Wegfindung, Entscheidungsfindung und Verhaltensanalyse in Multiagentensystemen

Carsten Hahn

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität
München

vorgelegt von
Carsten Hahn

eingereicht am
7. Mai 2021

Wegfindung, Entscheidungsfindung und Verhaltensanalyse in Multiagentensystemen

Carsten Hahn

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität
München

vorgelegt von
Carsten Hahn

1. Berichterstatter:	Prof. Dr. Claudia Linnhoff-Popien
2. Berichterstatter:	Univ.-Prof. Dr. Peter Reichl
Tag der Einreichung:	7. Mai 2021
Tag der Disputation:	29. März 2022

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Köln, 7. Mai 2021 Carsten Hahn

Danksagung

Die vorliegende Dissertation ist während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Mobile und Verteilte Systeme der Ludwig-Maximilians-Universität München entstanden. Während der Zeit am Lehrstuhl habe ich in ganz unterschiedlicher Form Unterstützung erhalten, für welche ich mich an dieser Stelle bedanken möchte.

Mein ganz besonderer Dank gilt Frau Prof. Dr. Claudia Linnhoff-Popien. Sie hat es mir ermöglicht, an ihrem Lehrstuhl zu arbeiten und zu promovieren. Ich habe das angenehme, freiheitliche und selbstbestimmte Arbeitsklima am Lehrstuhl immer sehr genossen. Insbesondere die Lehrstuhlworkshops zum Austausch und Hinterfragen von Ideen waren immer wieder Highlights.

Mein Dank gilt deshalb auch dem starken Team des Lehrstuhls, welches sich jederzeit gegenseitig unterstützt und dementsprechend auch mir über die Zeit sehr geholfen hat. Viele schöne Erinnerungen sind mit diesem Team verbunden, sowohl im fachlichen als auch im freundschaftlichen Sinne. Prinzipiell könnte ich an dieser Stelle den Namen jedes Lehrstuhlkollegens nennen und wie er mir geholfen hat, was aber den Rahmen sprengen würde. Namentlich hervorheben möchte ich an dieser Stelle Dr. Lenz Belzner, der mir mit seiner unglaublich positiven, bestärkenden und wissbegierigen Art immer wieder neue Denkanstöße und Themenfelder aufgezeigt hat. Ähnlich, hat mich Dr. Sebastian Feld in vielen Situationen motiviert und mit seinen Anregungen und Sichtweisen die gemeinsame Forschung bereichert. Bedanken möchte ich mich auch bei Thomy Phan und Dr. Thomas Gabor, die mir mit ihrer Kreativität und ihrem Fachwissen aus mancher kniffligen Situation geholfen haben. Zu erwähnen sind noch Dr. Lorenz Schauer, mit dem ich immer sehr die Mittagspausen genossen habe und welcher mir bei der Organisation der Lehre geholfen hat und Dr. Michael Beck, der mich vor allem zu Beginn meiner Zeit am Lehrstuhl gefördert hat. Des Weiteren ergeht mein Dank an Herrn Prof. Dr. Peter Reichl für sein Engagement als Zweitberichterstatter und die von ihm vorgebrachten Verbesserungsvorschläge und an Herrn Prof. Dr. Heinrich Hußmann für die Übernahme des Vorsitzes der Prüfungskommission.

Vor allem möchte ich mich auch bei meinen Eltern bedanken, die mich Zeit ihres Lebens in allen Lagen stets unterstützt haben und bei meinem Bruder Michael, der mir besonders beim Tod unseres Vaters beistand. Mein größter Dank gilt meiner wundervollen Partnerin Melanie, die mir liebevoll, motivierend und beratend zur Seite steht und mich insbesondere in der Zeit während des Schreibens dieser Dissertation unglaublich unterstützt hat.

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Wegfindung, Entscheidungsfindung und der Analyse des Verhaltens intelligenter Agenten, wenn diese in kontinuierlichen, zweidimensionalen Flächen bestimmte Ziele ansteuern sollen, ohne dabei mit statischen oder dynamischen Hindernissen zu kollidieren. Zunächst wird ein Verfahren entwickelt mit welchem ein einzelner Agent unter Verwendung eines erlernten Umgebungsmodells befähigt wird, intuitiv die zukünftigen Positionen beweglicher Objekte vorherzusagen, um so Kollisionen zu vermeiden bzw. Objekte direkt anzusteuern. Es wird vorgeschlagen, die erlernte Positionsvorhersage beweglicher Objekte mit einer baumartigen Suche nach günstigen Folgezuständen, indem der Ausgang von Aktionsfolgen simuliert wird, zu kombinieren. Zur Steigerung der Effizienz wird, aufbauend auf dem vorherigen Ansatz, das Verfahren modifiziert und die Positionsvorhersagen dynamischer Objekte über zeitabhängige Kantenkosten in einen Graph integriert, um so mit angepassten, klassischen Pfadplanungsalgorithmen, wie dem A* Algorithmus, Pfade in einer Raum-Zeit-Dimension planen zu können, die den Kontakt mit dynamischen Hindernissen von vornherein meiden.

Außerdem wird in dieser Arbeit der Frage nachgegangen, wie der zum Routing benötigte Graph auch in sich dynamisch veränderten, kontinuierlichen Umgebungen aktuell gehalten werden kann, wozu auf eine Methode namens Stable Growing Neural Gas zurückgegriffen wird. Diese ist von der gleichmäßigen Ausbreitung von Gas-Molekülen im Raum inspiriert und diskretisiert auf diese Weise einen Raum. Weiter wird behandelt, wie der Graph parallel und kollisionsfrei von multiplen Agenten zum Routing verwendet werden kann, wofür die Verwendung eines Potential Field Ansatzes vorgeschlagen wird. Neben Detailverbesserungen an der Methode des Stable Growing Neural Gas, werden vor allem die Synergien erarbeitet, die sich aus der Kombination der Methoden des Stable Growing Neural Gas, dem Potential Field Ansatz und der Verwendung des A* Algorithmus ergeben.

Um eine ganze Gruppe von Agenten zu kontrollieren, wird ein auf Reinforcement Learning basierendes Verfahren vorgeschlagen, um Agenten auf einen virtuellen, dynamischen Zielpunkt zuzusteuern. Dieses zeichnet sich durch eine hohe Anpassungsfähigkeit aus. In einem entwickelten Szenario, in welchem Agenten durch Kollisionen untereinander implizit benachteiligt werden, konnte gezeigt werden, dass die Agenten durch das entworfene Trainingsverfahren gelernt haben Kollisionen untereinander zu vermeiden, ohne explizit darauf trainiert zu werden.

Um die Interaktion lernender Agenten weiter zu untersuchen, wird in einem umgedrehten Szenario eine Gruppe von Agenten mittels Reinforcement Learning darauf trainiert, einem auf sie zukommenden Objekt auszuweichen. Unter der Prämisse, dass sich dieses von mehreren potenziellen Zielen ablenken lässt, hat sich wie im vorhergehenden Szenario emergent ein Schwarmverhalten unter den Agenten entwickelt, was mit Methoden der Spieltheorie weiter untersucht wurde und bei der Untersuchung sozialer Interaktionen und Dilemmata von Bedeutung ist.

Abstract

This thesis deals with the pathfinding, decision-making and behavior analysis of intelligent agents when they are supposed to approach certain targets in continuous, two-dimensional areas without colliding with static or dynamic obstacles. First, a method is developed to enable a single agent, using a learned environment model, to intuitively predict the future positions of moving objects in order to avoid collisions or directly target objects. It is proposed to combine the learned position prediction of moving objects with a tree-like search for favorable subsequent states by simulating the outcome of action sequences. To increase efficiency, building on the previous approach, the method is modified to integrate the position predictions of dynamic objects into a graph via time-dependent edge costs, allowing adapted classical path planning algorithms, such as the A* algorithm, to plan paths in a space-time dimension that avoid contact with dynamic obstacles in the first place.

In addition, this work explores the question of how the graph needed for routing can be kept up-to-date in dynamically changing, continuous environments, relying on a method called Stable Growing Neural Gas. This method inspired by the uniform distribution of gas molecules in space and discretizes a space in this way. Further, it is addressed how the graph can be used in parallel and collision-free by multiple agents for routing, for which the use of a Potential Field approach is proposed. In addition to detail improvements of the Stable Growing Neural Gas method, the synergies resulting from the combination of the Stable Growing Neural Gas methods, the Potential Field approach and the use of the A* algorithm are discussed.

In order to control a whole group of agents, a reinforcement learning based method is proposed to steer agents towards a virtual dynamic target point. This is characterized by high adaptability. In a developed scenario, in which agents are implicitly penalized by collisions among each other, it could be shown that the agents learned to avoid collisions among each other by the designed training procedure without being explicitly trained for it.

To further investigate the interaction of learning agents, in a reversed scenario, a group of agents is trained to avoid an approaching object using reinforcement learning. Under the premise that this can be distracted from multiple potential targets, swarming behavior emerged among the agents, as in the previous scenario, which was further investigated using methods from game theory and is important in the study of social interactions and dilemmas.

Inhaltsverzeichnis

1 Einleitung

Im Alltagsleben bzw. in der Wirtschaft nimmt die Automatisierung immer weiter zu, bis hin zu komplett autonom agierenden Robotern oder Fahrzeugen. Die Anwendungsfälle für sich autonom bewegende und organisierende Entitäten, im Folgenden abstrakt als Agenten bezeichnet, sind vielfältig. So ist es in Szenarien der Industrie 4.0 denkbar, dass Werkteile von Robotern vollständig autonom in Industriehallen von einer Maschine zur anderen bewegt werden [140]. Dabei darf es natürlich nicht zu Kollisionen kommen, weder zwischen den Robotern und statischen Hindernissen, noch zwischen Robotern und dynamischen Hindernisse, wie z.B., wiederum anderen beweglichen Robotern. Dementsprechend sollen die autonom agierenden Roboter dynamische Hindernisse bei ihrer Wegfindung einbeziehen und sich an geänderte Gegebenheiten anpassen. Solche Szenarien sind aber nicht nur in industriellen Anwendungen relevant, sondern auch für bereits gebräuchliche Staubsaug- oder Rasenmähroboter [71], in Computerspielen für Nicht-Spieler-Charaktere [107] aber auch bei Service Robotern, wie sie in Krankenhäusern, Altenheimen oder Geschäften zum Einsatz kommen könnten [76, 47].

Deshalb werden im Rahmen dieser Arbeit verschiedene Konzepte vorgestellt, wie autonome Agenten auf Freiflächen navigieren und dabei mit Hilfe von Methoden des maschinellen Lernens intuitiv Kollisionen aus dem Weg gehen können. Zudem wird darauf eingegangen, wie Freiflächen von vorneherein anpassungsfähig diskretisiert werden können, um sie algorithmisch verarbeitbar zu machen, und es wird ein ganzheitlicher Ansatz vorgestellt, bei dem verschiedene Methoden nutzbringend miteinander verbunden werden, um multiple Agenten kollisionsfrei zu navigieren. Die Arbeit endet mit der Steuerung ganzer Schwärme aus Agenten auf der Basis eines flexiblen Trainingsverfahrens und der Analyse emergenter Verhaltensweisen, welche aus der Interaktion autonom agierenden Agenten resultieren.

