Informatik
Hauptcampus
H O C H
S C H U L E
T R I E R

Simulationsprojekt Fahrstuhlsimulation Meilenstein 2

Frederik Mauren

Ausarbeitung Meilenstein 2 zum Modul "Reinforcement Learning"

Betreuer: Prof. Dr. Christoph Lürig

Trier, 14.06.2025

Kurzfassung

Im zweiten Meilenstein wurde das Simulationsmodell um eine lernbasierte Steuerungsstrategie auf Basis von Reinforcement Learning (PPO) erweitert. Zunächst wurde der RL-Agent für ein System mit nur einem Aufzug trainiert und die gewonnenen Erkenntnisse anschließend auf ein Szenario mit drei Aufzügen projiziert. Anstelle der zuvor eingesetzten Scanning-Strategie, bei der die Fahrstühle systematisch alle Etagen abfahren, lernt der Agent nun eigenständig, das Verhalten des Fahrstuhls zu optimieren, um die Warte- und Transportzeiten der Gäste zu minimieren.

Für das Training des RL-Agents wurde die Bibliothek Stable Baselines3 genutzt, wobei die Trainingsumgebung durch parallele Instanzen (SubprocVecEnv) beschleunigt und durch episodische Checkpoints und Logging überwacht wurde. Die Auswertung der Simulationsergebnisse erfolgte erneut mit pandas und matplotlib.

Der Vergleich beider Strategien zeigt: Der RL-Agent reduziert Warte- und Gesamtwartezeiten, während die Scanning-Strategie insgesamt schnellere Transportzeiten im Aufzug erreicht. Die Analyse zeigt, dass beide Steuerungsstrategien jeweils spezifische Stärken besitzen. Mit Reinforcement Learning lässt sich das Fahrstuhlsystem im Hinblick auf Wartezeiten weiter optimieren, sofern die zugrundeliegenden Parameter und Umgebungsbedingungen bekannt sind.

Für zukünftige Arbeiten bietet sich an, die Reward-Funktion gezielt anzupassen, um die Fahrzeiten weiter zu senken oder andere gewünschte Verhaltensweisen zu fördern. Darüber hinaus könnte der Reinforcement-Learning-Agent speziell auf ein System mit drei Aufzügen trainiert werden, um die Koordination und Effizienz weiter zu verbessern.

Inhaltsverzeichnis

1

Einleitung und Problemstellung

Im zweiten Meilenstein dieser Arbeit steht die Weiterentwicklung und Differenzierung der Simulationsumgebung für Fahrstuhlsysteme im Fokus. Ausgehend von der im ersten Meilenstein entwickelten Scanning-Strategie wurden drei unterschiedliche Umgebungen konzipiert: Zum einen die angepasste Scanning-Umgebung, in der die Parameter weiter verfeinert wurden, die Treppenoption entfernt wurde und Gäste nun zwischen den Etagen wechseln können; zum anderen eine spezielle Trainingsumgebung, die auf die effiziente Entwicklung eines lernenden Agenten ausgelegt ist, sowie schließlich eine eigene Reinforcement-Umgebung, in der das gelernte Modell evaluiert wird. Die grundlegende Steuerungslogik der Fahrstühle bleibt in allen drei Umgebungen identisch, wodurch die Ergebnisse und Kennzahlen direkt vergleichbar sind.

Hintergrund dieser Entwicklung ist die steigende Notwendigkeit, Fahrstuhlsteuerungen an dynamische und komplexe Bedingungen anpassen zu können. Insbesondere in modernen Gebäuden mit hohem Besucheraufkommen und wechselnder Auslastung reichen einfache, fest kodierte Steuerungsansätze häufig nicht mehr aus, um eine optimale Beförderung der Gäste zu gewährleisten. Während die klassische Scanning-Strategie alle Etagen systematisch abfährt und so eine einfache Grundlogik darstellt, zeigen sich in der Praxis gerade bei stark schwankender Auslastung oder variierendem Gästeaufkommen Defizite hinsichtlich der Warte- und Transportzeiten. Die Weiterentwicklung intelligenter Steuerungsverfahren leistet daher einen wichtigen Beitrag, um den Anforderungen moderner Gebäudetechnik gerecht zu werden und Ressourcen effizienter zu nutzen.

Ziel des zweiten Meilensteins ist es, ein lernbasiertes Modell zu entwickeln, das gezielt darauf optimiert wird, die durchschnittliche Gesamtwartezeit der Gäste zu minimieren und gleichzeitig auch die durchschnittlichen Warte- und Transportzeiten möglichst gering zu halten. Der Fokus liegt dabei auf dem praktischen Einsatz von Reinforcement Learning, einem Teilgebiet des maschinellen Lernens, das besonders für Entscheidungsprobleme in komplexen, dynamischen Umgebungen geeignet ist. Zu diesem Zweck wurde ein Reinforcement-Learning-Agent auf Basis von Proximal Policy Optimization (PPO) in die Simulationsumgebung integriert und zunächst in einer Umgebung mit einem Aufzug trainiert, bevor das erlernte Steuerungsverhalten auf ein realistischeres Szenario mit drei Aufzügen übertragen wurde.

