Hybride Lernstrategien für dateneffiziente Robotik

Ausarbeitung zum Proseminar „Informatik trifft Maschinenbau“

Zugeteiltes Paper:

Vision intelligence-conditioned reinforcement learning for precision assembly

Jesse Marekwica
Matrikelnummer: 238530
2026-01-10

Inhaltsverzeichnis

1. Einleitung & Motivation . . . 2
2. Systemarchitektur & Problemmodellierung . . . 2
   2.1. Markov Decision Process (MDP) . . . 2
   2.2. MDP zur Modellierung eines Montageproblems . . . 4
   2.3. Actor-Critic-Modell . . . 5
      2.3.1. Critic . . . 6
      2.3.2. Actor . . . 6
3. Effizientes Online Reinforcement Learning mit offline Daten - RLPD . . . 7
   3.1. Hybrides Buffer-System und Replay Ratio . . . 7
   3.2. Layer Normalization . . . 8
   3.3. Sample Efficient RL . . . 9
4. Diskussion und Evaluation . . . 10
   4.1. Fazit . . . 11
Bibliografie . . . 12

1. Einleitung & Motivation

Roboter in der Industrie, insbesondere der Fertigungstechnik, sind für Unternehmen essenziell geworden und reduzieren hohe Kosten durch Fachkräfte, erhöhen die Zeiteffizienz und sind in Zeiten der Industrie 4.0 weit etabliert. Dies lässt sich am Beispiel der Automobilindustrie veranschaulichen, in der Firmen wie KUKA bereits weit verbreitet mit ihren Automatisierungslösung sind. Dort werden hochautomatisierte Produktionsketten als Lösung angeboten, die hauptsächlich auf Flexible Robotik setzen, in denen Roboterarme Aufgaben wie Montage, Schweißen oder Lackierung übernehmen [Quelle]. Laut einer Analyse der International Federation of Robotics (IFR) erreichte der weltweite operative Bestand an Industrierobotern zuletzt mit rund einer Million Einheiten einen neuen Höchststand (2023) [1].

Trotz der guten Etablierung weisen diese Lösungen dennoch Nachteile bezüglich Feinmotorik, kontaktreicher und dynamischer Aufgaben auf. Die oben genannten Lösungen basieren meist auf statischen Regelwerken und gehen von deterministischen Abläufen aus. Tauchen in der Produktionskette kleinere Fehler auf, müssen diese meist durch Eingriffe von Menschen behoben werden, da klassische Robotiksysteme keine „intelligente“ Reaktion auf derartige Probleme geben. Eine Lösung zur statischen Programmierung und festen Regelwerken, könnte die Informatik liefern. Die vorliegende Arbeit untersucht das [Quelle] liefert eine Lösung, die Konzepte aus der Informatik wie Markowsche-Entscheidungsprobleme, Reinforcement Learning und neuronale Netze verwendet. Alle genannten Konzepte sind in der Informatik moderne Technologien und erfordern tiefes Wissen aus der Informatik.

Das Paper postuliert eine mögliche Lösung von Robotern, in der Präzisionsarbeit durch intelligente visuelle Verarbeitung von räumlicher Umgebung und „Human-in-the-Loop“ Reinforcement Learning erzielt wird. Dabei werden neuronale Netze, lernende Algorithmen und intelligente Impedanzcontroller verwendet, die gewisse Informatikkenntnisse voraussetzen. Demonstriert wird die Methode anhand der Montage von Computer-Hardware-Komponenten (konkret: RAM-Module und Kühlsystem auf einem Mainboard). Die Montage solcher kontaktreichen Komponenten mit der Kombination von Schrauben, korrektes Einsetzen und Widerstandserkennung beim RAM, würde einen großen Implementierungsaufwand beim klassisch statischen Programmieren aufweisen. Zusätzlich dazu wäre die Fehlerquote wahrscheinlich trotzdem hoch, da Abweichungen in Millimeterbereich bereits schwerwiegend sein könnten (ein RAM-Riegel, der 1mm daneben liegt, wird nicht passen). Trotz dieser Herausforderungen, erreichten die Autoren eine nahezu perfekte Erfolgsquote von über 98%.

Im Folgenden wird tiefer auf die Informatik des Papers eingegangen. Zuerst wird das eigentliche Problem als ein Markov Decision Process (MDP) modelliert. Dies ist nötig, da Reinforcement Learning Algorithmen auf solche MDP operieren. Des weiteren wird dann noch die verteilte Architektur, das Actor-Learner-Modell kurz erklärt und übergeleitet zur Umsetzung des lernenden Algorithmus. Der im Paper vorgestellte RLPD (Reinforcement Learning with Prior Data) bietet eine dateneffiziente Lösung an, die menschliches Eingreifen ermöglicht und einen Vision-Encoder verwendet, der auf ResNet-10, ein Convolutional Neural Network (CNN) basiert. Aufbauend wird noch kurz die Impedanzregelung vorgestellt. Abgeschlossen wird die Ausarbeitung mit einem Vergleich zu anderen Lösungen, kurzer Einschätzung und Folgerung für den Maschinenbau.

2. Systemarchitektur & Problemmodellierung

2.1. Markov Decision Process (MDP)

Das Paper modelliert das Montageproblem als einen Markov-Decission-Process (MDP). Dabei handelt es sich um eine formale, mathematische Definition eines Entscheidungsproblems, welches hier zur Optimierung der Montage verwendet wird. Ein MDP lässt sich als gerichteter Graph modellieren, wobei die Knoten als Zustände und die Kanten als Zustandsübergänge (Transitionen) interpretiert werden, die durch Handlungen ausgelöst werden. Zum besseren Verständnis wird ein bekanntes Beispiel aus

der Vorlesung von David Silver (DeepMind / UCL) betrachtet: Der beschriebene Graph [Abbildung 1] modelliert den Studienalltag.

