# Simulationsbasiertes Deep Reinforcement Learning für Modulare Produktionssysteme

Niclas Feldkamp[1*], Sören Bergmann[1], Steffen Straßburger[1]

[1]Informationstechnik in Produktion und Logistik, TU Ilmenau, Max-Plank-Ring 12, 98693 Ilmenau; *niclas.feldkamp@tu-ilmenau.de

**Abstract.** Modulare Produktionssysteme sollen die traditionelle Fließbandproduktion in der Automobilindustrie ablösen. Die Idee dabei ist, dass sich hochgradig individualisierte Produkte dynamisch und autonom durch ein System flexibler Arbeitsstationen bewegen können. Dieser Ansatz stellt hohe Anforderungen an die Planung und Organisation solcher Systeme. Da jedes Produkt seinen Weg durch das System frei und individuell bestimmen kann, kann die Implementierung von Regeln und Heuristiken, die die Flexibilität des Systems zur Leistungssteigerung ausnutzen, in diesem dynamischen Umfeld schwierig sein. Transportaufgaben werden in der Regel von fahrerlosen Transportsystemen (FTS) ausgeführt. Daher bietet die Integration von KI-basierten Steuerungslogiken eine vielversprechende Alternative zu manuell implementierten Entscheidungsregeln für den Betrieb der FTS. In diesem Beitrag wird ein Ansatz für den Einsatz von Reinforcement Learning (RL) in Kombination mit Simulation vorgestellt, um FTS in modularen Produktionssystemen zu steuern. Darüber hinaus werden Untersuchungen zu dessen Flexibilität und Skalierbarkeit durchgeführt.

## Einleitung

Insbesondere in der Automobilindustrie stellen modulare Produktionssysteme den nächsten Evolutionsschritt in der Produktionsorganisation dar, welcher die klassische Linienfertigung vierlorts abzulösen versucht. Heutzutage wird die Bedeutung der Flexibilität immer wichtiger, da die Anforderungen an Qualität, Vielfalt der Produktpalette und Individualisierbarkeit steigen [6]. Viele Automobilhersteller haben daher Pilotprojekte gestartet, um die Modernisierung des traditionellen Fließbandkonzepts hin zu einem flexibleren, werkstattähnlichen Produktionsschema zu voranzutreiben. Dies lässt sich zusammenfassen unter dem Begriff des modularen Produktionssystems [15–17]. In der Praxis finden sich viele weitere Begriffe, die alle mehr oder weniger den gleichen Ansatz beschreiben, darunter "modulare Montage" [3], "Flexi-Line" [22], "Full-Flex-Plant" [5] oder "Factory 56" [4]. Das Ziel dieser Ansätze ist es, nahezu jeden Fahrzeugtyp bzw. Variante im selben Werk zu produzieren, so dass auch bei der Einführung neuer Produkte kein langwieriger Um- oder Neubau der Produktion erforderlich ist.

Im besten Fall können neue Produkte sogar im laufendem Betrieb eingeführt werden, ohne dass die Produktion unterbrochen werden muss. Erreicht werden soll dies durch die Umsetzung einer selbstregulierenden Produktion im Sinne des Grundgedankens von Industrie 4.0. Hierbei werden die Produktionsaufträge von fahrerlosen Transportsystemen (FTS) durch ein System von matrixförmig angeordneten Stationen geleitet. Diese Stationen sind unterschiedlich ausgestattet und können daher jeweils verschiedenene Produktionsschritte anbieten. Produkte können dynamisch und individuell entscheiden, welchen Produktionsschritt sie als nächstes ausführen lassen wollen und welche Station diesen ausführen soll. Die gesamte Steuerungsstrategie wird nur durch technische Restriktionen des Produktionsprogramms hinsichtlich der Reihenfolge einiger Produktionsschritte limitiert. Zur Abbildung der Entscheidungsfindung der FTS ist die einfachste Methode die Implementierung von vordefinierten Heuristiken. Die Einbeziehung der sich dynamisch verändernden Umgebung und des Systemzustands in solche heuristischen Regeln kann in einem komplexen System recht schwierig sein und führt nicht in jedem möglichen Szenario zur besten Lösung. Daher ist die Verwendung einer auf künstlicher Intelligenz (KI) basierenden Steuerungslogik, die an einer großen Anzahl von Beispielen trainiert wird und in Echtzeit entscheiden kann, ein vielversprechender Ansatz. Auf der anderen Seite ist die ereignisdiskrete Simulation eine etablierte Methodik für die Modellierung, Planung und Steuerung von Produktionssystemen, insbesondere in der Automobilindustrie. Daher liegt es nahe, eine KI-basierte Steuerungslogik mit Hilfe eines geeigneten Simulationsmodells zu trainieren. In einem vorherigen Beitrag wurde hierzu ein Ansatz vorgestellt, welcher Reinforcement Learning in Verbindung mit einem Simulationsmodell zum Erlenen einer Steuerungslogik für die FTS nutzt [9]. Dieser Ansatz wird in diesem Beitrag weiter untersucht. Im Zentrum steht hierbei die Frage, wie flexibel und skalierbar die gelernte Strategie bei Änderungen im System

ist bzw. ob bei Änderungen des Ausgangssystems die Strategie neu angelernt werden muss.

