CODEN:LUTEDX/(TEIE-3097)/1-39(2021)

Electrical Engineering and Automation

Industrial

Användningen av Q-Learning för att minska energiförbrukningen hos industrirobotar

Mark Slipac

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Användningen av Q-Learning för att minska energiförbrukningen hos industrirobotar

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Industriell Elektroteknik och Automation

Examensarbete:
Mark Slipac

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2021

Sammanfattning

Examensarbetet utfördes på Lunds Tekniska Högskola, där möjligheten av att använda maskininlärningsmetoden ”Q-Learning”, för att energieffektivisera rörelserna av en industrirobot, undersöktes. Eftersom Q-Learning är en maskininlärningsmetod som inte endast försöker välja handlingar med bäst kortsiktig belöning, utan också tar hänsyn till framtida belöningar, ansågs denna metod passa in bra i uppgiften. I arbetet testas de två olika Q-Learnings metoderna, där den ena metoden använder sig utav neuronnät och den andra inte.

Simuleringen genomförs i programmet RobotStudio. RobotStudio är ett välkänt program utvecklat av ABB för att simulera diverse uppgifter för industrirobotar. Med hjälp av RobotStudio och dess funktioner är det möjligt att approximativt uppskatta energiförbrukningen av varje rörelse.

Under processen tränades och testades många olika neuronnät för att uppskatta energikostnaden för olika typer av rörelser. Genom att endast använda sig utav en startposition och slutposition för en rörelse, lyckades ett förvånansvärt noggrant sätt att uppskatta energikostnader uppnås, med hjälp av neuronnät.

Resultatet från examensarbetet visar att det är möjligt att använda Q-Learning som ett sätt att hitta energisnåla vägar, men eftersom precisionen är för låg för att användas i verkligheten bör fler tester genomföras.

Nyckelord: Q-Learning, Energiförbrukning, Robot, Maskininlärning, RobotStudio

Abstract

This thesis was conducted at the Faculty of Engineering, where the possibility of using the machine learning method known as “Q-Learning”, to increase energy efficiency of an industrial robot’s movements, was explored. Because Q-Learning is a machine learning method which not only tries to pick the action with the highest short-term reward, but also considers the future reward, it was considered to be a suitable match for this task. In the project, two different types of Q-Learning methods are tested and examined, where one method uses neural networks and the other one does not.

The simulation is conducted in the program RobotStudio. RobotStudio is a well-known program, developed by ABB, used to simulate different tasks for industrial robots. Using RobotStudio and its functions it is possible to estimate, to a fairly accurate degree, what the energy cost for each movement is.

During the process, many different neural networks were trained and tested in estimating the energy cost for different sets of movements. using only the starting and end position for a movement, a surprisingly accurate way of estimating the energy cost was achieved with neural networks.

The result of the project shows that it is possible to use Q-Learning as a way of finding energy efficient paths, but since the precision is still too low to use in practical applications, further testing is required.

Keywords: Q-Learning, Energy consumption, Robot, Machine Learning, RobotStudio

Förord

Examensarbetet har omfattat 22.5 högskolepoäng och har varit det avslutande momentet i min högskoleutbildning på LTH vid campus Helsingborg.

Jag vill tacka båda mina handledare, Mathias Haage och Christian Nyberg, som har varit väldigt hjälpsamma under arbetet och kommit med bra råd och tips då jag har varit i behov av det.

Jag vill tacka Mats Lilja för att ha tagit rollen som min examinator, men även för att han har varit en stor hjälp vid införskaffningen av diverse licenser till arbetet.

Jag vill även tacka ABB som har tillåtit mig använda RobotStudio i mitt arbete.

Lund, juni 2021

Mark Slipac

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Robotarmar inom industrier och företag är något som har blivit mer populärt de senaste 40 åren. Kanske är det för att de gör ett snabbare och mer effektivt arbete jämfört med människor, vilket gynnar produktionen. Kanske är det för att robotarna är multifunktionella och kan ställas om till att göra andra uppgifter, vilket ger ett företag möjlighet att vara mer flexibel, eller kanske så är det för att skador på arbetsplatsen minskar. [1]

För att använda dessa armar så måste de programmeras för att genomföra en uppgift, men att optimera roboten och dess rörelser så att den förbrukar så lite energi som möjligt är svårt. Att hitta det mest optimala läget där robotarmen förbrukar så lite energi som möjligt samtidigt som den utför sin uppgift är viktigt då detta kan spara mycket pengar och hjälpa miljön.

Ett sätt att angripa problemet kan vara genom att använda sig utav maskininlärning. Maskininlärning syftar på idén att ett system kan lära sig att utföra en uppgift genom att analysera data och ta beslut utifrån det.

Det finns många olika maskininlärningsmetoder med olika egenskaper. En av de viktigare egenskaperna för ett system som styr en robot, med energiförbrukningen i åtanke, är att systemet kan anpassa sig till förändringar. En annan viktig egenskap är att inlärningen kan ske både innan och under själva arbetet, dvs. inlärningen kan ske i en simulering men även när roboten arbetar i fabriken. Med detta i åtanke så har Q-Learning valts som inlärningsmetod.

Q-Learning är en inlärningsmetod som utvärderar handlingar som tas och justerar systemet beroende på värdet av handlingarna. Om implementeringen av Q-Learning lyckas kommer roboten få möjligheten att korrigera sin rutt vid till exempel ändringar i vikt eller ändringar i motstånd i roboten.

1.2 Syfte

Syftet med arbetet är att undersöka om det är möjligt att använda sig utav maskininlärningsmetoden ”Q-Learning”, för att hitta mer energisnåla rörelser och vägar som roboten kan ta. Det förväntade resultatet är en metod som kan minska förbrukningen av energi genom att ta smartare vägval. Förhoppningsvis bör metoden även korrigera sig i realtid om vissa motstånd eller laster ändrar sig.

1.3 Målformulering

Målet med arbetet är att, med hjälp av Q-Learning, skapa en metod för att hitta energisnåla sätt att röra en industrirobot på. Då Q-Learning består av två olika typer av maskininlärningsmodeller kommer båda testas och undersökas. Sedan kommer de anskaffade metoderna att utvärderas med precision och inlärningstid i fokus.

1.4 Problemformulering

De inledande frågeställningar till arbetet, som kommer besvaras under arbetets gång och hjälper arbetet att hålla sig avgränsat, är följande:

1a. Vilken typ av industrirobot ska undersökas?

b. Hur många axlar har den?

c. Hur beräknas energiförbrukningen?

2a. Vilket simuleringsprogram bör användas?

b. Hur implementeras Q-Learning I simuleringsprogrammet?

c. Hur implementeras neuronnät i simuleringsprogrammet?

3. Vad för slags simulering är det som kommer genomföras? Vilka typer av rörelser bör undersökas?

4. Hur evalueras metoderna?

1.5 Motivering av examensarbete

Examensarbetet valdes med samhällsnytta i åtanke. Eftersom energifrågan är väldigt viktig, och kommer bli viktigare med åren, är det bra att arbeta med något som inte nödvändigtvis löser nya problem utan effektiviserar gamla. Sedan valdes även detta projekt eftersom maskininlärning var i centrum och det är ett område som i snabb takt börjat växa under de senaste åren.

1.6 Avgränsningar

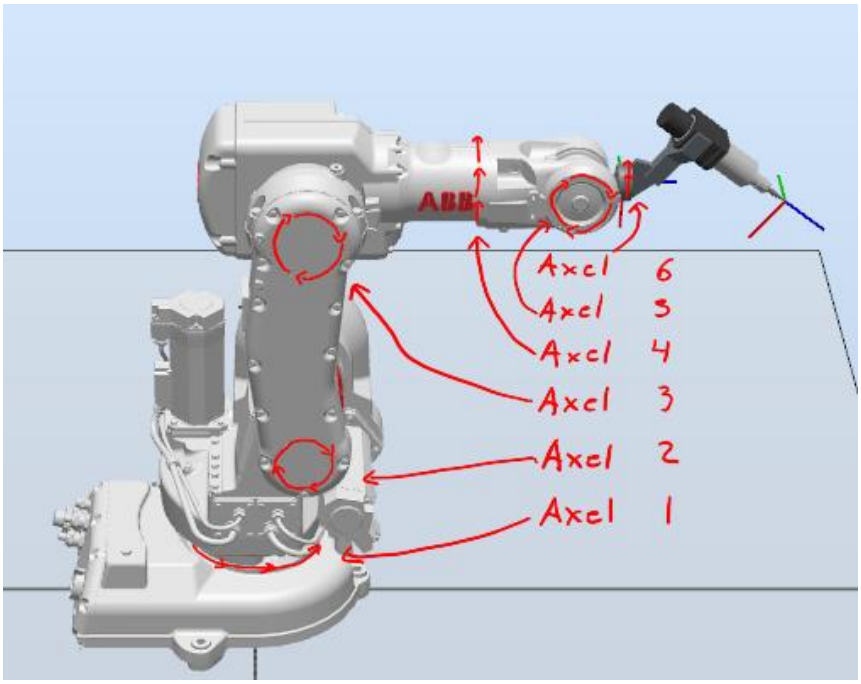
Då målet är att undersöka om det är möjligt med Q-Learning först och främst, så kommer kravet på precisionen av simuleringsprogram inte vara lika hög som om arbetet skulle jämföras med andra metoder. Sedan kommer bara väldigt enkla point-to-point scenarion undersökas, då mer komplicerade rörelser än så, med flera målpunkter, skulle kräva en längre arbetsperiod.