1.1 Motivation und Zielsetzung dieser Arbeit

In der vorliegenden Arbeit wird ein neuartiger Ansatz beschrieben, bei welchem Agenten mittels sogenannter intuitiver Physik Bewegungen von Objekten in ihrer Umgebung vorhersagen bzw. abschätzen und dadurch in ihre Wegplanung einbeziehen können. Bei intuitiver Physik handelt es sich um ein Themengebiet des maschinellen Lernens, bei welchem rein auf der Grundlage von Beobachtungen Modelle für physikalische Abläufe in der Umwelt auf makroskopischer Ebene gebildet werden. So konnten mit dieser Technik z.B. rein auf der Grund-

lage von Beispielen Modelle zur Vorhersage der Konfiguration eines Billardtisches nach einem Stoß oder zur Stabilität von Blocktürmen nach dem Auflegen eines weiteren Holzklotzes gebildet werden. Als Methodik wird auf künstliche neuronale Netze zurückgegriffen. Das Themengebiet ist an die Beobachtung angelehnt, dass während der Entwicklung eines Menschen, dieser rein auf der Basis von Beobachtungen bzw. Erfahrung lernt, z.B. die zukünftige Flugbahn eines geworfenen oder abprallenden Balles vorherzusagen. Im Verlauf dieser Arbeit wird ein Ansatz vorgeschlagen, welche diese Art der Modellbildung auf Grundlage von künstlichen neuronalen Netzen verwendet, um einen intelligenten Agenten mit einem Konzept über die Dynamiken seiner Umgebung auszustatten, welches er für die Planung seiner Wege verwenden kann. Diese Modelle können sehr schnell ausgewertet werden (im Hinblick auf die Rechenzeit), um Vorhersagen über Ortsänderungen oder Objektinteraktionen zu erhalten und sind trotzdem sehr flexibel in der Anpassung an geänderte Umgebungsbedingungen. Zunächst wird ein Verfahren vorgeschlagen, bei dem ein Agent durch die Simulation von Aktionsfolgen versucht, für ihn günstige Folgezustände zu finden, und die am erfolgversprechendste Abfolge von Aktionen dann entsprechend ausführt. Zur Erhöhung der Effizienz wird danach darauf eingegangen, wie die Vorhersagen des intuitiv erlernten Modells in einen Graphen integriert werden können, um so mit klassischen aber für diesen Zweck modifizierten Pfadplanungsalgorithmen, wie dem Dijkstra oder A* Algorithmus, Wege auf Freiflächen planen zu können.

Es werden in der Arbeit aber nicht nur Szenarien mit einzelnen, sondern auch mit multiplen autonomen Entitäten betrachtet. Dazu wird ein ganzheitlicher Ansatz vorgeschlagen, wie das Routing auf Freiflächen für multiple autonome Roboter geschehen kann.

Wie bereits erwähnt, werden Freiflächen zur effizienten Wegplanung in der Regel zunächst quantisiert, da gängige Algorithmen zur Pfadplanung auf Graphstrukturen arbeiten. Hierfür wurde auf ein Verfahren namens Stable Growing Neural Gas zurückgegriffen. Dieses zeichnet sich durch hohe Effizienz und hohe Anpassungsfähigkeit an geänderte Umgebungsbedingungen aus. Es wird gezeigt, wie der resultierende Graph gemeinsam von multiplen Entitäten kollisionsfrei verwendet werden kann und welche Vorteile sich gegenüber bestehenden Verfahren ergeben.

Soll eine große Anzahl an Agenten gesteuert werden, bietet es sich an, diese als Schwarm aufzufassen und als ein Ganzes zu steuern. In diesem Zusammenhang wurde untersucht, wie sich die einzelnen Entitäten zu einem Schwarm zusammenfassen lassen. Als Methodik zeigte sich Reinforcement Learning als besonders erfolgversprechend, erneut mit dem Ziel einer hohen Adaptivität. Es wird gezeigt, dass sich natürliches Schwarmverhalten erzeugen lässt, dass die einzelnen Entitäten darauf trainiert werden, die Distanz zu einem Zielpunkt bzw., von der Biologie inspiriert, zu einer gedachten virtuellen Futterquelle zu minimieren.

Ausgehend von diesen Betrachtungen wurde weiterhin untersucht, welche Ei-

genschaften einen Schwarm ausmachen und welche weiteren Beweggründe für die Bildung von Schwärmen bzw. Herden existieren. Dabei wurde herausgefunden, dass sich natürlich wirkende Schwärme dadurch bilden lassen, das multiple Agenten mittels Reinforcement Learning darauf trainiert werden, möglichst lange einem Fressfeind auszuweichen. Dabei handelt jeder Agent egoistisch, das heißt, er ist nur an seinem eigenen Überleben interessiert. Dennoch wurde gelernt, sich zu einer Gruppe zusammenzuschließen, da sich der Angreifer von mehreren möglichen Beutetieren ablenken lässt und dies das Überleben des Einzelnen verlängert. Empirisch konnte in diesem Szenario gezeigt werden, dass das Schwarmverhalten ein Nash Equilibrium besitzt, da ein Einzelner keinen Anreiz hat, von seiner Strategie abzuweichen und sich auf die Freifläche zu begeben, während alle anderen Agenten weiterhin die Schwarmbildung verfolgen. Daraus folgt, dass ein Schwarm ein selbsterhaltendes Konstrukt ist. Die angestellten Untersuchungen zu Schwärmen und deren Eigenschaften könnten z.B. in nachgelagerten Betrachtungen zu Evakuierungsszenarien oder dem Management ganzer autonomer Fahrzeugflotten Anwendung finden.

1.2 Aufbau dieser Arbeit

Im Folgenden wird der grundlegende Aufbau der vorliegenden Arbeit vorgestellt.

Im Kapitel 2 werden die benötigten Grundlagen für diese Arbeit erläutert. Dabei werden zunächst grundlegende Begriffe, wie der eines Graphen definiert und elementare Algorithmen zur Pfadplanung, zur Diskretisierung von Freiflächen und zur Kollisionsvermeidung erläutert. Zudem wird auch der Begriff eines intelligenten Agenten eingeführt und es werden grundlegende, sowie spezielle, künstliche neuronale Netze bis hin zu verschiedenen Reinforcement Learning Algorithmen beleuchtet.

Im daran anschließenden Kapitel 3 werden diese Grundlagen dann verwendet, um zwei Ansätze zu entwickeln, bei welchen Agenten, die mit einem künstlichen neuronalen Netz als Modell ihrer Umgebung ausgestattet sind, dieses Modell zur Planung von Pfaden auf einer frei begehbaren Fläche nutzen, um entweder Kollisionen mit sich dynamisch bewegenden Hindernissen zu vermeiden oder wiederum sich dynamisch bewegende Objekte gezielt anzusteuern. Im ersten Ansatz simuliert ein Agent mit dem ihm zur Verfügung stehenden Bewegungsmodell explizit zukünftige auf seine Aktionen folgende Zustände und wählt aus diesen Simulationen die für ihn günstigste Handlungsfolge. Im zweiten Ansatz werden die Vorhersagen des Bewegungsmodells über zeitabhängige Kantenkosten in einen Graph integriert, um auf diesem mit klassischen Algorithmen Pfadplanung zu betreiben, welche Kollisionen von vornherein vermeidet.

In Kapitel 4 wird ein Ansatz vorgeschlagen, bei welchem sich multiple Agenten effizient auf einer frei begehbaren Fläche bewegen können, ohne miteinander oder mit anderen Hindernissen zu kollidieren. Dieser besteht aus drei

Methoden, die miteinander kombiniert werden. Dies wäre zum einen die Überführung der Freifläche in einen Graph mittels einer Technik namens Stable Growing Neural Gas, um danach auf diesem Graph mittels etablierter Methoden wie dem Dijkstra oder dem A* Algorithmus die Wegplanung zu betreiben. Kollisionen zwischen den verschiedenen Agenten und gegebenenfalls weiteren Hindernissen werden mit Hilfe eines Potential Field Ansatzes verhindert. Durch die Kombination der genannten Methoden ergeben sich Synergien, welche ebenfalls in Kapitel 4 elaboriert werden.

Kapitel 5 stellt einen neuen Ansatz vor, mit welchem eine Gruppe von Agenten als Ganzes, das heißt als Schwarm, gesteuert wird. Dazu wurden Agenten, von der Biologie inspiriert, mittels Multi-Agent Reinforcement Learning auf einer kontinuierlichen, frei begehbaren, zweidimensionalen Fläche darauf trainiert, ein gedachtes Beutetier zu verfolgen bzw. die Distanz zu diesem zu minimieren. Das resultierende Verhalten wurde mit einem etablierten Ansatz zur simulativen Erzeugung von Schwärmen, mit welchem es Ähnlichkeiten aufweist, aber auch mit anderen Verhaltensstrategien verglichen. Der etablierte Ansatz namens Boids umfasst drei Regeln, welche ein Individuum auf lokaler Ebene befolgen muss, damit sich natürlich wirkendes Schwarmverhalten ergibt, ohne dass dieses explizit forciert wird. Dass das in Kapitel 5 beschriebene Vorgehen ähnliches Schwarmverhalten erzeugt wie der etablierte Ansatz, ebenfalls ohne dieses explizit zu erzwingen, liefert einen Ansatzpunkt zur Erklärung, warum die relativ abstrakten Boids-Regeln überhaupt Schwärme erzeugen bzw. gibt einen Einblick in die Beschaffenheit von Schwärmen allgemein. Zudem könnte das gewonnene und in Kapitel 5 beschriebene Modell für die weitere Untersuchung von z.B. Evakuierungsszenarien von Bedeutung sein.