# 1 Grundlagen und Stand der Forschung

## 1.1 Modulare Produktionssyteme

Der Begriff des modularen Produktionssystems ist ein Synonym für eine Reihe von Terminologien, die alle das gleiche Prinzip beschreiben. Bei diesem Ansatz ist die Flexibilisierung der Montage, z. B. in der Automobilindustrie, das Hauptziel. Das traditionelle Prinzip des festen Fließbandes wird zugunsten einer eher werkstatt-ähnlichen Produktion aufgelöst [15, 17]. Die eigentliche Montage der Produkte erfolgt an Montagestationen (oft auch als Fertigungsinseln bezeichnet), die sehr flexibel sind und verschiedene Aufgaben ausführen können, abhängig von ihrer Funktion, den Werkzeugen und dem zur Verfügung stehenden Personal. Jedes Produkt ist durch Aufgaben definiert, so dass die Produkte einen sehr hohen Grad an Individualisierung aufweisen können. Einzelne Produkte müssen also unterschiedliche Stationen durchlaufen und können daher individuelle Wege durch das System gehen. Die Montagestationen sind nicht durch ein festes Fördersystem verbunden. Das heißt, sowohl der Transport der Produkte als auch der Materialtransport wird durch FTS durchgeführt [21]. Hieraus ergibt sich ein hoher Grad an Flexibilität auf unterschiedlichen Ebenen. Die auszuführenden Montageschritte können sehr unterschiedlich sein und sind nicht an eine global einheitliche Systemzykluszeit gebunden, wie in Abbildung 1 dargestellt.

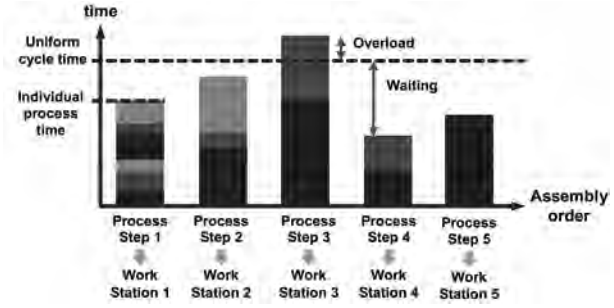


**Abbildung 1**: Individuelle Prozesszeiten gegenüber einheitlicher Zykluszeit [14, S.161].

Die verfügbaren Stationen bieten unterschiedliche Aufgaben an, die sie erfüllen können. Zudem können sich die Produkte infolge des Wegfalls eines festen Fördersystems frei bewegen, so dass die Montageschritte nicht unbedingt in einer festen Reihenfolge ausgeführt werden müssen, sofern keine technischen Einschränkungen gelten. Daraus ergibt sich ein hoher Freiheitsgrad für die Produkte, was die Flexibilität des Systems erhöht und die Kapazitätsauslastung der Stationen verbessert [14, 17]. Auf der anderen Seite wird dieser hohe Freiheitsgrad mit einer erhöhten Komplexität und Herausforderungen bei der Koordination, Kommunikation und Organisation zwischen den Systemkomponenten erkauft. Dies bezieht sich auf einen der Grundgedanken des Industrie 4.0-Konzepts, dass eine Vernetzung zwischen Maschinen, Geräten, Sensoren und Menschen proklamiert [19]. Einige Veröffentlichungen, welche sich bereits mit den simulationsbezogenen Aspekten und Modellierungstechniken modularer Produktionssysteme befasst haben [8, 26], kamen zu dem Schluss, dass ein geeignetes Simulationsmodell für die Entwicklung von Heuristiken zur Optimierung von Planungs- und Steuerungsproblemen in solchen Systemen unerlässlich ist und dies auch nach wie vor eine Herausforderung in Forschung und Praxis darstellt [10, 24]. Die Verwendung von KI-gestützten Steuerungslogiken trägt daher zur Lösung dieser Herausforderungen bei.

## 1.2 Deep Reinforcement Learning

Die Grundidee von Reinforcement Learning ist, dass ein Agent mit seiner Umgebung interagieren kann und auf der Grundlage des von ihm wahrgenommenen Zustands der Umgebung eine Aktion aus einer gegebenen Menge bestimmter Aktionen auswählt. Als Folge seiner Aktion erhält der Agent im Gegenzug eine Rückmeldung von seiner Umgebung. Diese Rückmeldung wird als Belohnung gewährt. Die Belohnung wird durch eine vorher ausgestaltete Formel berechnet und kann positiv oder negativ sein (Bestrafung statt Belohnung), abhängig vom gegebenen Zustand der Umwelt und der Aktion, die der Agent gewählt hat [28]. Diese Beziehung kann als ein sogenannter Markov-Entscheidungsprozess formalisiert werden, wie in Abbildung 2 dargestellt. Basierend auf der Wahrnehmung des Umgebungszustands $S_t$ aus allen möglichen Zuständen S zu diskreten Zeitpunkten t =1..n, wählt der Agent eine Aktion $A_t$ aus der Menge der möglichen Aktionen A. Dies führt zur Änderung des Zustands zu $S_{t+1}$. Der Agent erhält dann den neuen Zustand sowie die Belohnung $R_{t+1}$.