2 Teknisk Bakgrund

2.1 Robot

Roboten som användes i arbetet var en IRB 140. IRB 140 är en kompakt och kraftfull industrirobot med sex axlars rörlighet, tillverkad av ABB Robotics. Roboten kan lyfta upp till 6 kg och har en räckvidd på 0,81 meter. Roboten används huvudsakligen för svetsning, montering och packning. [2]

Med robot-styrenheten IRC5 kan alla ABB system programmeras. Programmeringen kan göras både på en operatörspanel som kallas för FlexPendant, eller så kan det programmeras i programvaran RobotStudio. [2]



Figur 1:Skärmbild av industriroboten IRB 140 med ett verktyg monterat. I figuren är även alla axlarna markerade.

2.1.1 Beräkning av energi

Beräkningen av energiförbrukning görs genom att använda formeln nedan, där P representerar den kinetiska energin som förbrukas. Symbolen τ representerar vinkelmomentet som belastar axeln och ω representerar vinkelhastigheten. Eftersom alla axlar kommer ha olika hastigheter och moment på sig, behöver vi genomföra beräkningarna för varje enskild axel åt gången. [1]

$$P = \tau * \omega \qquad \textit{Ekv. 1}$$

Sedan, eftersom roboten kommer vara i rörelse under mätningarna, och mätningarna ska ske i korta intervaller, ska beräkningarna göras på värdens delta.

2.2 RobotStudio

RobotStudio är en programmeringsmjukvara, utvecklad av ABB, som erbjuder ingenjörer möjligheten att arbeta med industrirobotar i simuleringsmiljöer och skapa program för robotarna att följa. RobotStudio har ett stort bibliotek med 3D-modeller av nästan alla industrirobotar som ABB har att erbjuda. Programmet har även en robust fysikmotor som gör det möjligt att simulera laster och krafter, som är en centralpunkt i examensarbetet. [3]

2.2.1 PC SDK

PC Software Development Kit, även kallat PC SDK, är ett verktyg inom RobotStudio som tillåter programmerare att, via Visual Studio, koppla sig till robotens kontroller. Via Visual Studio har man då tillgång till en mängd olika funktioner, t.ex. skriva till robotens Rapid kod. [4]

2.2.2 RobotStudio SDK

RobotStudio Software Development Kit är likt PC SDK, men skiljer sig i att man inte kopplar sig till robotens kontroller. Istället för att koppla sig direkt till robotens kontroller så fungerar RobotStudio SDK på det viset att man skapar ett program, en så kallad Add-In, som körs direkt på roboten. Programmeringen görs även här via Visual Studio med C# som programmeringsspråk. [4]

2.2.3 Rapid

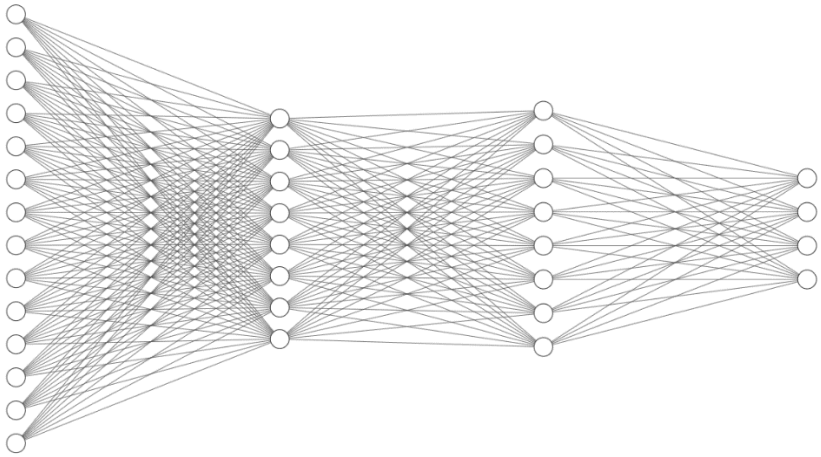
Rapid är ABB:s egna programmeringsspråk som är utvecklat för att styra deras industrirobotar. I programmeringsspråket finns många funktioner kring styrningen av roboten, men i arbetet används bara två funktioner. Den ena funktionen läser av varje axels vridmoment, medan den andra förflyttar axlarna. [3]

2.3 Neuronnät

Neuronnät används ofta inom maskininlärningssystem som ett sätt att processa stora mängder information och hitta samband mellan informationen. Näten är inspirerade av det biologiska nervsystemet som bygger upp hur vi processar information. Neuronnät har använts till allt fler tillämpningar de senaste åren och företag som Google och OpenAI har investerat mycket pengar i forskning om dessa nät. [5]

I figur 2 illustreras ett exempel på hur ett neuronnät kan se ut. Nätet är uppbyggt i flera lager av noder (cirklarna) med vikter (linjerna) som binder ihop dem. Första lagret brukar kallas för *Input*-lagret, det är här informationen kommer in. Sista lagret kallas för *Output*-lagret, här visas slutresultatet av processen. Ibland kan noderna i *Output*-lagret även vara förknippade med någon slags handling. [5]

Lagren mellan *Output* och *Input* kallas för osynliga lager. Deras funktion är att göra nätet mer komplext. Det är däremot viktigt att anpassa sitt nät för uppgiften och inte ha för många eller för få lager, då detta kan leda till att systemet inte ser sambandet tillräckligt bra eller att det tar för lång tid för nätet att lära sig sin uppgift. [5]



Figur 2: Exempel på hur ett neuronnät kan se ut.

2.3.1 *Forward propagation*

Forward propagation (FP) används för att beräkna vad lagren framför bör få för värden beroende på de förgående noder och vikter. Först används *Input*-lagret för att beräkna vad det första osynliga lagret bör bli och sedan används det lagret för att beräkna vad det nästa lagret blir. Detta fortsätter sedan tills *Output*-lagret är framtaget. Ekvationen för att genomföra FP på en nod i ett lager visas i ekvation 2. Uträkningen sker enskilt för varje nod i ett lager. [5]

$$z_i = b_i + \sum_{j=1}^{m} w_{ij} * n_j \qquad \textit{Ekv. 2}$$

w_{ij} representerar en vikt mellan noden n_j och n_i. Antalet noder i lagret representeras av m. Summan av alla vikter w_{ij} och noder n_j som är kopplade till noden n_i adderas sedan med b_i, som kallas för en bias. Bias ritas inte alltid ut i nätritningarna, men de finns där. De tillför en extra mängd komplexitet till nätet. [5]

Då summan av alla tillägg till noden är kända är det inte ovanligt att sätta det värdet i en aktiveringsfunktion (Ekv. 3). Dessa aktiveringsfunktioner gör att resultatet av en nod, beroende på z_i, är antigen väldigt nära 1 eller väldigt nära 0, för att få en ”av” och ”på” karaktäristik, likt en neuron. [5]

$$n_i = f(z_i) \qquad \textit{Ekv. 3}$$

Vid mer komplicerade nät kan ett *Output*, beroende på *Inputs* och vikter, bli nästan vad som helst. Nätet blir då likt en väldigt avancerad funktion. Med väldigt små ändringar i *Input*-lagret kan *Output*-lagret ändra sig mycket. Komplexiteten av nätet beror mycket på antalet lager och storleken på dem. [5]

2.3.2 *Backpropagation*

Backpropagation (BP) är metoden/algoritmen som används för att justera nätet så att det får det önskade värdet i *output*. Algoritmen används ofta i samband med neuronnät och maskininlärning och går ut på att hitta samband mellan vikterna och biasnoderna i nätet och den önskade *output*. Därefter justeras vikterna inkrementellt och efter ett par hundra gånger så är den faktiska *output* nära den önskade *output*. [5]

