Produktionssteuerung mit selbstlernenden Multiagentensystemen

Umsetzung anhand eines virtuellen Simulationsmodells

Inhalt

1. Motivation & Zielsetzung
2. Simulationsmodell & Steuerungsansätze
3. Ergebnisse
4. Kritik
5. Beantwortung der Forschungsfrage

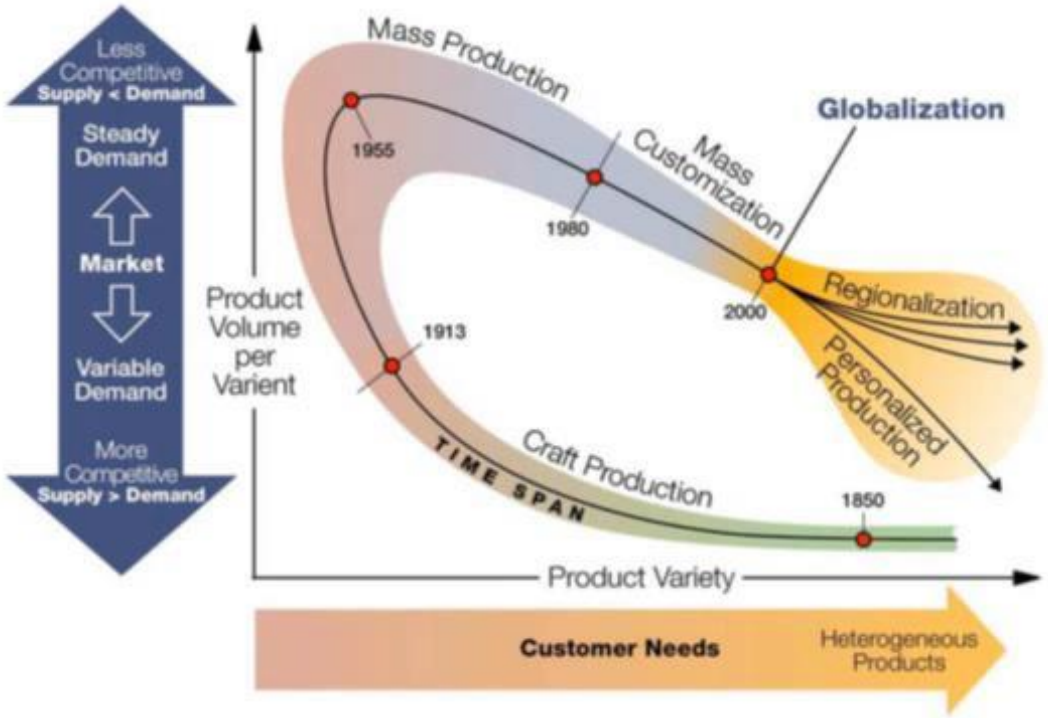
Motivation

Wandel der Anforderungen:

-> heterogenere & variablere Kundenbedürfnisse

-> Anpassung der unternehmensinternen Strukturen für mehr Flexibilität

Idee: Produktionssteuerung durch selbstlernende Multiagentensysteme



Forschungsfrage

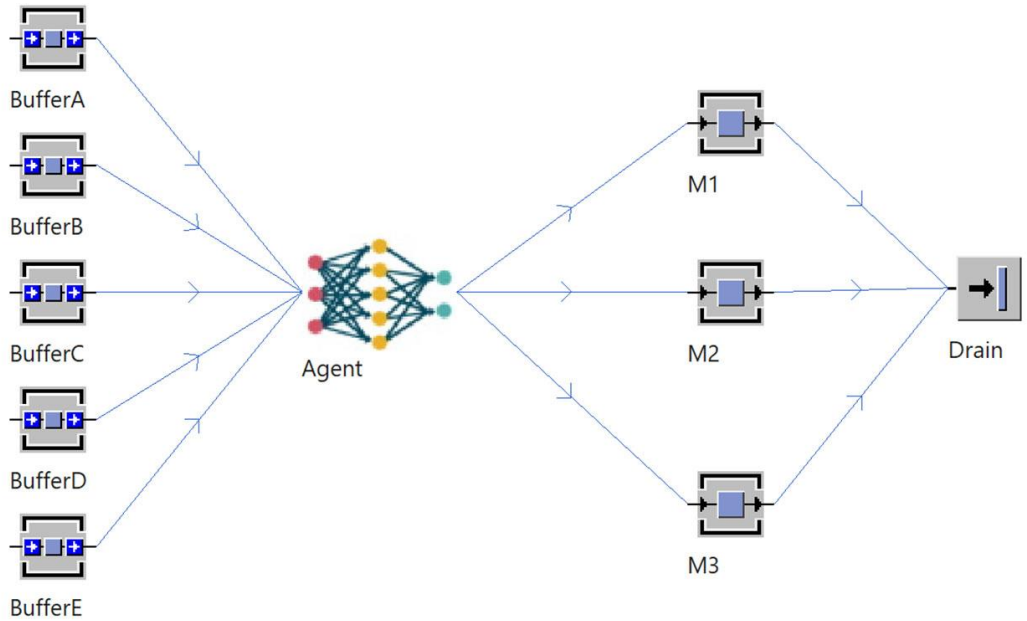
Wie leistungsfähig ist ein selbstlernendes Multiagentensystem in der Produktionssteuerung?