2

Verwendete Technologien und Bibliotheken

Für die Umsetzung und Weiterentwicklung der Simulation sowie für das Training und die Analyse des Reinforcement-Learning-Ansatzes wurden verschiedene Technologien und Python-Bibliotheken eingesetzt und ergänzt. Während im ersten Meilenstein der Fokus noch auf SimPy, pygame und klassischen Auswertungsbibliotheken lag, wurde das technische Fundament im zweiten Meilenstein insbesondere um moderne Werkzeuge für maschinelles Lernen erweitert.

Kern der RL-Implementierung ist die Bibliothek **stable-baselines3**, die aktuelle Reinforcement-Learning-Algorithmen wie **Proximal Policy Optimization (PPO)** bereitstellt und das Training von Agenten in benutzerdefinierten Umgebungen unterstützt. Im Rahmen dieser Arbeit wurde speziell **PPO** für das Training des RL-Agenten verwendet. Für Maskierungen und weitere Erweiterungen kam zusätzlich das Paket **sb3-contrib** zum Einsatz. Die **gymnasium**- beziehungsweise **gym**-Bibliothek diente zur Definition und zum Wrapping der RL-Umgebung, sodass das Fahrstuhlsystem nahtlos in den RL-Workflow eingebunden werden konnte. Die Trainingsumgebung wurde durch **SubprocVecEnv** parallelisiert, um mehrere Episoden gleichzeitig simulieren und so den Lernprozess erheblich beschleunigen zu können. Die zugrundeliegenden neuronalen Netze wurden mit **PyTorch** (über stable-baselines3) trainiert.

Die eigentliche Simulation, Visualisierung und Auswertung erfolgte wie bisher mit **SimPy** (Simulationslogik), **pygame** (grafische Darstellung), **numpy** und **random** (numerische und stochastische Prozesse), **copy** (Objektmanipulation), **pandas** (Datenanalyse) und **matplotlib** (grafische Auswertung). **tkinter** kam weiterhin zur flexiblen Einstellung von Simulationsparametern zum Einsatz.

Durch die Kombination dieser Bibliotheken konnte sowohl eine leistungsfähige Simulations- und Trainingsumgebung als auch eine umfassende Analyse- und Visualisierungsumgebung realisiert werden. So wurde es möglich, klassische Steuerungsansätze und moderne Reinforcement-Learning-Algorithmen direkt miteinander zu vergleichen und systematisch auszuwerten.

3

Training

In diesem Kapitel werden zunächst die Struktur des Action- und Observation-Spaces sowie die verwendete Reward-Funktion für den Reinforcement-Learning-Ansatz vorgestellt. Anschließend wird beschrieben, wie die Trainingsumgebung entwickelt wurde und welche Anpassungen für das Training des RL-Agenten vorgenommen wurden. Abschließend wird erläutert, wie das eigentliche Training durchgeführt wurde und nach welchen Kriterien die Leistung bewertet wurde.

3.1 Definition von Action Space, Observation Space und Reward-Funktion

In diesem Abschnitt werden der Action Space, der Observation Space und die Reward-Funktion des Fahrstuhlmodells vorgestellt, die als Grundlage für das Reinforcement-Learning-Training dienen.

3.1.1 Action Space

Der Action Space des Reinforcement-Learning-Agents wurde als MultiDiscrete-Space mit drei möglichen Aktionen pro Aufzug definiert. Für jeden Zeitschritt kann der Agent zwischen „Warten“ (wait), „nach oben fahren“ (up) und „nach unten fahren“ (down) wählen. Da im Training ausschließlich ein einzelner Aufzug betrachtet wurde, besteht der Action Space für diesen Fall aus genau drei diskreten Aktionen.

Die Auswahl zulässiger Aktionen wird durch eine Aktionsmaske eingeschränkt, die in jedem Zeitschritt dynamisch berechnet wird. Die Aktion „Warten“ steht dem Agenten dabei nur zur Verfügung, wenn Gäste auf der aktuellen Etage ein- oder aussteigen möchten und der Aufzug nicht gerade erst seine Türen geschlossen hat, also nicht unmittelbar zuvor schon gewartet wurde. Dies verhindert ineffiziente Folgeaktionen und sorgt dafür, dass „Warten“ tatsächlich nur dann gewählt werden kann, wenn es auch sinnvoll ist.

Die Aktionen „nach oben fahren“ und „nach unten fahren“ sind ausschließlich dann möglich, wenn die Türen geschlossen sind und sich der Aufzug nicht bereits auf der jeweiligen Endetage befindet. Durch diese gezielte Einschränkung des

Action Spaces wird sichergestellt, dass der Agent nur zulässige und sinnvolle Steuerbefehle auswählt, was die Effizienz und das Lernverhalten deutlich verbessert.

3.1.2 Observation Space

Der Observation Space des Reinforcement-Learning-Agents wurde so gestaltet, dass alle für die Steuerung des Aufzugs relevanten Informationen in kompakter Form abgebildet werden. Im Trainingsszenario mit einem einzelnen Aufzug umfasst der Observation Space insbesondere folgende Merkmale: die aktuelle Position des Aufzugs (Etage), den Belegungsstatus (Anzahl der sich im Aufzug befindenden Gäste), ein Histogramm der Ziel-Etagen der Passagiere im Aufzug sowie Informationen zu wartenden Gästen auf den einzelnen Etagen. Dadurch erhält der Agent in jedem Zeitschritt ein vollständiges Abbild des aktuellen Systemzustands.