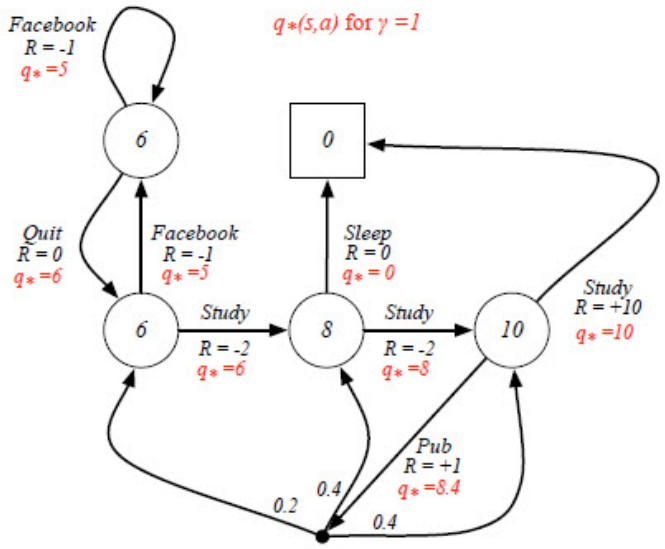

Abbildung 1: Optimal Action-Value Function for Student MDP - David Silver.

Um seinen Kurs bestehen zu können, müssen Studenten alle drei „Class"-Zustände erfolgreich durchlaufen. Die Kreise repräsentieren hierbei die Zustände. „Class 1" dient hier als Start-Zustand. In diesem Zustand kann der Agent (Student) nun eine Handlung wählen: Entweder „Facebook" oder „Study". Wählt er „Study", folgt eine Transition, die mit einer Wahrscheinlichkeit behaftet ist (in dem Fall implizit 1.0/100%). Der Reward (R) ist der Wert, den der Agent für das Verweilen in einem Zustand oder das Ausführen einer Aktion erhält. In „Class 1" kostet jeder Zeitschritt beispielsweise $R = -2$ (negativer Reward / Bestrafung). Sobald sich der Agent in „Class 3" befindet, kann er sich für „Pub" entscheiden. Von da aus landet er mit unterschiedlichen Wahrscheinlichkeiten in „Class" 1-3 (siehe Graph). „Sleep" ist ein terminierender Zustand, der das Ende markiert.

Das Ziel (die Optimierung) in einem MDP besteht darin, eine Strategie (Policy) zu finden, die die Summe der erwarteten Rewards maximiert.

Um dies auf das Paper zu übertragen, nutzen wir dessen formale Definition eines MDPs:

$$\boldsymbol{M = \{S, A, P, p, R, \gamma\}}$$

- $\boldsymbol{S}$ **(State Space)**: Beschreibt die Menge aller Zustände. In unserem Beispiel ist S alle dargestellten Knoten, wobei $s \in S$ ein konkreten Knoten beschreibt
- $\boldsymbol{A}$ **(Action Space)**: Beschreibt die Menge aller verfügbarer Aktionen. Im Zustand „Class 1" wäre das „Facebook" oder „Study".
- $\boldsymbol{P}$ **und** $\boldsymbol{p}$ **(Wahrscheinlichkeiten)**: Das große P beschreibt die **Startverteilung** (Initial State Distribution). Da es praktisch unendlich viele Startkonfigurationen geben kann, gibt P an, wie wahrscheinlich es ist, in einem bestimmten Zustand s_0 zu starten. Im Beispiel wäre „Class 1" unser Start-Zustand mit $P(Class1) = 1.0$. Das kleine p beschreibt **Systemdynamik** (Transition Probabilities). Es gibt die Erfolgswahrscheinlichkeit einer gewählten Aktion an. Im Beispiel: Wenn man „Pub" in „Class 3" wählt, ist $p = 0.4$ für den Übergang zu „Class 2" oder in „Class 1" für „Facebook" $p = 1.0$. Diese Defintion unterscheidet sich mit der von David Silvers Vorlesung. Um auf weiteres Vorgehen aufzubauen, wird sich an die Definition des Papers gehalten.
- $\boldsymbol{R}$ **(Reward Function)**: Bewertet die Qualität der Entscheidung. Im Beispiel erhält man $R = -2$ für „Study". Dieser Wert ist der entscheidene Parameter, an dem die Optimierung gemessen wird.

- γ **(Discount Factor)**: Dies ist der Gewichtungsfaktor $(0 \leq \gamma < 1)$, der bestimmt, wie wichtig zukünftige Belohnungen im Vergleich zu sofortigen sind. Ein γ nahe 0 macht den Agenten „kurzsichtig“ (nur der nächste Reward zählt), während ein γ nahe 1 den Agenten „weitsichtig“ macht (langfristige Ziele wie „Pass“ werden wichtiger als kurzes Facebook-Vergnügen). Im der Vorlesung von David Silver wurden beide Beispiele einmal gezeigt, wobei ein $\gamma = 0$ in einer Facebook-Schleife verfiel und ein $\gamma = 1$ in „Pass“ überging.