Beantwortung der Frage mittels:

- Modell eines Produktionssystems zur virtuellen Simulation
- Alternative Steuerungsansätzen zum Leistungsvergleich

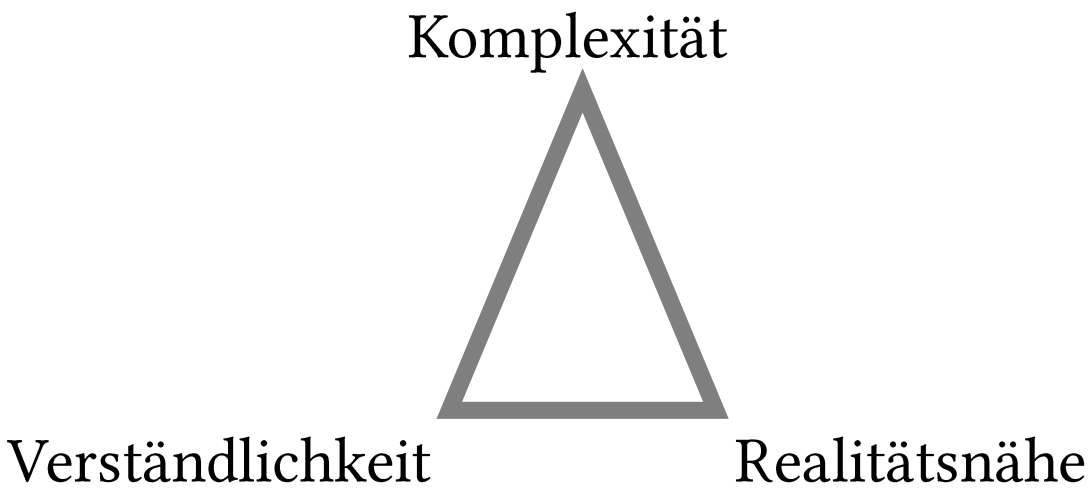
Vorgängerarbeit

- Einzelagent-System
- Aufgabe der Reihenfolgebildung
- Sortenreine Puffer
- Umrüstvorgänge notwendig
- Ziel: Minimierung der Gesamtbearbeitungszeit

