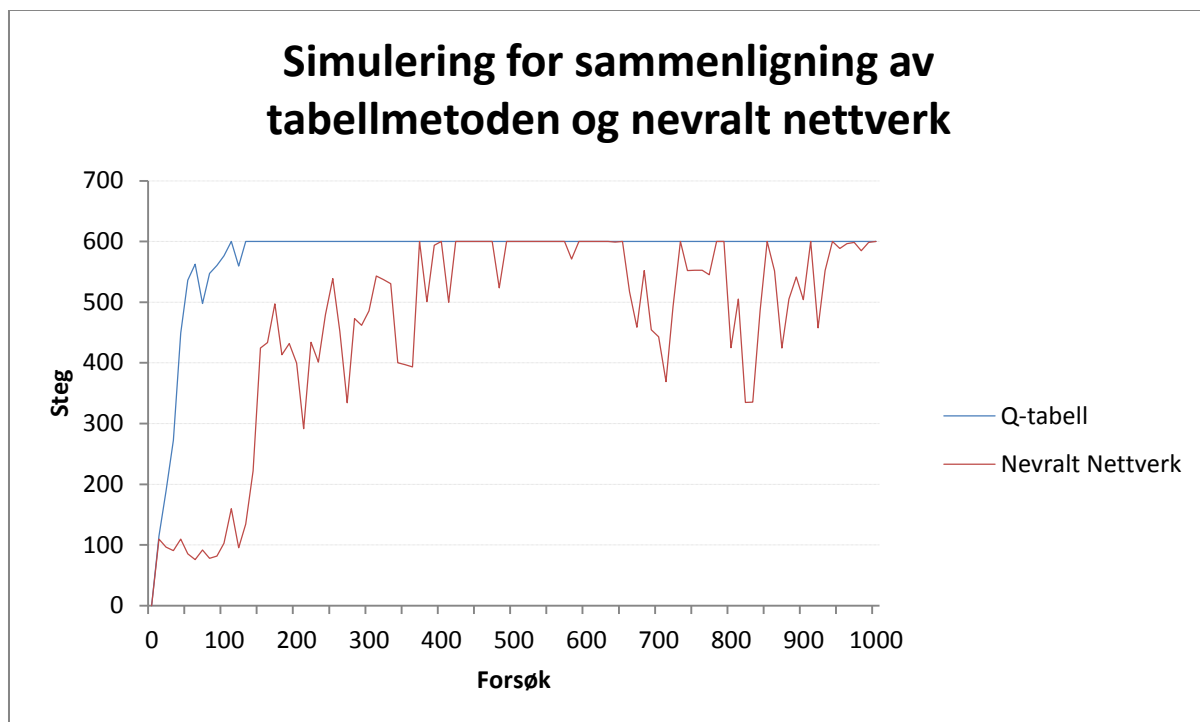
Gammaverdier	Totalt antall krasj	Totalt antall steg	Total belønning
0,01	205	201 986	-1 818
0,1	302	159 603	-1 057
0,3	301	176 135	-973
0,5	464	126 349	345
0,7	350	166 261	-153
0,9	195	216 209	1 229

5.3.3 Sammenligning

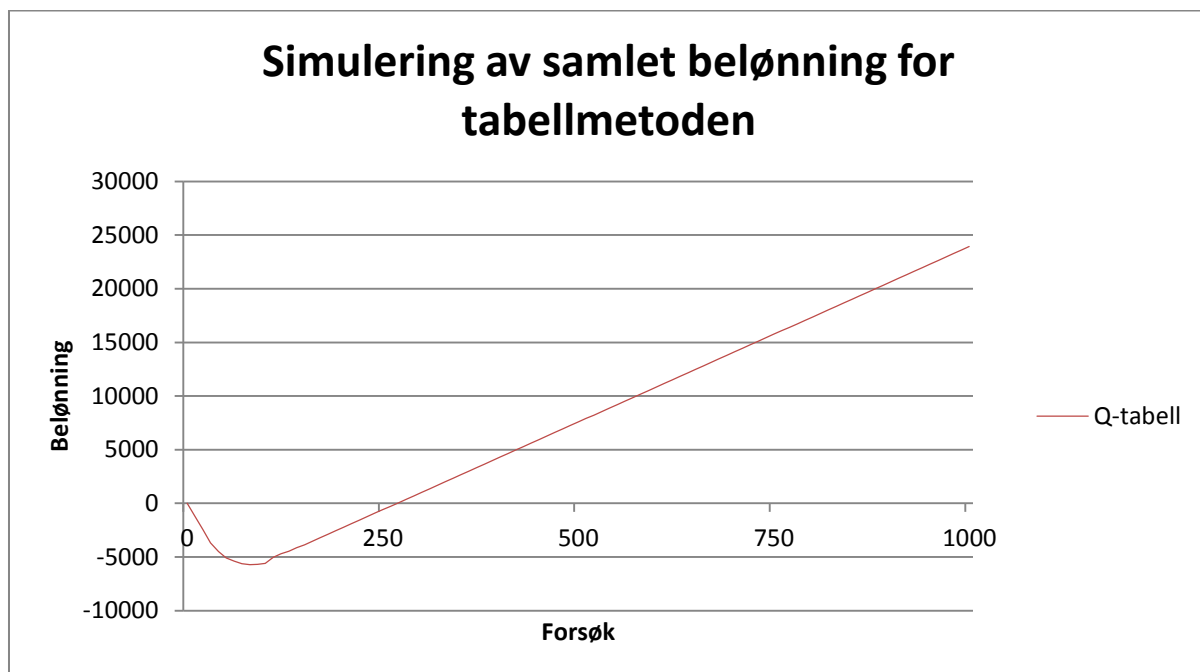
Tabell 13 viser totalt antall krasj og totalt antall steg for tabellmetoden og nevralt nettverk. Figur 5-8 sammenligner de to metodene basert på antall steg per forsøk. Beste verdier for alfa (0.5, 0.03/Q-tabell, nevralt nettverk) og gamma (0.9, 0.9/tabellmetoden, nevralt nettverk) for begge metodene er brukt i simuleringen. Nevralt nettverk og tabellmetoden er implementert med ulike belønningsfunksjoner. Belønningsfunksjonene og forskjellen mellom disse er beskrevet nærmere i kapittel 4.5. Simuleringene for akkumulert belønning for de to metodene er vist i to ulike figurer (5-9 og 5-10) for å bedre lesbarheten.

Tabell 13 – Resultat fra simulering av sammenligning av tabellmetoden og nevralt nettverk

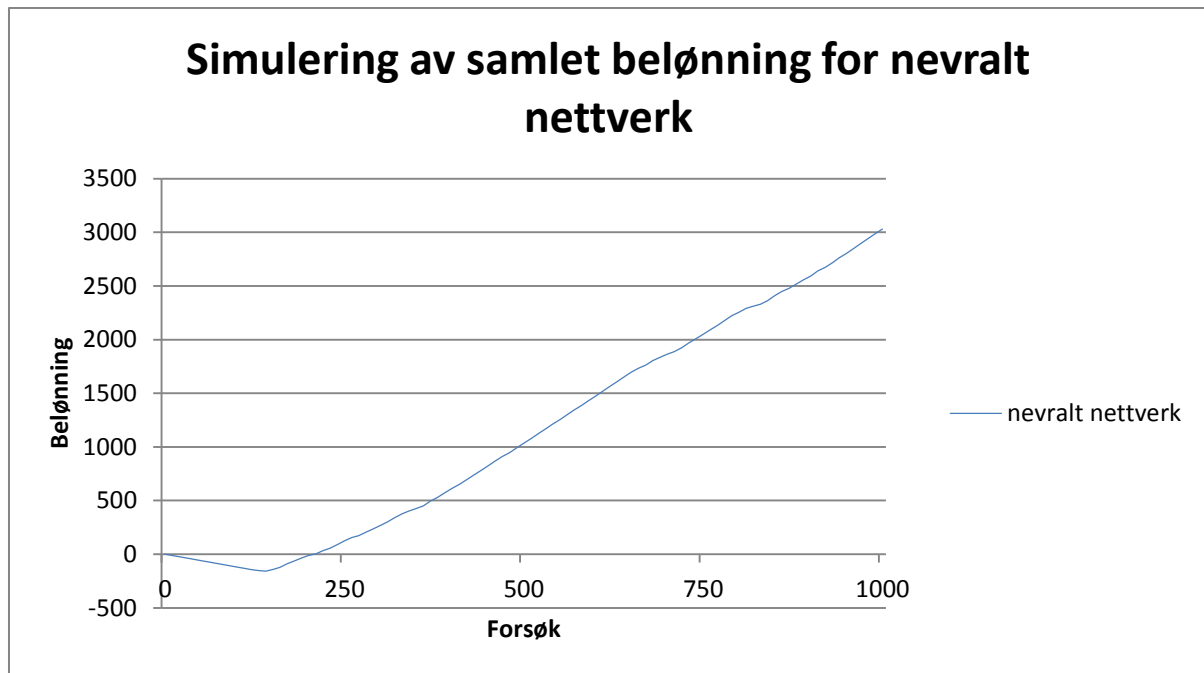
Metode	Totalt antall krasj	Totalt antall steg
Q-tabell	48	582664
Nevralt nettverk	299	466588