Die Gestaltung des Observation Spaces stellt sicher, dass dem Agenten alle Informationen zur Verfügung stehen, die für fundierte und effiziente Steuerungsentscheidungen notwendig sind. Gleichzeitig wurde darauf geachtet, die Dimensionalität des Zustandsraums so kompakt wie möglich zu halten, um das Training effizient zu gestalten und die Generalisierbarkeit des gelernten Verhaltens zu unterstützen.

Durch diese Definition des Observation Spaces ist gewährleistet, dass der Agent das Verhalten des Aufzugs und die Bedürfnisse der Gäste zuverlässig erfassen und darauf aufbauend optimierte Aktionen auswählen kann.

3.1.3 Reward-Funktion

Die Reward-Funktion ist so gestaltet, dass sie das Verhalten des Agents gezielt auf die Minimierung der Warte- und Transportzeiten der Gäste ausrichtet. Der Agent erhält eine positive Belohnung, wenn Gäste ihr Ziel erreichen oder zusteigen. Die Höhe der Belohnung hängt jeweils von der aufgewendeten Zeit ab: Je schneller ein Gast nach dem Einsteigen an seiner Ziel-Etage ankommt, desto höher fällt die Belohnung aus. Die Belohnung beim Aussteigen eines Gastes berechnet sich dabei wie folgt:

$$\text{Reward}_{\text{Aussteigen}} = \max(1,\ 20 - \frac{\text{Wartezeit in Schritten}}{60})$$

Für das Einsteigen eines Gastes erhält der Agent ebenfalls eine positive Belohnung, die umso größer ist, je kürzer der Gast auf den Aufzug gewartet hat:

$$\text{Reward}_{\text{Einsteigen}} = \max(1,\ 10 - \frac{\text{Wartezeit auf Aufzug}}{60})$$

Zusätzlich wurden negative Belohnungen (Strafen) eingeführt, um zu verhindern, dass die Aufzüge in einen ineffizienten Aktionskreislauf geraten, bei dem keine sinnvollen Aktionen mehr ausgeführt werden. So wird für jeden wartenden Gast auf den Etagen in jedem Zeitschritt eine Strafe von $-0{,}1$ vergeben. Für jeden Gast im Aufzug fällt pro Zeitschritt eine Strafe von $-0{,}05$ an.

Eine Episode endet entweder, wenn alle Gäste das Gebäude verlassen haben oder die maximale Episodenlänge erreicht ist. Durch diese Kombination aus Belohnungen und Strafen wird der Agent dazu angeregt, das Gesamtsystem effizient zu steuern und sowohl Warte- als auch Transportzeiten zu optimieren.

3.2 Konzeption und Aufbau der Trainingsumgebung

In diesem Abschnitt wird zunächst erläutert, wie die Logik der Trainingsumgebung für den Reinforcement-Learning-Agenten konzipiert und umgesetzt wurde. Anschließend wird beschrieben, wie auf Basis dieser Logik die eigentliche Trainingsumgebung entwickelt und in das Simulationssystem integriert wurde.

3.2.1 Trainings und Simulationsumgebungslogik

Die Trainings- und Simulationsumgebungslogik wurde so konzipiert, dass sie realistische Bedingungen für das Verhalten von Fahrgästen und Aufzügen in einem Bürogebäude abbildet und gleichzeitig effizientes Training des Reinforcement-Learning-Agenten ermöglicht. Ein Simulationsschritt entspricht jeweils einer realen Sekunde, wodurch zeitliche Abläufe präzise und nachvollziehbar modelliert werden können. Die Ankunft der Gäste im Gebäude erfolgt exponentialverteilt über einen Zeitraum von 120 Minuten, um natürliche Stoßzeiten und zufällige Verteilung des Besucheraufkommens abzubilden.

Die simulierten Gäste verbringen eine individuell festgelegte Arbeitszeit im Gebäude, die zwischen 6 Stunden 20 Minuten und 9 Stunden 38 Minuten liegt. Die für Fahrten mit dem Aufzug benötigte Zeit wird dabei von der Arbeitszeit der Gäste abgezogen, sodass die Aufenthaltsdauer realitätsnah abgebildet wird. Gäste können dabei auch auf der Etage 0 arbeiten; beim Betreten oder Verlassen dieser Etage wird kein Aufzug benötigt. Während ihres Aufenthalts wechseln die Gäste im Durchschnitt zwei Mal pro Stunde die Etage, um typische Bewegungsmuster in einem Bürogebäude zu simulieren.

Der Ablauf der Fahrstuhlnutzung ist ebenfalls detailliert modelliert: Das Öffnen und Schließen der Türen dauert jeweils vier Sekunden. Ein Etagenwechsel nimmt vier Sekunden pro Etage in Anspruch. Insgesamt ergibt sich daraus, dass ein ganzer Tag in der Simulation – also die Zeit, bis alle Gäste das Gebäude wieder verlassen haben – ungefähr 40.000 Simulationsschritte beziehungsweise etwa elf Stunden umfasst. Die Gäste sind zu Beginn zufällig auf alle Etagen des Gebäudes verteilt, was eine gleichmäßige Ausgangslage für das Training und die Auswertung sicherstellt.