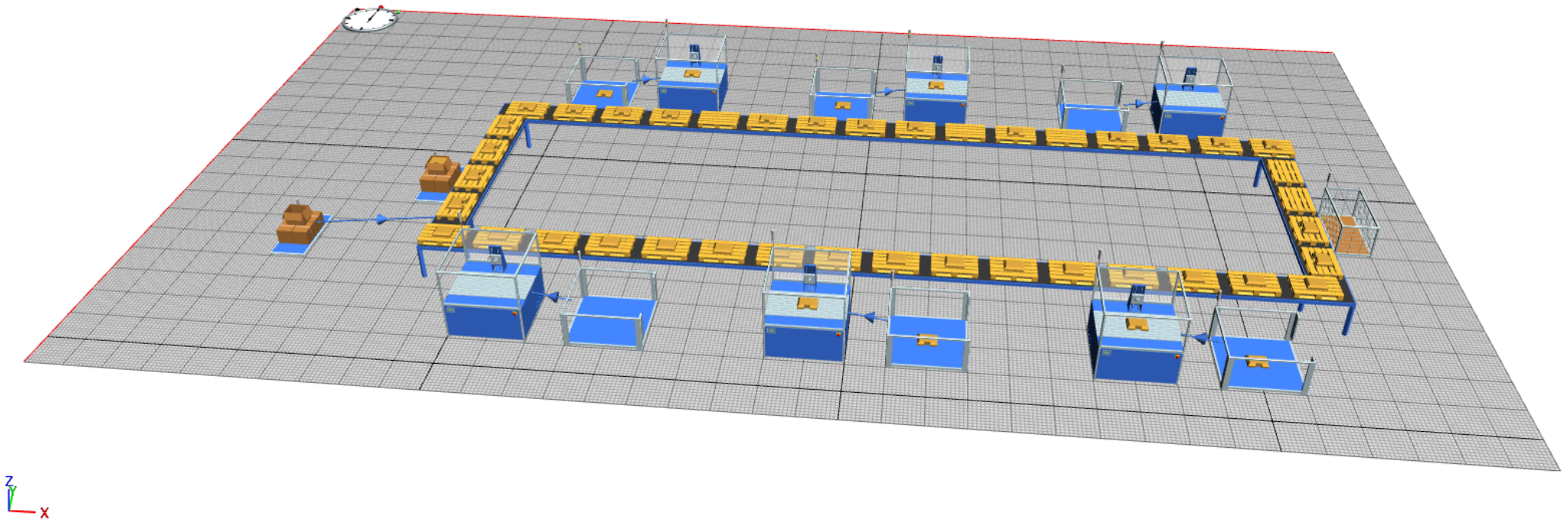


Anforderungen an das Simulationsmodell

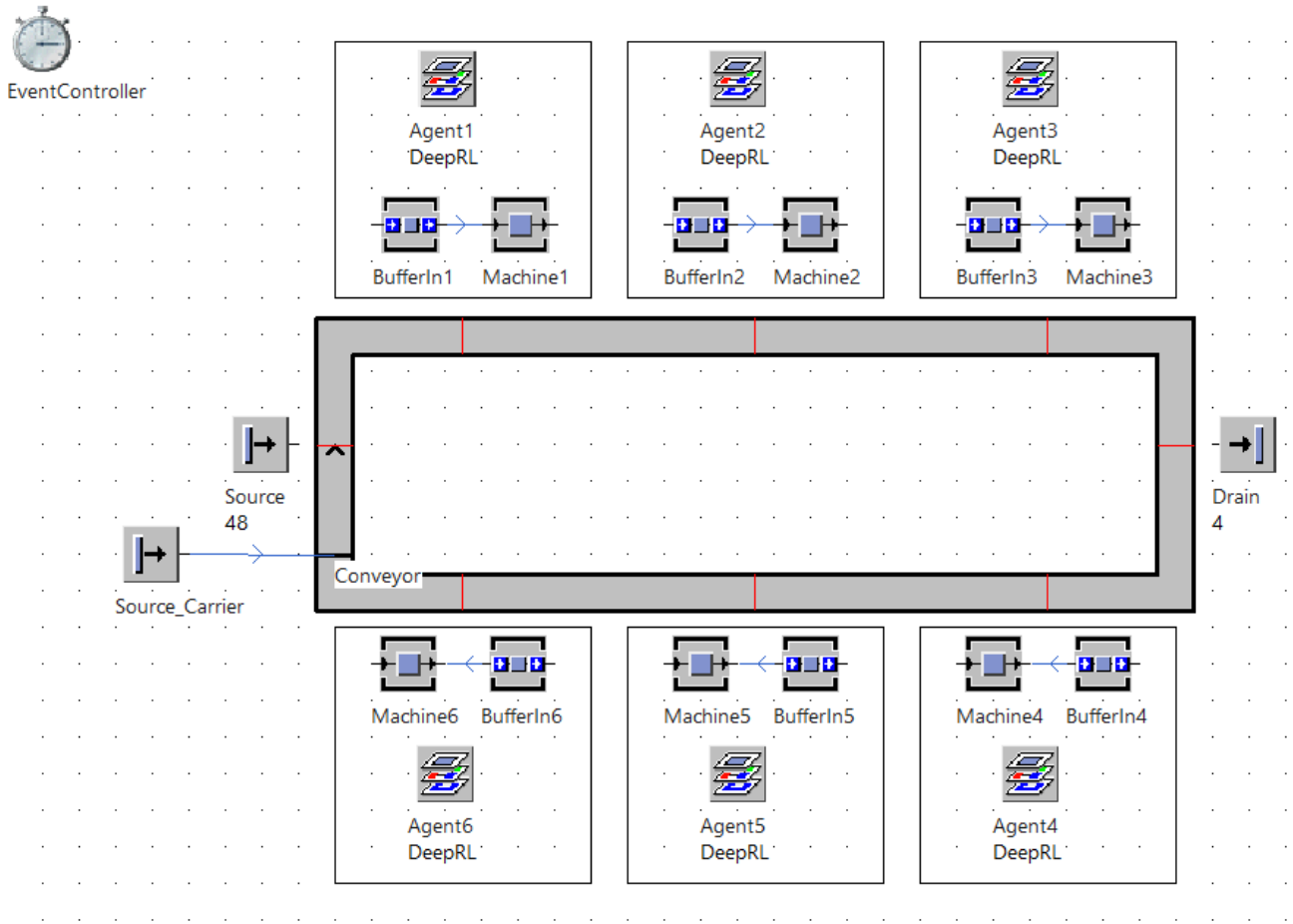
Spannungsverhältnis zwischen den Anforderungen:



Fiktives Produktionssystem



Gestaltung des Multiagentensystems



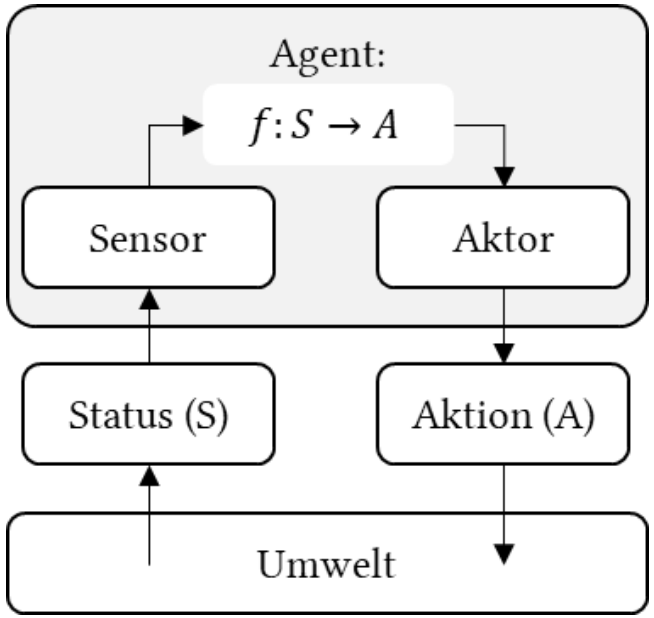
Verfügbare Steuerungsansätze

Statische Heuristiken:

1. nextValidAction
2. shortestRemainingTime

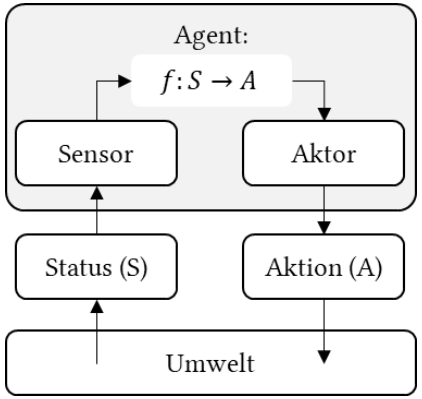
Selbstgelernte Heuristiken:

3. DeepRL



Aktionsraum

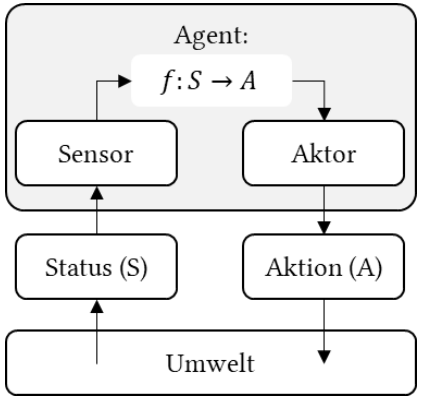
$$A = \begin{pmatrix} \textit{Transfer: Ladungsträger} - \textit{Eingangspuffer} \\ \textit{Transfer: Maschine} - \textit{Ladungsträger} \\ \textit{Kein Transfer} \end{pmatrix}$$