Figur 5-8 – Simulering for sammenligning av tabellmetoden og nevralt nettverk

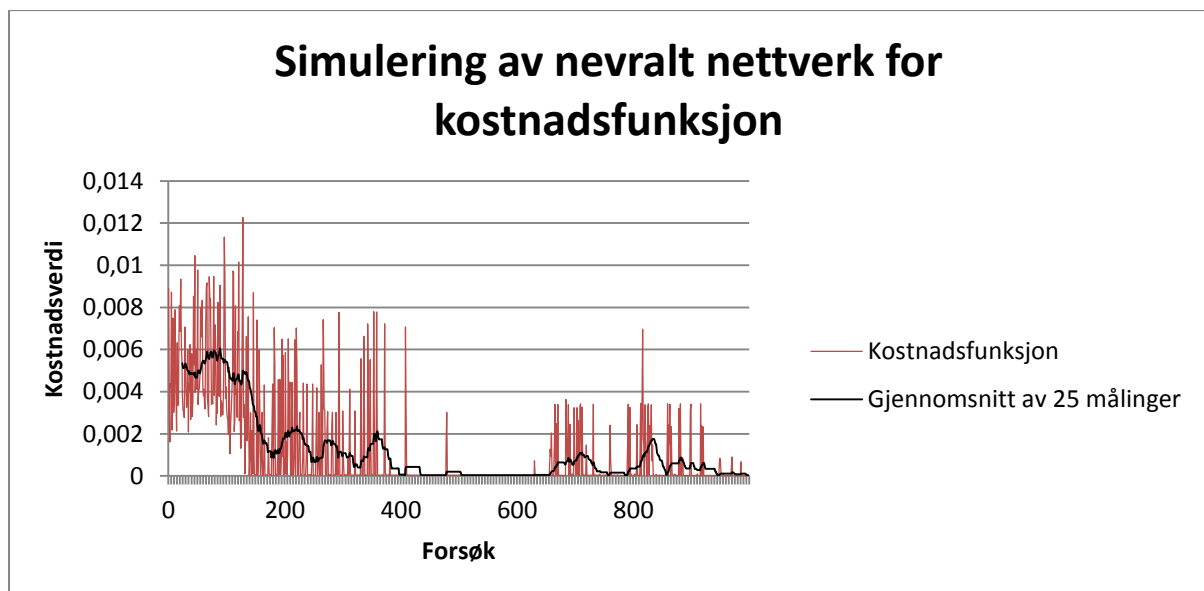


Figur 5-9 - Simulering av samlet belønning for tabellmetoden



Figur 5-10 - Simulering av samlet belønning for nevralt nettverk

Figur 5-11 viser summen av kostnadsfunksjonen for hver av de tre Q-funksjonene i nevrale nettverk. Beste verdiene for alfa (0.3) og gamma (0.9) er brukt i simuleringen som er kjørt 1000 forsøk. Den sorte grafen i figur 5-11 viser et gjennomsnitt av 25 forsøk for å illustrere en trend fremfor enkeltmålinger.



Figur 5-11 – Simulering av kostnadsfunksjon for nevralt nettverk

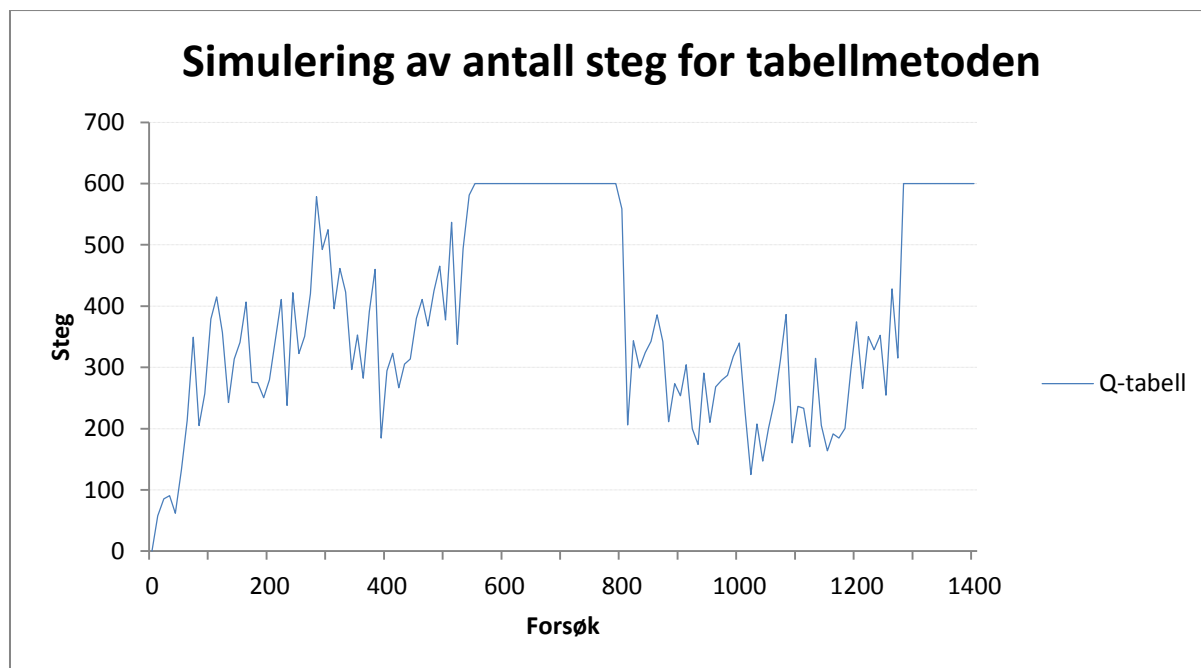
5.4 Simulering av system med dynamiske hindringer

Testene er utført med objekter som roterer i samme retning med konstant fart. Etter 800 forsøk endres retningen på de roterende hindringene. Simuleringen er kjørt med 500 steg og 1400 forsøk for begge metodene. Tabell 13 viser en oversikt over totalt antall krasj og totalt antall steg. Figur 5-12 og 5-13 viser simulering for antall steg og akkumulert belønning for Q-tabell, mens figur 5-14 og 5-15 viser det samme for nevralt nettverk. Samme verdier for alfa og gamma som presentert i kapittel 5.3.4 er benyttet ved simulering.

Figur 5-16 viser summen av kostnadsfunksjonen for hver av de tre Q-funksjonene i nevralt nettverk. Den sorte grafen i figur 5-16 viser et gjennomsnitt av 25 forsøk for å illustrere en trend fremfor enkeltmålinger.

Tabell 14 - Oversikt over totalverdier for dynamisk simulering

Metode	Totalt antall steg	Totalt antall krasj
Q-tabell	541211	384
Nevralt nettverk	287066	626