Hat man nun ein MDP formuliert, gilt es dieses zu lösen, wobei die Lösung bei einem Optimierungs-/Maximierungsproblem der höchstmögliche erreichbare Wert beschreibt, in unserem Fall der Reward R. Die gängiste Vorgehensweise bei dem lösen von MDP ist das Aufstellen einer Strategie, einer sogenannten **Policy** (π), denn nach einem bekannten Theorem [Quelle] gilt, dass es für jeden MDP eine optimale Policy (π) gibt, die besser oder gleich aller anderen Policies ist, also $\pi_* \geq \pi, \forall \pi$. Auf unser Bespiel bezogen, wird versucht eine Policy (π_*) gesucht, die den höchstmöglichen Wert in

$$E\left[\sum_{t=0}^{h} \gamma^t R(s_t, a_t)\right]$$

findet. Da die Übergänge zwischen den Zuständen stochastisch und dementsprechend Unsicherheiten vorhanden sind, bilden wir den Erwartungswert aller möglichen Verläufe. Das γ dient hier wie bereits angedeutet als Diskontierungsfaktor, der je nach Wahl über Zeitschritte t die Gewichtung immer kleiner bis nahe 0 einfließen lässt. Die Rewardfunktion mit $R(s_t, a_t)$ definiert für den derzeitigen Zustand s_t unter der Handlung a_t den Reward R. Dieser wird dann bis zum Angegebenen Horizont h akkumuliert.

Für das Beispiel eines Studentenleben ist eine optimale Policy (π_*) einfach zu bestimmen, indem wir über die **optimale Action-Value Function** $Q_*(s, a)$ kennt. Diese Funktion gibt an, welchen maximalen akkumulierten Reward man erwarten kann, wenn man im Zustand s die Aktion a wählt und danach optimal weitermacht. Ein Blick auf [Abbildung 1] verdeutlicht dies am Zustand „Class 3“:
- Der Wert für Lernen ist $q_*(\text{Class } 3, \text{Study}) = 10$.
- Der Wert für die Kneipe ist $q_*(\text{Class } 3, \text{Pub}) = 8.4$.

Da $10 > 8.4$, ist die Handlung „Study“ hier die optimale Wahl. Die Policy π_* wählt also stets gierig („greedy“) das Maximum über Q_*. Ein bekanntes Vorgehen für das bestimmen von Q_* in allen Zuständen ist der Beginn am Ende und zurückschauen in Vorgängerzuständen. Nachdem vereinfacht ausgedrückten Bellmann'schen Optimalitätsprinzip [Quelle] gilt nämlich, dass jede Teilpolicy π_{sub} einer optimalen Policy π_*, auch optimal ist. Zur Veranschaulichung: Wenn die kürzeste ICE Strecke von Dortmund nach München, über Leibzig ist, dann ist die kürzeste Strecke von Leibzig nach München die gleiche. Aufgrund dieser Eigenschaft können wir das Problem vom Ende her lösen („von München zurück nach Dortmund“). Man beginnt in den terminierenden Zuständen (z.B. „Sleep“ mit Wert 0) und berechnet die Werte der davorliegenden Zustände rekursiv rückwärts. Dabei gilt für jeden Schritt: Der Wert eines Zustands ist der sofortige Reward plus der (bereits berechnete) maximale Wert des Nachfolgezustands. So propagieren sich die korrekten Q_*-Werte von hinten nach vorne durch den gesamten Graphen, bis für alle Zustände $s \in S$ die optimale Entscheidung feststeht. Daraus leiten wir dann unsere optimale Policy $\pi_*(a|s)$ ab, also mit welcher Wahrscheinlichkeit wir Handlung a in Zustand s wählen.

2.2. MDP zur Modellierung eines Montageproblems

Da nun eine gewisse Grundlage für Verständnis von MDPs geschaffen wurde, wird nun das einfache Beispiel mit dem Montageproblems des Papers verglichen. Es existieren nämlich starke Unterschiede in der Bestimmung einer optimalen Policy π_*. Die Autoren modellieren ihr Montageproblem, ebenfalls als ein Markov-Decision-Process (MDP) mit $M = \{S, A, P, p, R, \gamma\}$. Während im Prinzip die Elemente des MDPs konzeptionell gleich bleiben, unterscheiden sie sich in der Umsetzung.

- S **(State Space)**: Die Zustandsmenge wird in dem Paper definiert als **State observation space**, also der gesamte beobachtbare Bereich der Montage über die Kameras und Zustand des Armes. Die Kameras nehmen Bilder auf, während der Roboter selbst auch die über ein ResNet-10, ein Convolutional Neural Network, in Vektoren übersetzt werden. Der Grund für die Übersetzung folgt aus der Komplexität der Lösungsmethode über Reinforcement Learning.
- A **(Action Space)**: Die Handlungsmöglichkeiten werden über die kartesische Koordinatenposition des Roboterarms und Griff-Zustand definiert.
- P **und** p **(Wahrscheinlichkeiten)**: Spielen hier eine deutlich relevantere Rolle. Der Roboterarm wird kaum immer von der gleichen Stelle aus Anfangen, zu montieren, noch wird das Mainboard immer exakten Millimeterbereich gleich liegen. Deshalb muss eine Wahrscheinlichkeitsverteilung $P(s)$ definiert werden, die unterschiedliche Start-Zustände modelliert, während p nicht mehr einfache Wahrscheinlichkeiten an einer Transition darstellt, sondern die gesamte Dynamik des Systems repräsentiert. Der Roboterarm wird egal wie präsize er ist, sich mit einer gewissen physikalischen Schwankung von der angegebenen Trajektorie abweichen. Aufgrund der physikalischen Komplexität, ist p uns nicht bekannt, sondern nutzen hier Reinforcement Learning um das p zu approximieren.
- R **(Reward Function)**: Die Autoren haben für R ein binäres Klassifizierungssystem gewählt, dass anhand von vorher tranierten Demos beurteilt, ob eine Montage erfolgreich, oder fehlgeschlagen ist. Die Reward Function bringt das PD in RLPD (RL with Prior Data) rein.
- γ **(Discount Factor)**: Erfüllt den exakt selben Zweck wie in unserem einfachen Beispiel.