- Der Aktionsraum ist bei allen Steuerungsansätzen gleich
- Nicht alle Aktionen sind immer ausführbar (Statusabhängigkeit)

1. nextValidAction

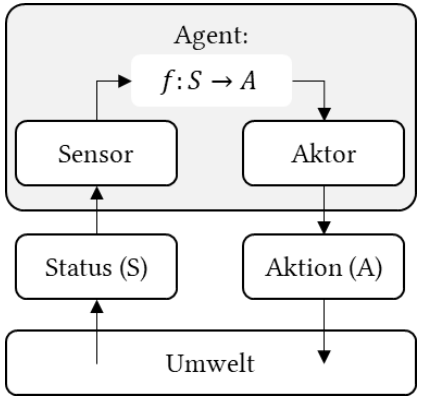
$$B(s_t) = \begin{pmatrix} Ladungsträger\ belegt? \\ Eingangspuffer\ belegt? \\ aktuelle\ Bearbeitung\ abgeschlossen? \\ Bearbeitung\ möglich? \end{pmatrix}$$



- Einfacher statischer Steuerungsansatz
- Zulässige Transfers werden direkt ausgeführt

2. shortestRemainingTime

$$B(s_t) = \begin{pmatrix} Ladungsträger\ belegt? \\ Eingangspuffer\ belegt? \\ aktuelle\ Bearbeitung\ abgeschlossen? \\ beste\ freie\ Wahl\ für\ priorisiertes\ Werkstück?\ ^{1} \end{pmatrix}$$

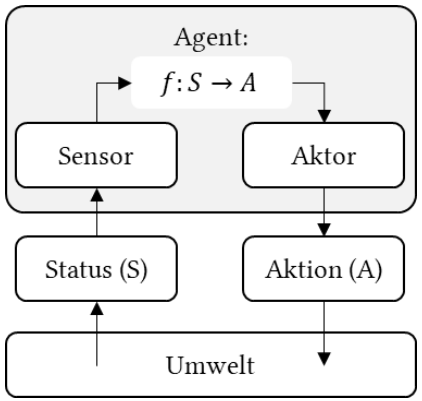


- leistungsfähigerer statischer Steuerungsansatz
- Priorisierung anhand der Restbearbeitungszeit

[1]allumfassendes Wissen über alle Werkstücke, Maschinenfähigkeiten und –belegung notwendig

3. DeepRL

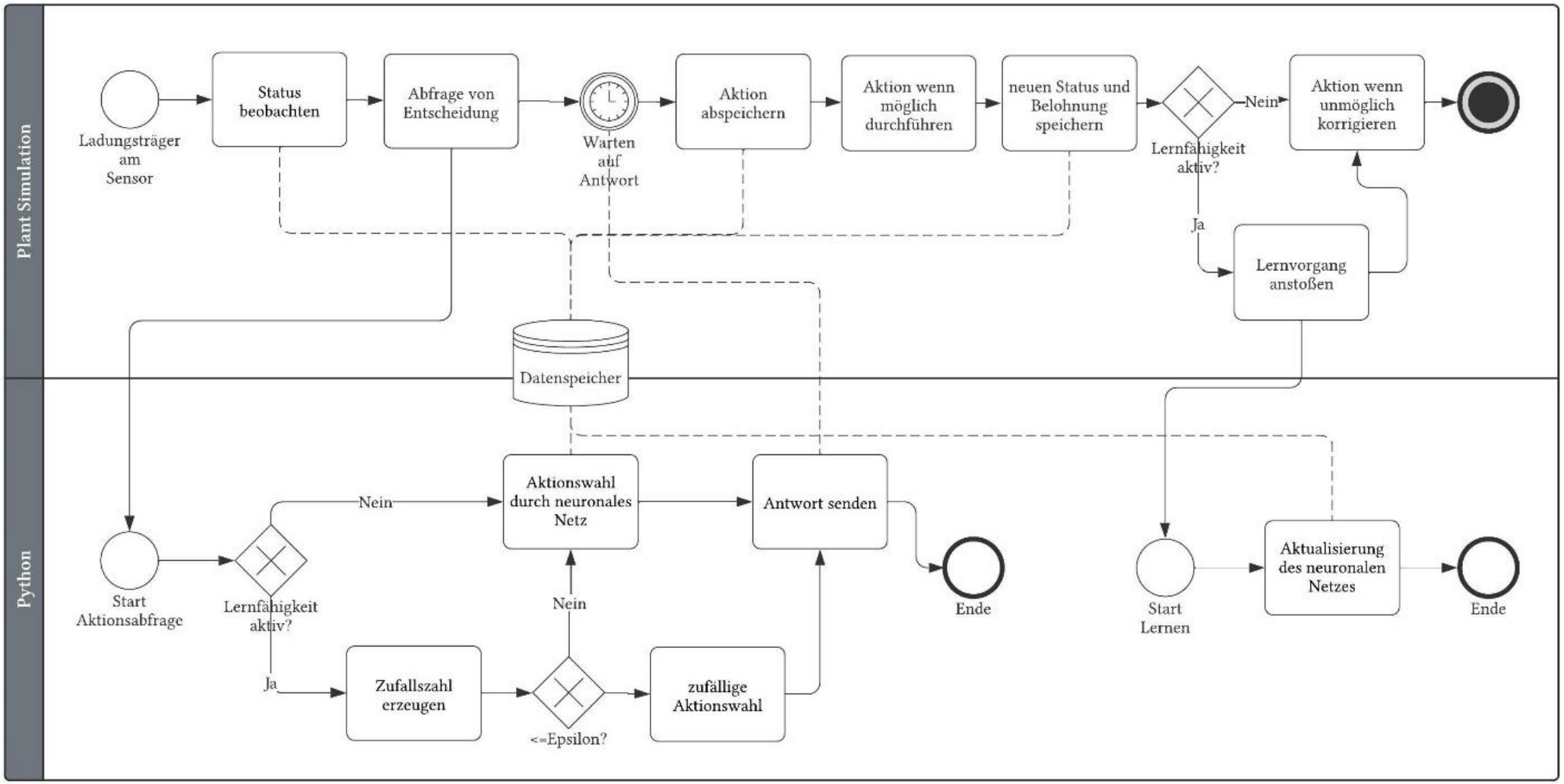
$$B(s_t) = \begin{pmatrix} Ladungsträger\ belegt? \\ Eingangspuffer\ belegt? \\ aktuelle\ Bearbeitung\ abgeschlossen? \\ Anzahl\ belegte\ Ladungsträger \\ Anzahl\ Werkstücke\ im\ System \\ Anzahl\ mögl.\ Bearbeitungsschritte \\ Anzahl\ möglicher\ Bearbeitungsschritte\ (n+1) \\ Anzahl\ verbleibende\ Bearbeitungsschritte \\ Anzahl\ mögl. Bearbeitungsschritte\ opt. Maschine^1 \end{pmatrix}$$