Innan beräkningar av hur vikterna/bias ska ändras, behöver man beräkna det totala felet som nätet ger. I ekvation 4 visas hur felet beräknas, där n^l är *output* vi får ut från en nod och *y* är det önskade värdet. [5]

$$E = (n^l - y)^2 \qquad \textit{Ekv. 4}$$

För att ta reda på vad den inkrementella justeringen av vikten *w* bör vara används ekvation 5, där resultatet är förhållandet mellan *E* och vikten *w*. De tre produkterna på andra sidan av ekvationen representerar olika förhållanden mellan vikter *w*, *input z* till särskilda noder, värden av noder n^l och felet *E*. [5]

$$\frac{\partial E}{\partial w} = \frac{\partial z}{\partial w} * \frac{\partial n^l}{\partial z} * \frac{\partial E}{\partial n^l} \qquad \textit{Ekv. 5}$$

Justeringen av biasnoderna sker på ett liknande sätt, förutom att den första produkten, $\frac{\partial z}{\partial w}$, inte finns. [5]

I ekvation 6 illustreras hur själva uppdateringen av vikten bör se ut. I vissa fall används α för att reglera hur mycket vikterna ändras då det ibland inte är optimalt att ändra för mycket. [5]

$$w_{new} = w_{old} - \alpha * \frac{\partial E}{\partial w} \qquad \textit{Ekv. 6}$$

2.3.2.1 *Momentum*

Momentum, eller tröghet, används i vissa sammanhang för att öka hastigheten av inlärningen för neuronnät, men även för att komma ut ur vissa inlärnings-fallgropar. Genom att implementera tröghet i ekvation 6 får man fram ekvation 7, som då bör ge snabbare inlärning. [9]

$$w_{new} = w_{old} - (\alpha * \frac{\partial E}{\partial w^l} + m * (\alpha * \frac{\partial E}{\partial w^{l-1}})) \qquad \textit{Ekv. 7}$$

Ekvationen är väldigt lik den tidigare ekvationen förutom att man adderar den föregående ändringen som genomfördes, multiplicerat med en tröghetsfaktor, med den nuvarande ändringen som håller på att genomföras. Detta skapar då en slags snöbollseffekt som för varje gång ökar storlek och ökar hur mycket en särskild vikt bör ändras, och när det sedan vänder och behöver gå i motsatt riktning så kräver det endast två iterationer. [9]

2.3.3 Initialisering av Neuronnät.

Innan inlärningen påbörjas med neuronnäten, så behöver vikterna ha något slags startvärde. Att initialisera alla vikter till 0 ger problem för både FP och BP. Att initialisera med ett slumpmässigt värde kan ge problem då fördelningen man väljer kan ge både mycket stora och mycket små tal i *Output*-lagret. Därför används ofta Xavier-metoden där man väljer begynnelsevärdena slumpmässigt men inom ett intervall. [7]

I ekvation 8 visas inom vilket område en vikt bör initialiseras till enligt Xavier-metoden. Vikten bör befinna sig slumpmässigt inom området som ekvationen visar, där n_i är antalet vikter i föregående lager, och n_{i+1} är antalet vikter som går ut ur lagret. [7]

$$[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}] \qquad \textit{Ekv. 8}$$

2.4 Q-Learning

Q-Learning är en typ av förstärkningsinlärningsalgoritm. Vad detta innebär är att algoritmen lär sig utföra uppgiften genom att testa sig fram och hitta lösningar baserat på föregående erfarenheter. Ett system för belöning och bestraffning implementeras för att erfarenheterna ska ge en förbättring av algoritmens beteende. Det som skiljer Q-Learning från andra algoritmer är att denna metod inte hela tiden letar efter handlingen som ger mest belöning, utan tar även hänsyn till framtida belöningar. [6]

Inom Q-Learning finns det två olika metoder för inlärning, den första är tabell Q-Learning och den andra är *Deep Q-Learning Network*. Dessa kommer förklaras mer ingående i de senare delarna av kapitlet. [6]

2.4.1 Epsilon-Greedy

Epsilon-Greedy är en populär policy för att angripa dilemmat mellan att utforska och att utnyttja. För att se till att systemet lär sig genom att exponeras för nya stimuli, så måste systemet ibland prova nya sätt att reagera på och inte bara falla tillbaka på sin erfarenhet. [6]

En konstant ε har ett värde mellan 1 och 0, med sannolikheten ε så använder algoritmen sin hittillsvarande erfarenhet och väljer det bästa alternativet som den redan har hittat och med sannolikheten 1-ε väljer den ett annat alternativ att utforska för att på så sätt utvidga sin erfarenhet. [6]

Att tidsanpassa värdet på ε ger systemet möjlighet att bli mer effektivt, då i början av inlärningsprocessen bör systemet utforska mer än vad det utnyttjar. Senare när systemet har införskaffat erfarenhet så kan det utforska mer vid de mer värderika områdena. [6]

2.4.2 Tabell Q-Learning

Tabell Q-Learning syftar på den mer normala typen av Q-Learning. I denna metod så är antalet tillstånd och handlingar kända och ändliga. Detta gör att vi kan sätta in alla möjliga tillstånd och handlingar i en tabell och uppdatera tabellen då vi kommer till de olika cellerna i den. [6]

Varje tillstånd i tabellen har ett visst antal handlingar som kan väljas, och varje kombination har ett särskilt Q-värde. I ekvation 9 visas hur Q-värden för varje par uppdateras. Belöningen som tillkom representeras av r_t och $maxQ(s_{t+1}, a)$ är värdet på den bästa handlingen att ta i det nuvarande tillståndet. [6]

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \alpha(r_t + \gamma * (maxQ(s_{t+1}, a) - Q^{old}(s_t, a_t))) \quad \textit{Ekv. 9}$$

Både γ och α är konstanter inom intervallet [1, 0]. α representerar inlärningshastigheten och γ representerar hur stor vikt de framtida handlingar ska ha i beslutet. Om γ är väldigt litet kommer inlärningsprocessen vara kort, men systemet kommer inte ta stor vikt i framtida handlingar, vilket kanske inte är optimalt. Å andra sidan, om γ är väldigt nära 1 kommer det ta lång tid tills alla Q-värden är kända. [6]

2.4.3 *Deep Q-Learning Network*

Deep Q-Learning Network (DQN) kombinerar Q-Learning med neuronnät för att uppnå ett system som inte förlitar sig på en tabell. Detta gör det möjligt att istället hitta ett samband mellan tillstånd och handlingar för att gissa vad Q-värdet bör vara. Då endast ett samband behövs, är det möjligt att lösa problem som tidigare hade behövt nästintill oändligt många tillstånd och handlingar i en tabell. [6]

Ett problem som dyker upp i användningen av DQN, till skillnad från tabell Q-learning, är att systemet kan fastna mellan två tillstånd och på så sätt höja varderas Q-värde oändligt högt. För att motverka detta används två separata nät, där det ena nätet uppskattar Q-värdet, $Q(s_t, a_t)$, och det andra uppskattar framtida Q-värden, $Q(s_{t+1}, a)$. Neuronnätet vars uppgift är att uppskatta framtida värden kallas för *Target Network* (TN). [6]

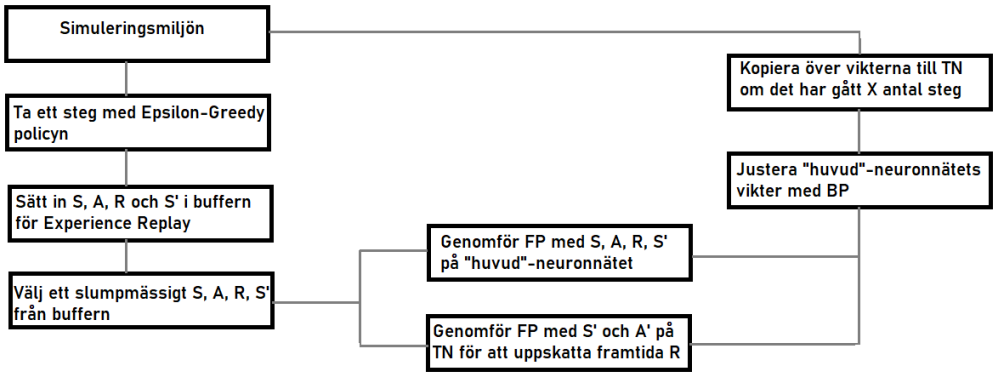
Det är värt att påpeka att beräkningen av felet *E* inför BP ser annorlunda ut för DQN. För att uppskatta felet *E* korrekt i denna metod behöver man ha i åtanke att de framtida värden också behöver uppskattas, nedan i ekvation 10 visas hur beräkningen ändras. [6]

$$E = (n^l + \gamma * (maxQ(s_{t+1}, a)) - y)^2 \quad \textit{Ekv. 10}$$

Det första nätet uppdateras sedan efter varje iteration, medan det andra uppdateras först efter en godtycklig mängd tid har passerat, så att det är säkerställt att systemet har tagit sig vidare och inte är kvar i samma tillstånd. [6]