Für die Simulationsumgebung wurden drei Aufzüge und 200 Gäste modelliert, um das Zusammenspiel mehrerer Aufzüge und ein hohes Gästeaufkommen zu untersuchen. In der Trainingsumgebung dagegen wurde das Szenario auf einen einzigen Aufzug und 80 Gäste reduziert, um die Komplexität während des Lernprozesses zu begrenzen und gezielte Steuerungsstrategien zu erlernen. Da es keine alternative Möglichkeit mehr gibt, zwischen den Etagen zu wechseln, sind die Gäste vollständig auf die Nutzung des Aufzugs angewiesen – die Option, die Treppe zu nehmen, wurde entfernt.

Besonders wichtig ist, dass die grundlegende Logik und Funktionsweise in beiden Umgebungen vollständig identisch umgesetzt ist. Lediglich die Anzahl der Aufzüge und Gäste wurde angepasst. Dadurch sind die Ergebnisse des Trainings direkt auf die größere Simulationsumgebung übertragbar und ermöglichen einen aussagekräftigen Vergleich der erlernten Strategien unter verschiedenen Bedingungen.

3.2.2 Trainingsumgebung

Die Trainingsumgebung wurde gezielt auf die Anforderungen des Reinforcement-Learning-Trainings zugeschnitten. Die grundlegende Logik der Trainingsumgebung entspricht dabei vollständig derjenigen der Scanning-Simulationsumgebung, sodass beide Ansätze direkt vergleichbar sind. Im Unterschied zur Simulationsumgebung wird bei der Trainingsumgebung jedoch auf die Nutzung von SimPy für die Ereignissteuerung und pygame für die grafische Darstellung verzichtet, um das Training effizienter und ressourcenschonender zu gestalten.

Der Fokus der Trainingsumgebung liegt auf der reinen Aufzugslogik und Steuerung. Während in der klassischen Simulationsumgebung die Steuerung über einen Dispatcher und ein ereignisorientiertes System mit Ereignissen und Warteschlangen realisiert wurde, übernimmt in der Trainingsumgebung nun der RL-Agent die zentrale Steuerungsfunktion. Der RL-Agent trifft auf Basis des aktuellen Systemzustands eigenständig Entscheidungen, welche Aktionen der Aufzug ausführen soll. Dadurch wird der Trainingsprozess beschleunigt, und die Lernschritte des Agenten können gezielt ausgewertet werden.

Durch diese Reduktion auf das Wesentliche und die enge Anlehnung an die ursprüngliche Logik werden eine hohe Effizienz beim Training und eine direkte Vergleichbarkeit der Ergebnisse gewährleistet.

3.3 Training des Modells

Das Training des Reinforcement-Learning-Modells wurde mit parallelen Trainingsumgebungen unter Verwendung von `SubprocVecEnv` durchgeführt. Durch die gleichzeitige Ausführung mehrerer Episoden konnten Daten effizienter gesammelt und der Lernprozess deutlich beschleunigt werden. Während des Trainings kam ein Episoden-Callback zum Einsatz, der nach dem Abschluss jeder Episode verschiedene Aktionen ermöglichte, wie etwa das Logging von Kennzahlen, die Evaluation des aktuellen Modells oder das frühzeitige Beenden des Trainings bei Erreichen bestimmter Kriterien. Zusätzlich wurde ein Checkpoint-Callback integriert, um das Modell in regelmäßigen Abständen automatisch zu speichern. Dies erleichtert sowohl die Sicherung von Zwischenergebnissen als auch eine spätere Analyse und den Vergleich verschiedener Trainingsstände.

Zu Beginn des Projekts wurde versucht, das gesamte komplexe Modell mit drei Aufzügen und 200 Gästen direkt zu trainieren. Dieser Ansatz erwies sich jedoch als ineffizient, da der Reinforcement-Learning-Agent in dieser umfangreichen Umgebung kein sinnvolles Verhalten erlernen konnte. Im weiteren Verlauf wurde deshalb

ein Curriculum-Learning-Ansatz gewählt, bei dem die Komplexität der Umgebung und die Anforderungen an den Agenten schrittweise erhöht wurden.

Zu den angepassten Parametern zählten insbesondere die Anzahl der Stockwerke, die Anzahl der Gäste, der Zeitraum, in dem Gäste spawnen, sowie deren Arbeitszeit und die Wahrscheinlichkeit für einen Wechsel des Stockwerks. Darüber hinaus wurden auch die Actionmask, die Reward-Funktion und die Episodenlänge mehrfach angepasst und aufeinander abgestimmt. Durch diese schrittweise Steigerung der Schwierigkeitsgrade konnte der Agent zunächst in einfacheren Szenarien grundlegende Strategien erlernen, bevor er in komplexeren Umgebungen eingesetzt wurde.