Kapitel 6 ist ebenfalls von der Biologie inspiriert und untersucht einen ähnlichen Sachverhalt wie Kapitel 5. Allerdings wird das Setting aus Kapitel 5 hier herumgedreht und soll generelles Schwarmverhalten in Multiagentensystemen weiter untersuchen. Zugrunde liegt die Annahme, dass ein Angreifer, der sich einem Schwarm von Beutetieren nähert, von der schieren Menge möglicher Ziele abgelenkt wird und damit der Schwarm den einzelnen Individuen eine gewisse Art von Schutz bietet bzw. deren Überlebenschance erhöht. Aufbauend auf dieser Annahme wird in Kapitel 6 untersucht, ob egoistisch handelnde Agenten im Beisein eines Räubers mit der genannten Eigenschaft lernen einen Schwarm zu bilden, wenn sie mittels Multi-Agent Reinforcement Learning darauf trainiert werden, möglichst lange zu überleben. In dem Szenario wird also das Überleben belohnt bzw. das Gefangenwerden bestraft aber nicht explizit die Schwarmbildung gefördert. Das resultierende, erlernte Verhalten gleicht wiederum jenen welches sich aus den anerkannten, statischen drei Schwarmregeln des Boids Algorithmus ergibt und liefert damit weitere Erkenntnisse über die Beschaffenheit von Schwärmen. Es liefert aber auch ein weiteres Beispiel für emergentes Verhalten, also ein unvorhergesehenes Zusammenspiel multipler Agenten, die mittels Reinforcement Learning auf ein gewisses Ziel hin trainiert wurden und eine Strategie entwickeln, die zwar versucht, die Ziel- bzw. Be-

lohnungsfunktion zu maximieren aber nicht dem vom Entwickler intendierten Verhalten entspricht. Das Kapitel liefert aber Einblicke in soziale Dilemmata, weil z.B. ein Agent keinen Anreiz bzw. keine Möglichkeit hat, von der Schwarmstrategie abzuweichen, wenn alle anderen daran festhalten, da er sich dann potentiell alleine auf die Freifläche begibt, was die Wahrscheinlichkeit erhöht, vom Angreifer als Ziel ausgewählt zu werden. Deshalb enthält Kapitel 6 zusätzlich eine Untersuchung zu Nash-Gleichgewichten im vorgeschlagenen Schwarm-Setting.

Abschließend fasst Kapitel 7 die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf sich daraus ergebende Herausforderungen und zukünftige Forschungsfragen.

1.3 Zugrundeliegende Vorarbeiten

Ein Großteil der Inhalte dieser Arbeit wurde bereits im Rahmen von Beiträgen zu internationalen Fachkonferenzen vorveröffentlicht. Dieser Abschnitt soll dazu dienen, den Inhalt der jeweiligen Artikel zu umreißen und den Eigenanteil des Autors der vorliegenden Arbeit (im Folgenden nur noch als „Autor“ bezeichnet) an den bereits publizierten Inhalten aufzuschlüsseln. Zudem wird ebenfalls darauf eingegangen, welche Teile der vorangegangenen Publikationen in die vorliegende Arbeit übernommen wurden und welche Inhalte der vorliegenden Arbeit über die bereits publizierten hinausgehen. Zusätzlich zu der Übersicht hier, wird zu Beginn eines jeden Inhaltskapitels, d.h. in 3.1, 4.1, 5.1 und 6.1 jeweils noch einmal eine detaillierte Übersicht über die einzelnen, bereits veröffentlichten Inhaltsteile gegeben.

Für alle im Folgenden aufgelisteten Veröffentlichungen gilt, dass Prof. Dr. Claudia Linnhoff-Popien – als Lehrstuhlinhaberin und Doktormutter des Autors – stets beratend als Kritikgeberin und mit wertvollen Gedanken, insbesondere in den Arbeiten, bei denen sie als Autorin aufgeführt ist, mitgewirkt hat.

Intuitive Pathfinding of Autonomous Agents [52]

Diese Arbeit beruht auf einer nicht publizierten Voruntersuchung des Autors aus dem Feld der sogenannten intuitiven Physik. In diesem werden anhand von Beispielen Modelle trainiert, die dann rein auf der Grundlage von Erfahrungen in der Lage sind physikalische Zusammenhänge auf makroskopischer Ebene vorherzusagen. So ist es mit Machine Learning Techniken, insbesondere fortgeschrittenen, künstlichen neuronalen Netzen möglich, z.B. das Abprallverhalten von elastischen Körpern vorherzusagen, selbst wenn sie Einflüssen wie Trägheit, Reibung oder Gravitation unterliegen, ohne dafür ein exaktes physikalisches Modell zu benötigen. Geht man davon aus, dass sich mit diesen Techniken Trajektorien von Objekten hinreichend genau vorhersagen lassen,

lässt sich diese Information verwenden, um sie in die Pfadplanung von autonomen Agenten zu integrieren. Die Idee, Ansätze der intuitiven Physik mit dem Planungsprozess autonomer Agenten zu verbinden stammt vom Autor dieser Arbeit. Sie ist inspiriert von Gesprächen zwischen dem Autor und Lenz Belzner zur Anwendung intuitiver Physik in Industrieprojekten und Arbeiten von Lenz Belzner zu zukunftsgerichteten, simulativen Planungsprozessen autonome Agenten mit bestimmten Zeithorizont. Die genannte Publikation umreißt, wie Ansätze der intuitiven Physik und Planungsprozesse autonome Agenten miteinander zu verbinden wären und welche Vorteile sich ergeben. Sie wurde vom Autor eigenständig und allein verfasst. Das in der genannten Veröffentlichung vorgestellte Konzept zur Verwendung künstlicher neuronaler Netze im Planungsprozess intelligenter Agenten ist in Kapitel 3 dieser Arbeit eingeflossen. Darüber hinaus sind in Kapitel 3 aber auch bisher nicht veröffentlichte Ergebnisse der vorangegangenen Untersuchungen zum Themengebiet der intuitiven Physik, genauer, zur Unsicherheit des künstlichen neuronalen Netzes über die Vorhersage zukünftiger Trajektorienpunkte, dargestellt.

Collision Avoidance using Intuitive Physics [53]

Diese Arbeit setzt auf die zuvor genannten Publikationen des Autors auf, darüber hinaus wird in ihr das zuvor beschriebene Konzept umgesetzt und evaluiert. Dazu wurde eine spezielle Form von künstlichen neuronalen Netzen namens LSTM mithilfe einer Physiksimulation darauf trainiert, den Ausgang von physikalischen Interaktionen zwischen Objekten, das heißt, deren zukünftige Position in diskreten Zeitschritten, vorherzusagen. Ein intelligenter Agent, bei dem davon ausgegangen wird, dass er über ein Modell seiner Umgebung verfügt, nutzt dieses, um seine zukünftigen Schritte zu planen. Genauer, durchsucht der Agent den Raum zukünftiger Zustände, indem er anhand des ihm zur Verfügung stehenden Modells den Ausgang von Aktionsfolgen simuliert, um so für ihn günstige zukünftige Zustände zu finden. Bei dieser Suche wird vom Agent ausgehend von seinem aktuellen Zustand ein Baum möglicher Folgezustände aufgespannt. Im konkret untersuchten Szenario bedeutet dies, dass der Agent auf seinem Weg mit gewissen sich dynamisch bewegenden Zielpunkten kollidieren kann bzw. dies mit anderen ebenfalls beweglichen Hindernissen vermeiden soll. Zur Evaluation werden die Performance-Werte miteinander verglichen, wenn der Agent über ein exaktes physikalisches oder ein auf trainierten künstlichen neuronalen Netzen beruhendes Modell verfügt. Die Erstellung des Konzepts, dessen Umsetzung und die Evaluation wurden vollständig vom Autor durchgeführt. Sebastian Feld stand bei der Evaluation mit hilfreichen Diskussionsbeiträgen zu Experimenten bzw. deren Ausrichtung beratend zur Seite. Zudem hat er bei der Erstellung der schriftlichen Arbeit und deren Strukturierung unterstützt. Die Inhalte und Ergebnisse der genannten Publikation sind vollständig in Kapitel 3 dieser Arbeit eingeflossen.

Predictive Collision Management for Time and Risk Dependent Path Planning [54, 55]

Die Publikation [54] bzw. deren erweiterte Version [55] basieren auf der Vorarbeit des Autors zur Abschätzung von Objektbewegung mittels intuitiver Physik und einer darauf aufbauenden Bachelorarbeit. Dabei ging es darum, den zuvor beschriebenen baumartigen Suchprozess durch den Raum möglicher Folgezustände durch konventionelle Pfadplanung auf Graphen zu ersetzen. Dementsprechend ging es darum, die vorhergesagten Objektbewegungen in den Graphen zu integrieren, damit sie bei der Pfadplanung Beachtung finden können. Dazu wurde vom Autor die dynamische Anpassung der Kantengewichte des Graphen vorgeschlagen. Der Verfasser der Bachelorarbeit, Hannes Schroter, legte besonderes Augenmerk auf das zeitliche Zusammenspiel zwischen Agenten- und Objektbewegungen. In Diskussion mit dem Autor entstand so ein System, bei welchem die Vorhersagen wahlweise eines Regressions- oder Klassifikationsmodells über eine zeitabhängige Kostenfunktion der Kanten des Graphen in diesen integriert werden. Dadurch ist es mittels eines leicht modifizierten, konventionellen Pfadplanungsalgorithmus möglich, Kollisionen auf einer globalen Ebene zu vermeiden, da Kollisionsobjekten schon während der Pfadplanung aus dem Weg gegangen werden kann.

Auf den Autor entfallen die Ideen der intuitiven Vorhersage von Objektbewegungen mittels adaptiver künstlicher neuronale Netze und der Integration von deren Bewegungsvorhersagen in den Graph über die Kantenkosten. Hannes Schroter übernahm die Recherche von Vorarbeiten zu zeitabhängigen Kantenkosten, zugehöriger Pfadplanungsalgorithmen und die Implementierung der Idee.

Basierend auf den Ergebnissen der Bachelorarbeit übernahm der Autor dieser Arbeit die Erstellung der Publikation, wobei Sebastian Feld bei der Strukturierung und Überarbeitung zur Seite stand. Die Inhalte und Ergebnisse der genannten Publikation sind vollständig in Kapitel 3 dieser Arbeit eingeflossen.

Dynamic Path Planning with Stable Growing Neural Gas [56]

Diese Publikation basiert zum Teil auf einer Bachelorarbeit die vom Autor dieser Arbeit ausgeschrieben wurde, um Limitationen der zuvor genannten Veröffentlichungen zu überwinden.

Die Anforderung, dass der Planungsprozess des Agenten auf einer frei begehbaren Fläche stattfindet, auf welcher sich auch dynamisch bewegende Hindernisse befinden können, blieb analog zur vorgenannten Veröffentlichung bestehen. Dementsprechend musste zunächst ein Ansatz bestimmt bzw. entworfen werden, der die Freifläche in einen Graph für die Pfadplanung überführt. Zudem musste ein geeigneter Pfadplanungsalgorithmus, der auf Graphen anwendbar ist, gewählt werden. Als dritten Teil der Lösung ging es darum, eine

geeignete Form der Kollisionsvermeidung zu integrieren.

