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
2026-01-09

Inhaltsverzeichnis

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.

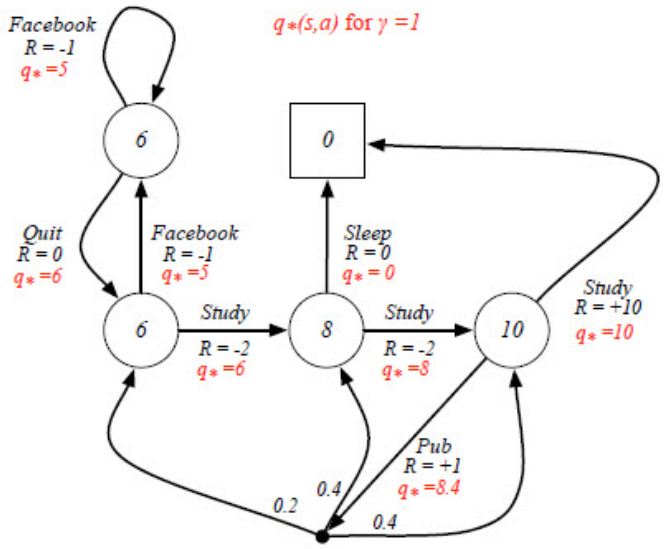

Abbildung 1: Optimal Action-Value Function for Student MDP - David Silver.

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.

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.

- $\boldsymbol{A}$ **(Action Space)**: Die Handlungsmöglichkeiten werden über die kartesische Koordinatenposition des Roboterarms und Griff-Zustand definiert.
- $\boldsymbol{P}$ **und** $\boldsymbol{p}$ **(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.
- $\boldsymbol{R}$ **(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.

2.2. 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].

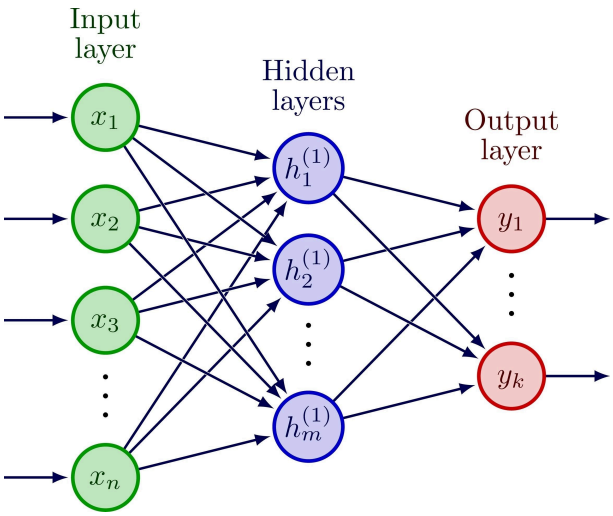

Abbildung 2: Neural Network - Shuttershock

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.

2.2.1. 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.

2.2.2. 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. Der Lernalgorithmus: RLPD mit „Human-in-the-Loop“