Während des Trainings wurde gezielt darauf geachtet, dass sich die durchschnittlichen Wartezeiten der Gäste kontinuierlich reduzierten und dass die erzielten Rewards im Verlauf des Trainings anstiegen. Dies diente als zentrales Kriterium zur Beurteilung des Lernerfolgs und der Effektivität der entwickelten Steuerungsstrategie. Das Training wurde dabei vollständig auf einer CPU durchgeführt.

Das finale Training des Modells umfasste insgesamt 22.531.560 Zeitschritte, was 1.318 vollständigen Episoden entspricht. Dieser Ansatz ermöglichte es, das Training effizient zu gestalten und letztlich ein robustes und leistungsfähiges Steuerungsmodell zu entwickeln.

4

Integration des trainierten Modells in die Reinforcement-Simulationsumgebung

Die finale Reinforcement-Simulationsumgebung wurde als Kombination aus der ursprünglichen Scanning-Simulationsumgebung aus Meilenstein 1 und der Trainingsumgebung konzipiert. Ziel war es, die Vorteile beider Ansätze miteinander zu verbinden und das trainierte Reinforcement-Learning-Modell unter realistischen Bedingungen zu evaluieren. Für die Umsetzung wurden sowohl `pygame` für die grafische Darstellung als auch `SimPy` für die ereignisorientierte Simulation übernommen und wieder integriert.

Im Unterschied zur Scanning-Simulationsumgebung agieren die Gäste in der Reinforcement-Simulationsumgebung nicht als eigenständige Prozesse. Stattdessen werden ihre Aktionen und Zustandsänderungen zentral gesteuert, wie bereits in der ursprünglichen Trainingsumgebung. Die Steuerung der Aufzüge erfolgt hingegen vollständig über das trainierte Modell. Dabei wird in jedem Simulationsschritt für jeden einzelnen Aufzug separat ein entsprechender Observation Space erstellt, der alle relevanten Informationen des jeweiligen Aufzugs abbildet. Basierend auf diesem individuellen Zustand wird für jeden Aufzug jeweils eine eigene Aktion bestimmt, sodass die Steuerungsentscheidungen des Modells für jeden Aufzug unabhängig getroffen werden.

Diese Vorgehensweise ermöglicht es, das ursprünglich nur auf einen einzelnen Aufzug trainierte Modell direkt auf ein Szenario mit mehreren Aufzügen (hier: drei Aufzüge) zu übertragen. Die Verallgemeinerung auf mehrere Aufzüge erfolgt, indem das Modell separat und unabhängig für jeden Aufzug verwendet wird. Diese Strategie wurde gewählt, da ein direktes Training mit mehreren Aufzügen die Komplexität der Lernaufgabe erheblich erhöhen würde und in der Praxis schwer zu realisieren wäre.

Durch diese Architektur kann das trainierte Modell flexibel in die Reinforcement-Simulationsumgebung eingebunden werden. Gleichzeitig bleibt die Nachvollziehbarkeit und Vergleichbarkeit mit der bisherigen Systemlogik erhalten. So lässt sich die Leistungsfähigkeit des RL-basierten Ansatzes im direkten Vergleich zu der in Meilenstein 1 entwickelten Scanningstrategie in der Scanning-Simulationsumgebung unter praxisnahen Bedingungen untersuchen.

5

Ergebnisse

In diesem Kapitel werden zunächst die Methodik und das Vorgehen zur Auswertung der Simulationsergebnisse erläutert. Im Anschluss werden die wesentlichen Resultate präsentiert und die Leistung der verschiedenen Steuerungsstrategien miteinander verglichen.

5.1 Methodik

Zur Ermittlung der Ergebnisse wurden jeweils zehn unabhängige Simulationsdurchläufe (Episoden) mit der Scanning-Strategie und zehn Episoden mit der Reinforcement-Learning-Strategie durchgeführt. Für jeden Durchlauf wurden zentrale Kennzahlen wie Wartezeiten, Transportzeiten und die Auslastung der Aufzüge aufgezeichnet. Anschließend wurden die Durchschnittswerte aller Episoden für beide Strategien berechnet, um eine robuste und aussagekräftige Vergleichsbasis zu schaffen.

Neben den Gesamtdurchschnittswerten pro Episode beziehungsweise pro Tag wurden auch stündliche Durchschnittswerte ermittelt. Dies ermöglicht eine detaillierte Analyse des zeitlichen Verlaufs der Systemleistung und zeigt, wie sich die beiden Steuerungsstrategien während verschiedener Phasen des Simulationstags verhalten.

5.2 Resultate

In diesem Abschnitt werden die zentralen Ergebnisse der durchgeführten Simulationen vorgestellt und die Leistungsfähigkeit der unterschiedlichen Steuerungsstrategien miteinander verglichen. Zunächst werden die wichtigsten Kennzahlen tabellarisch und textlich gegenübergestellt, bevor im Anschluss die zeitlichen Verläufe und weitere Charakteristika mithilfe von Diagrammen visualisiert und analysiert werden.

5.2.1 Vergleich zentraler Leistungskennzahlen

Die Auswertung der Simulationsergebnisse zeigt deutliche Unterschiede zwischen der Scanning-Strategie und der Reinforcement-Learning-Strategie. Im Mittel über

zehn Episoden beträgt bei der Scanning-Strategie die durchschnittliche Wartezeit der Gäste **47,32 Sekunden**, die durchschnittliche Fahrzeit **31,04 Sekunden** und die durchschnittliche Gesamtwartezeit somit **78,36 Sekunden**.