Bei näherer Betrachtung der Komponenten S und p zeigen sich die zentralen Herausforderungen dieses Ansatzes: die **stochastische Systemdynamik** (Nicht-Determinismus) und die **enorme Dimensionalität** des Zustandsraums. Letztere wird besonders bei den Sensordaten deutlich: Die beiden Handgelenkskameras (RealSense D405) liefern einen kontinuierlichen Strom an RGB-Bilddaten. Bei einer Frequenz von 30 Hz und einem Crop von 128×128 Pixeln müssen pro Sekunde knapp 1 Million Bildpunkte verarbeitet werden. Unter der Annahme einer Standard-Farbtiefe (8-Bit pro Kanal) entspricht dies einem Datenvolumen von ca. 3 MB/s. Anhand dieser Grundlage ist es nicht möglich, eine optimale Policy π_* zu finden. Stadessen ist es möglich über Reinforcement Learning eine optimale Policy (π_*) zu approximieren. Um sich aber einer optimalen Policy überhaupt annähren zu können, muss eine **parametrisierte Funktionsapproximation** erfolgen. Das Paper nutzt hierfür einen sogenannten **Actor-Critic-Ansatz**. Dabei wird das komplexe Optimierungsproblem auf zwei neuronale Netze aufgeteilt, die gegenseitig voneinander lernen. Im folgendes wird dieser genauer erläutert.

2.3. Actor-Critic-Modell

Die analytische Lösung eines MDPs basiert, wie in [Abbildung 1] gezeigt, auf der **Bellman-Gleichung**. Diese besagt, dass der Wert eines Zustands genau dem Erwartungswert aus dem Reward und dem Wert des Folgezustands entspricht: $Q_*(s,a) = E[R(s,a) + \gamma \max_{a'} Q^*(s',a')]$. Im klassischen Fall wird diese Gleichung iterativ gelöst, bis die Werte konvergieren. Für das Montageproblem im Paper ist dies aufgrund der hochdimensionalen Bilddaten und Systemdynamik nicht möglich. Daher müssen wir die analytische Funktion Q^* durch ein neuronales Netz Q_φ approximieren. Ein künstliches neuronales Netz (KNN) ist ein Modell des maschinellen Lernens, das in seiner Funktionsweise grob dem menschlichen Gehirn nachempfunden ist. Es dient dazu, Muster in Daten zu erkennen und Entscheidungen zu treffen [Quelle IBM]. Der einfachhaltshalber nehmen wir folgendes an: Ein neuronales Netz Q_φ lernt eine Funktion $f(X)$ durch Zuordnung eines Eingabevektors $X = (x_1, x_2, x_3, ...)$ um eine Reaktion Y vorherzusagen. Dabei besteht ein neural Network aus unterschiedlichen Schichten, einem Input Layer für X, mehrschichtigen versteckten Layer, das hauptsächlich für das lernen von $f(X)$ zuständig ist und einem Output Layer, indem Y ausgegeben wird [Quelle IBM].

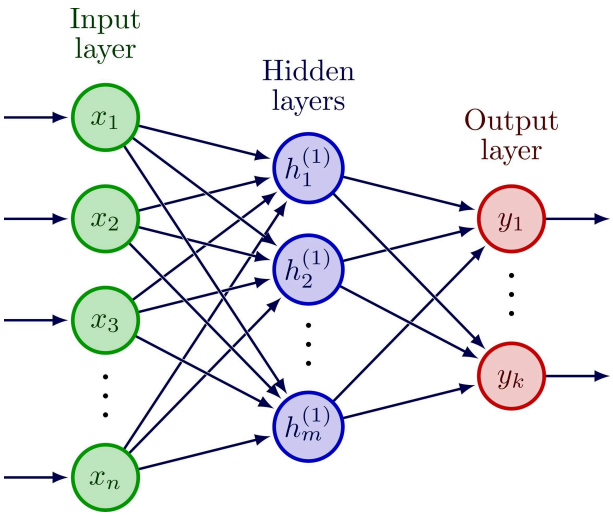

Abbildung 2: Neural Network - Shuttershock

Die Autoren des Papers nutzen zwei Neuronale Netzwerke, eines für den Actor und eines für den Critic in ihrer Umsetzung zur Approximation der Policy (π_*). Im folgenden wird tiefer auf die einzelnen Netzwerke eingegangen und ihre Wechselwirkung aufeinander analysiert.

2.3.1. Critic

Der Critic bewertet eine derzeitige Einschätzung des Netzwerks und vergleicht diese mit der tatsächlichen Situation und Zukunft. Dafür wurde folgende Loss-Funktion $\mathcal{L}_Q$ aufgestellt:

$$\mathcal{L}_Q(\varphi) = E_{s,a,s'}\left[\left(Q_\varphi(s,a) - \left(R(s,a) + \gamma E_{a'\sim\pi_\theta}\left[Q_{\overline{\varphi}}(s',a')\right]\right)\right)^2\right]$$