Figur 5-12 - Simulering av antall steg for tabellmetoden

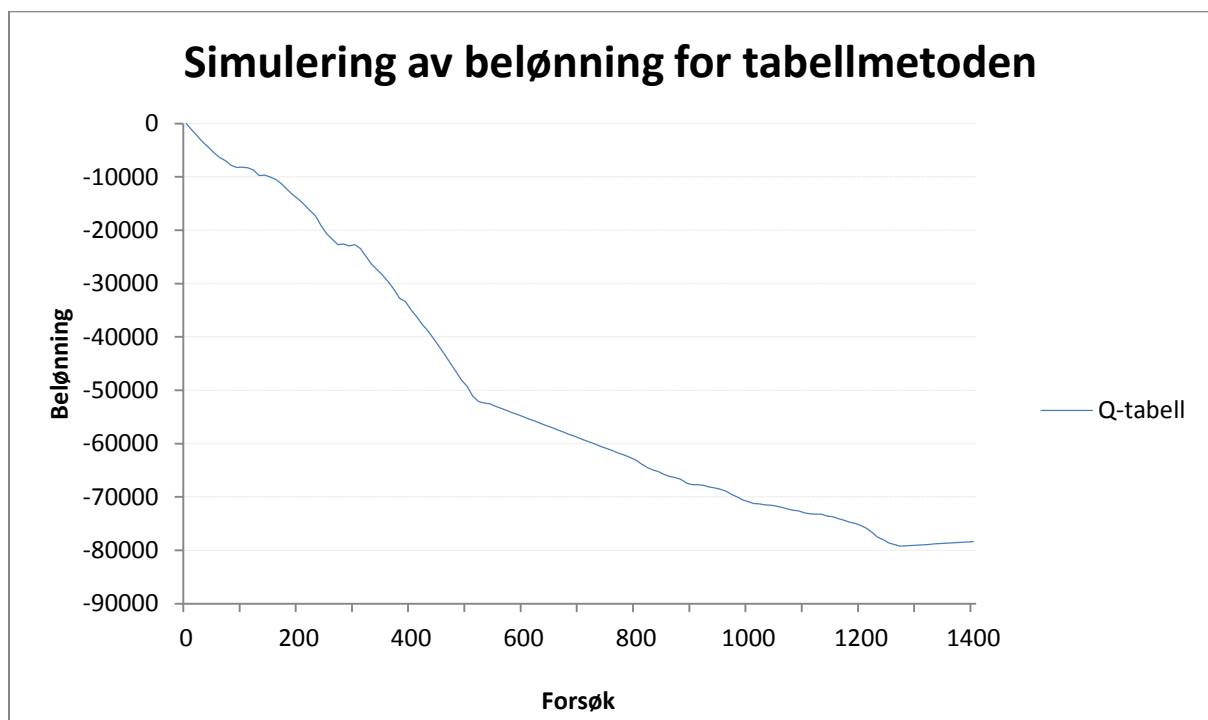
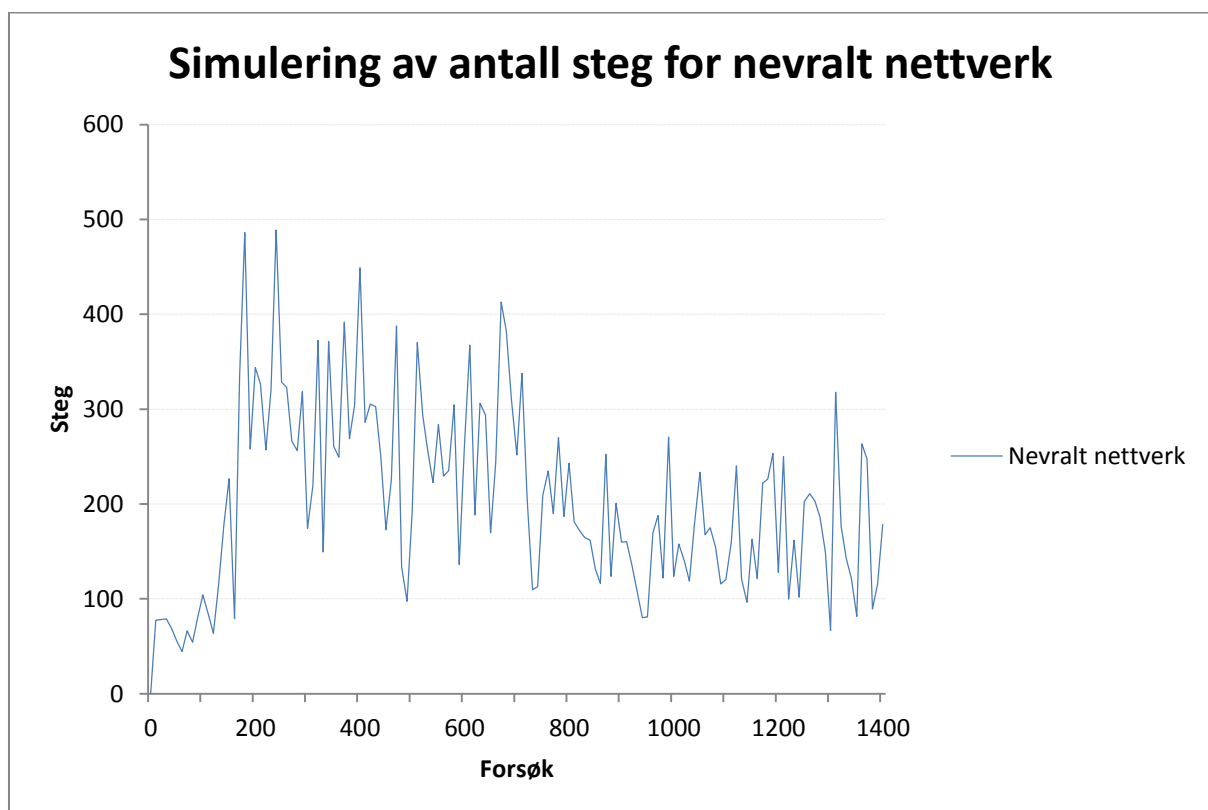
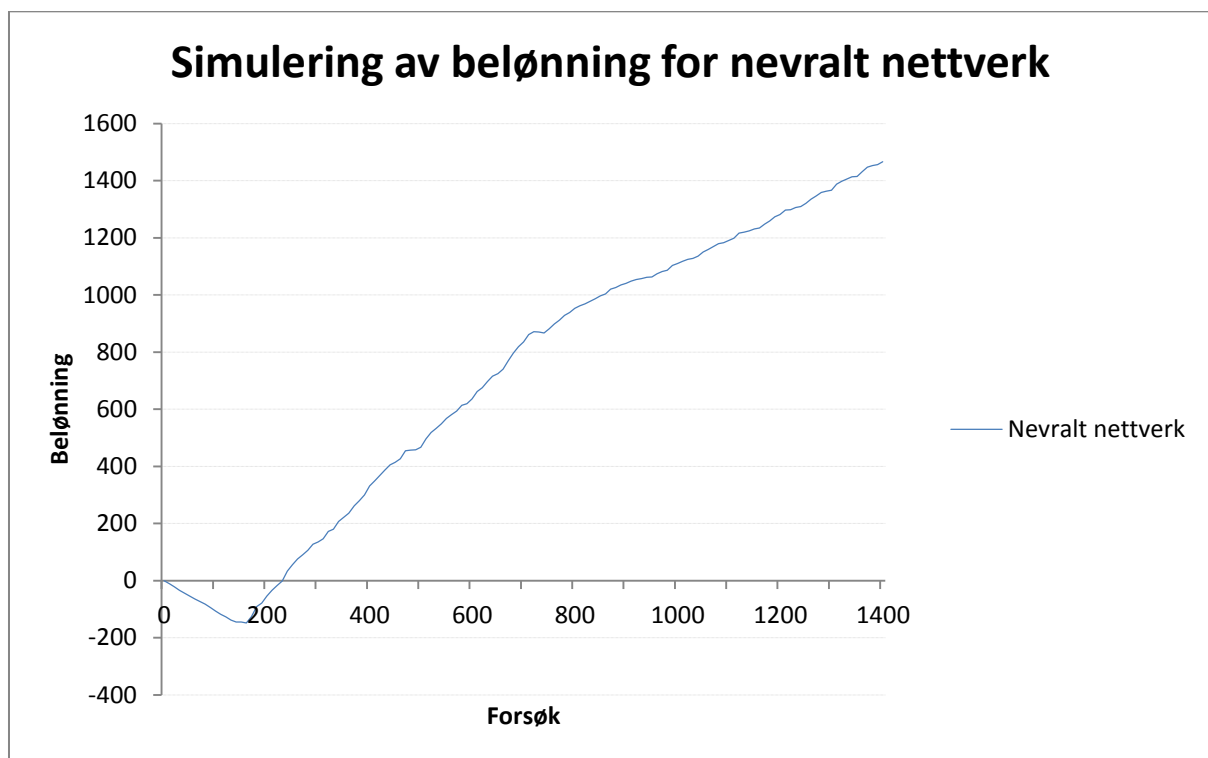


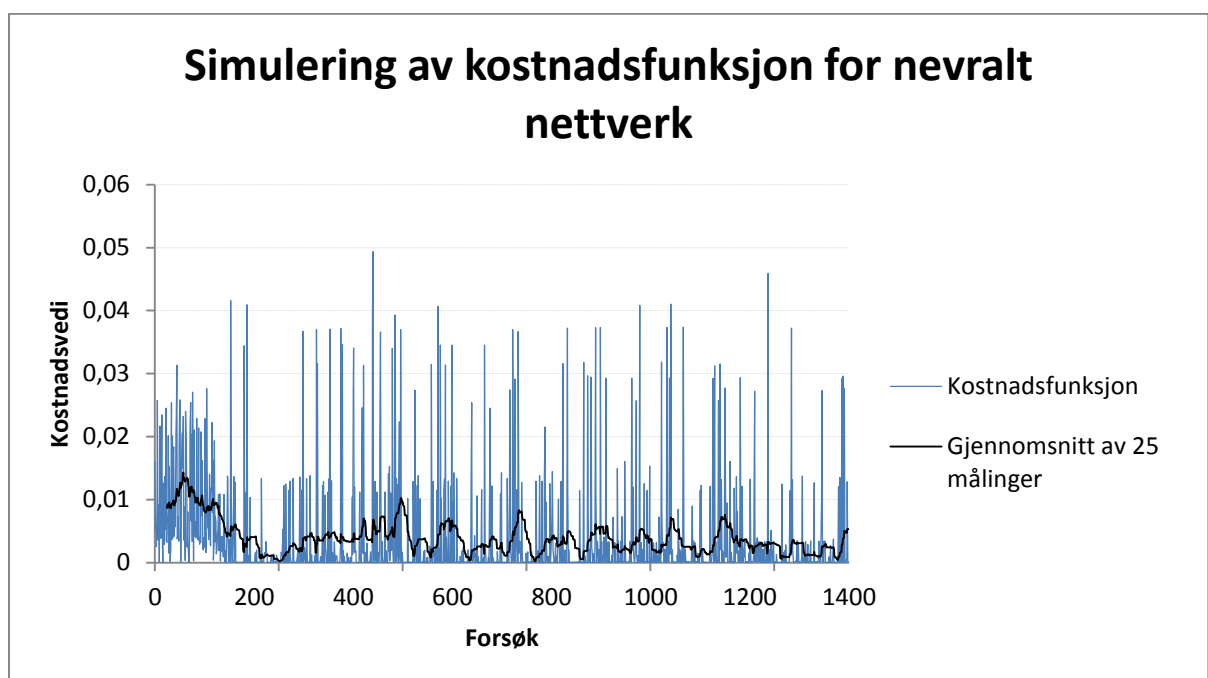
Figure 5-13 - Simulering av belønning for tabellmetoden