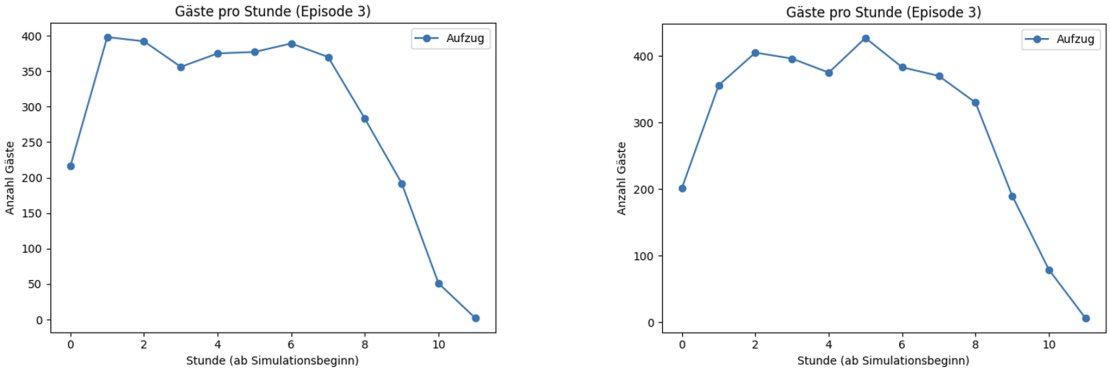
Bei Anwendung der Reinforcement-Learning-Strategie verringert sich die durchschnittliche Wartezeit deutlich auf **23,90 Sekunden**. Die durchschnittliche Fahrzeit steigt auf **48,35 Sekunden**, die durchschnittliche Gesamtwartezeit sinkt dennoch auf **72,25 Sekunden**.

Diese Ergebnisse verdeutlichen, dass der Reinforcement-Learning-Ansatz insbesondere bei der Reduzierung der Wartezeiten und der Gesamtwartezeit Vorteile gegenüber der klassischen Scanning-Strategie bietet. Die höhere Fahrzeit bei Reinforcement Learning weist darauf hin, dass der Agent offenbar Priorität darauf legt, Gäste möglichst schnell aufzunehmen und zu transportieren, selbst wenn die eigentliche Fahrzeit dadurch länger wird. Insgesamt zeigt sich, dass die lernbasierte Steuerung zu einer effizienteren Nutzung des Fahrstuhlsystems mit deutlich kürzeren Wartezeiten führt.

5.2.2 Grafischer Vergleich der Strategien

Für den grafischen Vergleich der beiden Strategien wurde jeweils die dritte von zehn simulierten Episoden ausgewählt, da diese hinsichtlich aller zentralen Kennzahlen sehr nah an den jeweiligen Durchschnittswerten lag. Somit bieten die gezeigten Diagramme einen repräsentativen Einblick in den typischen Verlauf der Simulationen sowohl für die Scanning-Strategie als auch für den Reinforcement-Learning-Ansatz.

Grafik 1: Anzahl Transportierter Personen pro Stunde



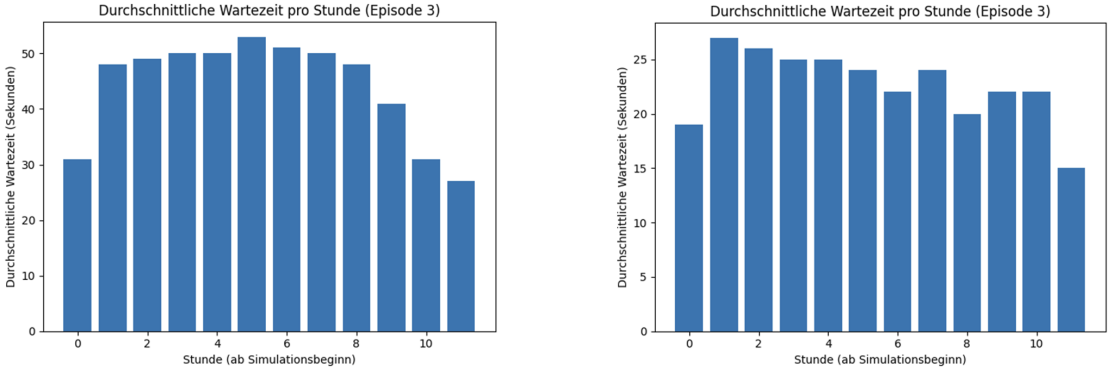
(a) Scanning-Strategie (b) Reinforcement-Learning-Strategie

Abbildung 5.1: Vergleich der Strategien

Die Anzahl der pro Stunde transportierten Personen verläuft bei beiden Steuerungsstrategien nahezu identisch. Im Spitzenbereich werden etwa 400 Personen pro Stunde befördert, was der in der Simulation vorgegebenen Wahrscheinlichkeit für

einen Etagenwechsel entspricht. Zu Beginn ist erkennbar, dass in den ersten beiden Stunden die Gäste exponentialverteilt im Gebäude ankommen.