2.4.3.1 *Experience Replay*

Experience Replay är en teknik som ofta används inom DQN för att öka inlärningshastigheten. Genom att använda sig utav en buffert med tillstånd, handlingar och belöning kan man kontinuerligt optimera nätet från slumpmässigt valda erfarenheter. Detta gör inlärningen mycket mer varierad och minskar risken att neuronnätet överoptimeras för ointressanta tillstånd som råkar passeras. [6]



Figur 3: Flödesschema för DQN algoritmen

3 Metod

I detta kapitel beskrivs de olika metoder som använts under examensarbetets gång samt vilka faser arbetet genomgick.

3.1 Genomförande

3.1.1 Kommunikation

Kommunikationen med handledarna och examinatorn skedde via e-mail för det mesta, med några enstaka zoom-möten. På grund av de rådande omständigheterna med Covid-19 var detta den bästa lösningen, och eftersom arbetet skedde i simuleringsmiljöer för det mesta så var det inte särskilt förödande.

3.1.2 Projektmodell

Arbetet planerades och utformades med Spiralmodellen som utgångspunkt. Spiralmodellen är uppbyggd på det sättet att arbetet genomförs i ett antal olika cykler, där varje cykel innehåller olika faser. Som exempel kunde faserna vara målsättning, förstudier, planering och testning. Sedan när den sista fasen är uppnådd då börjar man om.

3.1.3 Förstudier

Förstudierna skedde generellt sett i början av en ny cykel då en ny metod eller process skulle testas/implementeras. Genom att löpande genomföra förstudier inför varje ny cykel öppnades möjligheten att fördjupa sig i specifika områden som användes under arbetets gång.

Första förstudien under examensarbetet skedde i början av arbetet. Syftet med den första förstudien var att få en generell uppfattning av hur arbetet skulle genomföras. I förstudien ingick maskininlärningsmetoder, neuronnät, robotik, kinematik och liknande arbeten som använt liknande tekniker eller program. Detta gav en bra överblick av hela arbetet.

När området för arbetet hade smalnat av och några specifika metoder och processer hade blivit utvalda utfördes en fördjupad studie i dessa. Detta var viktigt att genomföra för att undersöka om det skulle finnas någon typ av ”återvändsgränd” som skulle göra arbetet mycket svårare eller i värsta fall, omöjligt.

Därefter gjordes några enstaka förstudier eller fördjupningar i början av en ny cykel i arbetet. Både för att friska upp minnet, men också för att fördjupa sig diverse specifika moment som, om de inte görs rätt, kan ha förödande effekter på arbetet.

3.1.4 Cykler

Arbetet genomgick fyra olika cykler med olika målsättningar. Den första cykeln av arbetet involverade den första förstudien och införskaffning av programvaror och licenser. Under denna period lades mycket tid på att hitta handledare och examinator för examensarbetet.

Andra cykeln av arbetet användes för att bli bekant med RobotStudio och Visual Studio, samt programmeringsspråken Rapid och C#. Cykeln omfattade programmeringen och implementeringen av funktioner som gjorde det möjligt att kommunicera med den virtuella kontrollern. Ett interagerbart test-fönster implementerades även för att kunna övervaka mätvärden och styra roboten.

Under cykel nummer tre skapades och testades olika funktioner kring neuronnät och maskininlärning. Detta gjordes då det var svårt att hitta bra bibliotek med funktioner för neuronnät och liknande metoder. För att säkerställa att metoderna fungerade som de ska genomfördes många med tester, då det minsta felet kunde orsaka stora fel.

I sista cykeln testades de olika Q-Learning metoderna. Inte bara de olika metoderna testades, utan även olika uppsättningar neuronnät och inlärningshastigheter som testades.

3.2 Programmering

Programmeringen av arbetet genomfördes i två olika utvecklingsspråk vilka var Rapid, i RobotStudio, och C#, i Visual Studio.

Programmeringen i Rapid bestod endast av några få rader kod som gjorde det möjligt att röra på roboten, samt gjorde det möjligt att avläsa vridmomentet för varje axel.

Största delen av programmering i arbetet bestod av C# i Visual Studio. Detta berodde på att det var besvärligt att hitta ett bra bibliotek med maskininlärningsfunktioner och funktioner kring neuronnät. Sedan behövdes även funktioner för vissa delar av Q-Learning algoritmen som var ännu mer besvärliga att hitta. Detta resulterade i att programmeringen slutligen gjordes från grunden

3.3 Källkritik

[1] *Handbook of Robotics* är en välkänd bok inom robotik som tar upp en hel del viktiga ämnen, så som kinematik, energiförbrukning och liknande. Boken finns i LUBcat och kan lånas som e-bok via Lunds universitet, vilket ger boken relativt hög trovärdighet. De formlerna som har använts i arbetet från denna bok har även förekommit i liknande arbeten och andra källor. Boken var även rekommenderad av två olika handledare/examinatorer.

[2] Produktspecifikationen kring industriroboten IRB 140 är skriven av ABB och eftersom ABB är ett stort företag med bra rykte när det gäller industrirobotar, så antas trovärdigheten på dokumentet väldigt hög.

[3][4] Databladen kring Rapid-koden, som även här är publicerade av ABB, antas vara pålitlig källa. Koderna och funktionerna som de erbjuder blivit testade för att säkerställa att de gör vad de påstås göra.

[5] *Introduction to multi-layer feedforward neural networks* är en vetenskaplig artikel där några metoder kring neuronnät förklaras. Artikeln anses vara legitim då ett flertal arbeten, både utom och inom Sverige, har refererat till arbetet. Metoderna är prövade innan arbetet för att säkerställa att de fungerar som angivet.

[6] *Deep Reinforcement Learning: Frontiers of Artificial Intelligence* finns i LUBcat att låna som en e-bok och boken är även publicerad av Springer, som är ett välkänt förlag, vilket i sin tur gör källan pålitlig.

[7] Artikeln går igenom hur Xavier initialisering fungerar och varför det fungerar så bra som det gör. Metoden förekommer i många artiklar och är publicerad av universitetet i Montreal, vilket ger källan en hög grad av trovärdighet.

[8] Kapitlet som refereras är ett kapitel från en bok som många forskare har varit delaktiga med att skriva. Boken finns att låna i LUBcat som e-bok och är den första boken som dyker upp då man söker ”Neural Network”.

[9] Rapporten är citerad ofta i arbeten kring maskininlärning fortfarande trots att rapporten kom ut 1999. Detta anses ge rapporten en viss nivå av trovärdighet.

4 Analys

4.1 Målsättning

Målet med arbetet är att undersöka möjligheten av att använda sig utav maskininlärningsmetoder för att uppskatta energiförbrukningen hos en industrirobot och för att sedan uppskatta den mest energisnåla handlingen som industriroboten kan ta.

Kraven för att metoden ska anses vara lyckad är att tiden för träningen inte ska överstiga en viss gräns, och för att systemet sedan kan nå en tillräckligt hög grad av precision i att approximera energisnåla rörelser.

4.2 Motivering för val av verktyg

I detta kapitel ska några val av simuleringsmiljöer och programmeringsspråk utvärderas och kritiseras.

Valet av RobotStudio som simuleringsmiljö skedde genom en rekommendation från handledare samt mot bakgrund att programmet ofta används i liknande arbeten då industrirobotar är i fokus. Det andra möjliga valet skulle kunnat vara en spelmotor som Unity, eftersom Unity har implementerat nya verktyg för att underlätta programmeringen av industrirobotar skulle det inte vara helt fel. I det fallet skulle däremot mer tid behöva läggas på att implementera nya sätt att mäta energin som roboten förbrukar, vilket inte skulle passat tidsramen för examensarbetet.

Att verifiera funktionerna hos RobotStudio går att göra i simuleringsmiljön, men för att vara helt säker på vissa av dem så skulle en verklig industrirobot behövas, vilket i detta fall inte var möjligt.