Figur 5-14 – Simulering av antall steg for nevralt nettverk

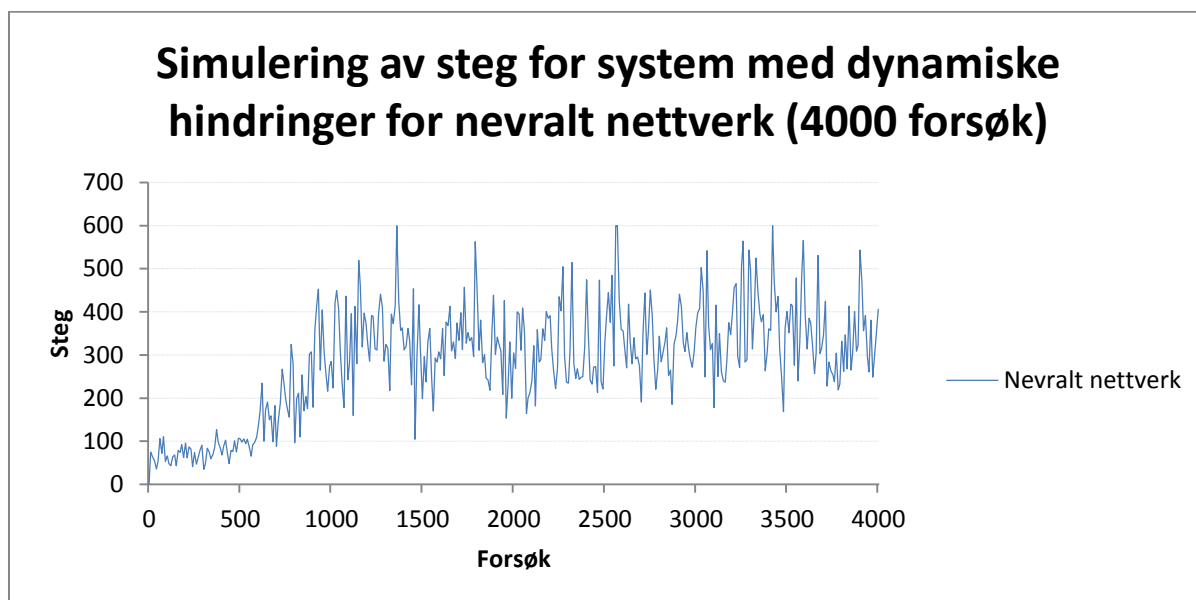


Figur 5-15 – Simulering av belønning for nevralt nettverk

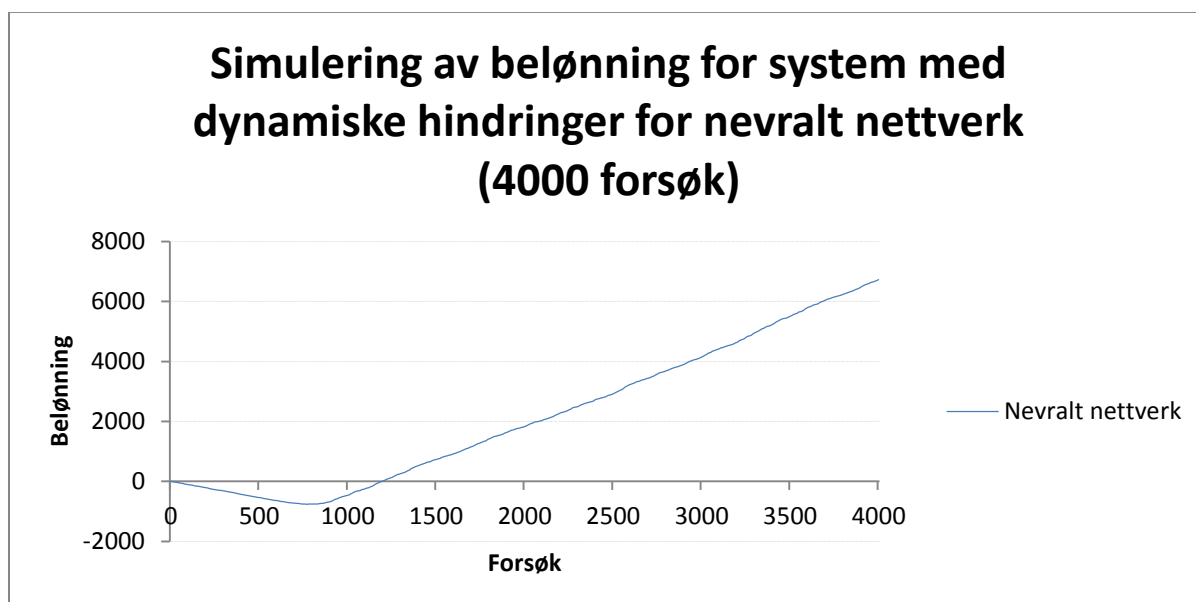


Figur 5-16 – Simulering av kostnadsfunksjon for nevralt nettverk

Simulering av akkumulert belønning (figur 5-17), totalt antall steg (figur 5-18) og kostnadsfunksjonen (figur 5-19) for nevralt nettverk med dynamiske hindringer er inkludert for å finne ut om systemet konvergerer til en løsning. Simuleringen er gjort over 4000 forsøk for å se om nevralt nettverk finner en stabil løsning i løpet av flere forsøk enn for øvrige simuleringer.

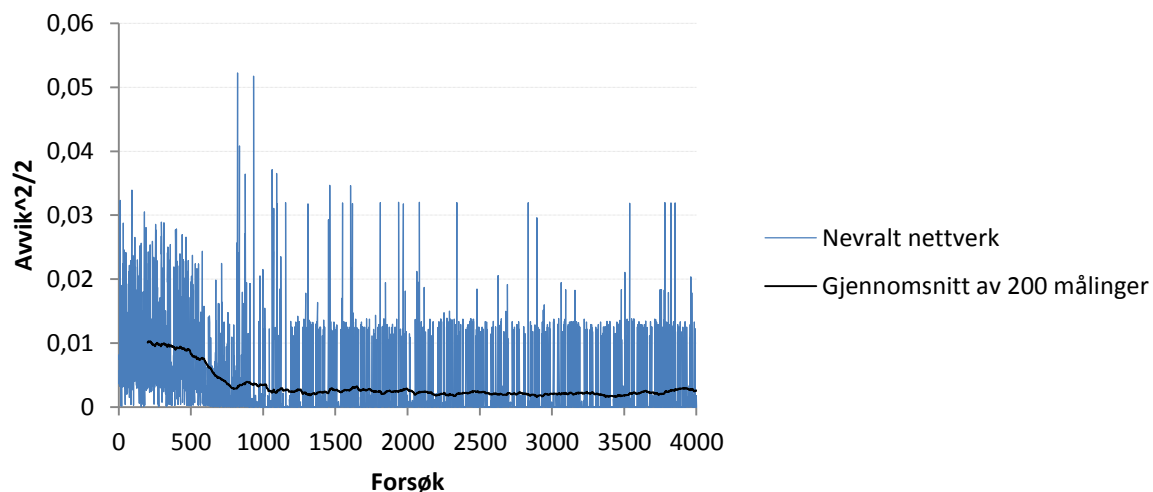


Figur 5-17 - Simulering av steg for system med dynamiske hindringer for nevralt nettverk (4000 forsøk)



Figur 5-18 - Simulering av belønning for system med dynamiske hindringer for NN (4000 forsøk)

Simulering av kostnadsfunksjon for system med dynamiske hindringer for nevralt nettverk (4000 forsøk)



Figur 5-19 - Simulering av kostnadsfunksjon for system med dynamiske hindringer for NN (4000 forsøk)

6 Diskusjon

6.1 Sammenligning av demonstrator og simulering

Simuleringen er laget mest mulig lik de faktiske forhold agenten ville møte i den fysiske arenaen. Størrelsen på arenaen og agenten i simuleringen er proporsjonale med forholdene i den fysiske demonstratoren. Allikevel er det noen elementer, som for eksempel friksjon og strømbegrensninger, som gjør at de to systemene er ulike. I simuleringen er systemets hastighet kun begrenset av datamaskinens ressurser. I demonstratoren kommer det til en hastighet hvor agenten vil mistet grepet på kjørebane eller situasjoner der agenten vil kjøre for langt mellom hver sensormåling. Den fysiske modellen er begrenset av batterikapasitet, i motsetning til simuleringen som vil kjøre så lenge det er tilgang på strøm.

Et konkret eksempel på en ulikhet er løsningen for dynamiske hindringer i de to modellene. Dette illustreres best ved først å beskrive simuleringen, for deretter å sammenligne den med den fysiske modellen. Et program utføres sekvensielt. Det vil si at én funksjon kjøres av gangen, men ofte så fort at det oppleves som om alt skjer samtidig. Agenten må reagere tidligere for å unngå dynamiske hindringer med retning mot den, enn den må for statiske. For å finne ut om en hindring er i bevegelse eller statisk stoppes agenten. Deretter gjøres to målinger. Basert på om avstanden har økt, er redusert eller uendret på de to målingene vil agenten skille ut objekter i bevegelse før den kjører videre og prosessen gjentas. Simuleringen oppleves allikevel som sømløs fordi denne sekvensen gjøres mange ganger i sekundet. I den fysiske modellen ville samme fremgangsmåte trolig fungert like godt, men agentens bevegelse ville vært preget av å kontinuerlig starte og stoppe.

Så snart mennesket introduseres i et system vil unøyaktighet være et faktum. Menneskets bevegelser er ikke like repeterbare og nøyaktige som hos en datamaskin. Det har i løpet av prosjektet blitt laget en demonstrasjonsvideo (vedlagt på CD) hvor agenten plasseres på et avmerket sted i arenaen for deretter å kjøre mot arenaveggen og over flere forsøk lære seg å unngå disse veggene. Etter hvert forsøk flyttes agenten tilbake til startpunktet sitt før den begynner neste forsøk. I simuleringen hadde dette punktet blitt nøyaktig det samme hver gang, mens det i den fysiske modellen avhenger dette av hvor mennesket plasserer agenten. Sted og vinkel i forhold til veggen vil trolig variere for hver gang, og med det introdusere agenten for andre situasjoner enn det som er meningen i forsøket.

6.1.1 Sensorer

Agentens fem ultrasoniske sensorer utgjør dens synsfelt. Sensorene påvirkes av støy og andre faktorer som gjør at målingene i noen situasjoner vil gi feil verdier. Sensoren oppfatter en hindring når vinkelen mellom agenten og hinderets overflate ikke overskrider $\pm 20^\circ$ normalt på agenten. I den fysiske modellen var det i noen situasjoner vanskelig for agenten å oppfatte hindringene på grunn av

denne faktoren. Uten å ha utført tester antas det at hindringenes overflate kan ha noe å si for evnen til å reflektere sensorens lydbølger.

Ulike frekvenser gir ulik avbøying av lydbølgene på en overflate. En løsning på problemet ovenfor kan derfor være enten å endre frekvensområde, måle på flere frekvenser eller benytte en helt annen metode. Andre metoder kan for eksempel være kamera, andre sensorer (f.eks IR) eller ferdige sensorløsninger som for eksempel Kinect fra Microsoft.

Agentens synsfelt er begrenset av fem sensorer i front. Agenten får ingen informasjon om hindringer utenfor synsfeltet. Dette kan medføre kollisjon ved at agenten svinger mot hindringen den ikke ser. Dette vises ofte i den fysiske modellen. Problemet kunne vært løst ved å montere flere sensorer som utvider synsfeltet, for eksempel bak og på siden av agenten, men flere sensorer fører til flere tilstander (større Q-tabell). Dette vil igjen medføre større krav til prosesseringen av data.

Sensorene i simuleringen er teknisk sett feilfrie og gir et direkte utslag med en gang en hindring kommer i veien, uavhengig av vinkel og overflate. De fysiske faktorene kunne til en viss grad vært inkludert i simuleringen for å redusere ulikhetene mellom den fysiske og den simulerte modellen.