Die Funktion Q-Loss bestimmt die Fehlerquote der Parameter φ im neuronalen Netzwerk Q_φ, indem es anhand einer Stichprobe (einem Batch aus dem Replay Buffer) den **mittleren quadratischen Fehler** (Mean Squared Error) zwischen zwei Werten berechnet. Einmal dem Wert der eigenen Vorhersage $Q_\varphi(s,a)$, also eine Einschätzung des Netzwerks über die Handlung a in Zustand s und zum anderen dem **Bellman-Zielwert** (Target), der sich zusammen aus dem tatsächlich erhaltenen Reward $R(s,a)$ und der diskontierten Prognose des **Target-Networks** $Q_{\overline{\varphi}}$ für den Folgezustand ergibt. Wichtig zu beachten ist, dass es sich beim Target-Network um ein anderes Network handelt als Q_φ. Das liegt daran, dass eine sofortige Aktualisierung der Werte zum gleichzeitigen Veränderung des Netzwerkes und Zieles führen würde. Deshalb wird eine Kopie $Q_{\overline{\varphi}}$ erstellt, wodurch das Training stabilisiert wird. Dadurch wird verhindert, dass das Ziel („Moving Target“) während des Updates zu stark schwankt, indem die Parameter $\overline{\varphi}$ nicht direkt optimiert werden, sondern den Hauptparametern φ folgen und über einen gleitenden Durchschnitt (Soft Update) aktualisert werden [Quelle].

Der Critic dient als Leiter für den Actor, der die tatsächlichen Bewegungen ausführt bzw. die Policy (φ_θ) aktualisert, auf dem die Bewegungen basieren.

2.3.2. Actor

Der Actor steuert den Roboter, da er die tatsächliche Policy (φ_θ) definiert, die das Montageproblem löst. Dafür wurde folgende Loss-Funktion $\mathcal{L}_\pi$ aufgestellt:

$$\mathcal{L}_\pi(\theta) = -E_s\left[E_{a\sim\pi_\theta(\theta)}\left[Q_\varphi(s,a)\right] + \tau\Phi(\pi_\theta(.\mid s))\right]$$

Die Funktion Policy-Loss setzt sich aus zwei Zielen zusammen: Einmal Gierig (Exploitation) zu sein und andereseits Neugierig (Exploration). Das $(-)$ zu beginn der Funktion wandelt das Maximierungsproblem in ein Minimierungsproblem um, da Computer besser darin sind, Fehler zu minimieren. Denn

die Maximierung vom Reward, also dem suchen eines globalen Optimus einer Funktion $f(x)$ ist für uns gleichbedeutend wie das suchen des globalen Minimums $-f(x)$, nur einfacher für Computer umzusetzen. Der Teil, der die Gier des Actors steuert, ist in diesem Teil enthalten: $E_{a\sim\pi_\theta(\theta)}\left[Q_\varphi(s,a)\right]$. Das $a \sim \pi_\theta(\theta)$ hat dabei eine relativ wichtige Bedeutung. Es ist nicht mathematisch nicht möglich, Rückpropagierung (Backpropagation) in einem neuronalen Netz zu betreiben, wenn Stochastik zugrundeliegt, denn wir können aus einem Sample a keine Rückschlüsse zur Zufallsverteilung ziehen und plausible Anpassungen am Neuronalen Netz vornehmen. Die Autoren bedienen sich hier dem Trick der Reparametrisierung (Reparameterization), indem grob gesagt der Zufall in ein Standard-Rauschen ϵ ausgelagert, wodurch der stochatischen Sample, differenzierbar wird [Quelle]. Durch diesen Trick kann über den Critic Q_φ der Actor lernen, sich anzupassen. Der zweite Teil der Funktion $\tau\Phi(\pi_\theta(. \mid s))$ ist zuständig für die Exploration. Damit wird vorgebeut, dass sich der Actor nicht zu früh in einer approximierten Lösung festsetzt, sondern nach anderen, eventuell besseren sucht. Der Hyperparameter τ (Temperatur) steuert dabei die Balance: Ein hohes τ fördert Exploration, während ein niedriges τ die Policy stärker auf die Nutzung des besten bekannten Weges (Exploitation) fokussiert. Die Entropie Φ gibt die Standardabweichung σ vor, also wie „Experimentierfreudig“ der Actor ist. Diese Exploration wenden wir auf unseren Zustand s an unter der Berücksichtigung aller möglichen Handlungen a (hier gekennzeichnet durch $(.|s)$, innerhalb der Policy (π_θ)).

Der Actor leitet den Roboter unter der Berücksichtigung des Critics und eigener „Neugier“.

3. Effizientes Online Reinforcement Learning mit offline Daten - RLPD

Deep Reinforcement Learning (RL) konnte in vielen Feldern bereits Erfolge verzeichnen wie in Atari [Quelle] oder Go [Quelle]. In diesen Beispielen werden hohe Erfolge durch Reinforcement Learning und viele Online Interaktionen erzielt, dass durch Simulationen gut umsetzbar ist. Leider sind Probleme, wie das Montageproblem von Liu & Wang, in der Realität oft deutlich komplexer, als in einer Simulation. Reward sind meist absthrahiert, während sie in der Realität schwer greifbar und hochdimensional sind. Die Autoren des Papers [Quelle] postulieren den Ansatz von **RLPD**. Dieser Unterscheidet sich von Deep RL und SAC + Offline Daten. Liu & Wang stützen sich stark mit ihrer Architektur auf den Ansatz aus dem Paper [Quelle] von Ball et al., wobei in RLPD drei Grundkonzepte den Ansatz prägen. Im folgenden werden wir die Motivation hinter diesen Erweiterungen anschauen und deren Umsetzung von Liu & Wang.