Für die Diskretisierung wurde ein Ansatz namens Stable Growing Neural Gas (SGNG) gewählt, der sich fortlaufend ausführen lässt, ohne an Effizienz zu verlieren und somit den Graph auch über die Zeit an veränderliche Räume anpassen kann. Zur Pfadplanung auf dem resultierenden Graph kamen die allgemein anerkannten Dijkstra bzw. A* Algorithmen zum Einsatz. Für diese ergeben sich allerdings noch Verbesserungen wenn sie im Zusammenhang mit SGNG verwendet werden, was in der Publikation herausgearbeitet wird. Zur Kollisionsvermeidung wurde im Gegensatz zur vorherigen beschriebenen Publikation nicht ein Ausweichverfahren auf der Grundlage der Vorhersage von künstlichen neuronalen Netzen verwendet, sondern es kam ein sogenannter Potential Field Ansatz zum Einsatz. Der Vorteil an dieser Technik ist, dass sie sich leicht mit den vorher gewählten Methoden verbinden lässt und, dass sowohl dynamische Hindernisse als auch andere Agenten relativ simpel als Kollisionsobjekte zueinander betrachtet werden können und sich so Kollisionen zwischen ihnen auf einer lokalen Ebene vermeiden lassen. Der Beitrag dieser Veröffentlichung besteht darin, dass die drei genannten Methoden in dieser Art noch nicht miteinander kombiniert wurden und sich bei ihrer Kombination Detailverbesserungen ergeben, die in der Publikation dargestellt werden.

Die Aufgaben- bzw. Problemstellung der Bachelorarbeit wurde vom Autor dieser Arbeit vorgeschlagen. Die Diskussion möglicher Lösungskomponenten und deren Auswahl erfolgten im Dialog zwischen dem Autor und Manuel Zierl, als Verfasser der Bachelorarbeit. Der experimentelle Vergleich mit verwandten Arbeiten, der für die aus der Bachelorarbeit resultierende Veröffentlichung essentiell war, wurde vor allem vom Autor dieser Arbeit durchgeführt. Bei der Erstellung des Manuskripts zur Veröffentlichung und auch zum Teil bei der Betreuung der Bachelorarbeit stand Sebastian Feld beratend zur Seite. Die Erstellung des Textes erfolgte etwa zu gleichen Anteilen zwischen dem Autor und Manuel Zierl. Sebastian Feld war durch Diskussionen über die Strukturierung von diesem involviert. Die Inhalte der hier beschriebenen Veröffentlichung sind vollständig im Kapitel 4 dieser Arbeit eingeflossen.

Emergent Escape-based Flocking Behavior using Multi-Agent Reinforcement Learning [60]

Diese Publikation ist von der Biologie inspiriert, genauer von Fisch- bzw. Vogelschwärmen. In einer Unterhaltung unter Kollegen warf Thomy Phan die Frage auf, ob Agenten, die mittels Reinforcement Learning darauf trainiert werden, im Beisein eines Räubers möglichst lange zu überleben, einen Schwarm bilden würden. Dies beruht auf der Annahme, dass die schiere Anzahl an möglichen Beutetieren das Raubtier verwirrt, so dass es nur schwierig ein einzelnes herauspicken kann, was wiederum die Überlebenschance eines Beutetieres erhöht. Diese Fragestellung bildet den Anstoß für die hier aufgeführte Publikation [60]. Um die Fragestellung zu untersuchen, wurde vom Autor zunächst eine

Simulationsumgebung implementiert, in der ein Räuber nach statischen Verhaltensregeln aus den in seiner Umgebung befindlichen Agenten einen zufällig auswählt und ihn für eine gewisse Zeit folgt, bis er den Agent fängt oder eine erneute zufällige Wahl eines Ziels aus der Umgebung als Beutetier erfolgt. Dies soll simulieren, dass sich der Räuber von mehreren Agenten in seiner näheren Umgebung ablenken bzw. verwirren lässt. Ein Agent gilt als gefangen, wenn er mit dem Räuber kollidiert. Die Agenten können sich frei in der Umgebung bewegen, da es keine Hindernisse gibt, Kollisionen untereinander nicht behandelt werden und sie auch nicht von Wänden gestoppt werden (der Raum hat die Eigenschaft eines Torus, so dass Agenten, die ihn verlassen würden, wieder unmittelbar auf der gegenüberliegenden Seite eintreten). Die Fragestellung selbst aber auch der kontinuierliche Zustandsraum unterscheidet die Publikation von verwandten Arbeiten. Parallel zur Entwicklung der Umgebung wurde ein Verfahren zum Training multipler autonome Agenten ausgewählt und integriert. Der Trainingsprozess und die Umgebung wurden so aufeinander abgestimmt, dass die Vermutung der Schwarmbildung im Beisein eines Räubers unter den gegebenen Bedingungen bestätigt werden konnte. Zur Evaluation wurden die aus dem Lernprozess resultierenden Schwärme mit anderen Verhaltensstrategien, unter anderem einen etablierten Ansatz zur algorithmischen Schwarmgenerierung, verglichen Während des Entwicklung und Evaluationsprozess standen Thomy Phan, Thomas Gabor und Lenz Belzner als Diskussionspartner beratend zur Seite. Die beiden erstgenannten waren auch an der Erstellung des Manuskriptes zur Veröffentlichung beteiligt. Die Ergebnisse der genannten Veröffentlichung bilden einen Teil des Kapitels 6 dieser Arbeit, wo sie mit den Inhalten der nachfolgend beschriebenen Publikation verknüpft und in Beziehung gesetzt werden.

Nash Equilibria in Multi-Agent Swarms [59]

Aus der vorweg beschriebenen Arbeit entstammt die Beobachtung, dass Agenten, die mittels Reinforcement Learning darauf trainiert werden, im Beisein eines Räubers möglichst lange zu überleben, unter gewissen Bedingungen dazu neigen, einen Schwarm zu bilden, um ihre jeweilige Überlebenschance zu erhöhen (ohne dass die Schwarmbildung explizit forciert wird). Sie hat aber auch gezeigt, dass dieses Schwarmverhalten nicht immer die optimale Strategie sein muss, da andere Verhaltensweisen dem einzelnen Agenten eine längere Überlebenszeit versprechen. Während einer Diskussion kam die Frage auf, ob dies mit Nash-Gleichgewichten in Schwärmen in Verbindung stehen könnte, so dass der Reinforcement Learning Algorithmus unter Umständen in diesem „lokalen Optima“ hängen bleibt. Der Autor ist dieser Fragestellungen nachgegangen und hat das zuvor beschriebene Szenario so verändert, dass sich Agenten, die verschiedene Strategien verfolgen, gemeinsam in der Umgebung befinden und miteinander interagieren können, genauer, solche Agenten, die durch Reinforcement Learning gelernt haben einen Schwarm zu bilden und solche,

die das Verhalten der anderen Agenten nicht einbeziehen und sich unabhängig von diesen vom Räuber entfernen, um zu überleben. Die durchschnittliche Überlebenszeit der Agenten in den einzelnen Gruppen wurde als Auszahlung eines Normalformspiels im Sinne der Spieltheorie interpretiert, bei dem sich die Agenten für eine der Strategien schwärmen oder nicht schwärmen entscheiden. Dadurch konnte gezeigt werden, dass Schwarmverhalten im Sinne der Überlebenszeit in diesem Setting sehr wohl ein Nash-Gleichgewicht darstellt, da ein einzelner Agent keinen Anreiz hat vom Schwarmverhalten abzuweichen und sich auf die unsichere Freifläche zu begeben, während andere Agenten weiterhin an der Schwarmstrategie festhalten. Es konnten aber auch weniger offensichtliche Nash Gleichgewichte gefunden werden, die vom Mischungsverhältnis der schwärmenden und nicht schwärmenden Agenten, der Gesamtanzahl der Agenten und der ihnen zur Verfügung stehenden Fläche abhängen. Die Implementierung des Settings und die Konzeption der Experimente wurden vom Autor durchgeführt. Thomy Phan, Sebastian Feld und Thomas Gabor waren über Diskussionen und bei der Ausarbeitung der Publikation beteiligt. Christoph Roch steuerte Beiträge zur spieltheoretischen Modellierung bei, Fabian Ritz bereicherte die Veröffentlichung durch Beiträge zu verwandten Arbeiten und sozialen Dilemmata und Andreas Sedlmeier arbeitete letzte Korrekturen und Anmerkungen in die Veröffentlichung ein. Der Inhalt der Veröffentlichung ist in Kapitel 6 dieser Arbeit eingeflossen, wo er mit den Ergebnissen der vorangegangenen Publikation in Beziehung gesetzt wird.

Foraging Swarms using Multi-Agent Reinforcement Learning [61]

In dieser auf den vorher beschriebenen Publikationen basierenden Arbeit wird der Frage nachgegangen, ob sich Schwärme aus lernenden Agenten mittels Reinforcement Learning auch auf Grundlage anderer Kriterien bilden, als die erhöhte Überlebenschance in Anwesenheit eines Räubers. Diese Fragestellung ergibt sich relativ natürlich aus den zwei vorweg beschriebenen Veröffentlichungen, und wurde auf diese aufbauend hauptsächlich vom Autor dieser Arbeit vorangetrieben. Konkret, sollte in einer ausgeschriebenen Bachelorarbeit untersucht werden, ob sich das Räuber-Beute Szenario der vorangegangenen Publikationen umdrehen lässt, so dass mehrere autonome Agenten bildlich einem virtuellen Futterpunkt folgen und sofern dies positiv ist, betrachtet werden, inwieweit die Formation der sich bewegenden Agenten gewisse Schwarmeigenschaften erfüllt. Die Bachelorarbeit basierte stark auf den vorangegangenen Veröffentlichungen, weshalb der bearbeitenden Studentin, Paula Wikidal, auch der im Rahmen von diesen erstellte Quellcode der Simulationsumgebung zur Verfügung gestellt wurde. Ihre Aufgabe war es dementsprechend ein zur Aufgabenstellung passendes Trainingsverfahren zu entwickeln und das resultierende Verhalten der Agenten zu evaluieren bzw. zu analysieren. Bei der Bearbeitung standen der Autor dieser Arbeit und Fabian Ritz als Diskussions-

partner und Betreuer der Bachelorarbeit beratend zur Seite. Die Vermutung der Schwarmbildung durch Agenten die mittels Reinforcement Learning in einem kontinuierlichen Zustandsraum auf eine Art Futtersuche trainiert wurden, konnte durch die Bachelorarbeit bestätigt werden, wobei die Herangehensweise und die abschließenden Experimente stark an die vorangegangenen Publikationen des Autors angelehnt waren. Dadurch konnte eine Brücke von anderen Arbeiten des Autors über die Wegfindung einzelner und multipler Agenten hin zur Steuerung ganzer Schwärme bzw. zur Verhaltensanalyse in Multiagentensystemen geschlagen werden. Bei der Erstellung der aus der Bachelorarbeit resultierenden Publikation waren Thomy Phan und Thomas Gabor als Diskussionspartner beteiligt, steuerten neben Fabian Ritz aber auch Teile des Textes bei. Der Großteil des Textkörpers der Publikation entfällt auf Paula Wikidal und den Autor dieser Arbeit, wobei der Autor federführend bei der Erstellung der Publikation war. Die Inhalte und Ergebnisse der Publikation sind zugleich auch die Kerninhalte von Kapitel 5.

Im Folgenden sollen noch einmal kurz die Kerninhalte der Vorarbeiten den einzelnen Inhaltskapiteln zugeordnet werden. Dabei wird auch angegeben, welche Tabellen und Abbildungen bereits in den Veröffentlichungen enthalten waren.