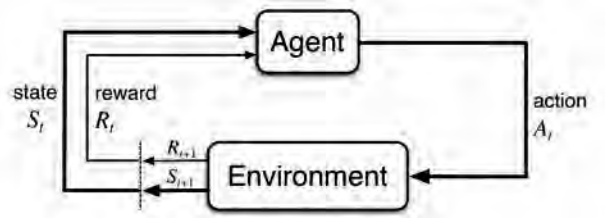


**Abbildung 2:** Agent-Umgebungsbeziehung in einem Markov-Entscheidungsprozess [28, S.48].

Der Agent strebt an, seine Handlungen so zu wählen, dass er die Summe seiner Belohnungen über den betrachteten Zeitraum maximieren kann und entwickelt so letztlich eine Strategie zur Lösung des Problems [28, 31]. Tatsächlich ist Reinforcement Learning aber ein Oberbegriff für eine Vielzahl unterschiedlicher Methoden und Algorithmen. Die meisten dieser Algorithmen lassen sich in zwei Kategorien einteilen: Modellbasierte und modellfreie Ansätze. Modellbasierte Ansätze zielen darauf ab, ein Modell der zugrundeliegenden Regeln und der Dynamik der jeweiligen Umgebung zu erlernen, um daraus eine optimale Strategie abzuleiten. Modellfreie Ansätze hingegen zielen darauf ab, die beste Aktion für jeden Zustand zu finden, um eine globale Strategie über alle Zustände zu erstellen oder einen gewünschten Zustand so schnell wie möglich zu erreichen [1, 18].

Für komplexe und dynamische Systeme, die mit stochastischer Unsicherheit konfrontiert sein können (wie modulare Produktionssysteme), sind modellbasierte Ansätze in der Regel nicht sehr geeignet, da der Zustandsraum zu groß ist, die interne Modellbildung des Algorithmus zu ungenau sein könnte und der Rechenaufwand zu hoch ist [12]. Einer der bekanntesten Vertreter der modellfreien Ansätze ist Q-Learning. Hier lernt der Agent für jedes Zustands-Aktions-Paar einen Wert (sog. Q-Wert), so dass er eine Wertfunktion $Q(s,a)$ approximieren kann, aus der er eine optimale Strategie ableiten kann [30]. Eine adäquate Erweiterung des traditionellen Q-Learnings ist Deep-Q-Learning (DQL), bei dem die Beziehung zwischen Zuständen und Aktionen durch ein künstliches neuronales Netz abgebildet wird. Dies ermöglicht das Prinzip des Q-Learning auch für große Zustands- und Aktionsräume [23, 27].

### 1.3 DQL in Verbindung mit ereignisdiskreten Simulationsmodellen

Gabel und Riedmiller konnten zeigen, dass DQL in der Lage ist, Standard-Optimierungsprobleme im Kontext des Produktionsmanagements zu lösen. In einem Job-Shop-Scheduling-Szenario mit parallelen Maschinen konnten sie klassische heuristische Regeln schlagen [11]. Über die Kombination von DQL und ereignisdiskreten Simulationsmodellen, der bevorzugten Simulationtechnik für die Modellierung von Produktionssystemen [8], gibt es nur wenig Forschung, so dass hier insgesamt noch viel Forschungsbedarf besteht. Gosavi weist in diesem Zusammenhang auf die komplexe Herausforderung der Synchronisation von Simulationsfortschritt und Lernalgorithmus hin [12]. Zur Lösung einfacher Optimierungsprobleme wurden bereits erfolgreich DQL-Algorithmen in Verbindung mit Simulationsmodell angewendet [12, 13]. Dies bezog sich allerdings auf das Finden einer globalen optimalen Lösung für ein festgelegtes Problem.

DQL bietet in Kombination mit Simulation aber auch vielversprechende Möglichkeiten für die dynamische Entscheidungsfindung in Echtzeit in einer sich dynamisch verändernden und unsicheren Umgebung. Zhang et al. setzten DQL und Simulation erfolgreich für die Bestimmung des nächsten auf einer Maschine zu verarbeitenden Auftrags [32, 33] sowie für Losgrößenbestimmung in Echtzeit ein [34]. In diesem Zusammenhang weisen Xie et al. darauf hin, dass es viele Herausforderungen bei der Kombination von Simulation und DQL gibt, die noch nicht abschließend und allgemeingültig gelöst sind, zum Beispiel die Quantifizierung von Zuständen und Aktionen, die Gestaltung der Belohnungsfunktion und die Sicherstellung der Konvergenz des Lernalgorithmus [32].

## 2 Reinforcement Learning und Simulation zur Steuerung von FTS in modularen Produktionssystemen

### 2.1 Konzeptionelle Überlegungen

Im Folgenden wird unterstellt, dass ein FTS ein Produkt/Auftrag an der Quelle aufnimmt und wieder freigibt, nachdem alle erforderlichen Montageaufgaben ausgeführt wurden, bevor es wieder ein neues Produkt aufnimmt. Das Produkt und das FTS bilden insofern eine Einheit. D. h., das Produkt überlässt die gesamte Entscheidungsfindung im Prozess dem FTS und seiner Steuerungslogik.

Der Entscheidungsfindungsprozess für FTS kann wiederum in mehrere Ebenen unterteilt werden, wie in Abbildung 3 dargestellt.