Programmeringen skedde i Visual Studio då det var det enda sättet att skapa ett avancerat styrprogram till RobotStudio som styr det i realtid. Ett annat sätt som arbetet skulle kunnat genomföras på, är att skapa ett litet program inom Rapidkod som då skulle spara undan data från vissa handlingar, som förklaras sen. Efteråt skulle Python kunnat användas att analysera datan. Eftersom Python har flera välutvecklade bibliotek, utformade för att exekvera olika funktioner kring neuronnät, skulle det inte vara dåligt. I slutändan valdes däremot Visual Studio, då förmågan att skriva ett mer avancerat program och kunna göra all programmering i ett språk övervägde det andra.

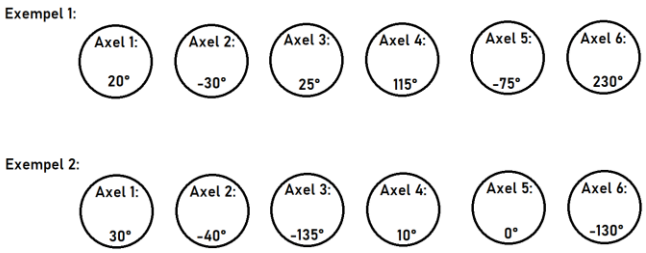
Utvecklingen av programmet skedde med hjälp av två separata bibliotek i Visual Studio, RobotStudio SDK och PC SDK. Anledningen till detta var att RobotStudio SDK upplevdes ha något snabbare svarstid till ett par funktioner. Eftersom RobotStudio SDK fungerar som ett Add-In och PC SDK måste skicka information via kontrollern så var detta inte så långsökt påstående.

4.3 Experiment

4.3.1 Tabell Q-Learning

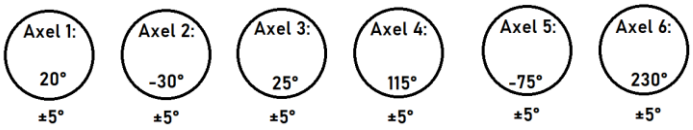
Det första experimentet som genomförs är med den ”vanliga” typen av Q-Learning, alltså Tabell Q-Learning. Kraven för detta experiment är att tabellen bör innehålla tillstånd och för varje tillstånd bör det finnas en handling som i sin tur innehåller ett Q-värde.

För att implementera metoden till vår simuleringsmiljö så tilldelas en uppsättning av axelvinklar ett tillstånd. Detta betyder att varje kombination av de olika axelvinklar som roboten kan anta i varje axel, blir ett tillstånd. Då detta blir väldigt många tillstånd tillverkas först en tabell där axelvinklarna skiljer sig med 5°, se figur 4 för exempel.



Figur 4:Illustrerar två olika exempel på uppsättningar av axelvinklar som i sin tur representerar ett tillstånd.

Handlingarna i experimentet består av en förflyttning från ett tillstånd till ett annat tillstånd. För vårt experiment blir detta en rörelse som roboten genomför. Förflyttningen till ett annat tillstånd kan endast genomföras till intilliggande tillstånd. Detta betyder att en handling kommer bestå av en förflyttning i axellederna, med 5° i antingen positiv eller negativ riktning, se Figur 5.



Figur 5:Visar hur på ett ungefär hur varje axel kan ändra sig.

Under rörelsens gång spelas energiförbrukningen in och det är detta värde som representerar belöningen r_t från ekvation 9. Eftersom det är höga belöningar som eftersträvas och inte låga så multipliceras belöningen med minus ett. Detta gör då att mer energikrävande rörelser får ett sämre värde, medan energisnåla rörelser får ett större värde.

Målet, eller uppsättningen av axelvinklar som metoden ska försöka röra sig till, tilldelas en belöning som är mycket högre än de andra. Detta leder till att alla

handlingarna som leder till just den uppsättningen av axelpositioner kommer få högre belöning.

Q-Tabell		Handlingar			
		-5, -5, -5, -5, -5, -5	-5, -5, -5, -5, -5 , 0	-5, -5, -5, -5, -5, +5	. . .
Tillstånd	-180, -90, -230, -200, -115, -400	Q-Värde: 312,23	Q-Värde: -20,34	Q-Värde: 12,52	. . .
	. . .	. . .	. . .	. . .	. . .
	20, -30, 25, 115, -75, 230	Q-Värde: 211,32	Q-Värde: -11,23	Q-Värde: 11,42	. . .
	. . .	. . .	. . .	. . .	. . .
	180, 110, 50, 200, 115, 400	Q-Värde: 401,46	Q-Värde: 64,93	Q-Värde: 15,56	. . .

Figur 6:Exempel hur Q-tabellen kan se ut

För att få ett tillfredställande resultat behöver roboten först röra sig mellan alla möjliga tillstånd med alla möjliga handlingar. När alla handlingar sedan har genomförts så börjar inlärningen som pågår tills Q-värden har konvergerat. En tidsgräns på 8 timmar ställs in för att se till att metoden inte tar för lång tid. Se Algoritm 1 för förklaring i pseudo-kod.

Algoritm 1 Tabell Q-Learning

```
Skapa tabellen med varje tillstånd och handling
Sätt alla Q-värden till 0 förutom alla handlingar som leder till
ett mål
Gå igenom alla möjliga rörelser i alla möjliga tillstånd och
spara belöningen rt från varje
För (varje episod, tills Q-värden har konvergerat eller
tidsgränsen har nåtts)
       Gå till startpositionen som är (0,0,0,0,0,)
       För (för varje steg, tills 30 steg har tagit eller målet
har nåtts)
               Välj ett steg med Epsilon-Greedy policyn
                Observera handlingen at, belöningen rt och nästa
tillstånd st+1
                Sätt in värden i Ekvation 9 och uppdatera Q-
värdet för tillståndet i tabellen
```

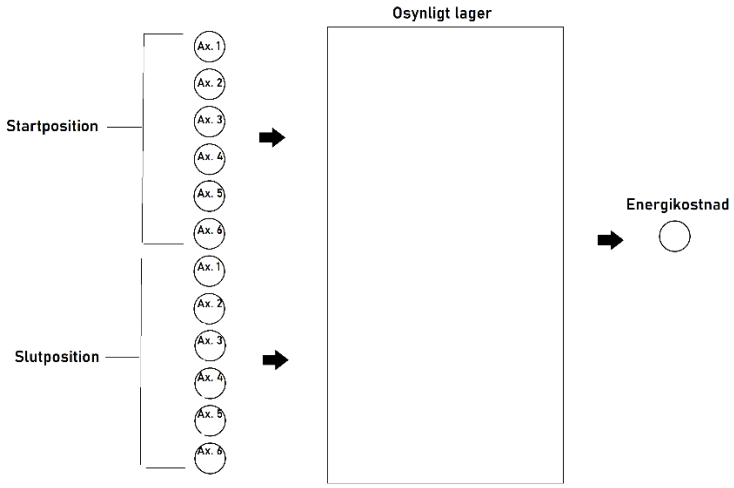
En annan variant av Tabell Q-Learning, som möjligtvis har snabbare inlärningstid, testas även. Metoden använder sig utav större variationer av axelvinklar i början, alltså istället för 5° används 50°. Sedan när Q-värden i tabellen har konvergerat till ett visst värde sållas de tillstånden med lägst värden bort. Tillstånden som är kvar delas då upp i mindre tillstånd där varieringen av axelvinklar minskar till t.ex. 25°. Handlingarnas räckvidd minskas även till ±25°. Processen av att konvergera, sålla ut de dåliga och dela upp de resterande, fortsätter tills processen har kommit ner till ±5° och även där konvergerat.

4.3.2 Träning av Neuronnät

Detta experiment genomförs för att undersöka om det är möjligt, och till vilken grad, att uppskatta energikostnaden av en slumpmässig rörelse med hjälp av neuronnät. Detta experiment genomförs då detta är viktigt att testa och dokumentera innan nästa experiment då nästa experiment annars inte kan fungera.

Inför experimentet slumpas först 1000 par av uppsättningar av axelpositioner för roboten. Varje par av uppsättningar är då två olika tillstånd med en rörelse som binder dem samman. Ett program som kör varje handling och sparar energiförbrukningen i ett text-dokument skapas. Detta dokument används som träningsdata. Sedan genomförs samma process igen, där vi slumpar 1000 nya uppsättningar och kör igenom dem och sparar informationen i ett separat text-dokument. Detta dokument kommer sedan användas som testdata, för att verifiera och evaluera effektiviteten av neuronnätet.