Die Kernidee des dritten Kapitels, Methoden der intuitiven Physik für die vorausschauende Pfadplanung von autonomen Agenten zu verwenden wurde bereits in [52] publiziert. In [53] wurde dieses Konzept dann weiter ausgebaut, implementiert und evaluiert. Die Integration der Vorhersagen in einen zeitabhängigen Graph zur kollisionsvermeidenden Pfadplanung wurde in [54] publiziert. Diese Inhalte sind ebenfalls in Kapitel 3 eingeflossen. Die Tabelle 3.1 sowie die Abbildungen 3.1, 3.4, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12 und 3.13 waren bereits in [53] enthalten. Die Abbildungen 3.18, 3.21, 3.25, 3.26, 3.30a und 3.30b wurden bereits in [54] veröffentlicht bzw. wurden die Abbildungen 3.14, 3.15, 3.16, 3.17, 3.19, 3.20, 3.22a, 3.22b, 3.23, 3.24, 3.27, 3.28a, 3.28b, 3.29a, 3.29b, 3.31 und 3.32 sowie die Tabellen 3.2 3.3 3.4 3.5 3.6 3.7 in der erweiterten Version der Veröffentlichung [55] publiziert.

Die Kerninhalte von Kapitel 4, das heißt, die Diskretisierung eines Raumes mit Stable Growing Neural Gas, der Pfadplanung auf dem sich ergebenden Graphen mit Hilfe des Dijkstra bzw. A* Algorithmus' und der Kollisionsvermeidung unter Verwendung eines Potential Field Ansatzes sowie die Evaluation der Kombination dieser Methoden, wurde bereits in [56] vom Autor publiziert. Die mit dem Kapitel verbundenen Algorithmen 3 und 4 sowie die Abbildungen 4.2a, 4.2b, 4.3, 4.4a, 4.4b, 4.6, 4.7, 4.8a, 4.8b, 4.9, 4.10, 4.11, 4.12 und 4.13 waren bereits in [56] enthalten.

Die Kerninhalte von Kapitel 5 zur Steuerung eines ganzen Schwarms autonome Agenten, indem diese von der Biologie inspiriert mittels Multi-Agent Reinforcement Learning darauf trainiert werden einem gedachten Futterpunkt zu folgen wurden in zuvor vom Autor in [61] veröffentlicht. Die damit verbundenen Tabellen 5.1, 5.2 sowie die Abbildungen 5.2, 5.3a, 5.3b, 5.3c, 5.4, 5.5,

5.6a, 5.6b, 5.6c und 5.6 waren ebenfalls bereits in [61] enthalten. Die Tabelle 5.3 und die Abbildungen 5.1a und 5.1b sind zur besseren Verständlichkeit und Darstellung in dieser Arbeit hinzugekommen

In Kapitel 6 sind die Publikationen [60] und [59] eingeflossen. Die Veröffentlichung [60] behandelt die Frage, ob egoistisch handelnde Agenten im Beisein eines gedachten Raubtieres einen Schwarm bilden würden, um ihre eigene Überlebenschance zu erhöhen. Daran schloss sich direkt eine Untersuchung zu Nash-Gleichgewichten in Schwärmen an, mit der Idee, dass ein Agent keinen Anreiz hat von der Schwarmstrategie abzuweichen, wenn alle anderen an dieser festhalten und in einer Gruppe zusammen bleiben, welche in [59] veröffentlicht wurde. Die Tabelle 6.1 sowie die Abbildungen 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10 und 6.11 waren bereits in [60] enthalten. Die Tabelle 6.2 und die Abbildungen 6.12, 6.13, 6.14 sind bereits in [59] zu finden.

Die Publikationen [51, 57, 58], bei welchen der Autor federführend beteiligt war sowie die Publikationen [37, 38, 123, 145], bei welchen dieser als Koautor beteiligt war, sind inhaltlich kein Bestandteil der vorliegenden Arbeit.

2 Grundlagen der Pfadplanung mit Hilfe maschinellen Lernens

In diesem Kapitel sollen die Grundlagen für das Verständnis der weiteren Arbeit geschaffen werden, weshalb auf grundlegende verwandte Arbeiten und Methoden eingegangen wird. Dazu wird im Abschnitt 2.1 zunächst auf den Begriff eines intelligenten Agenten eingegangen, welcher insbesondere für Kapitel 3 von Bedeutung ist, in welchen eben solche Agenten ihre zukünftigen Handlungen planen. Danach wird in Abschnitt 2.2 auf grundlegende Pfadplanungsalgorithmen auf der Basis von Graphen eingegangen, bevor in Abschnitt 2.3 erläutert wird, wie Freiflächen zu Graphen diskretisiert werden können, um darauf Pfadplanung zu betreiben. Abschnitt 2.4 befasst sich mit grundlegenden Methoden der Kollisionserkennung und -vermeidung. Die in dieser Arbeit zum Einsatz kommenden künstlichen neuronalen Netze werden im Abschnitt 2.5 beleuchtet. Insbesondere in Kapitel 5 und 6 werden durch Reinforcement Learning ganze Schwärme von Agenten gesteuert bzw. darauf trainiert dynamisch intelligenten Hindernissen bzw. Verfolgern auszuweichen und Kollisionen mit diesen sowie untereinander zu vermeiden. Deshalb werden Grundlagen zu dieser Technik im Abschnitt 2.6 ausgeführt.

Über die grundlegenden Methoden und verwandten Arbeiten in diesem Kapitel hinaus, welche für die gesamte Arbeit relevant sind, wird am Anfang eines jeden Kapitels auf fremde Arbeiten eingegangen, welche für das jeweilige Kapitel von Bedeutung sind.

2.1 Intelligente Agenten

Russell und Norvig definieren einen Agenten als alles, was seine Umgebung (durch Sensoren) wahrnehmen und auf diese Umgebung (durch Aktuatoren) einwirken kann [126]. Sie identifizierten das Konzept der rationalen Agenten als zentral für ihren Ansatz der künstlichen Intelligenz – Systeme, die man vernünftigerweise als intelligent bezeichnen kann. Sie kategorisieren Agenten nach ihrer inneren Struktur. In dieser Kategorisierung enthalten alle fortgeschrittenen Typen ein Modell der Welt, in der sie agieren. Der Zustand dieses Modells repräsentiert das aktuelle Weltbild der Agenten und wird durch ihre Wahrnehmungen aktualisiert. Aber dieses Modell erlaubt es dem Agenten auch,

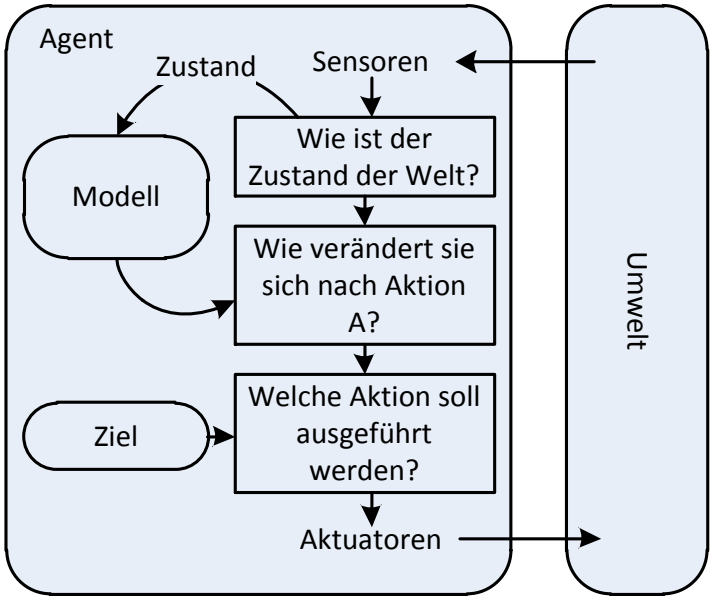


Abbildung 2.1: Überblick über die innere Struktur eines modellbasierten, zielbasierten Agenten (adaptiert von [126]). Er verfügt über ein Modell, um den Zustand der Welt zu verfolgen und abzuschätzen, wie sich die Welt während seiner Aktionen entwickelt. Mit einer Zielfunktion ist er in der Lage, zu beurteilen, ob eine gewählte Handlung zur Erreichung seiner Ziele führt.

abzuschätzen, wie sich die Welt entwickelt, entweder unabhängig vom Agenten oder verursacht durch eine Handlung des Agenten. Um vernünftig handeln zu können, benötigt der Agent auch eine Art Zielinformation, die Situationen beschreibt, die wünschenswert sind. Ausgestattet mit einer solchen Zielfunktion kann der Agent abschätzen, ob eine Handlung den Zustand der Welt so verändert, dass sie (schließlich) zur Erreichung seiner Ziele führt. Die Struktur eines modellbasierten, zielbasierten Agenten ist in Abb. 2.1 skizziert. Im weiteren Verlauf dieser Arbeit werden Agenten behandelt, die ihre zukünftigen Aktionen auf der Grundlage verschiedener Weltmodelle planen und bewerten.

2.2 Pfadplanung

In den folgenden Unterabschnitten sollen zunächst elementare Begriffe der Graphentheorie definiert werden, wonach auf Single-Source Shortest Path Algorithmen eingegangen wird, mit welchen die kürzesten Wege zwischen einem gegebenen Startknoten und den übrigen Knoten eines Graphen berechnet werden können.

2.2.1 Graphen

Ein gerichteter Graph $G = (V, E)$ im Sinne der Graphentheorie ist ein Tupel aus einer endlichen Menge von Knoten V (englisch für Vertices) und Kanten E (englisch für Edges). Dabei ist eine Kante $e = (u, v) \in E$ ($E \subseteq V \times V = \{(v,w) \mid v,w \in V\}$) eine gerichtete Verbindung vom Knoten u zum Knoten v. Zudem gibt es ungerichtete Graphen, welche als vereinfachte Version gerichteter Graphen aufgefasst werden können, in welchen für jede gerichtete Kante (u,v) die entsprechende Gegenkante (v,u) vorhanden ist. Ungerichtete Kanten werden dann in der Regel als Menge $\{u, v\}$ notiert. Veranschaulicht werden Graphen im Allgemeinen, indem die Knoten als Kreise bzw. Punkte und die Kanten als sie verbindende Linien (mit Pfeil im gerichteten Fall) gezeichnet werden. Kanten können mit Objekten aus einer Menge M beschriftet bzw. „markiert“ werden. Diese repräsentieren dann Bezeichner, Längen, Gewichte, Kosten oder Kapazitäten. Im Folgenden spielen vor allem gewichtete Graphen eine Rolle, bei denen den Kanten mit einer Gewichtsfunktion eine reelle Zahl zugeordnet wird $w : E \mapsto \mathbb{R}$. Die Beziehungen zwischen den Knoten bzw. die Gewichte der Kanten werden in der Regel in einer quadratischen Gewichtsmatrix, der Adjazenzmatrix, oder in einer sogenannten Adjazenzliste (auch Nachbarschaftsliste) gespeichert. Zwei Knoten u und v heißen adjazent bzw. benachbart, wenn zwischen ihnen eine Kante $(u, v) \in E$ existiert. Ein *Weg* oder *Pfad* $p = (v_0, v_1, \ldots, v_k)$ durch den Graphen ist eine Folge von Knoten mit der Eigenschaft, dass aufeinanderfolgende Knoten mit einer Kanten aus der Menge E verbunden sind, d.h. $(v_{i-1}, v_i) \in E$ für $i = 1, 2, \ldots, k$ (für gerichtete Graphen).