**Abbildung 3**: Hierarchie der Entscheidungsfindung für FTS in modularen Produktionssystemen [7, S.397].

Die erste Entscheidungsebene ist für die Auswahl der nächsten Aufgabe (welcher Montageschritt als nächstes auszuführen ist) und des Ziels (welche Station für die Ausführung der Aufgabe zu wählen ist) verantwortlich. Diese Entscheidung führt kaskadenartig zu untergeordneten Entscheidungen in Bezug auf Routing und Wegfindung, Kollisionserkennung und -vermeidung bis hin zu den eigentlichen Sensoren und Aktoren des FTS. Der im Folgenden vorgestellte Ansatz zielt auf die erste Ebene der Entscheidungsfindung ab und überlässt die unteren Ebenen anderen Systemen bzw. dem Simulator. Dies schließt jedoch nicht aus, dass diese Ebenen auch durch KI-gestützte Systeme gesteuert werden können. Das FTS muss folglich nach jeder abgeschlossenen Aufgabe oder nach Aufnahme eines neuen Auftrags an der Quelle eine Entscheidung treffen. Wenn eine Entscheidung ansteht, wird die Simulation angehalten und der aktuelle Systemzustand an den RL-Agenten gesendet. Der Agent trifft dann die Entscheidung über die nächste anzufahrende Station und sendet sie zurück an die Simulation, woraus dann das nächste Ziel für das FTS festgelegt und die Simulation fortgesetzt wird, bis eine neue Entscheidung erforderlich ist. Hierbei wird zur Entscheidungsfindung, wie oben erläutert, DQL für das Lernen des RL-Agenten verwendet. Das bedeutet, dass ein mehrschichtiges neuronales Netz verwendet wird, um die Wertfunktion zwischen Systemzuständen und Aktionen zu modellieren. Der Input des Netzes wird durch den aktuellen Systemzustand repräsentiert. Theoretisch könnte eine große Anzahl von möglichen Systemparametern übergeben werden. In unserem Ansatz wurde der Input reduziert auf die Aufgaben, die derzeit ausgeführt werden können, die verfügbaren Stationen (d. h., jene Station, die ausgestattet sind, um eine der verfügbaren Aufgaben auszuführen) und die aktuelle Länge der einzelnen Warteschlangen. Die Ausgabeschicht des Netzes stellt dann die Stationen dar, so dass das Ausgangsneuron mit dem höchsten Wert die nächste Station für die aktuelle Entscheidung bestimmt.

Um die Leistung des Lernalgorithmus weiter zu steigern, wird Double Deep Q-Learning (DDQN) eingesetzt. Dabei werden zwei identische neuronale Netze verwendet: Ein lokales Netz, das die nächste Aktion vorhersagt, und ein Zielnetz, das den Q-Wert für den Trainingsschritt berechnet. Diese Methode verhindert, dass der Agent suboptimale Strategien überschätzt und führt zu einer besseren Gesamtlernleistung [29].

Der Agent lernt im Laufe mehrerer Episoden. Eine Episode endet, wenn der Agent eine ungültige Entscheidung trifft oder das Zeitlimit der Simulation erreicht ist. Der eigentliche Trainingsprozess ist angelehnt an das bekannten Lernproblem, bei dem ein Agent versucht, den Ausgang eines Labyrinths zu finden, ohne in eine Falltür zu geraten [2]. Für jede richtige Entscheidung, die der Agent trifft, erhält er eine Belohnung von 0. Wenn ein Produkt vollständig zusammengebaut und freigegeben wird, erhält er eine Belohnung von 1 (der Ausgang des Labyrinths gilt als gefunden), was den Agenten dazu bringt, so schnell wie möglich zum Ausgang zu gelangen. Wenn eine ungültige Aktion gewählt wird (was bedeutet, dass die gewählte Station keine der aktuell verfügbaren Aufgaben ausführen kann), wird eine Belohnung von -1 gegeben, die Episode endet und die Simulation beginnt von vorne. Auf diese Weise wird sichergestellt, dass die korrekten Produktionsregeln und -beschränkungen erlernt werden. Darüber hinaus wurden der Belohnungsfunktion zusätzliche leistungsbezogene Komponenten hinzufügt, nachdem sich ein stabiler Lernprozess etabliert hatte. Eine Aktualisierung der Gewichte der neuronalen Netze und damit der Q-Funktion direkt nach jeder Entscheidung ist jedoch nicht ratsam. Zeitliche Korrelationen zwischen dem Lernen aus Erfahrungen und dem Sammeln von Erfahrungen können zu unerwünschten Verzerrungen führen, die den Agenten dazu neigen lassen, ältere Vorkommnisse bestimmter Zustands-/Aktionspaare zu vergessen. Aus diesem Grund wurde ein Mechanismus implementiert (sog. Memory-Replay-Buffer), der jede Erfahrung, die der Agent macht, in einem großen Puffer speichert. Gespeichert wird hierbei der jeweilige Systemstatus, die Aktionen des Agenten, den neuen Systemstatus nach der Aktion und die gewonnene Belohnung. Nach jeder Episode wird eine Zufallsstichprobe von Erinnerungen aus diesem Puffer entnommen, um das neuronale Netz zu trainieren [20]. Nach erfolgreichem Training kann dann das Echtsysteme an die Stelle des Simulationsmodells treten, sodass der RL-Agent die realen FTS steuern kann.

## 2.2 Implementierung

Für die Erprobung der Implementierung anhand einer

Fallstudie wurde ein Beispielmodell entwickelt, welches ein typisches Layout für modulare Produktionssysteme in der Automobilindustrie darstellt. Abbildung 4 zeigt einen Screenshot des Modells, welches mit Siemens Plant Simulation implementiert wurde. Das Modell enthält eine 50 m x 35 m große Fabrikhalle mit 9 Montagestationen. Die FTS können sich vor den Stationen in eine Warteschlange einreihen, wenn sie auf die Bearbeitung warten. Es gibt 4 verschiedene Produkttypen, die jeweils bis zu 20 einzelne Montageaufgaben mit unterschiedlichen Bearbeitungszeiten benötigen. Jede der Stationen bietet ein Portfolio von 2 bis 4 Aufgaben, welche sie ausführen kann. Für jedes Produkt gibt es spezifische Vorranggraphen für die Montage, so dass manche Aufgaben in einer bestimmten Reihenfolge ausgeführt werden müssen, während die Reihenfolge für andere Aufgaben frei gewählt werden kann. Die Art des Auftrags wird zufällig ausgewählt, bevor er einem freien FTS zugewiesen wird und von diesem an der Quelle abgeholt wird.