6.2 Dynamiske hindringer

Dynamiske hindringer har som tidligere nevnt vært utfordrende i den fysiske modellen. Agenten ser avstanden til hindringene, men ikke hvilken retning og hastighet de har. Fordi agenten ikke vet hvilken vinkelretning hinderet har i forhold til agenten er det vanskelig å si om den utgjør en kollisjonsfare.

En tenkelig løsning ville vært å benytte seg av konseptet Computer Vision (CV). CV handler om å analysere og tolke bilder gjennom videoprosessering. Ved hjelp av ett enkelt kamera i front ville agenten dermed sett omtrent like mye som den nå gjør med de fem sensorene. Ettersom hindringene ikke ville beveget seg fra en sektor til en annen, men istedet forblitt i samme bilde, ville det trolig gjort det enklere å si noe om hinderets retning og hastighet.

Andre løsninger kunne vært å gi de dynamiske hindringene en annen farge enn de statiske (for å skille ved hjelp av fargesensorer), eller utstyre hindringene med sendere som opplyser agenten om de ulike hindringenes posisjon. På bakgrunn av egen fart og retning ville agenten kunne se en retning på hindringene, relativ til seg selv, og gjøre den handlingen som må til for å unngå hinderet.

Grunnet prosjektets tidsbegrensning og omfang ble det benyttet ultrasoniske sensorer. Disse krever, i motsetning til for eksempel CV, lite ressurser og opplæring.

6.3 Metodevalg

Tidsperspektiv og anvendelsesområder for systemet er viktig når metode og parametere skal velges. Noen metoder krever mye opplæring for å konvergere til en god løsning, mens andre bruker mindre tid på å konvergere til en løsning med mindre potensiale på sikt. I de fleste systemer må parametere og metoder tilpasses sitt bruk, og valget inneholder ofte et kompromiss mellom læringsrate, langsiktighet, nøyaktighet og stabilitet.

6.3.1 Feillæring og tilfeldigheter

Ulike valg av parametere stiller også ulike krav til nøyaktighet og stabilitet i et system. Et problem som kan oppstå i den fysiske modellen på grunn av fysiske forutsetninger eller støy, er feillæring. Feillæring oppstår når agenten får belønning for handlinger som ikke er en god handling, eller straff for riktig handling i den gitte situasjonen. Det kan ta lang tid å gjenopprette slike feil. Fysiske tilpasninger er nødvendig for hvert enkelt system og ved hjelp av diskretisering av inndata kan feillæring til en viss grad unngås. I simuleringen vil ikke dette problemet oppstå.

Systemløsningene beskrevet i denne rapporten vil alltid påvirkes av tilfeldige valg av handlinger. Det er ønskelig med et system som utforsker lenge nok til at det konvergerer til den beste løsningen. Diagrammene i resultatkapittelet viser at utvikling i belønning og steg over tid vil variere med hver simulering, og det viser seg å være mest hensiktsmessig å se på statistikk over totalt antall krasj og totalt antall steg for vurdering og sammenligning.

6.3.2 Utforskningsfunksjon og valg av parametere

Softmax konvergerer raskere til en løsning og gir en større akkumulert belønning enn epsilon (figur 5-2). Det er også den funksjonen som gir mest stabile verdier ved simulering av antall steg per forsøk (figur 5-1), og er dermed den beste utforskningsfunksjonen for dette systemet.

Simuleringer for alfa- og gammaverdier blir presentert i resultatkapittelet. For tabellmetoden er ikke valg av alfa- og gammaverdier kritisk, og systemet vil konvergere til en løsning for alle gyldige verdier (vist i figur 5-3 og figur 5-4). Resultatene for nevralt nettverk (5.3.1 Alfa og 5.3.2 Gamma) viser at alfa- og gammaverdiene er avgjørende for å finne en stabil løsning.

Resultatene for alfa- og gammaverdiene viser ikke nødvendigvis optimale verdier, men nøyaktigheten er vurdert å være tilstrekkelig. Dette er vist gjennom simulering for nevralt nettverk, samt simulering og fungerende demonstrator for tabellmetoden.

6.3.3 Diskretisering

Enkel og effektiv kategorisering av inndata er viktig og essensielt for å gjøre et system effektivt. Diskretiseringen beskrevet i kap. 4.2 viser seg å være et godt kompromiss mellom nøyaktighet og effektivitet for den statiske løsningen. Systemets diskretisering for dynamiske hindringer fungerer, men kunne vært optimalisert ved for eksempel å diskretisere hindringenes hastighet og/eller retning.

Et dynamisk system inneholder en rekke flere momenter enn statiske, og har derfor større krav til nøyaktige inndata og diskretisering. Det er nødvendig å vite agenten og hinderets hastighet, i tillegg til hinderets retning.

6.3.4 Prosjektets resultater

Etter utforskningsperioden velger metodene beste erfarte handlinger. Graden av økning i belønning beskriver hvor god løsningen er. Hvor jevnt grafen stiger eller synker, i tillegg til totalt antall steg per forsøk, viser derimot i hvilken grad løsningen er stabil. Grafen i figur 5-9 viser at maksimalt antall steg per nevralt nettverk oppnås etter ca 380 forsøk. Metoden beholder ikke denne løsningen grunnet valgt oppbygning av nettverket og valgte parametere. Tabellmetoden konvergerer til en stabil løsning etter ca 150 forsøk, og vil for dette systemet raskere nå en stabil løsning (figur 5-9). Etter at agenten har funnet en fungerende løsning vil den ikke fortsette å utforske miljøet som ville gjort at den finner andre eller bedre løsninger. Dette er en utfordring ved bruk av tabellmetoden.

Simuleringen av system med dynamiske hindringer er gjennomført ved at hindringene roterer i samme retning for deretter å endre retning etter 800 forsøk. Simulering viser at tabellmetoden finner en løsning etter 400 forsøk, men denne er ikke optimal ettersom belønningsverdien kontinuerlig synker (figur 5-12 og figur 5-13). Nevralt nettverk finner en løsning, men løsningen er ikke stabil (figur 5-14). Figur 5-16 viser at kostnadsfunksjonen ikke konvergerer mot 0, og at systemet derfor ikke finner en stabil løsning.

Belønningsfunksjonene for de to metodene er ulike, og for å se om løsningen er god, må belønningsverdien for den dynamiske simuleringen sammenlignes med simulering av statiske for samme metode.

Simulering av 4000 forsøk er gjennomført for å se om nevralt nettverk konvergerer til en stabil løsning over tid (figur 5-17, 5-18 og 5-19). Simuleringen viser at metoden i løpet av de 4000 forsøkene ikke finner en stabil løsning. Det antas at en en simulering over flere enn 4000 forsøk vil gi samme resultat, og at problemet derfor er relatert til metodens oppbygning og parametervalg.

6.3.5 Videre arbeid

Som videre arbeid i prosjektet er det naturlig å gjøre en vurdering på om tabellmetoden er en egnet løsning for et system med dynamiske hindringer, eller om andre løsninger, som for eksempel fuzzy logikk kunne fungert bedre. Fuzzy logikk kan håndtere en større mengde inndata og har ikke behov for like nøyaktig informasjon som tabellmetoden. Metoden krever derimot en større forståelse av sammenhengen mellom inn- og utdata for definering av logiske regler.

Det vil også være naturlig å vurdere om Q-læring kan implementeres ved bruk av metoden beslutningstre (eng: decision tree), eller andre former for funksjonsapproximasjon for å effektivisere systemet. Flerstegs Q-læring (eng: multi-step Q-learning) er metoder som er designet for å planlegge flere steg frem i tid, og dette er noe som kunne vært implementert for å bedre agentens prestasjon i form av mer langsiktig planlegging.

Av konkret videre arbeid kan det implementeres flere handlinger i form av grader av retningsendring eller ulike hastigheter for å gjøre agentens handlinger jevnere. Den selvlærende algoritmen kan effektiviseres med fokus på å korte ned tiden mellom hver sensormåling, og det kunne vært gjort mer omfattende testing og optimalisering av belønningsfunksjonen for hver av metodene. På grunn av prosjektets omfang er det ikke gjort testing av hvordan belønningsfunksjonen kunne vært tilpasset hver av de to modellene (fysisk og simulert).

Prinsippet bak den selvlærende algoritmen beskrevet i denne oppgaven, kunne vært utgangspunkt for systemer med et mer praktisk formål. Ved å være selvlærende kan agenten manøvrere og finne veien i ukjente omgivelser. Eksempler på praktiske bruksområder kan være: støvsugerroboter, geografisk kartlegging, detektering av miner, sosiale roboter, eller annet bruk der det er behov for et selvlærende systemer som opererer i et ukjent eller dynamisk miljø.

6.4 Uavhengig agent

I dagens løsning sendes sensorverdiene fra agenten til en datamaskin, hvor beregningene gjøres, før agentens neste handling returneres. Om man relaterer prosjektet til den virkelige verden er det naturlig å tenke seg at agenten med fordel kunne vært konstruert uavhengig. Løsningen vil være at datamaskinen som nå mottar, bearbeider og sender data integreres i selve agenten.

Dagens datamaskiner kan være både små og kraftige, og det finnes rimelige løsninger med lav vekt som i dag kan ha omtrent samme kapasitet som en bærbar datamaskin. Raspberry Pi er en slik lavkost datamaskin med integrert skjermkort og USB-porter. Den er på størrelse med en smarttelefon og har

samme prosessorkapasitet som en middels laptop. Denne kunne dermed vært implementert som en del av agenten.

Overføringsmediet Bluetooth Bee, som er benyttet i prosjektet, har en overføringshastighet på ca 300kb/s og kan med det overføre data raskt nok til både live-presentasjon av q-tabellen og eventuelle andre data fra demonstratoren. Dermed ville agenten vært uavhengig samtidig som data ville vært tilgjengelig.

Agenten ville trolig blitt noe større som følge av den tilførte elektronikken. Økningen i størrelse og tyngde ville igjen vært på bekostning av bevegelse, strømforbruk og arealstørrelse. Mer elektronikk ville også gitt økte kostnader.