Die Länge l eines Pfades wird im Folgenden als die Summe der Kantengewichte der im Pfad enthaltenen Kanten angesehen:

$$l(p) = \sum_{i=1}^{k} w((v_{i-1}, v_i)) \ .$$

In Analogie zu einem Straßennetz können die Straßen als Kanten und Kreuzungen als Knoten aufgefasst werden. Die Gewichte der Kanten repräsentieren in dieser Analogie bzw. auch im Folgenden dieser Arbeit den Euklidischen Abstand der Knoten bzw. die Distanz zwischen den Kreuzungen (unter der vereinfachenden Annahme von kartesischen Koordinaten).

Ein Pfad heißt einfach, wenn alle enthaltenen Knoten paarweise verschieden sind, d.h., $v_i \neq v_j$ für $i, j \in \{1, \ldots, n-1\}$ mit $i \neq j$. Man spricht von einem Kreis, wenn $(v_0, v_1, \ldots, v_k)$ ein einfacher Pfad ist und gilt $v_0 = v_k$, d.h., dass Start- und Endknoten des Pfades übereinstimmen. Graphen ohne Kreise heißen azyklisch oder kreisfrei. Ein ungerichteter Graph wird als zusammenhängend bezeichnet, wenn von jedem Knoten u ein Pfad zu jedem anderen Knoten v existiert, d.h. falls es zu je zwei beliebigen Knoten $u, v \in E$ einen ungerichteten Weg in G mit u als Startknoten und v als Endknoten gibt. Ein ungerichteter Graph $G = (V,E)$ heißt (stark) zusammenhängend, falls es zwi-

schen zwei beliebigen Knoten u und v aus G sowohl einen gerichteten Pfad von u nach v sowie auch einen gerichteten Weg von v nach u in G gibt.

Bäume sind spezielle Graphen im Sinne der Graphentheorie. Ein Baum ist ein zusammenhängender kreisfreier Graph. Sie lassen sich bei ungerichteten Graphen unter anderem dadurch charakterisieren, dass jedes Paar von Knoten des Graphen durch einen einzigen einfachen Pfad verbunden ist. Existiert in einem Baum ein besonderer, ausgezeichneter Knoten, der von den anderen Knoten unterschieden wird, bezeichnet man den Baum als gewurzelten Baum. Der ausgezeichnete Knoten stellt die Wurzel dar. In Visualisierungen von Graphen wird diese häufig oberhalb der anderen Knoten dargestellt. Für gerichtete Graphen lässt sich noch einmal unterscheiden, ob die Pfade in Richtung der Wurzel zeigen (In-Tree) oder von dieser weg (Out-Tree). Für die Beziehungen zwischen den Knoten eines gewurzelten Baumes beziehungsweise eines Out-Trees sind die folgenden Bezeichnungen gebräuchlich. Betrachtet man einen Pfad von der Wurzel w zu einem Knoten y, wird jeder Knoten x auf diesem Pfad als Vorfahr von y bezeichnet. Ist x ein Vorfahr von y, dann ist y ein Nachfahr von x. Der von y ausgehende Teilbaum mit Wurzel y ergibt sich aus dem Knoten y und dessen Nachfahren im ursprünglichen Baum. Wird eine einzelne Kante (x,y) auf dem Pfad von der Wurzel w zum Knoten y betrachtet, heißt x Vater von y und y Kind von x.

Das Problem den kürzesten Weg zwischen zwei Knoten in einem Graph zu finden (Shortest Path Problem) ist in der Informatik von zentraler Bedeutung, da sich viele Problemstellungen darauf abbilden lassen. Ein offensichtliches Problem aus der realen Welt wäre die kürzeste Route (im Sinne der geographischen Distanz) zwischen zwei Orten in einem Straßennetz zu finden. Im Folgenden sollen Algorithmen betrachtete werden, die dieses Problem lösen.

2.2.2 Dijkstra Algorithmus

Der Dijkstra Algorithmus wurde 1959 von Edsger W. Dijkstra [28] vorgeschlagen und löst das Single-Source-Shortest-Paths Problem, d.h., er berechnet den kürzesten Weg von einem Startknoten zu allen anderen Knoten eines Graphen, bzw. je nach Implementierung den kürzesten Pfad zu allen Knoten, deren Abstand vom Startknoten kleiner gleich dem Abstand eines bestimmten Zielknotens zum Start ist. Gegeben sei dazu ein gewichteter, gerichteter Graph $G = (V, E, w)$ mit nichtnegativen Kantengewichten und ein Startknoten $s \in E$. Ein gerichteter Pfad in $p = (v_0, v_1, \ldots, v_k)$ in G habe die Länge/Kosten $l(p) = \sum_{i=1}^{k} w((v_{i-1}, v_i))$ und der (gerichtete) Abstand von zwei Knoten u,$v \in E$ sei $d(u,v) = min\{l(p) \mid p$ Pfad von u nach v in $G\}$ (bzw. ∞, falls kein Pfad von u nach v in G existiert). Der Dijkstra Algorithmus (siehe Algorithmus 1) verläuft in Runden und hält dabei eine Menge an Knoten S aufrecht, für welche die finale Länge des kürzesten Pfades bereits bestimmt wurden. Der Algorithmus wählt wiederholt einen Knoten $u \in V - S$ mit der minimalen Pfadlängenschätzung, fügt ihn zu S hinzu und führt dann eine

Algorithmus 1 Dijkstra

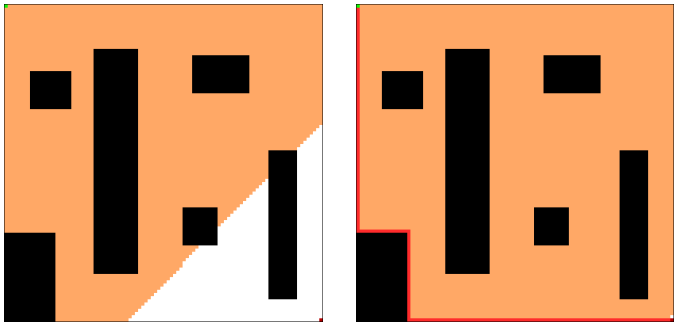
Require: gerichteter, gewichteter Graph $G = (V, E, w)$
Startknoten s

1: $S \leftarrow \emptyset$, dist$[s] \leftarrow 0$, p$[s] \leftarrow -2$
2: **for each** $v \in V - \{s\}$ **do**
3: | dist$[v] \leftarrow \infty$
4: | p$[v] \leftarrow -1$
5: **end for**
6: **while** $\exists u \in V - S$: dist $< \infty$ **do**
7: | $u \leftarrow$ ein Knoten u mit minimalen dist$[u]$
8: | $S \leftarrow S \cup \{u\}$
9: | **for each** $v \in V - S$ mit $(u, v) \in E$ **do**
10: | | dd $\leftarrow$ dist$[u] + w(u,v)$
11: | | **if** dd $<$ dist$[v]$ **then**
12: | | | dist$[v] \leftarrow$ dd
13: | | | p$[v] \leftarrow u$
14: | | **end if**
15: | **end for**
16: **end while**

Return.: dist$[1..n]$ # *Länge $d(s,v)$ der kürzesten Wege*
p$[1..n]$ # *Vorgängerknoten $p(v)$*

sogenannte Relax-Operation (innere For-Schleife in Algorithmus 1) auf allen von u ausgehenden Kanten durch, wodurch die Schätzung der Pfadlänge und der Vorgänger für Knoten v aktualisiert wird, wenn der kürzeste bisher zu v gefundene Pfad mit dem Durchqueren von u verbessert werden kann. Im Algorithmus kommt eine dist-Array zum Einsatz, um Schätzungen den aktuellen Pfadlängen vorzuhalten. Aus diesem muss in Zeile 9 des Algorithmus 1 ein Knoten mit minimalem Entfernungswert gefunden werden, der in V aber nicht in S enthalten ist. Wird anstelle des dist-Arrays eine Prioritätswarteschlange verwendet, die als Binärheap realisiert ist, hat der Dijkstra Algorithmus eine Laufzeit von $O((n+m)logn)$, um in einen gerichteten Graphen $G = (V, E, w)$ die kürzesten Wege von einem Startknoten s aus zu allen anderen Knoten im Graph zu berechnen. Wobei n die Knotenanzahl ($|V|$) und m die Kantenzahl $|E|$ ist. Weil der Dijkstra Algorithmus immer den „nächsten“ Knoten aus $V-S$ auswählt, um ihn zu S hinzuzufügen, gehört er zu den Greedy Algorithmen. Diese Klasse von Algorithmen muss nicht immer die optimale Lösung liefern. Für den Dijkstra Algorithmus kann jedoch gezeigt werden, dass er unter den genannten Voraussetzungen, die kürzesten Pfade findet. Dies würde an dieser Stelle allerdings zu weit führen, weshalb auf [21] verwiesen, woraus (zusammen mit [27]) auch die vorherigen Aussagen entnommen wurden.

Die Abbildung 2.2 zeigt die Ausführung des Dijkstra Algorithmus auf einem Graphen, welcher aus einer Rastergrafik erzeugt wurde. Auf die Trans-



(a) Expansion der Menge S

(b) Resultierender Pfad zwischen Start- (links oben) und Zielknoten (rechts unten).

Abbildung 2.2: Ausführung des Dijkstra Algorithmus auf einem mit der in Abschnitt 2.3 beschrieben Methode erzeugten Graphen.

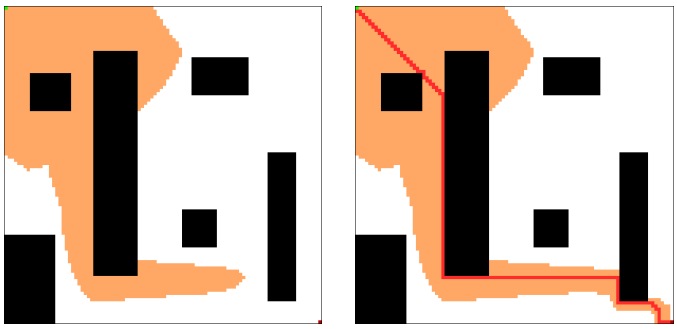
formation von Rastergrafiken in Graphen wird in Abschnitt 2.3 eingegangen. Dadurch, dass alle Kantenkosten homogen 1 betragen und durch die Struktur des Dijkstra Algorithmus, erfolgt die Expansion der Menge S (orange Pixel) gleichmäßig vom Start aus (grüner Pixel in der linken, oberen Ecke) über die begehbare Fläche, bis ein Pfad (rote Linie) zum Ziel (roter Pixel in der rechten, unteren Ecke) gefunden wird.