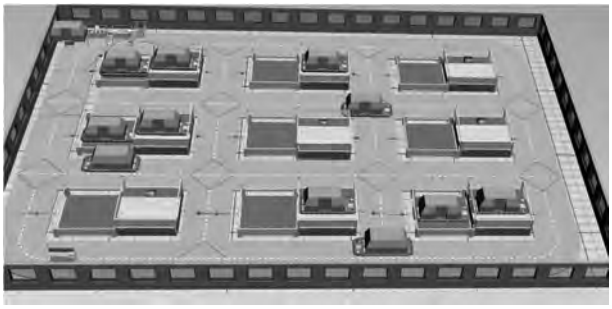

**Abbildung 4**: Screenshot des Plant Simulation Modells.

Die Implementierung des DQL-Algorithmus wurde in der Python-Bibliothek Tensorflow/Keras vorgenommen. Die Verbindung zwischen Plant Simulation und Python wurde über eine TCP/IP-basierte Schnittstelle realisiert. Dabei fungiert die Python-Seite als Server, das Plant-Simulation-Model als Client. Beim Starten des Servers werden die erforderlichen Elemente, einschließlich des neuronalen Netzes, initialisiert. Der Server wartet dann auf eingehende Anfragen des Simulators. Bei jeder Anfrage werden eine Reihe von Daten übertragen, darunter der Zustand des Systems sowie einige Metadaten wie Simulationszeit und Systemkennzahlen. Dieser Zustandsvektor wird an den RL-Algorithmus weitergeleitet, welcher dann eine Aktion auswählt, die dann wiederum als Antwort auf die TCP/IP-Anfrage an den Simulator zurückgesendet wird. Die Serverseite ist auch für die Berechnung von Belohnungen sowie für das Anhalten, Zurücksetzen und Starten der Simulation über in die TCP/IP-Antwort eingebettete Schlüsselwörter verantwortlich. Für die Feinabstimmung des Versuchsaufbaus wurden einige Vorversuche durchgeführt. Eine der größten Herausforderungen ist beispielsweise die Verwaltung des Memory-Buffers in Verbindung mit der Abfolge der Lernepisoden. Beim Lernen befinden sich RL-Agenten in einem Dilemma zwischen Exploration und Exploitation. Exploration bedeutet, dass der Agent neue Aktionen ausprobiert, während Exploitation bedeutet, dass der Agent an bereits gelernten Strategien festhält. Um Exploration zu gewährleisten und zu verhindern, dass der Agent in suboptimalen Strategien stecken bleibt, sollten Entscheidungen mit einer bestimmten Wahrscheinlichkeit $\varepsilon$ zufällig gewählt werden. Diese Wahrscheinlichkeit sollte anfangs hoch sein (in der Regel 90 %) und dann über die Dauer der Episoden abnehmen.

### 2.3 Ergebnisse

Um die Performanz der erlernten Strategie des Agenten bewerten und in Kontext setzen zu können, wurde diese zum einen mit zufälligem Entscheiden der nächsten Aktion sowie einer einfachen, heuristischen Regel verglichen. Bei ersterem wählt das FTS aus der Menge der möglichen Ziele ein zufälliges Ziel aus. Für die Heuristik wird die nächste gültige Station mit der kürzesten Warteschlange und der geografisch nächstgelegenen Station ausgewählt, wenn mehrere Stationen zum Zeitpunkt der Entscheidungsfindung die gleiche Warteschlangenlänge haben. Für den Vergleich wurden die benötigten Durchlaufzeiten gemessen, jeweils mit einem Stichprobenumfang von 100. Das Ergebnis dieses Vergleichs ist in Abbildung 5 in Form eines Box-Whisker-Plots dargestellt. Hierbei sind die Ergebnisse mit 12 FTS sowie mit 14 FTS dargestellt. Es hat sich gezeigt, dass bei weniger als 12 FTS die Auslastung der Stationen zu gering ist, als dass irgendeine gesonderte Strategie einen nennenswerten Effekt auf die Durchlaufzeit hätte. Auf der anderen Seite führen mehr als 14 FTS zu Stau- und Blockadesituationen, sodass das System mit mehr als 14 FTS nicht sinnvoll zu betreiben ist. Wie in Abbildung 5 zu erkennen, ist die zufällige Entscheidungsauswahl in beiden Szenarien am schlechtesten. Im 12-FTS-Szenario ist die RL-basierte Strategie im Durchschnitt gleichwertig mit der heuristischen Regel. Allerdings ist sie in einigen Situationen sogar in der Lage gewesen, deutlich kürzere Durchlaufzeiten zu produzieren. Zudem sind die Ausreißer nach oben etwas weniger stark vertreten als bei der heuristischen Regel. Dies lässt vermuten, dass die heuristische

Regel im Schnitt schon eine sehr gute Grundstrategie darstellt, es aber Situation gibt, in denen dies eben nicht die beste Strategie darstellt und dieses Potenzial entsprechend vom RL-Agenten ausgenutzt wird. Im 14-FTS-Szenario nähern sich alle drei Strategien in der Spannweite aller Beobachtungen an, da hier offenbar generell weniger Optimierungspotenzial, welches durch eine geschickte Strategie ausgenutzt weden kann, besteht.