3.1. Hybrides Buffer-System und Replay Ratio

Beim klassischen Deep Reinforcement Learning kommt die Dauer des Lernprozesses vor allem aus der Anfangsphase, in denen der Algorithmus erst eine gewisse Richtung ermitteln muss, in der sich die optimale Policy befindet. Die Richtungfindung, kann aber minimiert werden, indem beim Deep RL im vorraus bereits suboptimale Policies oder menschliche Demonstrationen die Richtung vorgeben. Mit einher gehen jedoch Probleme mit diesen Demonstrationen. Während der RL-Algorithmus lernt, werden in kurzer Zeit bereits eine vielzahl von Druchläufen in den Replay-Buffer, den Speicher, in dem alle Epochen gespeichert werden, geladen und überwiegen schnell die Anzahl an selbstgelandener Daten, wodurch sie in der Gewichtung rausfallen. Die Autoren aus dem Paper [Quelle] erkannten das ebenfalls und entwarfen basierend auf Ross & Bagnell (2012) [Quelle] ein symmetrische Replay-Buffer Architeuktur. Dabei werden statt nur einem Replay-Buffer, zwei Buffer angelegt, wobei einer die Online Daten speichert und aufnimmt, die der RL-Algorithmus erstellt und einem Offline Buffer, der die eben genannten menschlichen Demonstrationen oder suboptimale Policies mit einer hohen Abdeckung. Das hat vor allem den Vorteil, dass unter der schnell groß werdenen Menge an RL-Abläufen, die vorgefertigten Richtungsgeber nicht untergehen. Damit aber auch beim Samplen die Gewichtung erhalten bleibt, wird aus beiden Buffern die gleiche Menge an Daten entnommen, genau genommen jeweils 50% aus

den Offline und 50% aus den Online Daten. Diese simple Designentscheidung erweist sich als besonders effektiv, wie in der Evaluation zu erkennen ist, wenn auch diese Designentscheidung alleine nicht ausreichend ist [Quelle].

Dieses Vorgehen des symmetrischen Samples wird auch von Liu & Wang verwendet, wie in Abbildung 3 zu erkennen [Bild idk].

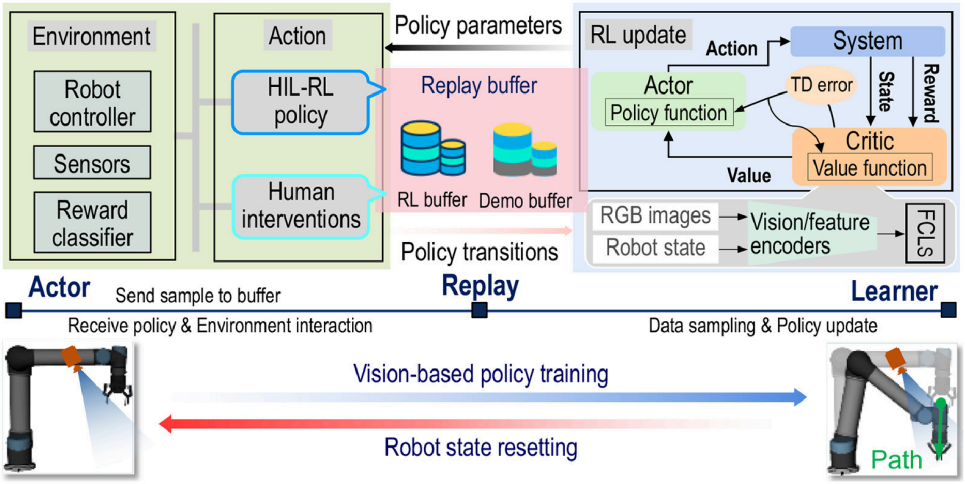

Abbildung 3: Arctor-Learner Architektur mit zwei Buffern

Näher wird auch erklärt, dass es sich um menschliche Demonstrationen handelt, die im vorraus erstellt wurden. Dabei wurden für jede Montageaufgabe (CPU-Kühlkörper, RAM, Lüfter) jeweils 30 erfolgreiche Trajektorien, also volle Bewegungsabläufe einer Montage, verwendet. Hierbei wird „Erfolgreich“ druch zwei Kritierien definiert. Zum einen darf sich das zu montierende Objekt nicht mehr als 0.1mm der Zielposition entfernt befinden. Zum anderen, sollte Ersteres erfüllt sein, muss der vorher tranierte Binary Classifier, ebenfalls mit einer Wahrscheinlichkeit von min. 97% die Montage als „1“, also erfolgreich bewerten. Hier wurde sich dazu entschieden, hoch qualitative menschliche Abläufe als Offline Daten zu verwenden. Die genaue Vor- und Nachteile varrieren stark nach zu optimierenden Problem, unter der Grundlage jedoch, dass Liu & Wang das Modell in der Realität tranieren, ist dieses vorgehen wenn auch aufwendig, durchaus sinnvoll und erfolgreich, wie der Evaluation der Versuchsabläufe zu entnehmen ist.

3.2. Layer Normalization

Eine fundamentale Schwäche von Deep Reinforcement Learning ist der Umgang mit unbekannten Daten, das sogenannte **Out-of-Distribution (OOD)** Problem. Bei Actor-Critic-Architekturen mit OOD Daten sind diese Daten meist nicht definiert während dem Lernprozess von RL. Dabei passiert es, dass der Critic die neuen eingehenden Daten stark „überschätzt“, da dieser die Daten mit den Offline Daten aus dem Demo-Buffer in Beziehung bringt [Quelle Schwarz]. Folge daraus in der Praxis sind Instabilitäten im Traning und Divergierung des Critics, während er versucht den immer größer werdenen Werten zu folgen. Die Lösung von Ball et al. nutzt Normalisierung der Werte innerhalb des Neuronalen Netzes, damit Werte innerhalb eines Zahlenbereiches (meist mit einem Mittelwert $\mu = 0$ und Standardabweichung $\sigma = 1$) bleiben. Dabei ist es möglich, dass der RL-Ansatz mit Layer Normalization trotzdem neues lernen kann, ohne zu stark von den Demo Daten limitiert zu sein [Quelle 2016]. Mathematisch wurde das wie folgt bewiesen und umgesetzt: $\|Q(s,a)\| \leq \|w\|$. Dies bedeutet, dass der vorhergesagte Q-Wert niemals größer sein kann als die Norm der Netzwerkgewichte w.