Neuronnätet som används under experimentet består av tolv *inputs*, något osynligt lager och en *output* nod. De tolv *Input*-noderna representerar startpositionen och slutpositionen av roboten. Varje nod representerar en vinkel på en särskild axel. Värdet på *Output*-noden bör, vid lyckade fall, representera energikostnaden av rörelsen mellan startpositionen och slutpositionen. Se figur 7 för illustrering av uppbyggnaden.



Figur 7:Uppbyggnaden av neuronnätet. Det osynliga lagret är i detta fall fortfarande okänt.

Under testerna prövas olika uppsättningar av det osynliga lagret och olika inlärningshastigheter α, för att hitta den mest effektiva kombinationen. I detta fall testas enbart en till tre lager då mer än detta anses vara onödigt komplext för detta experiment. Antalet noder i varje lager varierar från 10 till 500 och inlärningshastigheterna som testas är 0,01, 0,001, 0,0001 och 0,00001 med tröghetsfaktor *m* på konstant 0,9.

Efter att en uppsättning av det osynliga lagret och en inlärningshastighet har valts, påbörjas träningen av nätet. Träningen genomförs på så sätt att informationen från text-dokumentet med träningsdata matas in i neuronnätet. Först passerar startpositionen och slutpositionen in genom *input*-noderna, där det då genomförs ett FP, vilket sedan

resulterar i att *Output*-noden fått fram en gissning på vad det tror energiförbrukningen är. Efteråt genomförs ett BP för att justera nätet så att det till nästa gång kan få fram en bättre gissning.

För att ge en siffra på hur fel gissningen av neuronnätet var används ekvation 11, där först skillnaden mellan gissningen T och det faktiska värdet R beräknas. Skillnaden divideras sedan med *T* för att få fram en procentuell skala på storleken av felet. Eftersom negativa värden på *Loss* inte är önskvärda tas absolutbeloppet av skillnaden.

$$Loss = \frac{|T - R|}{T} \qquad \textit{Ekv. 11}$$

För att få en bättre överblick av hur neuronnäten presterar används ett linjediagram där storleken på felet plottas utan träningen.

Under träningen av de olika neuronnät och inlärningshastigheterna sparas de neuronnäten och inlärningshastigheterna vars fel gick mot noll. Om felet under träningen inte konvergerar mot noll är det misslyckat och den uppsättningen bör inte testas vidare.

Vid nästa steg genomförs samma test, men efter varje gång neuronnätet anpassar och rättar sig efter träningsdatan genomförs enbart ett FP på testdatan. Felet som produceras plottas även ut i linjediagrammet, men av en separat linje. Om felet för testdatan konvergerar är det ett godkänt test, men om felet divergerar betyder det att neuronnätet antigen har misslyckats eller tränat för länge. Algoritm 2 visar förenklat stegen för processen.

Algoritm 2 Träning av neuronnät

```
Slumpa 1000 olika par av uppsättningar och spara energiförbrukningen
av rörelsen mellan dem
Slumpa 1000 olika par av uppsättningar och spara energiförbrukningen
av rörelsen mellan dem
Välj en uppsättning av neuronnät och en inlärningshastighet
Initialisera neuronnätet
While (true)
       För varje träningsdata
              FP
              BP
              Skriv ner det genomsnittliga felet
        För varje testdata
               FP
               Skriv ner det genomsnittliga felet
```

För att säkerställa att neuronnäten får förbättrade resultat vid en större mängd träningsdata testas även en ökning och sänkning av mängden rörelser i träningsdatan.

4.3.3 *Deep Q-Learning Network*

Det sista experimentet som genomförs är likt en kombination av det första experimentet med Tabell Q-Learning och det andra experimentet med neuronnät. Experimentet är uppdelat i två delar där olika tester genomförs, men neuronnäten som används i dessa test är samma som de neuronnät som fick bäst resultat i det föregående experimentet i delkapitel 4.3.2.

I den först delen genomförs ett liknande test som innan, men istället för att slumpa fram olika uppsättningar av axelpositioner används axelpositioner som skiljer sig med 10º från varandra, likt första experimentet. Detta betyder att varje startposition och slutposition som matas in i neuronnätet kommer skilja sig 10º i minst en axel.

Eftersom målet med algoritmen här är att röra sig mot en specifik axelposition, med en så låg energiförbrukning som möjligt, behöver alla handlingar som resulterar i den positionen ge en väldigt hög belöning. Vi kallar den positionen för mål-positionen och då detta är ett point-to point experiment som genomförs behövs också en hem-position som väljs valfritt.

Likt det föregående experimentet, spelas 1000 olika rörelser in för att skapa träningsdata. Sättet rörelserna väljs ut är däremot annorlunda och är i detta fall inte slumpmässigt valda. Eftersom neuronnätet behöver kunna skilja på olika handlingar i samma positioner behövs slumpmässiga handlingar tas vid liknande positioner. Detta görs genom att spela in slumpmässiga handlingar som börjar från hem-positionen. Efter att ett antal rörelser har tagits återgår roboten till hem-positionen och tar ett antal nya slumpmässiga rörelser. Efter att 1000 rörelser har spelats in till träningsdata, spelas ytterligare 1000 rörelser in till testdata. Algoritm 3 förklarar processen för inspelning av rörelser.

Algoritm 3 DQN inspelning av rörelser

```
Välj en uppsättning av neuronnät och en inlärningshastighet
Starta i hem-positionen
For (X antal steg eller tills målet nås)
        Spela in startpositionen, slutpositionen och
energiförbrukningen av rörelsen och överför
det till ett text-dokument.
```

Anledningen till att detta experiment genomförs är för att undersöka om det gör någon skillnad på inlärningen om rörelserna i test- och träningsdatan har liknande start- och slutpositioner. Experimentet görs också på detta vis för att det ger systemet en ändlig mängd handlingar, vilket i detta fall är alla kombinationer av 10 graders vridningar på axlarna, som är 728 stycken.

Sedan, i andra delen av experimentet, genomförs samma test med träningsdata och testdata, fast med TN aktivt. Syftet med TN är att uppskatta framtida handlingars värde. TN är en kopia av neuronnätet som används i vanliga fall men vikterna i nätet uppdateras endast efter att en viss tid har gått. Detta förhindrar att TN hamnar i en loop där det uppskattar högre och högre värden, tills det går mot oändligheten. Algoritm 4 förklarar hur träningen av DQN genomförs.

Algoritm 4 DQN träning med TN

```
Välj en uppsättning av neuronnät och en inlärningshastighet
Initialisera neuronnätet
While (true)
       För varje träningsdata
                FP
                Kolla vad den bästa handlingen vid slutpositionen är
och använd det resultatet i BP
                BP
                Skriv ner det genomsnittliga felet
        För varje testdata
                 FP
                 Kolla vad den bästa handlingen vid slutpositionen är
och använd det resultatet i beräkningen av felet
                 Skriv ner det genomsnittliga felet
```

5 Resultat

I detta kapitel redovisas de mest väsentliga resultaten från experimenten som utfördes.

5.1 Tabell Q-Learning

Då träningstiden, även med olika modifikationer för att minska tiden, alltid översteg tidsgränsen på 8 timmar fick testerna avbrytas.

Vid en snabb beräkning av hur lång tid genomgången av de olika tillstånden och deras tillhörande handlingar skulle ta, kom det upp till över 300 timmar.

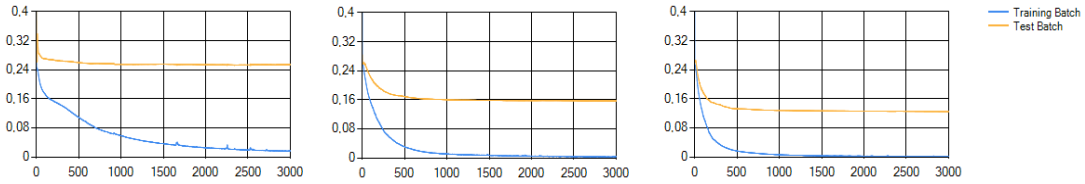
5.2 Träning av Neuronnät

Under testerna för att ta fram ett effektivt neuronnät till uppgiften blev det till slut ett neuronnät med ett osynligt lager som innehöll 300 noder. Inlärningshastigheterna, α, som ansågs fungera bäst var de inom intervallet av [0,00001, 0,0001]. Dessa α gav tillräckligt snabba och stabila resultat.

Under träningen av neuronnät med två och tre osynliga lager vägrade felet konvergera mot noll. Felet planade istället ut vid en nivå där ett medelvärde hittades och storleken på felet ändrades inte, även när olika inlärningshastigheter applicerades.