- Algorithmus passt Steuerungsheuristik dynamisch an
- Beobachtungen werden als Eingangsdaten für KNN verwendet

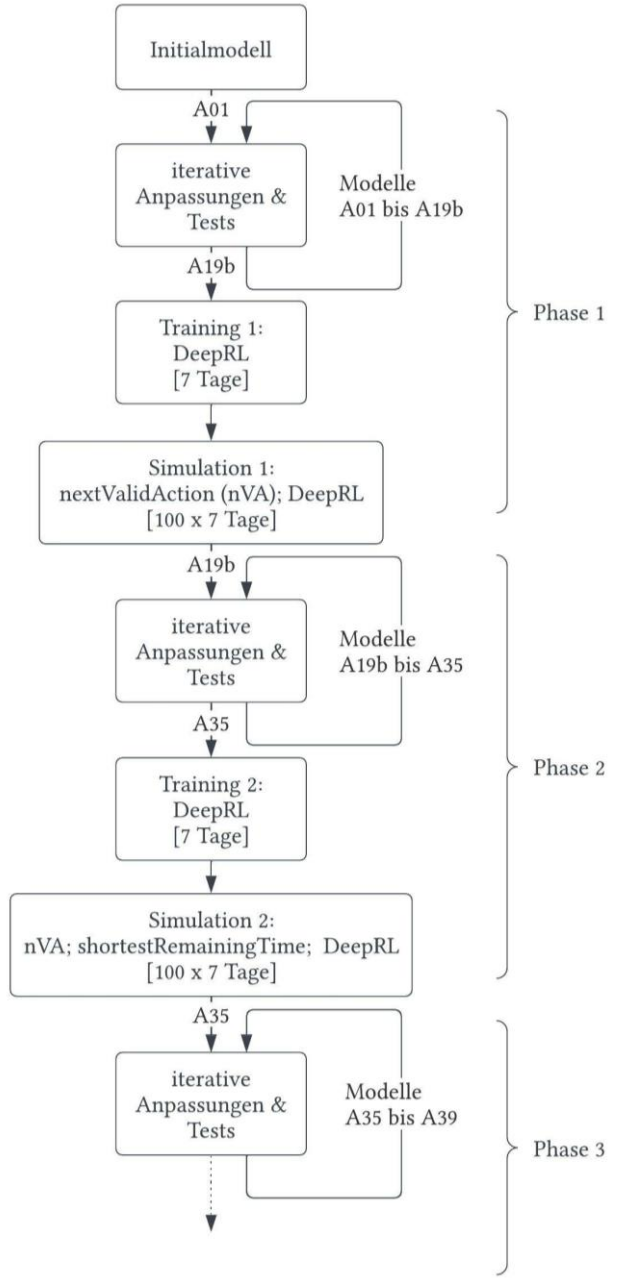
[1]allumfassendes Wissen über alle Maschinenfähigkeiten notwendig

3. DeepRL


Plant Simulation
Ladungsträger am Sensor
Status beobachten
Abfrage von Entscheidung
Warten auf Antwort
Aktion abspeichern
Aktion wenn möglich durchführen
neuen Status und Belohnung speichern
Lernfähigkeit aktiv?
Nein
Aktion wenn unmöglich korrigieren
Ja
Lernvorgang anstoßen
Datenspeicher
Python
Start Aktionsabfrage
Lernfähigkeit aktiv?
Nein
Aktionswahl durch neuronales Netz
Antwort senden
Ende
Ja
Zufallszahl erzeugen
<=Epsilon?
Nein
zufällige Aktionswahl
Start Lernen
Aktualisierung des neuronalen Netzes
Ende