Die Umsetzung von Liu & Wang der Layer Normalization ist Anzunehmen, da ansonsten eine Umsetzung von RLPD nicht möglich wäre. Wie die Gleichung der Q-Loss Funktion zeigt, nutzen die Autoren die standardmäßige Soft-Bellman-Optimierung des SAC-Algorithmus:

$$\mathcal{L}_Q(\varphi) = E_{s,a,s'}\left[\left(Q_\varphi(s,a) - \left(R(s,a) + \gamma E_{a'\sim\pi_\theta}\left[Q_{\overline{\varphi}}(s',a')\right]\right)\right)^2\right]$$

Diese Funktion minimiert lediglich die Differenz zwischen der Vorhersage Q_φ und dem Zielwert. Sie beinhaltet jedoch keinen Mechanismus, der das Netzwerk vor der erwähnten **Overestimation** bei unbekannten Daten schützt. Die Stabilisierung muss daher strukturell innerhalb der Funktion $Q_\varphi(s,a)$ selbst erfolgen. Während Ball et al. [Quelle] hierfür explizit **Layer Normalization** vor der letzten Ausgabeschicht vorschreiben ($\|Q\| \leq \|w\|$), lassen Liu und Wang die genaue Innenarchitektur ihres Critics im Paper unerwähnt. Es ist möglich, dass die Autoren sich hier implizit auf die Beschaffenheit des **Reward Classifiers** verlassen, der den Reward hart auf das Intervall $[0,1]$ begrenzt. Unter der Annahme, dass sie die RLPD-Methodik vollständig adaptiert haben, ist der Einsatz dieser Normalisierungsschichten zwingend erforderlich, da der Critic sonst bei der Verarbeitung der Daten aus den zwei Buffern zur Divergenz neigen würde.

3.3. Sample Efficient RL

Die Designentscheidung eines zweiten Offline Buffers führt dazu, dass Sampling aufwendiger wird. Zur Minimierung des Aufwands wird die Geschwindigkeit der Lernprozesses modifiziert. Eine Möglichkeit dabei ist den UTP-Wert zu erhöhen, den Wert der vorgibt, wie viele Lernschritte (Critic passt Gewichte an) pro Arbeitsschritt (Actor führt Handlung aus) durchführt werden. Jedoch läuft man damit Gefahr, dass der RL-Algorithmus in **Überanpassung (Overfitting)** verfällt. Die Überanpassung beschreibt einen Zustand, indem ein Modell sich zu stark an einem lokalen Optimum angepasst hat und irrelevante Faktoren berücksichtigt [Quelle]. Zum besseren Verständnis wird als veranschaulichtes Beispiel oft der Unterschied zwischen „Verstehen“ und „Auswendig“ aufgewiesen. Wenn ein RL-Algorithmus zu exakt gerlernt hat, eine Aufgabe zu lösen, sorgt der Zustand der Überanpassung dafür, dass die spezifisch gelernte Aufgabe mit einer hohen Genaugikeit gelöst wird, jedoch bei leichten Änderungen bereits scheitert. Als Lösung dafür nennen Ball et. at. einige Möglichkeiten [Quelle], wobei sie sich für die Methoden mit **Random Ensemble Distillation** und **Random Shift Augmentations** entscheiden. Ersters beschreibt die Verwendung mehrer Critics, die in Wechselwirkung zueinander sich vor Divergenz schützen. Zweiteres wird genutzt, da Ball et. al. ebenfalls Bilder zum Traning verwendet. Dabei werden Die Bilder um wenige Pixel zufällig verschoben. Das ist besonders nützlich, da so der RL-Algorithmus lernt den RAM-Sockel wirklich als RAM-Sockel zu erkennen und nicht statisch auf Pixelpositionen versteift (Overfitting).

Präventionen für Overfitting im Paper von Liu & Wang geht vor allem hervor bei der Extraktion von Daten aus Bildern. Sie nutzen dafür ein vortraniertes ResNet-10 Netzwerk zur Bilderkennung, dass bereits schon auf Millionen von Daten traniert wurde und davon auszugehen ist, dass dieses bereits gegen Overfitting besteht [Quelle]. ResNet-10 ist eine vereinfachte Version des bekannten ResNet, dass Bilder auf relevante Daten extrahiert, mit denen viele RL-Algorithmen arbeiten können. Im Paper werden die Kamerabilder des Robters ins ResNet gesetzt, wodurch das Problem der geringen Datenmenge aus 1400 Bildern, vorgebeugt wird. Diese Daten werden dann in den Binary Classifier eingespeist, der dann Abläufe des Actors bewertet und mit in die Learner geliefert wird. Zusätzlich dazu wird **„Image Cropping“** verwendet, wodurch Bilder auf 128×128 in relevante Bereiche verkleinert werden. Jedoch geht keine direkte Nutzung von **Random Shift Augmentations**, **Random Ensemble Distillation** oder anderer genannter Methoden aus dem Paper von Ball et. al. hervor, die als essenzielle Designentscheidung postuliert wurden, um RLPD umzusetzen und Overfitting vorzubeugen. Besonders die **Random Shift Augmentation** wäre eine robuste Verbesserung und würde die Statik, die in den Bildern von Liu & Wang gegeben ist.