Nedan i figur 8 visas tre grafer som visar hur storleken av felet ändras med inlärningstiden. I graferna syns det även hur stort felet blir för både träningsdata och testdata. Anledningen till att det är tre grafer är för att illustrera hur platån som testdatan når blir lägre och lägre vid ändring av storleken på träningsdatan.

I den första bilden har 250 rörelser använts som träningsdata, i den andra har 500 rörelser använts och i den sista har 1000 använts. För alla grafer har samma storlek av testdata behållits på 1000 rörelser



Figur 8:Visar tre olika grafer där olika storlekar av träningsdata har använts under träningen. I experimentet undersöktes slumpmässiga rörelser.

Tabell 1 visar tiden det tog att genomföra alla FP och BP på tränings- och testdatan hundra gånger.

Antal rörelser i träningsdata	250	500	1000
Tid per 100 steg	~47sek	~56sek	~1min och 12sek

Tabell 1: exekveringstid för 100 pass genom träning/testdata. I experimentet undersöktes slumpmässiga rörelser

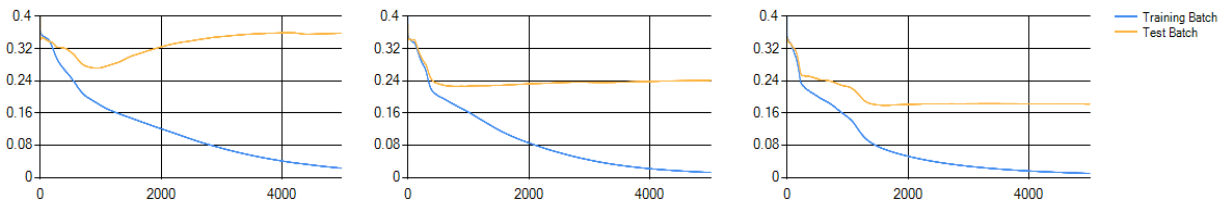
5.3 *Deep Q-Learning Network*

Under de två testerna som genomfördes på DQN-metoden så användes samma uppsättning av neuronnät och inlärningshastigheter som var ett neuronnät med endast ett osynligt lager och 300 noder i det lagret.

I första testet som genomfördes så testades neuronnätets förmåga att uppskatta energiförbrukning av olika rörelser med en variering av 10º i minst en axel. Förmågan att uppskatta energiförbrukningen av olika rörelser med liknande startpositioner testades även här då det var viktigt för att få ett bra resultat. Detta test var sig väldigt likt det förra.

Nedan i figur 8 visas tre olika grafer som redogör för hur felet för uppskattningarna av energiförbrukningen ändras under träningstiden. De olika graferna representerar felet som träningsdatan får fram och felet som testdatan får fram. Det värt att notera att graferna representerar olika mängder av rörelser i träningsdata, likt det förra experimentet och dess motsvarande grafer.

I den första bilden har 250 rörelser använts som träningsdata, i den andra har 500 rörelser använts och i den sista har 1000 använts. För alla grafer har samma storlek av testdata behållits på 1000 rörelser.



Figur 9:Visar tre olika grafer olika mängder av träningsdata påverkar testdatans konvergering. I experimentet undersöktes rörelser som flyttade 10º på minst en axel.

Tabell 2 nedan visar tiden det tog att genomföra alla FP och BP på tränings- och testdatan, hundra gånger.

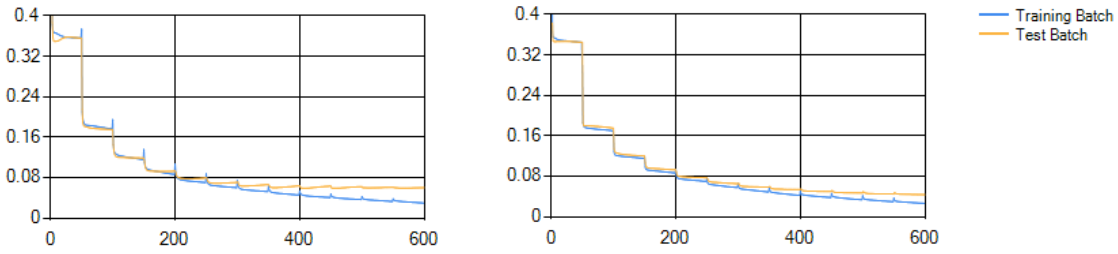
Antal rörelser i träningsdata	250	500	1000
Tid per 100 steg	~50	~1min och 2sek	~1min och 18sek

Tabell 2: exekveringstid för 100 pass genom träning/testdata. I experimentet undersöktes rörelser som flyttade 10º på minst en axel.

I andra testet som genomfördes undersöktes hur implementeringen av TN och förmågan att ta framtida belöningar i åtanke, påverkade resultatet. I detta test användes samma träningsdata och testdata som i det föregående testet.

I figur 10 nedan redovisas två olika grafer med olika mängder träningsdata, likt förra testet. Graferna visar hur felet ändras under träningstiden på både träningsdatan och testdatan, som representeras av olika linjer. Det är viktigt att notera att uppdateringen av TN skedde efter 50 steg.

I den första bilden användes 250 rörelser som träningsdata och i den andra har 500 rörelser använts. Det tredje testet var tvunget att avbrytas då det översteg tidsgränsen på 8 timmar.



Figur 10: Visar två olika grafer med olika mängder träningsdata som genomförs med implementeringen av ett target network.

Tabell 3 visar tiden det tog att genomföra alla FP och BP på tränings- och testdatan hundra gånger. I detta test ingick även tiden det tog för neuronnäten att uppskatta värdet på den mest värdefulla framtida handlingen.

Antal rörelser i träningsdata	250	500
Tid per 100 steg	~25 minuter	~50 minuter

Tabell 3: exekveringstid för 100 pass genom träning/testdata med target network aktivt.

6 Slutsats

6.1 Diskussion

6.1.1 Metod

Under examensarbetets gång var det tre faktorer som minskade precisionen och noggrannheten av resultaten. Det första var beslutet att programmera neuronnäten och funktionerna av neuronnäten själv. Problemet med att programmera dessa funktioner från grunden är inte att resultatet blir fel, utan att processtiden blir mycket längre.

Den andra faktorn som bidrog till ett mindre noggrant resultat var mätningarna av energiförbrukningen. Vid en undersökning där noggrannheten av mätningarna gjordes så testades först några olika rörelser som mättes upp till att kosta ungefär 20 000 Joule. Dessa rörelser spelades sedan upp 20 gånger och den procentuella skillnaden mellan det högsta och lägsta värdet var varje rörelse jämfördes. Den procentuella skillnaden låg då i intervallet [0,4%, 2%].

Samma test gjordes sedan på rörelser som kostade ungefär 2000 och 300 Joule. Resultatet för dessa blev [2%, 6%] och [4%, 10%]. Anledningen till att kostnaderna varierade så pass mycket kan ha berott på att mätningarna av vinkelhastighet och mätningarna av vinkelmoment inte var synkroniserade.

Det sista som minskade noggrannheten av arbetet var användningen av rörelser som endast skiljde sig med 5º och 10º. Eftersom ju större vinklar som används desto mindre intressant blir arbetet då inget företag styr industrirobotar med stora variationer i vinklar, men samtidigt så minskade mätfelet ju större vinklar användes. Detta blev lite av ett dilemma orsakat av det förgående problemet.

Det är däremot viktigt att påpeka att detta endast försämrade resultaten lite och inte påverkade slutsatserna i slutändan.

6.1.2 Tabell Q-Learning

Anledningen till att Tabell Q-Learning inte kunde användas i simuleringen var att oavsett vilken metod som användes blev antalet tillstånd och handlingar för många. Eftersom alla möjliga handlingar och tillstånd måste genomföras minst en gång för att ta reda på energikostnaden behövs väldigt mycket tid. Det blir alltså för många rörelser som måste genomföras.

Om det skulle vara möjligt att snabbspola simuleringen eller på något annat sätt veta i förväg vad energikostnaderna är för varje handling, så skulle denna metod möjligtvis kunna fungera.

Däremot om metoden skulle tränas med en industrirobot som inte har några laster på sig så skulle samma Q-Learning tabell kunna återanvändas flera gånger, och om olika start- och slutpositioner skulle användas skulle det gå att justera det. I detta fall skulle

en Q-Learning tabell behöva tas fram varje gång en ny last på roboten används och roboten skulle inte ha förmågan att anpassa sig till ändringar i miljön.