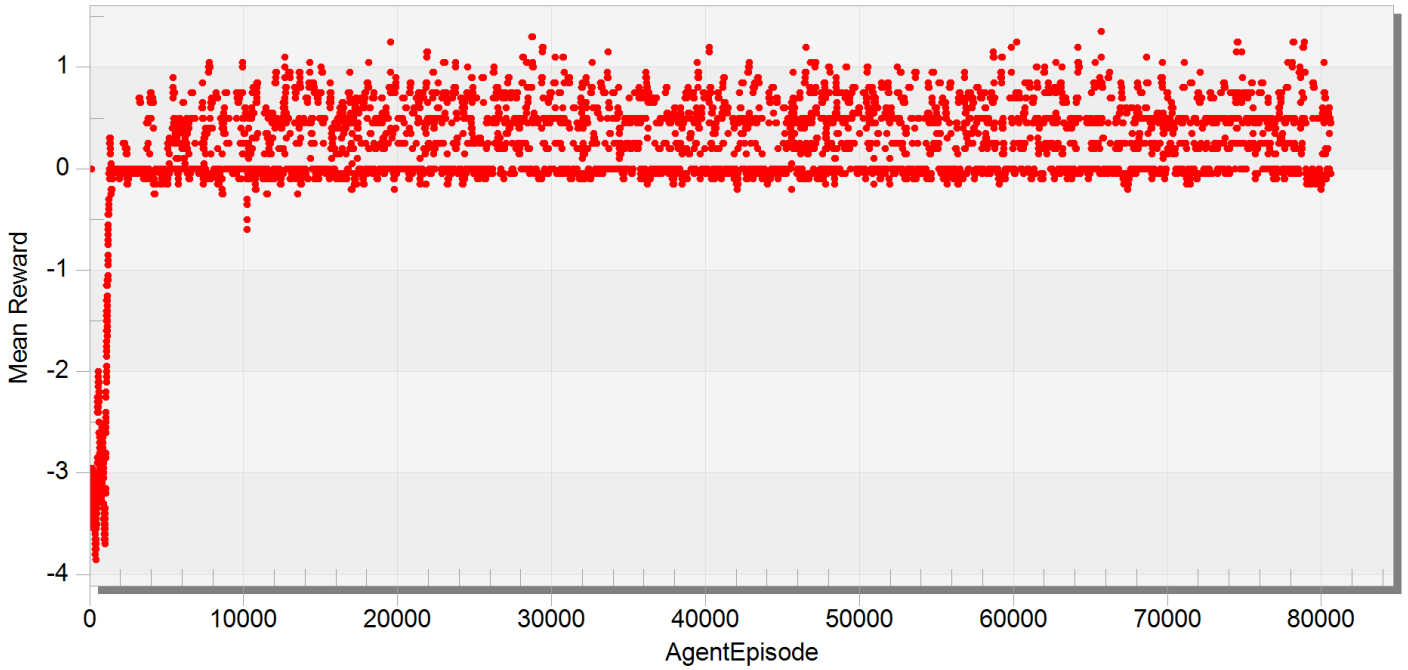
Experimentieren

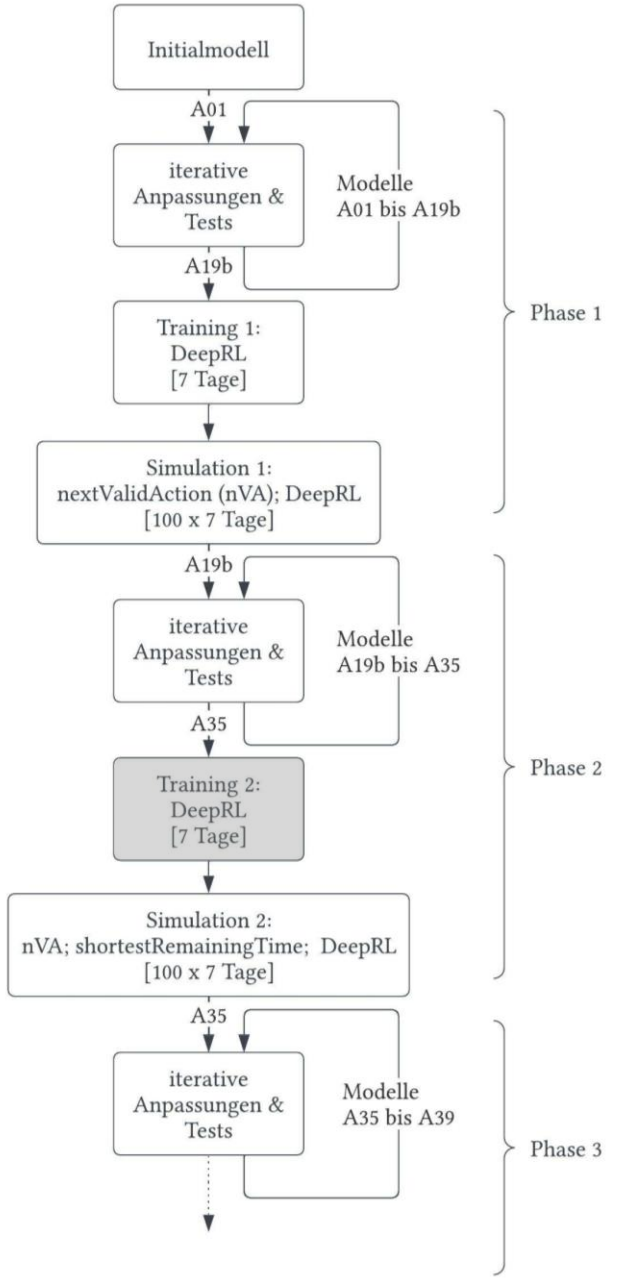
a) Iterative Modellentwicklung
b) Training des selbstlernenden Algorithmus über 7-Tage Produktion
c) Ausführliche Simulationsexperimente (Lernfähigkeit deaktiviert)