7 Konklusjon

Denne rapporten viser Q-læringsmetoden implementert både ved tabellmetoden og nevralt nettverk for statiske og dynamiske hindringer. Det er gjort teoretisk arbeid i rapporten som danner et grunnlag for å designe lignende selvlærende systemer og videre arbeid. Rapporten beskriver inngående hvordan Q-læring fungerer og hvordan denne kan implementeres ved de to ulike metodene.

Simuleringsresultatene for metodene i et statisk miljø viser at tabellmetoden gir klart best resultater sammenlignet med nevralt nettverk. Metoden lærer mer effektivt og konvergerer raskere til en god løsning. Dette gjelder generelt der hvor tilstandstabellen er liten nok til å være håndterbar. Resultatene for de dynamiske simuleringene viser at tabellmetoden finner en løsning som fungerer, men ikke den optimale løsningen. Ved nok utforskning ville tabellmetoden teoretisk sett konvergere til en bedre løsning.

Nevralt nettverk begynner å konvergere til en løsning, men får for liten tid til å utforske miljøet. I teorien vil nevralt nettverk konvergere til en mer nøyaktig og bedre løsning enn Q-tabell, gitt at den har nok tid til utforskning. Nevralt nettverk har tradisjonelt vært brukt i større og mer komplekse systemer, og konvergerer til en god løsning når systemet har tid til nødvendig opplæring. Q-tabell er oversiktlig og enkel å forstå, og egner seg godt til selvlærende systemer med liten inndatamengde.

Opplæringsfasen i selvlærende systemer bestemmer i stor grad hvor godt systemet presterer på sikt. Metodene i denne oppgaven er designet med utforskningsfunksjoner basert på tilfeldige handlinger i starten. Det gjør at systemet presterer noe ulikt fra gang til gang i opplæringsfasen. Det finnes også utforskningsfunksjoner som ikke baserer seg på tilfeldigheter, men disse er ikke beskrevet i oppgaven grunnet prosjektets tidsomfang. Ved tilstrekkelig tid til utforskning, vil funksjoner basert på tilfeldigheter, teoretisk sett prestere likt som funksjoner som ikke er basert på tilfeldigheter.

Det finnes mange kjente metoder som kan brukes for å designe selvlærende systemer. De mest kjente av disse er beskrevet i teorikapitlet. Det er ikke mulig å trekke andre slutninger angående valg av metode, annet enn at Q-læring fungerer for det designede systemet.

7.1 Videre arbeid

Arbeidet gjort i dette prosjektet er viktig for å fremheve aktualiteten til selvlærende systemer. Arbeidet viser hvordan enkle systemer som skal håndtere lite informasjon, kan dra nytte av å være designet med en selvlærende metode. Det er gjort lite annet arbeid som belyser bruken av Q-læring for å unngå dynamiske hindringer, og dette prosjektet viser bruken av metoden for dynamiske systemer.

Arbeid med dynamiske hindringer introduserer flere faktorer som må tas hensyn til, sammenlignet med for statiske objekter. Flere faktorer gir en større tilstandstabell, som igjen gjør at systemet krever lengere tid for å konvergere til en løsning. Dette er noe som gjenspeiles i resultatene for dynamiske objekter i denne rapporten. Dynamiske systemer har behov for lengere opplæring fordi de har større tilstandstabell, og må besøke flere tilstander for å finne optimale Q-estimer.

Det er mange interessante aspekter av selvlærende systemer som ikke er omtalt i denne oppgaven. Blant annet lagring av Q-verdier ved hjelp av beslutningstre (eng.: decision tree) eller andre former for funksjonsapprosimasjon. Ulike metoder har ulike bruksområder, og for arbeid med dynamiske hindringer kunne det vært interessant å sammenligne Q-læring med andre metoder, som for eksempel, fuzzy logikk. På grunn av prosjektets tidsomfang har det ikke vært mulig å gjennomføre noe studie på dette.

Videre arbeid for prosjektet kan vurderes å omfatte implementering av et større antall handlinger for agenten. Ved å introdusere hastighet og grader av retningsendring, antas agenten å håndtere flere tilstander, herunder dynamiske hindringer. Det kunne blitt utført mer omfattende testing og optimalisering av belønningsfunksjonen, da det er ikke kjent i hvilken grad den fysiske modellen skiller seg fra den simulerte.

7.1.1 Ressurser

Desto mer komplekst systemet er, desto mer ressurser trenger det. Resultatene i denne rapporten viser at en tilsynelatende enkel metode som tabellmetoden fremstår mer effektiv enn nevralt nettverk for dette systemet.

8 Litteraturliste

12. Learning: Neural Nets, Back Propagation (2014). [Videoklipp]. Hentet fra:

<https://www.youtube.com/watch?v=q0pm3BrIUFo>

2.5 - Gradient Descent - [Machine Learning] By Andrew Ng (2014). [Videoklipp]. Hentet fra:

<https://www.youtube.com/watch?v=eikJboPQDT0>

Bie, T. (2015, 30. April). Intet norsk selskap har gjort dette før. *IT avisen*. Hentet fra

<http://www.itavisen.no/nyheter/intet-norsk-selskap-har-gjort-dette-før-386486>

Chen, X (2006). Reinforcement Learning Overview. *University of Hawaii*. Hentet 28.03.15 fra

<http://www2.hawaii.edu/~chenx/ics699rl/grid/rl.html>

Caihong Li, Jingyuan Zhang, & Yibin Li (2006) Application of Artificial Neural Network Based on Q-learning for Mobile Robot Path Planning. *Information Acquisition, 2006 IEEE International Conference*, 978-982. Hentet fra: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4097803>

Ganapathy, V., Soh Chin Yun, & Joe, H.K. (2009) Neural Q-Learning Controller for Mobile Robot. *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference*, 863-868. Hentet fra: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5229901>

Hatem, M. & Foudil, A. (2009) Simulation of the navigation of a mobile robot by the QLearning using artificial neuron networks. *Proceedings of the 2nd Conférence Internationale sur l'Informatique et ses Applications (CHIA 2009) Vol. 547*. Hentet fra: <http://ceur-ws.org/Vol-547/81.pdf>

Huang, B.-Q., Cao, G.-Y., Guo, M. (2005), Reinforcement learning Neural Network to the problem of autonomous mobile robot obstacle avoidance. *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference Vol. 1*, 85 – 89. Hentet fra: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1526924>

Jose Antonio Martin H. (2011). A Reinforcement Learning Environment in Matlab: (QLearning and SARSA). Hentet 10.03.2015, fra:

http://jamh-web.appspot.com/download.htm#Reinforcement_Learning

Lecture 10 Reinforcement Learning I (2014). [Videoklipp]. Hentet fra:

<https://www.youtube.com/watch?v=IXuHxkpO5E8>

Lecture 11: Reinforcement Learning II (2014). [Videoklipp]. Hentet fra:

<https://www.youtube.com/watch?v=yNeSFbE1jdY>

Linan Zu , Peng Yang, Lingling Chen, Xueping Zhang, & Yantao Tian (2007) Obstacle Avoidance of Multi Mobile Robots Based on Behavior Decomposition Reinforcement Learning. *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference*, 1018-1023. Hentet fra:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4522303>

Linear regression (2): Gradient descent (2013). [Videoklipp]. Hentet fra:

<https://www.youtube.com/watch?v=WnqQrPNYz5Q>

Marcello Restelli (2015) *Reinforcement Learning Exploration vs Exploitation*. Dipartimento Di Elettronica, Informazione e Bioingegneria Hentet 15.05.2015, fra

<http://home.deib.polimi.it/restelli/MyWebSite/pdf/rl5.pdf>

Marsland, S. (2009) *Machine Learning An Algorithmic Perspective* (1. utg.). [Boca Raton, FL]: CRC Press

Mitchell, T (1997) *Machine Learning* (1. utg.). [USA]: McGraw-Hill Science/Engineering/Math

Ng, A., Ngiam, J., Yu Foo, C., Mai, Y., Suen, C., Coates, A., ... Tandon, S. (2015). Multi-Layer Neural Network. Hentet 30.03.2015, fra

<http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press. Hentet fra:

<http://neuralnetworksanddeeplearning.com/index.html>

Pack Kaelbling. K og Littman. M. L. (1996) Reinforcement Learning: A Survey Hentet 30.04.2015, fra: <http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node25.html>

Poole, D. & Mackworth, A. (2010). Artificial Intelligence: Foundations of Computational Agents, *11.3 Reinforcement Learning*. Cambridge University Press. Hentet fra

http://artint.info/html/ArtInt_262.html

Poole, D. & Mackworth, A. (2010). Artificial Intelligence: Foundations of Computational Agents, *11.3.3 Q-Learning*. Cambridge University Press. Hentet fra http://artint.info/html/ArtInt_265.html

Poole, D. & Mackworth, A. (2010). *Artificial Intelligence: Foundations of Computational Agents*, 11.3.4 *Exploration and Exploitation*. Cambridge University Press. Hentet fra http://artint.info/html/ArtInt_266.html

RSS2014: 07/16 09:00-10:00 Invited Talk: Andrew Ng (Stanford University): Deep Learning (2014). [Videoklipp]. Hentet fra: <https://www.youtube.com/watch?v=W15K9PegQt0>

Russel, S. & Norvig, P. (2010). *Artificial Intelligence, A Modern Approach* (3. utg). [Upper Saddle River, New Jersey, USA]: Pearson Education, Inc.