Wie bereits ausführlich erklärt, ist es nicht möglich einem Roboter, der derart feinmotorische und milimetergenaue Montageaufgaben bewältigen soll, statisch zu programmieren. Deshalb wurde im Paper Reinforcement Learning als Lösung genommen, in der ein Algorithmus selbständig lernt eine Strategie (Policy) zu entwickeln, die ihm vorgibt in einer Situation (Zustand) eine nahezu ideale Aktion vorzunehmen. Jedoch sind selbst klassische RL-Ansätze, wie auch der hier zugrundeliegende **SAC-Ansatz (Soft Actor Critc)** meist nicht ausreichend, um praxistauglich eingesetzt werden zu können. Grund dafür ist das das Erlernen einer Strategie oft Wochen von Laufzeit erfordert. Dateneffizienz ist bereits seit langer Zeit ein Problem beim Reinforcement Learning. Die Autoren bedienen sich dafür einer Erweiterung des SAC-Ansatzes, der sich rund um die Nutzung von Demonstrationen dreht. Der Algorithmus **RLPD (Reinforcement Learning with Prior Data)**, der in diesem Paper verwendet wurde, addressiert das Problem der Dateneffizienz, indem es bereits mit Vorkenntnissen der angepeilten Lösung arbeitet. Der größte Vorteil dabei ist, dass der RL-Algorithmus nicht von selbst die Richtung der Strategie erlernen muss, sondern bereits durch die Demos, in die Richtung der gewollten Strategie gedrückt wird. Die Struktur des Actor-Critic-Ansatzes wurde bereits ausführlich im MDP-Abschnitt behandelt. Im folgenden wollen wir uns mit den hier genutzten Besonderheiten des RLPD mit menschlichen Eingriffen beschäftigen. Zuerst wird Dafür wird zuerst der allgemeine RLPD Ansatz aus einem Paper [Quelle] betrachtet, auf den sich auch die Autoren stützen. Aufbauend darauf wird die Umsetzung betrachtet, Darunter fällt die Bereitstellung der Demos und wie diese in den RLPD eingebunden werden und anschließend auf die „Human-in-the-Loop“ Umsetzung eingenagen.

3.1. Effizientes Reinforcement Learning mit offline Daten [Quelle]

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 in der Realität oft deutlich komplexer, als in einer Simulation. Rewards 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 in der Form von vier Erweiterungen:

- **Hybdrid Replay-Buffer-System:** Beim RLPD werden statt einem Buffer, zwei genutzt. Einen für unsere Online Daten, die der RL Algorithmus selbst erstellt und ein zweiter Offline Buffer, indem menschliche Demos oder große Mengen suboptimaler Daten liegen. Diese Trennung hat den Vorteil, dass Daten nicht vermischt werden, wodurch die Offline Daten nicht in der Masse der entstehenden Online Daten verloren gehen.
- **Replay Ratio Sampling:** Das ziehen von Batches aus bereits vorhandenen Daten ist essenziell zum lernen von RL-Algorithmen. Da zwei Buffer vorhanden sind, besteht ein Batch zwingend aus 50% der Online Daten und 50% Offline Demos. Dadurch bleibt die Gewichtung ausgeglichen zwischen den beiden Datenmengen.
- **Layer Normalization:** Da die Daten aus unterschiedlichen Quellen kommen können, ist eine Normierung der Daten wichtig. Ansonsten kann es passieren, dass der Critic divergiert, indem er sich an immer höherwerdene Werte versucht anzupassen, die aufgrund von Überschätzung des Critcs der Daten ausgeht [Quelle].

Aufbauend auf diesem theoretischen Gerüst implementieren Liu und Wang spezifische Komponenten, um den Algorithmus für die Montageaufgabe nutzbar zu machen. Dies umfasst eine visuelle Belohnungsfunktion und die Einbindung menschlicher Korrekturen.

3.2. Reward Classifier

Der Reward oder Binary Classifier ist zuständig für die Bewertung von Montageabläufen, sowohl vor, als auch während dem Lernprozess. Der Classifier selbst ist ebenfalls ein Neuronales Netz, dass mit einer Wahrscheinlichkeit, eine binäre Zuweisung von Erfolg (1) und Misserfolg (0) bewertet.

Der Classifier selbst wird mit 100 erfolgreichen und 1300 fehlgeschlagen Datenpunkten traniert. Datenpunkte beschreibt hier die visuelle Übersetzung eines RGB-Bildes über ResNet-10 (Convolutional Neural Network) in räumliche Merkmale, in Form von Vektoren. Mit diesen Vektoren wird der Classifier über 100 Epochen und einem Adam Optimiser traniert, worauf hin dateneffizient und mit hoher Genaugikeit, der Classifier nun für das eigentliche RL-Policy Traning verwendet werden konnte.

3.3. Replay-Buffer

3.4. Mensch als Korrektiv

4. Evaluation

Ehrlich gesagt kauf ich das Paper nicht ganz ab. Aber mal schauen.

Bibliografie

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