6.1.3 Träning av Neuronnät

Träningen av neuronnätet med slumpmässiga rörelser gav förvånansvärt bra resultat då felet på testdatan konvergerade närmare noll vid ökningar av träningsdata. En annan anledning för att systemet lyckades så pass bra var för att i rapporten har endast start- och slutpositioner använts som *input*-värden i neuronnäten, men i liknande arbeten och rapporter har andra värden använts, så som acceleration och hastighet. Detta visar då att det är möjligt att använda sig av mindre information för att uppskatta energiförbrukning.

Det resultat som var mest förvånande var att användningen av fler osynliga lager än ett aldrig fungerade. Det kan ha berott på ett flertal anledningar, men det enda som kommer i tanke är att metoden för initialisering av vikterna inte var särskilt bra i just detta fall. Detta kan också ha berott på att inlärningshastigheten som var för hög och kanske systemet behövde mer träningstid men detta är då orsakat av den ineffektiva programmeringen av neuronnäten och dess funktioner.

6.1.4 *Deep Q-Learning Network*

Det första testet genomfördes för att säkerställa att neuronnätet hade förmågan att skilja på olika handlingar i samma startpositioner, till skillnad från det föregående testet som endast genomfördes för att undersöka om neuronnätet hade förmågan att uppskatta energikostnaden från slumpmässiga rörelser. Eftersom platån för testdatan minskade för varje ökning av träningsdata visade det sig att neuronnätet även kan tränas för detta.

Det är även intressant att undersöka om platåerna är lägre mellan detta test och det föregående. Vid jämförelse av graferna visade det sig att det föregående experimentet hade lägre platåer. Detta skulle kunna bero på att neuronnätet har svårare att tränas upp om det behöver skilja på liknande rörelser i liknande positioner.

Det andra testet som genomfördes var för att undersöka om det är möjligt att uppskatta framtida handlingar med DQN. I graferna syns det hur felet konvergerar mot noll och det syns även hur felet når en lägre platå vid ökning av träningsdata, vilket är ett bra tecken då detta antyder att vid tillräckligt hög träningsdata är det möjligt att använda sig av metoden.

Resultatet som var mindre tillfredställande var tiden som det tog för andra experimentet. Det tog till och med så pass lång tid att det tredje testet, det med 1000 rörelser i träningsdatan, var tvunget att avbrytas. Anledningen till detta var att algoritmen hela tiden behövde, med hjälp av TN, uppskatta den framtida mest värda handlingen. Eftersom det finns 728 möjliga handlingar att ta vid varje position så tar det lång tid men detta kan också ha varit orsaken av en ineffektiv programmering.

Vid träning av ett DQN finns det en konstant γ som representerar hur mycket framtida belöningar bör tas hänsyn till. Konstanten kan anta vilket värde som helst i intervallet

[0, 1] och vid högre värden tas framtida belöningar mer hänsyn till, men vid högre värden ökar även inlärningstiden. Eftersom 0.99 användes i arbetet kan det möjligtvis ha varit en bidragande faktor till varför det tog så pass lång tid för felet att konvergera.

6.2 Reflektion över problemformulering

1a. Vilken typ av industrirobot ska undersökas?

Industriroboten IRB 140 från ABB undersöks

b. Hur många axlar har den?

IRB 140 är en 6-axlad robot

c. Hur beräknas energiförbrukningen?

Energiförbrukning beräknas med hjälp av ekvation 1 från arbetet

2a. Vilket simuleringsprogram bör användas?

Simuleringsprogrammet RobotStudio som är tillverkat av ABB används

b. Hur implementeras Q-Learning I simuleringsprogrammet?

Simuleringsprogrammet ansluts till Visual Studio som i sin tur är värden för Q-Learning algoritmen.

c. Hur implementeras neuronnät i simuleringsprogrammet?

Simuleringsprogrammet ansluts till Visual Studio som i sin tur är värden för neuronnäten och funktionerna kring neuronnäten.

3. Vad för slags simulering är det som kommer genomföras? Vilka typer av rörelser bör undersökas?

De rörelser som undersöks är axelrörelser. Det finns ett par andra sätt att röra industriroboten på, men de undersöks inte i detta arbete. Sedan är det point-to-point tester som genomförs.

4. Hur evalueras metoderna?

Metoderna evalueras på noggrannhet och inlärningstid. Noggrannheten eller precisionen av programmet övervakas genom att felet som metoderna uppskattar plottas i grafer.

6.3 Etiska Aspekter

Ingenjörernas hederskodex är tio punkter som en ingenjör bör sträva efter att följa i sitt arbete. Hederskodexen finns för att fungera som en slags riktlinje för hur arbeten bör utföras och för att föra information och tekniker vidare till framtida generationer. I

detta examensarbete har de flesta punkterna i hederskodexen försökt att följas, men den som har varit i extra fokus har varit punkten kring att ens teknik bör ha någon samhällsnytta. [10]

Metoderna som har undersökts, har undersökts i hopp om att upptäcka nya sätt att styra industrirobotar på ett energisnålt sätt. Detta anses göra samhället nytta då detta möjligtvis kan spara el och energi. Hur mycket energi som egentligen sparas av industriroboten och används av processorn för beräkningar är en annan fråga.

Sedan, ur ett säkerhetsperspektiv, kanske metoden inte är särskilt försvarbar då detta är en experimentell teknik som, om används på fel sätt, kan leda till skador.

6.4 Framtida utvecklingsmöjligheter

Metoden med DQN är med stor sannolikhet möjlig att implementera för att spara energi, men det behöver undersökas vidare. För att bli bättre än de matematiska metoderna behöver systemet ha en så pass snabb inlärningstid att det kan justera sig vid ändringar i miljön, så som ändringar i motstånd och laster, i realtid samtidigt som det kan kartlägga en lika bra rörelseväg som de andra metoderna.

Metoden kan fungera bättre i andra sammanhang där experimentet inte är ett point-to-point experiment utan innehåller flera punkter/hållplatser som behöver nås. I ett sådant experiment kanske denna metod skulle vara mer lämplig, då att planera olika depåstopp är svårt matematiskt, men det skulle behöva undersökas.

Vid vidare undersökning av metoden skulle effektivare programmering och högre precision av mätningar på energi först behöva uppnås, i delkapitel 4.2 motiveras andra val av program och metoder som skulle kunna bidra till detta. Sedan skulle ökningar av träningsdata testas vidare för att kartlägga hur mycket träningsdata det behövs för en viss precision av neuronnätet på testdatan, och till sist behöver metoden jämföras med andra energisparande metoder. Detta skulle då ge en bättre bild för hur väl metoden fungerar men än så länge har endast några steg mot det målet tagits.

7 Terminologi

DQN = Deep Q-Learning Network

TN = Target Network

FP = Forward Propagation

BP = Backpropagation

8 Källförteckning

[1] B. Siciliano, O. Khatib. *Handbook of Robotics*. 2nd ed. Springer Publishing Company. 2016.

[2] ABB automationsföretag, *IRB 140 Product Specification*. 2020-12-16

https://search.abb.com/library/Download.aspx?DocumentID=3HAC041346-001&LanguageCode=en&DocumentPartId=&Action=Launch (hämtad 2021-04-19)

[3] ABB automationsföretag. *RobotStudio*.

https://new.abb.com/products/robotics/robotstudio (hämtad 2021-03-26)

[4] ABB automationsföretag. *Developer Center*. 2020-09-22

https://developercenter.robotstudio.com/ (hämtad 2021-03-26)

[5] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feedforward neural networks", *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[6] Mohit Sewak. *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. 1st ed. Springer Publishing Company; 2019.

[7] Glorot, X. & Bengio, Y.. (2010). "Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics", in *Proceedings of Machine Learning Research* 9:249-256 Available from http://proceedings.mlr.press/v9/glorot10a.html

[8] Russell C. Eberhart, Roy W. Dobbins and Larrie V. Hutton. "Performance Metrics". In: Russell C. Eberhart and Roy W. Dobbin, editors. *Neural Network PC Tools*. 1st ed. Academic Press 1990.

[9] Ning Qian, "On the momentum term in gradient descent learning algorithms", *Neural Networks*, Volume 12, Issue 1,1999, Pages 145-151, ISSN 0893-6080, https://doi.org/10.1016/S0893-6080(98)00116-6.

[10] Sveriges Ingenjörer. *Hederskodex*. 2019-09-13. https://www.sverigesingenjorer.se/om-forbundet/sveriges-ingenjorer/hederskodex/ (hämtad 2021-05-17)