Grafik 2: Durchschnittliche Wartezeit pro Stunde

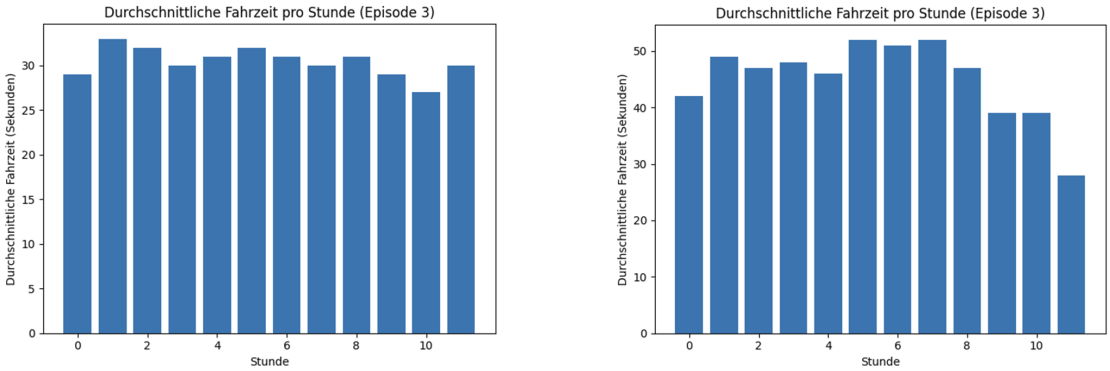


(a) Scanning-Strategie (b) Reinforcement-Learning-Strategie

Abbildung 5.2: Vergleich der Strategien

Im gesamten Simulationsverlauf zeigen sich mit der Reinforcement-Learning-Strategie deutlich geringere Wartezeiten auf den Aufzug im Vergleich zur Scanning-Strategie.

Grafik 3: Durchschnittliche Fahrtzeit pro Stunde

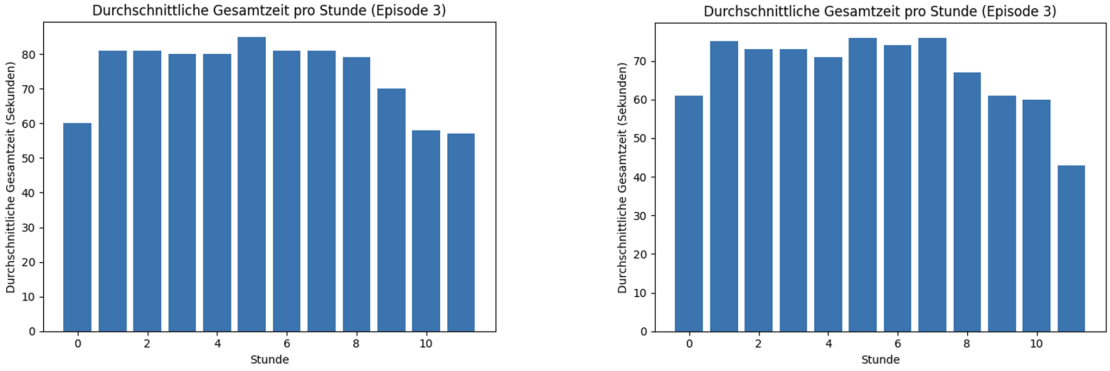


(a) Scanning-Strategie (b) Reinforcement-Learning-Strategie

Abbildung 5.3: Vergleich der Strategien

Über die gesamte Simulation hinweg erzielt die Scanning-Strategie geringere Fahrzeiten im Vergleich zur Reinforcement-Learning-Strategie.

Grafik 4: Durchschnittliche Gesamtzeit pro Stunde



(a) Scanning-Strategie

(b) Reinforcement-Learning-Strategie

Abbildung 5.4: Vergleich der Strategien

Die durchschnittlichen Gesamtzeiten, bestehend aus Warte- und Fahrzeiten, fallen bei der Reinforcement-Learning-Strategie geringer aus als bei der Scanning-Strategie.

6

Analyse

Die Analyse der Simulationsergebnisse zeigt, dass sich die beiden Steuerungsstrategien in ihren grundlegenden Funktionsweisen und Auswirkungen deutlich unterscheiden. Die Reinforcement-Learning-Steuerung positioniert die Aufzüge häufig in den mittleren Etagen, wodurch die durchschnittlichen Wartezeiten der Gäste deutlich verringert werden. Im Gegensatz dazu fahren die Aufzüge bei der Scanning-Strategie systematisch alle Etagen in eine Richtung ab und nehmen dabei die Gäste auf, die in Fahrtrichtung unterwegs sind. Dies führt zu insgesamt kürzeren Fahrzeiten, da weniger Umwege entstehen.