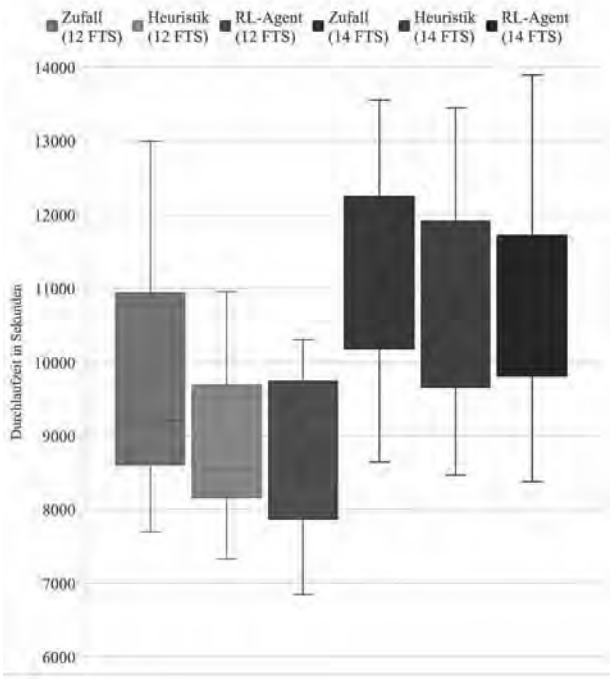


**Abbildung 5:** Vergleich von Durchlaufzeiten (in Sekunden) für unterschiedliche Steuerungsmechanismen und Szenarien.

Da sich der durchschnittliche Durchsatz zwischen den Szenarien nur unwesentlich unterscheidet (Abbildung 6), scheint der Einsatz von 14 FTS im gegebenen System nicht sinnvoll zu sein. Es existiert wenig Optimierungspotential, welches über das der heuritschen Regel hinausgeht. Interessant ist dennoch, dass im 14-FTS-Szenario Mittelwert und Median der gemessenen Durchlaufzeiten der RL-Agenten-Strategie deutlich unter denen der Heuristik liegen. Durch die zum Teil deutlich höheren Ausreißer liegt der durchschnittliche Durchsatz aber nur unwesentlich höher. Die RL-Agenten-Strategie scheint sich also bezogen auf die Entscheidungsfindung von der Heuristik zu unterscheiden, kann in Summe jedoch keine bessere Performanz erreichen.

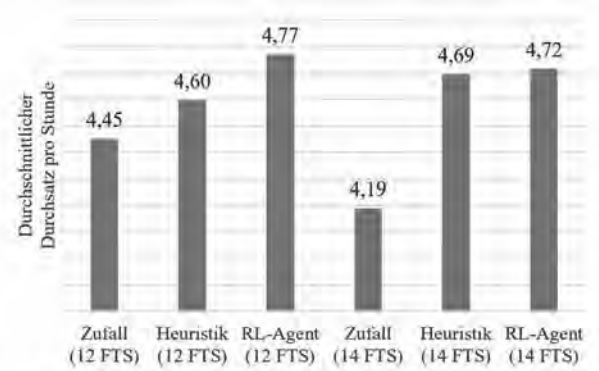


**Abbildung 6:** Durchschnittlicher Durchsatz von gerfertigen Produkten pro Stunde.

Im nächsten Schritt wurde untersucht, wie flexibel der Agent mit seiner erlernten Strategie auf veränderte Rahmenbedingungen, d. h. eine veränderte Anzahl von FTS im System, reagiert. Dies ist in Abbildung 7 dargestellt.

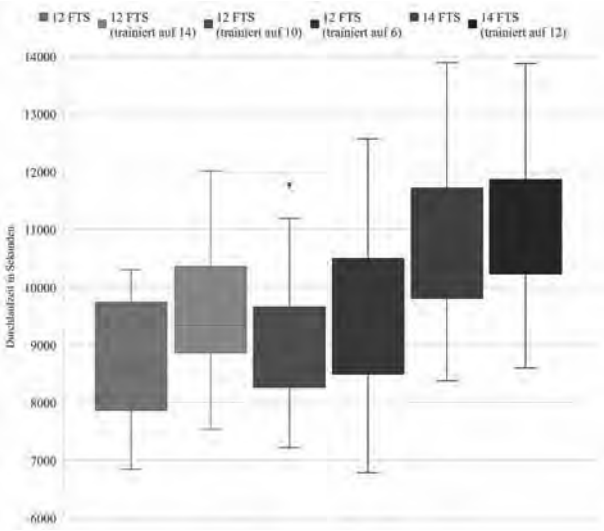


**Abbildung 7:** Vergleich der Durchlaufzeiten (in Sekunden) verschiedener RL-Agenten bei geänderte Rahmenbedingungen.

Hierbei wurde ein jeweils auf einer bestimmten Anzahl von FTS trainierter Agent in ein neues Umfeld mit veränderter FTS-Zahl eingesetzt, sowohl für das Szenario mit 12 FTS als auch für das Szenario mit 14 FTS. Hierbei wurde deutlich, dass der Agent in keiner der Varianten in der Lage war, mit der Performanz der für das eigentliche Szenario antrainierten Strategie mitzuhalten. Dies zeigt also, dass die jeweils antrainierten Strategien im Hintergrund keine allgemein gültige, optimierte Strategie gefunden haben, vielmehr ist diese vermutlich sehr stark auf das jeweilige Szenario spezialisiert. In der Konsequenz heißt dies, dass hier generell bei Änderungen am Szenario jedes Mal neu antrainiert werden sollte, um eine

bestmögliche Performanz zu erzielen.