2.2.3 A* Algorithmus

Neben dem Algorithmus von Dijkstra existieren noch solche, die mit negativen Kantengewichten umgehen können, ohne sich in negativen Kreisen/Zyklen zu verlieren bzw. dies zumindest zu detektieren (z.B. der Bellman-Ford Algorithmus) oder das All-Pairs-Shortest-Paths Problem, d.h. den kürzesten Pfad zwischen allen Knoten des Graphen, in einer kürzeren Laufzeit löst, als es z.B. mit der wiederholten Ausführung des Dijkstra Algorithmus der Fall wäre (z.B. der Floyd-Warshall Algorithmus).

Da in dieser Arbeit aber in der Regel keine negativen Kantengewichte bei der Pfadplanung auftreten und das All-Pairs-Shortest-Paths Problem nicht vorkommt, soll im Folgenden auf eine etablierte Verallgemeinerung bzw. Erweiterung des Dijkstra-Algorithmus, der A* Algorithmus (kurz A*), betrachtet werden.

Der A* gehört zur Klasse der informierten Suchalgorithmen und dient wie dessen Grundlage, der Dijkstra Algorithmus zur Lösung des Single-Source-Shortest-Paths Problems in Graphen mit nichtnegativen Kantengewichten. Während der Dijkstra Algorithmus ausschließlich die Kantengewichte heranzieht, um eine Abschätzung über die Länge des kürzesten Pfades zu einem Knoten vorzuhalten (im Algorithmus 1 das dist-Array, im Folgenden $d(u)$ genannt), verwendet der A* eine zusätzliche Heuristik $h(u)$, welche einbezogen



(a) Expansion des A* mit Euklidischen Distanz zum Ziel als Heuristik

(b) Resultierender Pfad zwischen Start- (links oben) und Zielknoten (rechts unten).

Abbildung 2.3: Ausführung des A* Algorithmus auf einem mit der in Abschnitt 2.3 beschrieben Methode erzeugten Graphen.

wird, um zu bestimmen welcher Knoten u als nächstes expandiert bzw. relaxiert wird. Dementsprechend wird als nächstes der Knoten gewählt, dessen Wert

$$f(u) = d(u) + h(u)$$

am geringsten ist.

Mit der Heuristik $h(u)$ sollen die Kosten vom Knoten u zum Zielknoten geschätzt werden. Deshalb eignen sich in einem kartesischen Koordinatensystem die euklidische Distanz bzw. die Großkreis-Distanz auf einer Kugeloberfläche (wie man sie idealisiert für die Erde annimmt). Es sind auch andere Heuristiken möglich, wobei eine Bedingung an diese ist, dass durch sie die tatsächlichen Kosten nicht überschätzt werden. Die Heuristik der Euklidischen Distanz bzw. die Großkreis-Distanz dient dazu, dass der A*, im Gegensatz zum Dijkstra Algorithmus, Knoten nicht kreisförmig um den Startknoten expandiert, sondern mit einer Gewichtung in Richtung des Zielknotens erfolgt. Bei einer beispielhaften Pfadplanung in einem Straßennetz wird dadurch die Suche in Richtung des Zielortes gelenkt, womit der kürzeste Pfad in der Regel schneller gefunden wird als mit dem Dijkstra Algorithmus und die Suche terminiert.

Die Abbildung 2.3 zeigt die Ausführung des A* Algorithmus auf einer Rastergrafik. Auf die Erzeugung eines Graphen aus einer solchen Grafik wird im folgenden Abschnitt 2.3 eingegangen. Dadurch, dass alle Kantenkosten homogen 1 betragen und durch die Struktur des A* Algorithmus, erfolgt die Expansion (orange Pixel) mit einer Tendenz in Richtung Ziel (roter Pixel in der rechten, unteren Ecke). Es kam die Euklidische Distanz in Richtung Ziel als Heuristik zum Einsatz, was Knoten (Pixel), die in der Richtung des Ziels liegen, attraktiver erscheinen lässt als andere. Dies geschieht, bis das Ziel gefunden wurde und ein Pfad (rote Linie) zwischen Start (grüner Pixel in der linken, oberen Ecke) und Ziel erstellt werden kann.

2.3 Diskretisierungsmethoden

Um Freiflächen $A \subset \mathbb{R}^n$, wie sie in Gebäuden, im Straßenverkehr z.B. auf Parkplätzen oder in Computerspielen vorkommen, für Planungsalgorithmen greifbar zu machen, müssen diese in eine algorithmisch verarbeitbare Repräsentation überführt werden. Für eine solche verarbeitbare Repräsentation eignet sich die zuvor eingeführte Datenstruktur der Graphen.

In der Regel werden frei begehbare Flächen diskretisiert indem sie in eine endliche Anzahl an Regionen unterteilt werden. Diese repräsentieren dann jeweils eine Menge an Punkten im $\mathbb{R}^n$. Nimmt man den Mittelpunkt einer Region als Knoten und verbindet die Knoten benachbarter Regionen mittels Kanten, ergibt sich ein Graph, wobei Kanten nur gebildet werden dürfen, wenn sich auf der Strecke zwischen den beiden benachbarten Regionenmittelpunkten kein Hindernis befindet.

Damit ein, aus einer solchen direkten Aufteilung der Freifläche in Regionen, resultierender Graph valide zur Pfadplanung genutzt werden kann, sollte der Graph bzw. die Aufteilung die Eigenschaften der Quantisierung und die der Lokalisierung erfüllen [98].

Die Eigenschaft der Quantisierung sagt aus, dass es möglich sein muss, jeden Punkt im Raum einem Knoten des Graphen zuzuordnen. Es darf kein Punkt existieren, der sich in keiner Regionen befindet. Also, alle Regionen zusammen müssen den gesamten Raum füllen. Mathematisch ausgedrückt bedeutet dies, dass für den Raum $A \subset \mathbb{R}^n$ und die gebildeten Regionen $R_1, R_2, \ldots, R_k \subset A$ gelten muss, dass ihre Vereinigung wieder den quantisierten Raum ergibt $R_1 \cup R_2 \cup \ldots \cup R_k = A$. Die Eigenschaft der Lokalisierung verlangt die Umkehrbarkeit der Abbildung, so dass jeder Knoten des Graphen in eine Position des Raums überführt werden kann, so dass sich ein Agent, der den Graph benutzt, damit im Raum lokalisieren kann.

Grundsätzlich kann die Aufteilung des frei begehbaren, kontinuierlichen Raumes in diskrete Regionen händisch erfolgen. Das Ziel ist es, den Raum automatisiert in einen Graph zu überführen. Eine der ersten und einfachsten Diskretisierungsmethoden ist sogenannte „Tile Graphs“. Bei einer Tile (engl. für Fliese/Kachel) basierten Diskretisierung des Raumes wird dieser komplett in reguläre, gleich große Kacheln unterteilt. Dies sind in der Regel Quadrate (in höheren Dimensionen Würfel oder den entsprechenden Hyperwürfel), es lassen sich aber auch Ebenen mit regulären Hexagonen lückenlos parkettieren, wodurch diese durchaus auch Anwendung finden. Um eine Fläche lückenlos mit Quadraten zu kacheln, wird diese in der Regel einfach in ein regelmäßiges Gitter unterteilt. Kacheln sind dann entweder betretbar, wenn sie zur Freifläche gehören oder nicht betretbar, wenn sich an dieser Stelle ein Hindernis, wie z.B. eine Wand befindet. Das Gitter aus freien und belegten Kacheln lässt sich einfach in einer Matrix speichern z.B. mit einer 0 für nicht betretbare/mit Hindernissen blockierte Kacheln und einer 1 für freie, betretbare Kacheln. Ein jeder Punkt $p \in \mathbb{R}^n$ wird durch den Wert der Matrix, an der Position $\lfloor p/s \rfloor$

definiert wobei s die Größe eines Quadrats ist. Die Funktion $\lfloor \ldots \rfloor$ gibt den höchsten ganzzahligen Wert zurück, der kleiner, oder gleich, zu ihrem Argument ist. In einem zweidimensionalen Raum ($A \in \mathbb{R}^2$) erhält man durch diese Unterteilung ein schachbrettartiges Muster.

Ein solches Schachbrett lässt sich leicht in einen Graph überführen, indem der Mittelpunkt jedes freien Quadrates als Knoten angenommen wird und Kanten zwischen freien, begehbaren Quadraten aufgebaut werden, die eine Seite teilen. Damit erhält man einen (endlichen) Graphen, der den gesamten begehbaren Raum repräsentiert. Die Darstellung des schachbrettartigen Kachelmusters in Form einer Matrix mit 0/1-Werten entspricht essentiell auch der Darstellung von schwarz-weißen Rastergrafiken im Computer. Liegt ein Grundriss als Rastergrafik vor, ist damit auch einfach klar wie dieser in ein Graph zu überführen ist. Jeder weiße Pixel der Freifläche wird in einen Knoten des Graphen überführt und über Kanten mit Knoten anderer, angrenzender weißer Pixel verbunden. Ein Beispiel für einen solchen resultierenden Graph ist in Abbildung 2.4 gegeben. Der Nachteil an diesem Vorgehen ist, dass das zugrundeliegende Gitternetz sehr fein sein kann und damit entsprechend sehr viele Knoten entstehen, was wiederum die Pfadplanung auf dem resultierendem Graph verlangsamt. Wird die Pfadplanung auf solchen, aus Pixeldarstellungen entstandenen Graphen, ausgeführt, tendieren die Pfade dazu, direkt auf dem nächsten weißen Pixel neben einer Wand entlang zu führen. Dies kann bereits in Simulationen mit Agenten, die einen gewissen Durchmesser haben, zu Problemen führen, ist aber spätestens in realen Situationen mit Robotern nicht wünschenswert und erfordert weitere Schritte der Nachbearbeitung. „Tile Graphs“ stellen also eine leicht zu überblickende und implementierende Methode dar, die bei statischen, kleinen und einfach aufgebauten Räumen ihre Berechtigung hat. Andererseits ist sie bei vielen Knoten rechenintensiv und wenig skalierbar, so dass es sinnvoll sein kann z.B. mehrere Kacheln in einem weiteren Schritt zusammenzufassen. Steigt die Komplexität des Raumes oder ist dieser nicht statisch, sondern weist unvorhersehbare Veränderungen über die

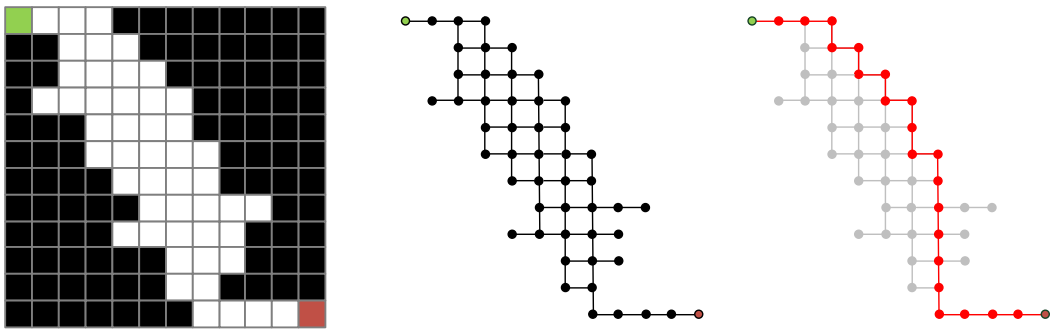


Abbildung 2.4: Überführung einer Rastergrafik in einen Graphen mit einem beispielhaften kürzesten Pfad zwischen Start (grün) und Ziel (bordeauxrot).