4. Diskussion und Evaluation

Die im Paper präsentierte Ergebnisse und Methodik zur Lösung eines Montageproblems, indem Feinmotorik, Schrauben und präzise Kraftverhältnisse benötigt werden, wurden mit Technologien und modernen Ansätzen der Informatik beeindruckend angegangen. Die wissenschaftliche Grundlage der Problematik, die zuerst Formal mit einem MDP formuliert und anschließend per RLPD approximiert wurde, ist technisch anspruchsvoll und schneidet stark mit Transfergebieten wie Maschinenbau, Elektortechnik, Physik und der Informatik. Der als Kern gewählte RLPD Ansatz, wurde aus Gründen der Dateneffizienz und Lerngeschwindigkeit gewählt. Die Umsetzung des RLPDs wurde in der Ausarbeitung einmal anhand des genutzten Papers von Bell et. al. für RLPD [Quelle] grundlegend erklärt und anschließen mit der Umsetzung von Liu & Wang verglichen. Dabei spielt die Reproduzierbarkeit von Arbeiten eine wichtige Rolle in der Wissenschaft. Es fällt auf, dass Liu & Wang auf eine detaillierte Spezifikation ihrer Critic-Architektur verzichten und diese Designentscheidung unerwähnt lassen. Da der Standard-SAC-Algorithmus ohne diese Modifikation in einem hybriden Setting zu Instabilitäten neigt, bleibt unklar, durch welchen Mechanismus die Autoren die hohe Erfolgsquote des Modells sicherstellen. Diese Intransparenz erschwert nicht nur die Reproduktion der Ergebnisse, sondern lässt auch offen, ob der Erfolg auf einer robusten Architektur oder auf der Verwendung von Human-in-the-Loop (HIL) zurückzuführen ist.

Die Verwendung von HIL in den Paper ist an vielen stellen sehr diskret und geht nicht in Tiefe darauf ein, wie diese menschlichen Eingriffe aussahen. In vielen Passagen werden die Eingriffe angesprochen und erwähnt, aber wie viele tatsächlich es sind und diese aussehen, wird offen gelassen, obwohl grade die Menge menschlicher Einwirkungen die Qualität der konstruierten RLPDs unterstreichen würden. Das die Lerngeschwindkeit deutlich erhöht wird, ist beim mehrfachen Eingreifen und korrigieren des Menschen selbstverständlich. Dementsprechend ist der Vergleich zwischen anderen RL-Ansätzen wie BC, SAC und DP unzureichend, wenn der RLPD + HIL verglichen wird. Eine Ausnahme scheint hier der HG-Dagger Vergleich zu sein, da hier die „gleiche Anzahl an Eingriffen wie RL“ [Quelle], vorgenommen wurden, wobei auch hier wieder die vage Äußerung von „Interventions“ problematisch bei der Validierung des Lernprozesses darstellen.

Zudem fehlt es an Variantion im Lernprozess und Test-Abläufen. Während des Lernprozesses werden identische Bauteile mehrmals unter gleicher Ausgangs- und Montageposition montiert. Bilder werden statisch ins System geladen, wodurch Overfitting eine Begründung für die hohen Erfolgsergebnisse sein könnte. In der Literatur zum visuellen Reinforcement Learning with Prior Data [Quelle] gilt Datenaugmentation als Standard, um zu verhindern, dass das neuronale Netz lediglich statische Pixel-Konstellationen auswendig lernt. Da der Versuchsaufbau in einer kontrollierten Laborumgebung stattfand, besteht der begründete Verdacht, dass die Policy π_θ weniger eine intelligente Reaktionsfähigkeit auf Systemdynamiken erlernt hat, sondern vielmehr eine Überanpassung an die fixe Kameraperspektive und Beleuchtung vorgenommen wurde. Ohne den Nachweis, dass das System auch unter veränderten visuellen Bedingungen (z.B. leichte Kameraverschiebungen, Lichtwechsel) funktioniert, ist die Aussagekraft der 98%-Quote für reale Industrieanwendungen als eingeschränkt zu bewerten.

Der Ansatz der Autoren stützt sich zudem auf einen Demo-Buffer, der mit 30 erfolgreichen, menschlichen Trajektorien gefüllt ist. Während dieses Vorgehen die Dateneffizienz des Lernprozesses zweifellos steigert, verschiebt es den Aufwand lediglich vom Training auf die Datenerhebung. Ein System, das auf hochpräzise menschliche Abläufe angewiesen ist, widerspricht teilweise dem Ziel der autonomen Robotik, die auch mit imperfekten Daten umgehen sollte. Verstärkt wird dieser Argument durch den „Human-in-the-Loop“ (HIL) Ansatz, der ebenfalls während der Traningsphase, hohe menschliche Aufmerksamkeit und Expertenwissen fordert, wobei hier die Häufigkeit der Eingriffe deutlichen Aufschluss liefern würde. Dies ist industriell nicht einfach Umsetzbar und kostspielig.

4.1. Fazit

Die vorliegende Arbeit untersuchte die Anwendung von hybriden Lernstrategien im Kontext der Präzisionsmontage. Die Modellierung des Montageproblems als Markov Decision Process (MDP) und die Nutzung von RLPD zeigen das Potenzial auf, starre Regelwerke durch lernfähige Algorithmen zu ersetzen. Für zukünftige Arbeiten an der Schnittstelle von Informatik und Maschinenbau bleibt die Herausforderung bestehen, solche Systeme nicht nur im Labor unter Idealbedingungen zu validieren, sondern ihre Robustheit durch Variation der Umweltparameter nachzuweisen.

Bibliografie

[1] I. I. F. o. Robotics, „International Federation of Robotics“. Zugegriffen: 7. Januar 2026. [Online]. Verfügbar unter: https://ifr.org/