## 3 Fazit und Ausblick

In diesem Beitrag wird gezeigt, dass die Entscheidungsfindung von FTS in modularen Produktionssystemen mittels Deep-Reinforcement-Learning in Kombination mit ereignisdiskreter Simulation verbessert werden kann. Die erlernte Strategie ist in der Lage, einfache heuristische Entscheidungsregeln zu übertreffen. Grundsätzlich ist die erlernte Strategie jedoch spezialisiert auf das zugrundeliegende System und reagiert empfindlich auf veränderte Rahmenbedingungen. Ein trainierter Agent sollte daher immer nur in demselben Umfeld eingesetzt werden, für welches er trainiert wurde. Grundsätzlich zeigt dieser Beitrag das Potenzial für KI-basierte Steuerung in modularen Produktionssystemen auf, es bietet sich allerdings auch noch viel Raum für weitere Forschung. Beispielsweise hängt der Lernerfolg des RL-Agenten stark von der Ausgestaltung der Belohnungsfunktion ab, welche wiederum vom angestrebten Optimierungsbedarf abhängt. In der hier gezeigten Fallstudie wurde sich auf das Ziel der Verbesserung der Durchlaufzeiten konzentriert. Mehrdimensionale Zielsysteme, wie Durchsatz, Durchlaufzeiten und Systemauslastung sind deutlich schwieriger darzustellen nicht nur bezogen auf die Komplexität der Belohnungsfunktion, sondern auch auf die Menge an Informationen, die der RL-Algorithmus verarbeiten muss. Ein weiteres Problem ist, dass der RL-Agent immer nur auf Grundlage seines individuellen Optimums trainiert und dabei eine mögliche globale Strategie, die zu besseren Ergebnissen führen könnte, vernachlässigt wird. Für weitere Forschung hierzu gibt es grundsätzlich zwei Möglichkeiten, dieses Problem anzugehen: Die erste Option wäre, einen Agenten zu trainieren, der nicht nur ein individuelles FTS, sondern das gesamte System auf einmal steuert. Dies würde theoretisch zu einer globalen Strategie führen, aber auch die Komplexität des Eingangsvektors des neuronalen Netzes stark erhöhen. Zudem wäre hier die Synchronisation der Entscheidungsanforderung und -findung zwischen Simulation und RL-Agent deutlich schwieriger darzustellen. Die andere Option wäre die Implementierung von Elementen klassischer Multiagentensysteme, so dass ein Agent mit anderen Agenten interagieren und verhandeln kann. Diese Ansätze werden aktuell als Multi-Agent Deep Reinforcement Learning (MADRL) zusammengefasst und wurden bereits in Anwendungen wie Flotten- und Verkehrsmanagement oder auch Energieverteilungsoptimierung eingesetzt [25]. Eine Nutzung solcher Algorithmen für die FTS-Entscheidungsfindung in modularen Produktionssystemen verspricht daher einen vielversprechenden Anknüpfungspunkt.

### References


[1] Achiam, J. and Morales, M. 2018. *OpenAI Spinning Up – Part 2: Kinds of RL Algorithms*. https://spinningup.openai.com/en/latest/etc/author.html. Accessed 14 April 2020.

[2] Atienza, R. 2020. *Advanced Deep Learning with TensorFlow 2 and Keras*. Packt Publishing, Birmingham, UK.

[3] Audi AG. 2019. *Die Modulare Montage - Fertigungsinseln statt Fließband*. https://www.audi-illustrated.com/de/smart-factory/Die-Modulare-Montage. Accessed 28 January 2019.

[4] Daimler AG. 2018. *Factory 56 – Mercedes-Benz Cars increases flexibility and efficiency in operations*. https://www.daimler.com/nnovation/production/factory-56.html.

[5] Daimler AG. 2018. *Groundbreaking for first Full-Flex Plant*. https://www.daimler.com/innovation/case/connectivity/full-flex-plant.html. Accessed 28 January 2019.

[6] ElMaraghy, H., Schuh, G., ElMaraghy, W., Piller, F., Schönsleben, P., Tseng, M., and Bernard, A. 2013. Product variety management. *CIRP Annals* 62, 2, 629–652.

[7] Feldkamp, N., Bergmann, S., and Straßburger, S. 2019. Modellierung und Simulation von modularen Produktionssystemen. In *Simulation in Produktion und Logistik 2019*, M. Putz and A. Schlegl, Eds. Verlag Wissenschaftliche Scripten, Auerbach, 391–401.

[8] Feldkamp, N., Bergmann, S., and Strassburger, S. 2019. Modelling and Simulation of Modular Production Systems. In *Simulation in Produktion und Logistik 2019*, M. Putz and A. Schlegel, Eds. Verlag Wissenschaftliche Scripten, Auerbach, 391–401.

[9] Feldkamp, N., Bergmann, S., and Strassburger, S. 2020. Simulation-Based Deep Reinforcement Learning for Modular Production Systems. In *Proceedings of the 2020 Winter Simulation Conference*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 1596–1607.