Shadbolt, P. (2015, 21. Januar). Scientists upload a worm's mind into a robot. *CNN, Technology, Make Create Innovate*. Hentet fra: <http://edition.cnn.com/2015/01/21/tech/mci-lego-worm/index.html>

St. Olavs Hospital, (udatert). *Om læring og mestring*. Hentet 25. April 2015 fra <http://www.stolav.no/StOlav/Avdelinger/Laering%20og%20mestring/dokumenter/sentrale%20begreper%20i%20LMS.pdf>

Strauss, C. & Sahin F. (2008) Autonomous navigation based on a Q-learning algorithm for a robot in a real environment. Hentet fra: <http://scholarworks.rit.edu/other/673>

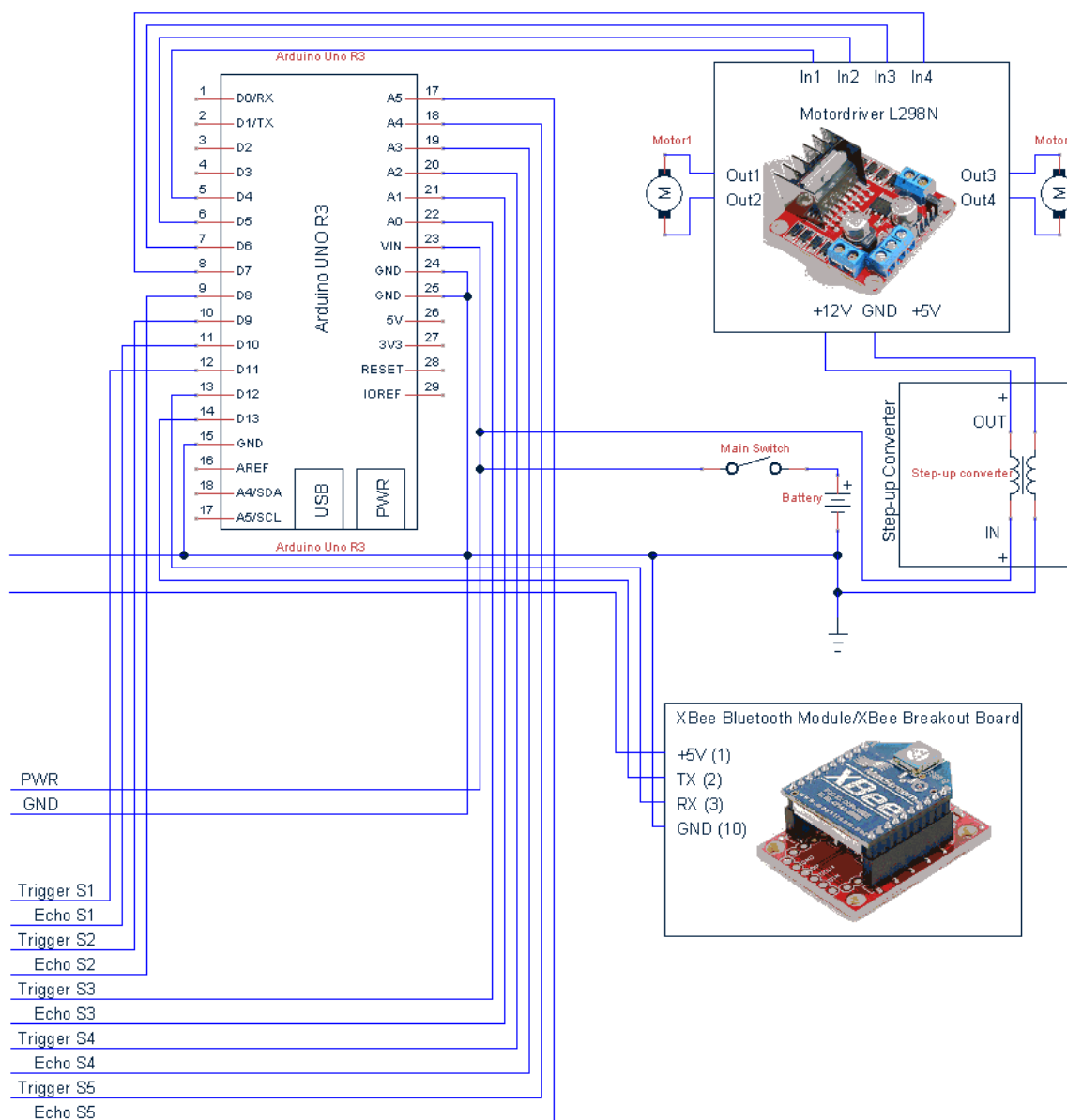
Teknomo, K. (2005). Q-learning by Examples. Hentet 12.05.2015 fra <http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/Q-Learning.htm>

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., James Diebel, . . . Stang, P. (2006). Stanley: The Robot that Won the DARPA Grand Challenge. *Stanford Artificial Intelligence Laboratory, Stanford University*. Hentet 22.04.2015 fra <http://robots.stanford.edu/papers/thrun.stanley05.pdf>

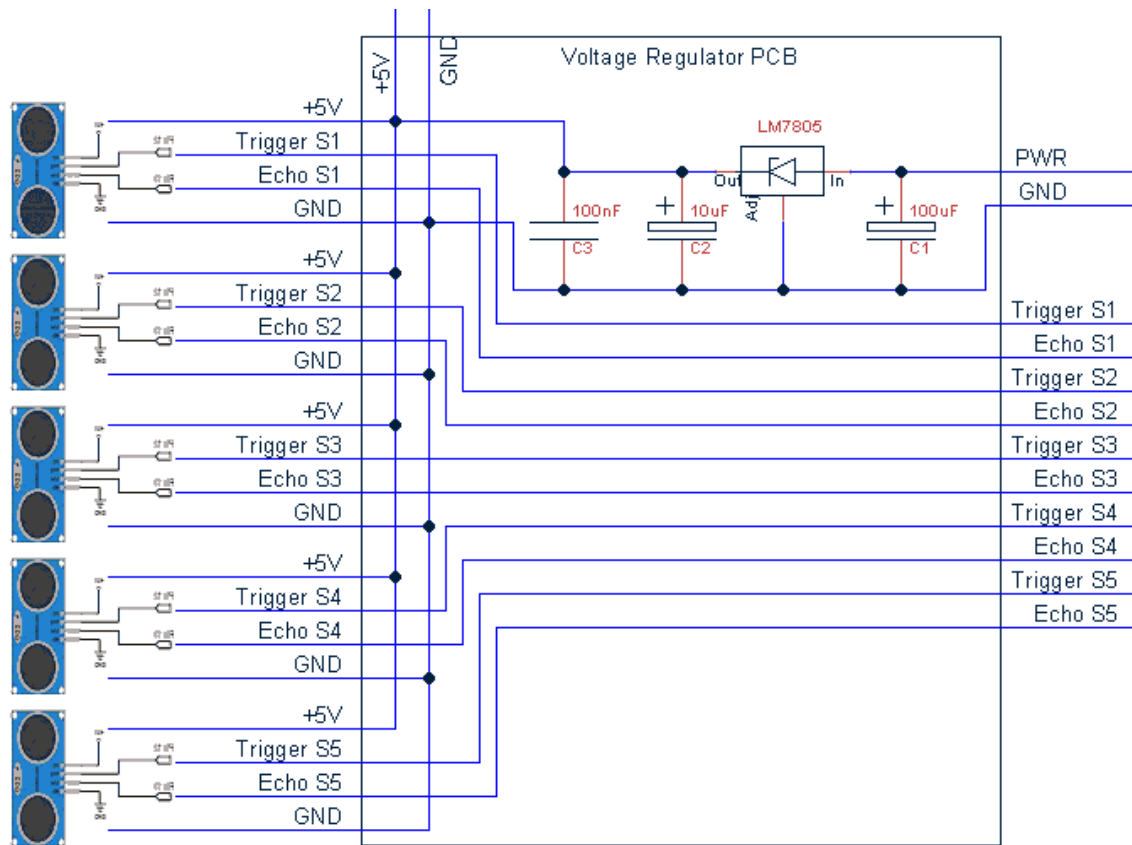
Ullah, M. (2006) SETPROD. Hentet 15.03.2015, fra: <http://www.mathworks.com/matlabcentral/fileexchange/5898-setprod>