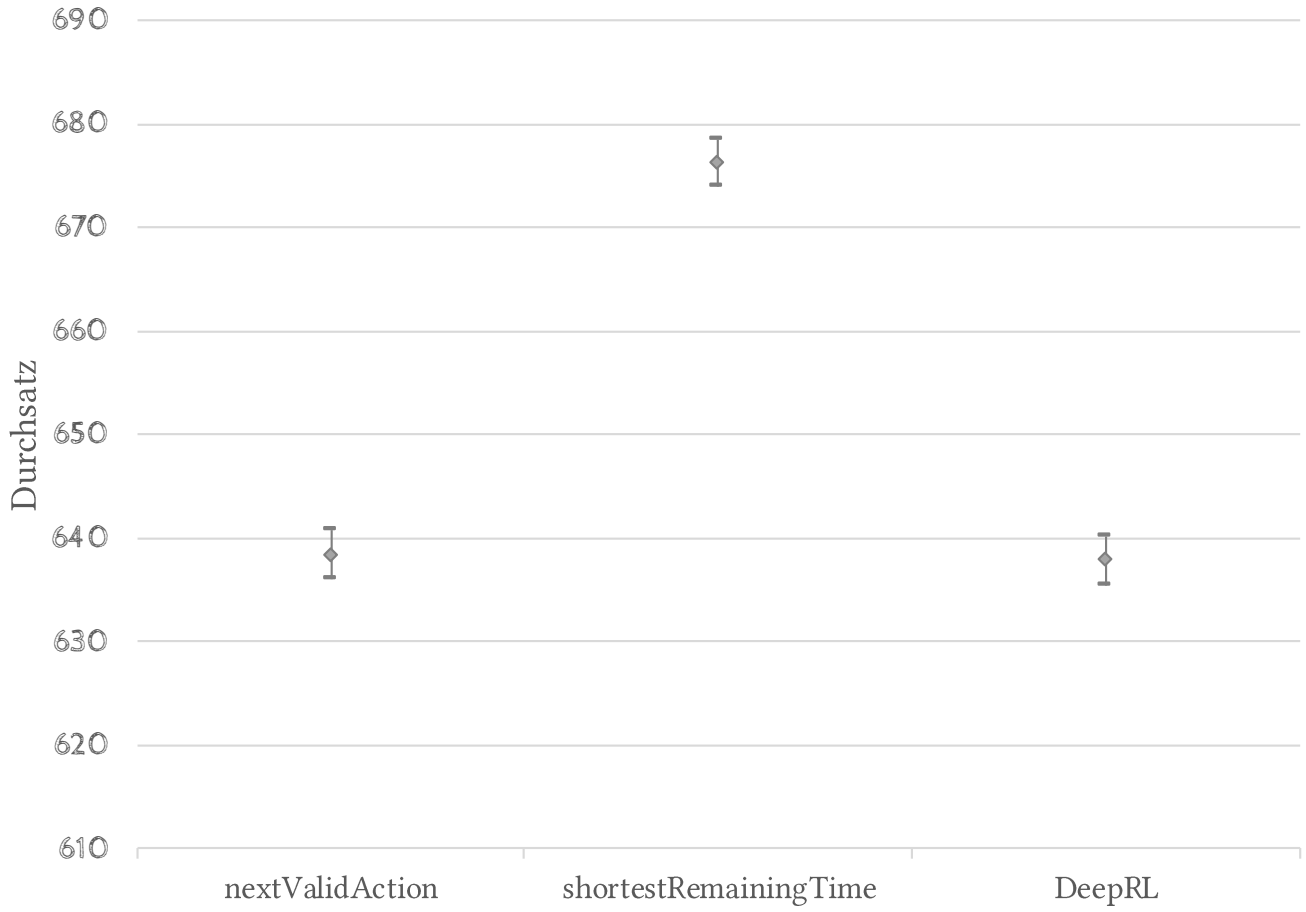
Nachweis der Lernfähigkeit

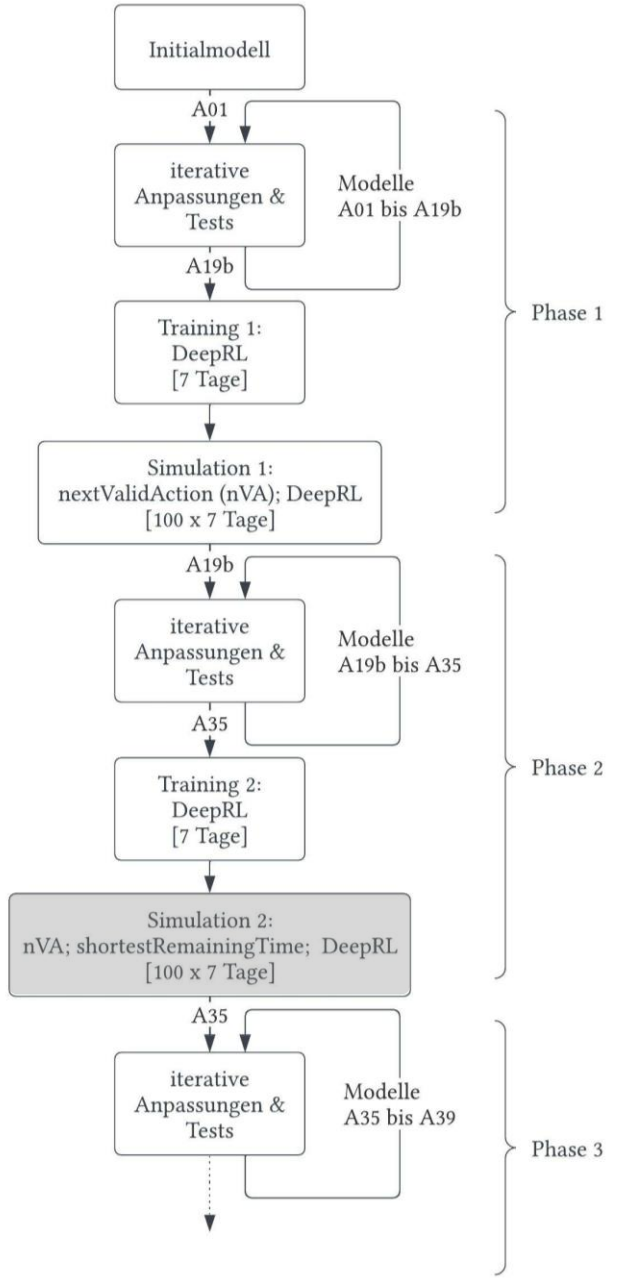
Mean Reward over the last 100 decisions





Simulationsergebnisse





Interpretation der Ergebnisse

Der selbstlernende Algorithmus lernt:

- Unzulässige Aktionen zu vermeiden
- Mögliche Transfers werden direkt durchgeführt
- Keine Priorisierung der Aufträge
- Keine Zusammenarbeit der Agenten untereinander

Möglichkeiten zur Leistungssteigerung

Faktoren	Gamma	Lernrate	KNN-Design	Epsilon Greedy (ε/decay/ε-min)	Belohungsfunktion
Mögliche Faktorenstufen	0,9	0,001	10x24x24x3	1/0,99/0,01	Bestrafung (-5) für unmögliche Wahl; Belohnung (+25) für möglichen Transfer und (10*x) Anzahl an hintereinander durchführbaren Operationen, kein Transfer neutral (0)
	0,99	0,01	10x64x24x3	1/0,99/0,05	Bestrafung (-5) für unmögliche Wahl; Belohnung (+5) für möglichen Transfer; kein Transfer neutral; (+100*x) für x=Zuwachs des Durchsatzes seit letzter Entscheidung
	0,999	0,1	10x24x12x6x3	1/0,999/0,1	Bestrafung (-5) für unmögliche Wahl; kein Transfer Bestrafung (-2); (+100*x) für x=Zuwachs des Durchsatzes seit letzter Entscheidung