Ein wichtiger Unterschied zwischen den beiden Ansätzen besteht darin, dass der Reinforcement-Learning-Agent über deutlich mehr Informationen verfügt als die klassische Scanning-Strategie. So kann der RL-Agent bei seiner Entscheidungsfindung sowohl die Anzahl der Passagiere im Aufzug als auch die Zahl der wartenden Personen auf den einzelnen Etagen mit einbeziehen. Diese umfassende Informationsbasis ermöglicht flexiblere und gezieltere Steuerungsentscheidungen. Allerdings ist zu berücksichtigen, dass diese Informationslage im praktischen Betrieb so nicht ohne Weiteres verfügbar ist. In realen Fahrstuhlsystemen ist es in der Regel nicht möglich, jederzeit die exakte Zahl der wartenden Personen auf allen Etagen oder die genaue Belegung jedes Aufzugs zu kennen. Dadurch profitiert der RL-Agent in der Simulation von einer idealisierten Beobachtungsgrundlage, was die Übertragbarkeit der Ergebnisse auf reale Systeme einschränken kann.

Darüber hinaus ist zu beobachten, dass die RL-Strategie vermutlich weniger robust auf sich ändernde Rahmenbedingungen reagiert als die klassische Scanning-Strategie, da sie stark auf die im Training verwendeten Parameter optimiert ist. Änderungen wie ein anderes Passagieraufkommen, eine abweichende Anzahl an Etagen oder veränderte Bewegungsmuster der Gäste könnten die Effizienz der RL-Steuerung daher beeinträchtigen.

Ein zentraler Aspekt der Analyse betrifft die Ausgestaltung der Reward-Funktion: Wartende Gäste auf den Etagen werden im aktuellen Belohnungsschema stärker bestraft als Passagiere, die sich bereits im Aufzug befinden. Dies führt dazu, dass der RL-Agent seinen Fokus insbesondere auf kurze Abholzeiten legt, während die eigentlichen Transportzeiten nachrangig behandelt werden. Entsprechend werden Gäste möglichst schnell aufgenommen, selbst wenn sich dadurch längere Fahrzeiten ergeben.

7

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Simulationsmodell für ein Fahrstuhlsystem entwickelt, mit dem Ziel, verschiedene Steuerungsstrategien hinsichtlich ihrer Effizienz und ihres Einflusses auf Warte- und Fahrzeiten der Gäste zu analysieren. Aufbauend auf einer klassischen Scanning-Strategie wurde ein Reinforcement-Learning-Ansatz implementiert, trainiert und in die Simulationsumgebung integriert. Die entwickelte RL-Steuerung konnte dabei die durchschnittlichen Wartezeiten und die Gesamtwartezeiten der Gäste im Vergleich zur Scanning-Strategie deutlich reduzieren, auch wenn die durchschnittlichen Fahrzeiten im Aufzug teilweise anstiegen.

Ein zentrales Ergebnis der Analyse ist, dass die RL-Steuerung die Aufzüge häufig in zentralen Etagen positioniert, um kürzere Wartezeiten zu ermöglichen, während die Scanning-Strategie zu insgesamt geringeren Fahrzeiten führt. Zudem zeigte sich, dass die RL-Strategie flexibel auf die aktuelle Situation reagieren kann, jedoch im Vergleich zur klassischen Lösung eine ideale Informationslage voraussetzt, die in realen Systemen meist nicht gegeben ist. Dies schränkt die Übertragbarkeit der Simulationsergebnisse auf reale Fahrstuhlsysteme ein und stellt eine zentrale Limitation des Ansatzes dar.

Die Erfahrungen mit der Entwicklung und Integration der Mensch-Maschine-Schnittstelle (insbesondere die Visualisierung mittels pygame) zeigten, dass ein interaktives und anschauliches Feedback für die Nachvollziehbarkeit des Systemverhaltens sehr hilfreich ist. Auch die Auswertung und das Logging der Simulationsdaten erwiesen sich als wertvolle Hilfsmittel, um die Performance der verschiedenen Strategien systematisch vergleichen zu können.

Kritisch zu bewerten ist, dass die Performance des RL-Agenten stark von der Gestaltung der Reward-Funktion und den während des Trainings verwendeten Parametern abhängt. Änderungen an diesen Bedingungen könnten zu deutlich anderen Ergebnissen führen und die Robustheit des Modells einschränken. Weiterhin ist zu beachten, dass das Training und die Anwendung des RL-Agents ausschließlich auf einer CPU erfolgten, was längere Trainingszeiten zur Folge hatte.

Als Ausblick ergeben sich mehrere Ansatzpunkte für weiterführende Arbeiten: Zukünftige Projekte könnten darauf abzielen, den RL-Agenten robuster gegenüber sich ändernden Systemparametern zu machen, etwa durch den Einsatz von domänenübergreifendem oder Transferlernen. Eine realistischere Gestaltung des Observation Space – also der Informationsbasis des Agenten – wäre ebenso

wünschenswert, um den Praxisbezug zu erhöhen. Darüber hinaus könnten alternative Reward-Funktionen und weitergehende Optimierungsziele, wie zum Beispiel die Minimierung des Energieverbrauchs oder eine adaptive Priorisierung verschiedener Nutzergruppen, untersucht werden. Zudem wäre es sinnvoll, den RL-Agenten zukünftig direkt auf ein System mit drei Aufzügen zu trainieren, um die Koordination zwischen den Aufzügen zu verbessern und die Steuerung insgesamt noch effizienter zu gestalten. Nicht zuletzt wäre es interessant, den Ansatz auf reale oder experimentelle Fahrstuhldaten anzuwenden und die erzielten Verbesserungen direkt im Betrieb zu validieren.