9 Vedlegg

9.1 A) Koblingsskjema

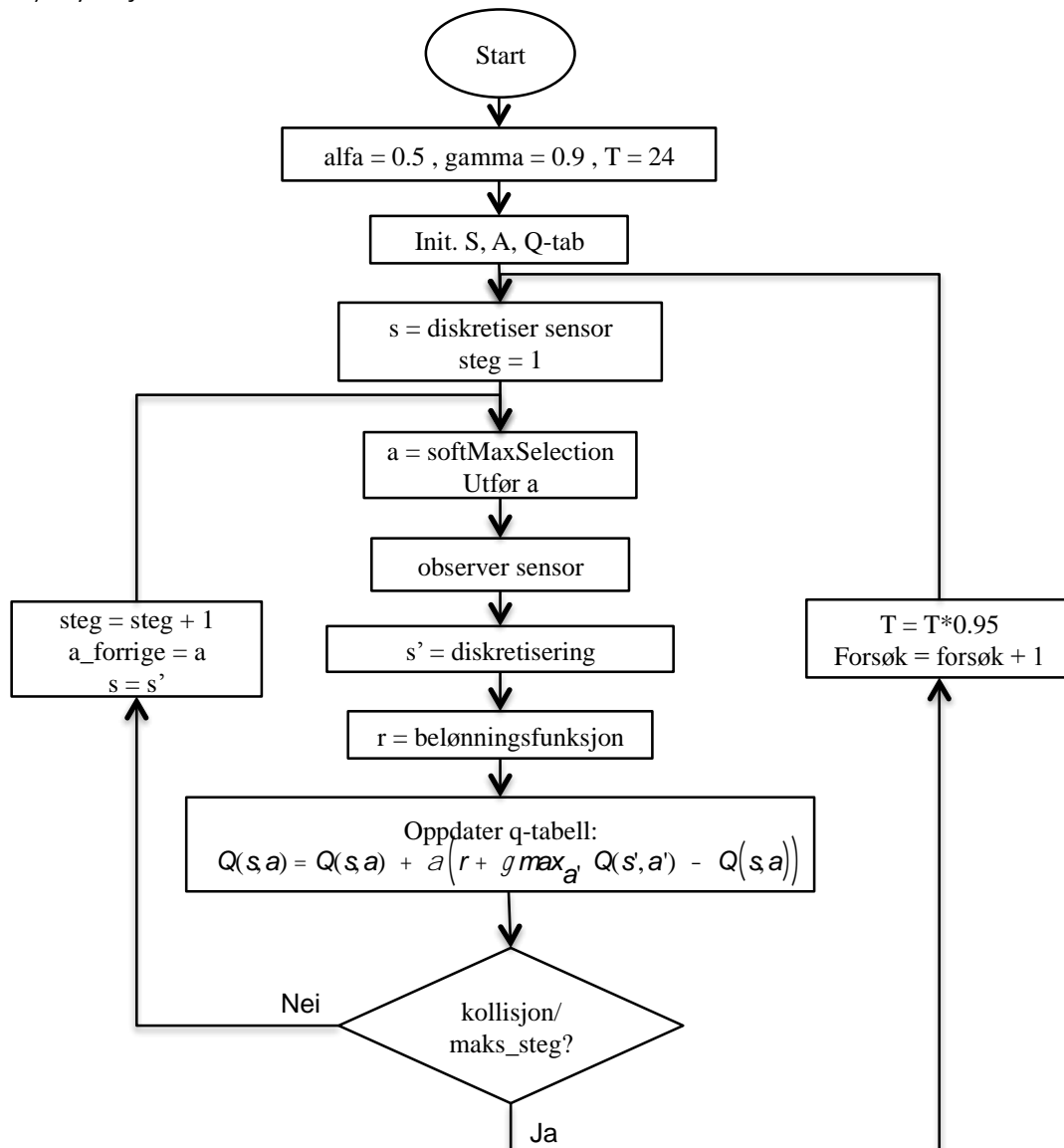


Figur 9-1 - Koblingsskjema for øvrig elektronikk (Del 1)



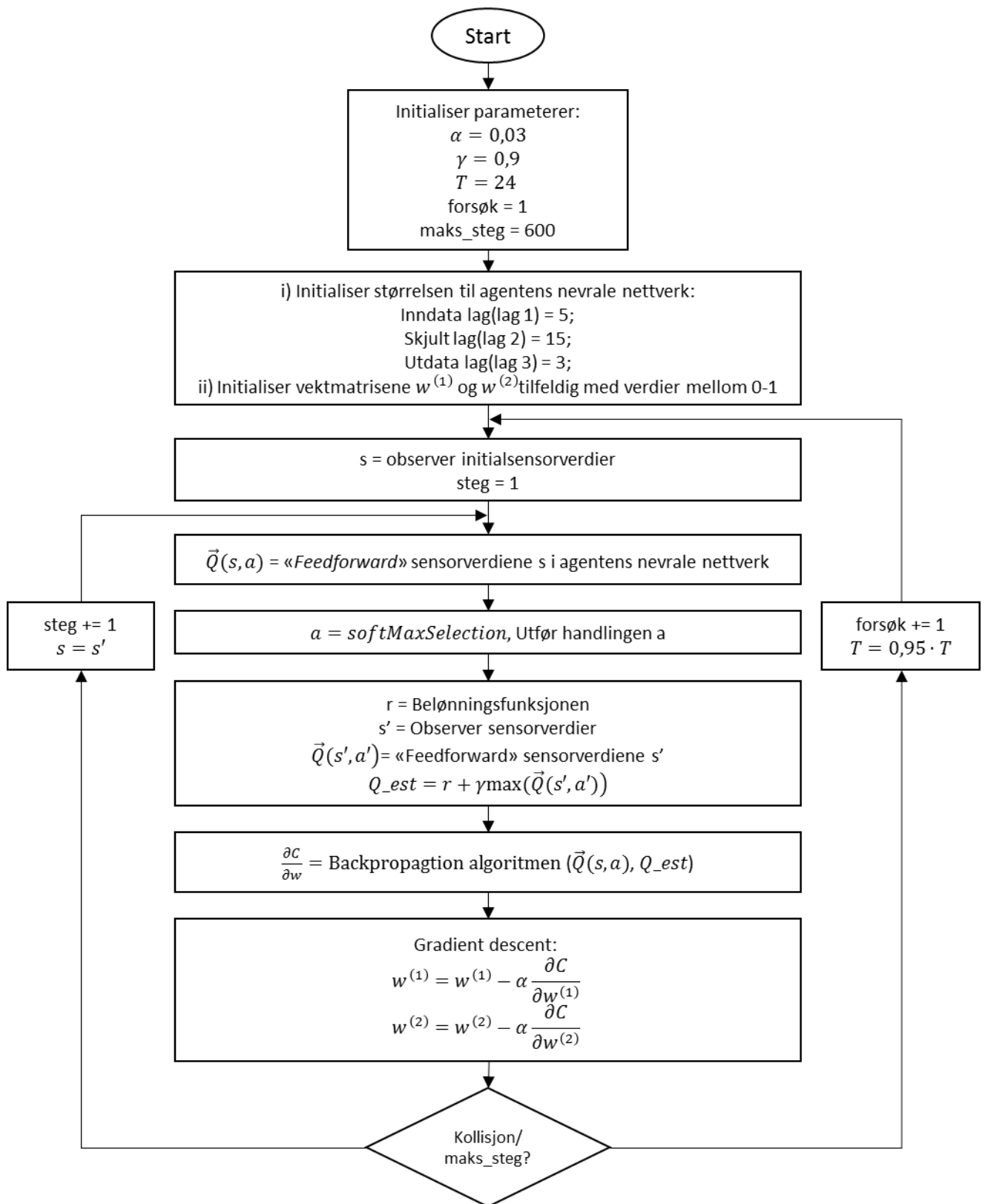
Figur 9-2 - Kablingsskjema for spenningsregulator og sensorer (Del 2)

9.2 B) Flytskjema for tabellmetoden



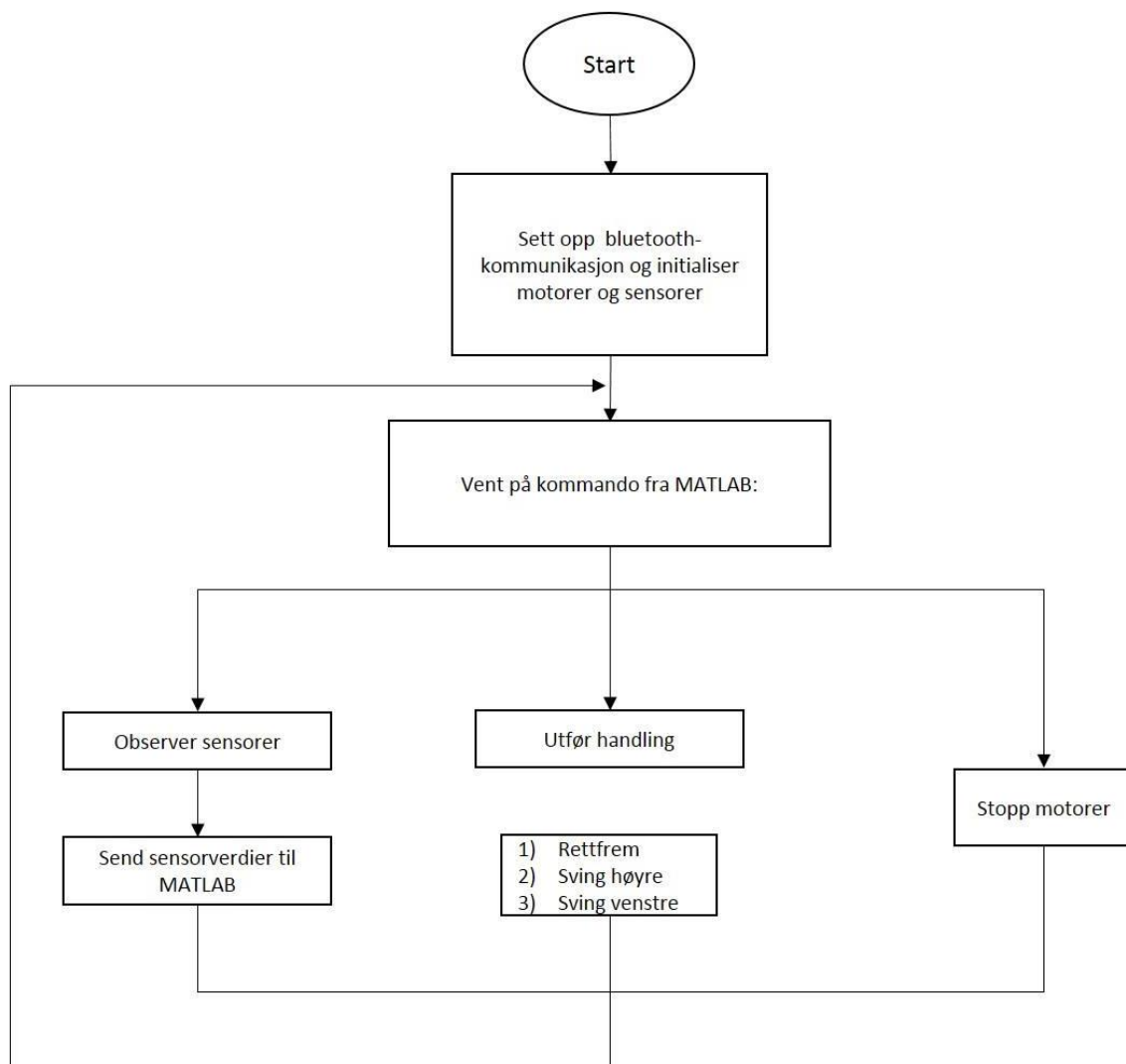
Figur 9-3 - Flytskjema for tabellmetoden

9.3 C) Flytskjema for nevral nettverk



Figur 9-4 - Flytskjema for nevral nettverk

9.4 D) Flytskjema for fysisk agent



Figur 9-5 - Flytskjema for fysisk agent