			10x48x24x12x3		Bestrafung (-5) für unmögliche Wahl; Belohnung (+100*x) für x=Zuwachs des Durchsatzes seit letzter Entscheidung

Kritik

- Eingeschränkte Nutzbarkeit durch hohe Berechnungszeiten

-> meiste Zeit für Entscheidungsabfrage aus Python
(unabhängig von de-/aktivierter Lernfähigkeit)

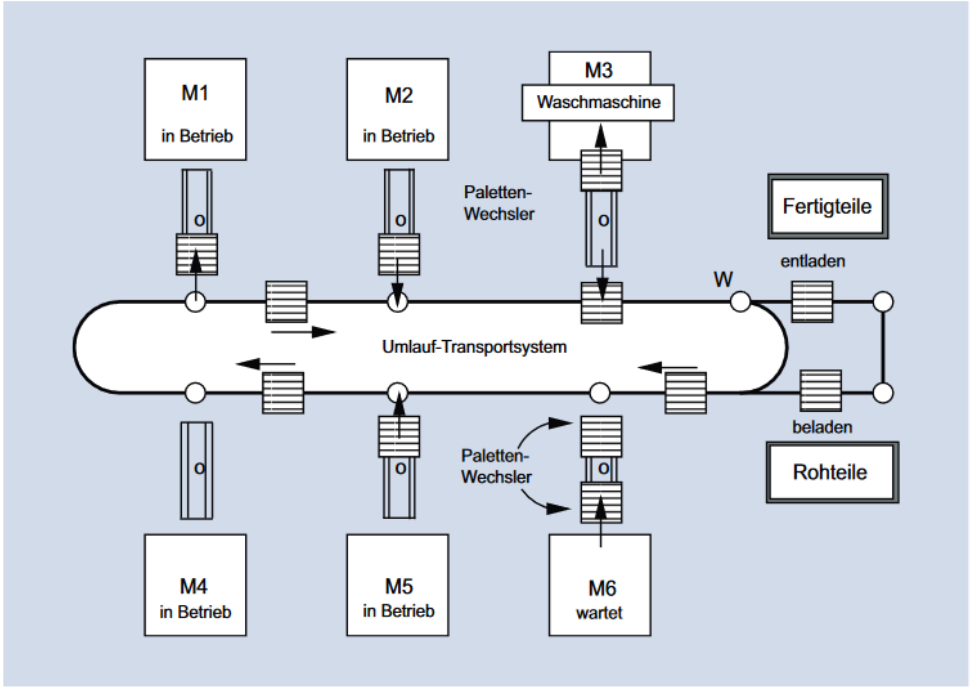
- Reduktion der Leistung auf die Kenngröße Durchsatz

-> Multi-Kriterien Optimierung (Auslastung, Termintreue, etc.)

Kritik

- Realitätsferne durch kontinuierliche Umlaufförderung („zufällige“ Entscheidungsabfrage)

-> eigenständiger Agent entscheidet über Transporte



Zusammenfassung und Ausblick

Wie leistungsfähig ist ein selbstlernendes Multiagentensystem in der Produktionssteuerung?

Mit geringem Aufwand[1] lässt sich das Leistungsniveau eines einfachen statischen Steuerungsansatzes[2] erreichen.

Die Anpassung des Lernverfahrens kann zu Leistungssteigerungen führen.

[1] Bestehende Softwarebausteine

[2] Mögliche Bearbeitungsschritte werden direkt durchgeführt (keine Priorisierung, aber auch keine unzulässigen Entscheidungen)