[10] Filz, M.-A., Herrmann, C., and Thiede, S. 2019. Simulation-Based Data Analysis to Support the Planning of Flexible Manufacturing Systems. In *Simulation in Produktion und Logistik 2019*, M. Putz and A. Schlegel, Eds. Verlag Wissenschaftliche Scripten, Auerbach, 413–422.

[11] Gabel, T. and Riedmiller, M. 2008. Adaptive Reactive Job-Shop Scheduling with Learning Agents. *International Journal of Information Technology and Intelligent Computing* 24, 4.

[12] Gosavi, A. 2015. *Simulation-Based Optimization. Parametric Optimization Techniques and Reinforcement*

*Learning*. Operations Research/Computer Science Interfaces Series 55. Springer, New York.

[13] Gosavi, A. 2015. Solving Markov Decision Processes via Simulation. In *Handbook of Simulation Optimization*, M. C. Fu, Ed. Springer New York, New York, NY, 341–379. DOI=10.1007/978-1-4939-1384-8_13.

[14] Greschke, P., Schönemann, M., Thiede, S., and Herrmann, C. 2014. Matrix Structures for High Volumes and Flexibility in Production Systems. *Procedia 5CIRP6* 17, 160–165.

[15] Grunert, D., Jung, S., Leipold, T., Göppert, A., Hüttemann, G., and Schmitt, R. 2019. Service-oriented Information Model for the Model-Driven Control of Dynamically Interconnected Assembly Systems. In *Tagungsband des 4. Kongresses Montage Handhabung Industrieroboter*. Springer Berlin Heidelberg, Berlin, Heidelberg, 23–33.

[16] Hüttemann, G., Buckhorst, A. F., and Schmitt, R. H. 2019. Modelling and Assessing Line-less Mobile Assembly Systems. *Procedia 5CIRP6* 81, 724–729.

[17] Hüttemann, G., Gaffry, C., and Schmitt, R. H. 2016. Adaptation of Reconfigurable Manufacturing Systems for Industrial Assembly – Review of Flexibility Paradigms, Concepts, and Outlook. *Procedia 5CIRP6* 52, 112–117.

[18] Kaelbling, L. P., Littman, M. L., and Moore, A. W. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4, 1, 237-285.

[19] Kagermann, H., Helbig, J., Hellinger, A., and Wahlster, W. 2013. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Securing the future of German manufacturing industry ; final report of the Industrie 4.0 working group*. Forschungsunion; Geschäftsstelle der Plattform Industrie 4.0, Berlin, Frankfurt/Main.

[20] Kalyanakrishnan, S. and Stone, P. 2007. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*. ACM Press, New York, New York, USA, 1. DOI=10.1145/1329125.1329241.

[21] Kern, W., Lämmermann, H., and Bauernhansl, T. 2017. An Integrated Logistics Concept for a Modular Assembly System. *Procedia Manufacturing* 11, 957–964.

[22] Mayer, B. 2018. *Ich bin ein Fan von Effizienz*. https://www.automobil-produktion.de/interviews-734/porsche-produktionsvorstand-reimold-wir-ziehen-alle-register-126.html?page=4. Accessed 28 January 2019.

[23] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540, 529–533.

[24] Mueller, D., Mieth, C., and Henke, M. 2018. Quantification of Sequencing Flexibility Based on Precedence Graphs for Autonomous Control Methods. In *Proceedings of the 4th International Conference on Industrial and Business Engineering - ICIBE' 18*. ACM Press, New York, New York, USA, 211–220. DOI=10.1145/3288155.3288162.

[25] Nguyen, T. T., Nguyen, N. D., and Nahavandi, S. 2020. Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE transactions on cybernetics*.

[26] Schönemann, M., Herrmann, C., Greschke, P., and Thiede, S. 2015. Simulation of matrix-structured manufacturing systems. *Journal of Manufacturing Systems* 37, 1, 104–112.

[27] Shrestha, A. and Mahmood, A. 2019. Review of Deep Learning Algorithms and Architectures. *IEEE Access* 7, 53040–53065.

[28] Sutton, R. S. and Barto, A. 2018. *Reinforcement Learning: An Introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, MA, London, England.

[29] van Hasselt, H., Guez, A., and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. AAAI Press, 2094–2100.

[30] Watkins, C. J. C. H. and Dayan, P. 1992. Q-learning. *Machine Learning* 8, 3-4, 280-292.

[31] Wiering, M. and van Otterlo, M. 2012. *Reinforcement Learning and Markov Decision Processes*. Adaptation, Learning, and Optimization 12. Springer, Heidelberg.

[32] Xie, S., Zhang, T., and Rose, O. 2019. Online Single Machine Scheduling Based on Simulation and Reinforcement Learning. In *Simulation in Produktion und Logistik 2019*, M. Putz and A. Schlegel, Eds. Verlag Wissenschaftliche Scripten, Auerbach, 59–68.

[33] Zhang, T., Xie, S., and Rose, O. 2017. Real-time Job Shop Scheduling Based on Simulation and Markov Decision Processes. In *Proceedings of the 2017 Winter Simulation Conference*. IEEE Inc, 3899–3907.

[34] Zhang, T., Xie, S., and Rose, O. 2018. Real-Time Batching in Job Shops based on Simulation and Reinforcement Learning. In *Proceedings of the 2018 Winter Simulation Conference*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 3331–3339.