Zeit auf, eignen sich andere Methoden zur Diskretisierung des Raumes.

Deshalb soll im Folgenden auf Techniken eingegangen werden, die Freiflächen generischer und weniger feingliedrig diskretisieren, das heißt, in einen Graph für die Pfadplanung überführen, und dadurch besser skalieren.

Im Folgenden werden zwei der weitverbreitetsten Ansätze für die Diskretisierung und Pfadplanung in einer kontinuierlichen Domäne vorgestellt. Probabilistic Roadmaps [66] und Rapidly Exploring Random Trees [82] sind zwei Beispiele für fortgeschrittenere Ansätze für die Diskretisierung eines kontinuierlichen Bereichs.

2.3.1 Probabilistic Roadmap

Die Probabilistic Roadmap (PRM) Methode [66] ist ein Pfadplanungsalgorithmus für kontinuierliche Räume. Der Algorithmus quantifiziert den Raum, indem er zufällig Knoten setzt und alle vorhandenen Knoten innerhalb eines vordefinierten Abstands mit einer Kante verbindet, wenn der Raum dies zulässt. Eventuell auftretende disjunkte Graphen werden in einem späteren Schritt verbunden. Der resultierende Graph kann mit graphbasierten Planungsalgorithmen zur Planung eines Pfades zwischen einer Start- und einer Endposition innerhalb des Raumes verwendet werden.

Der Algorithmus besteht aus zwei Phasen, einer Lernphase und einer Anfragephase. Die Lernphase ist wiederum in zwei Schritte unterteilt. Der erste Schritt erzeugt zufällig Knoten innerhalb des Raums und versucht, jeden Knoten mit allen anderen Knoten mit einer Kante innerhalb eines vordefinierten Abstands zu verbinden. Eine solche direkte Verbindung wird natürlich nicht hergestellt, wenn sie durch einen ungültigen Bereich (z.B. eine Wand) verlaufen würde. Dieser Prozess wird für eine vordefinierte Anzahl an Iterationsschritten wiederholt. Mit einer höheren Anzahl an Iterationen erhöht sich die Anzahl an Knoten und Kanten und damit die Wahrscheinlichkeit, dass der Graph die Freifläche präzise repräsentiert aber auch den Speicherbedarf des Graphen und die Laufzeit (ebenfalls im Hinblick auf die spätere Pfadplanung). Der zweite Schritt kompensiert das Auftreten von disjunkten Graphen, die das Ergebnis des ersten Schrittes sein können. Dies kann zum Beispiel geschehen, wenn relativ weitläufige Teile des frei begehbaren Raumes nur durch eine Engstelle miteinander verbunden sind. Um die unter Umständen entstandenen Teilgraphen zu verbinden, werden Knoten ausgewählt, die in den gerade beschriebenen „schwierigen“ Zonen liegen. In deren näherer Umgebung werden dann neue Knoten erzeugt und versucht, diese mit dem Graphen zu verbinden. Um Knoten in schwierigen Zonen (in denen zwei Teilgraphen aneinander grenzen) zu finden, schlagen die Autoren von [66] verschiedene Strategien vor. Eine davon ist, den nächsten Knoten zu finden, der mit dem betrachteten Knoten nicht verbunden ist und den Abstand zwischen den beiden zu berechnen. Ist dieser sehr gering, ist die Wahrscheinlichkeit hoch, dass sich der betrachtete Knoten in einer „schwierigen“ Zone befindet.

In der Abfragephase des Algorithmus wird der neu konstruierte Graph verwendet, um einen Pfad zwischen einer Start- und einer Endposition innerhalb des Raumes zu bestimmen. Wenn die Abfragephase fehlschlägt, ist dies ein Hinweis darauf, dass der in der Lernphase gebildete Graph keine gute Darstellung des Raums ist. Dies geschieht entweder, wenn die Start- oder Endposition nicht mit dem Graphen verbunden werden kann, oder wenn es keinen möglichen Pfad durch den Graphen gibt (letzteres könnte auf einen disjunkten Raum hinweisen). In beiden Fällen ist es notwendig, die Lernphase zu wiederholen (ggf. mit einer höheren Anzahl von Iterationen).

Das Verfahren kann viele redundante Kanten erzeugen, die bestimmte Bereiche des Raumes dicht bedecken, wohingegen andere Regionen nur spärlich überdeckt sind.

Da der durch die Probabilistic Roadmap erzeugte Graph nur eine Momentaufnahme des Raumes ist, stellt sich die Frage, wie mit sich dynamisch verändernden Räumen umgegangen werden soll. In diesem Fall wäre es notwendig, die Roadmap komplett neu zu berechnen, es sei denn, man verfügt über spezifische Informationen über die Art der Veränderungen. Eine solche vollständige Neuberechnung ist sehr ineffizient. Es gibt Verfahren, die die Neuplanung von Probabilisic Roadmaps effizienter machen [8]. Diese basieren im Wesentlichen auf der Beschleunigung der Pfadplanungsphase durch die Verwendung dynamischer, graphbasierter Pfadplanungsalgorithmen, die keine vollständige Neuberechnung des Pfades erfordern [90].

Die Abbildung 2.5 zeigt die beispielhafte Diskretisierung einer Freifläche mittels PRM. Zu sehen ist der resultierende Graph, mit dicht und dünn besetzten Regionen, die aus der zufälligen Wahl der Knotenpostionen resultieren.

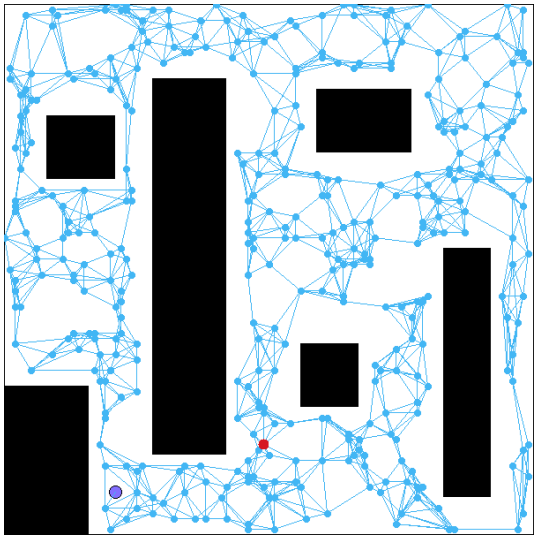


Abbildung 2.5: Beispielhafte Diskretisierung einer Freifläche mittels PRM und einem Agenten (roter Punkt), welcher auf dem resultierenden Graphen zu einem Ziel (lila Kreis) navigiert.

2.3.2 Rapidly Exploring Random Trees

Ähnlich wie Probabilistic Roadmaps basieren auch die sogenannten Rapidly Exploring Random Trees (RRT) [82] auf der zufälligen Generierung von Wegpunkten. Die Absicht hinter RRT war es, physikalische Phänomene, die die reale Pfadplanung beeinflussen, direkt in den geplanten Pfad einzubeziehen. Der durch eine PRM erzeugte Pfad kann plötzliche Wendungen mit Winkeln aufweisen, die ein realer Agent (z.B. ein Auto) mit Eigenschaften wie Geschwindigkeit, Beschleunigungs- oder Bremsweg möglicherweise nicht fahren kann.

RRT eignet sich besonders für solche Realwelt Systeme, da ein Knoten im Graphen als Konfiguration des Agenten und eine Kante als möglicher Weg von einem vorherigen Zustand in diese Konfiguration hinein angesehen werden kann. Bereits beim Erstellen des Knotens und der verbindenden Kante, kann darauf geachtet werden, dass Beschränkungen, die für die Bewegung des Agenten gelten, eingehalten werden. Wie beschrieben, können PRMs mehrere disjunkte Graphen liefern, die erst nach einer hohen Anzahl von Iterationen miteinander verbunden werden können. Das RRT-Verfahren löst dieses Problem durch die Verwendung einer anderen Datenstruktur. Ausgehend von der Startposition des Agenten wird ein einziger Baum erstellt, mit dem jeder Knoten, der generiert wird, verbunden werden muss bis das Ziel erreicht wird.

Der Algorithmus erstellt den Baum wie folgt: Der erste Knoten ist die Startposition des Agenten. Dieser Knoten stellt auch die Wurzel des Baumes dar, der gebildet wird. Mit Hilfe einer Wahrscheinlichkeitsverteilung wird ein zufälliger neuer Punkt im Raum erzeugt. Zu diesem Punkt wird der Knoten aus dem Baum gesucht, der den geringsten Abstand zu ihm hat. Dann wird versucht, eine direkte Kante zwischen den beiden Punkten zu erzeugen. Dies ist nur möglich, wenn der gesamte Abstand der Kante im Raum liegt (ohne Hindernisse) und der Weg alle vorgegebenen Bedingungen, z.B. Beschleunigung und Geschwindigkeit, erfüllt. Dieser Vorgang wird für eine vordefinierte Anzahl von Iterationsschritten wiederholt, was zu einem Baum führt. Mit Hilfe einer angepassten Wahrscheinlichkeitsverteilung, mit der neue Knoten im Raum erzeugt werden, kann die Erzeugung neue Knoten in Richtung des Ziels gelenkt werden. Dies ist ähnlich der distanzbasierten Heuristik beim A* (siehe Abschnitt 2.2.3). Das Verfahren endet nach der vordefinierten Anzahl an Iterationen oder wenn das Ziel und mit einer Kante als Knoten zum Baum hinzugefügt wurde.

Mit Hilfe von graphenbasierten Pfadfindungsalgorithmen könnte dann ein Pfad über den Baum und damit auch im Raum berechnet werden. Aufgrund der Baumstruktur ist die Pfadberechnung allerdings einfacher. Man kann den Pfad leicht vom Zielknoten über die Iteration über die jeweiligen Elternknoten im Baum zurückverfolgen, bis die Wurzel des Baumes, d.h. die Position des Agenten, erreicht ist. Die Umkehrung dieses Pfades entspricht dann dem Pfad, den der Agent zum Ziel zurücklegen muss.

Die Abbildung 2.6 zeigt einen beispielhaften RRT, bei welchem ausgehend

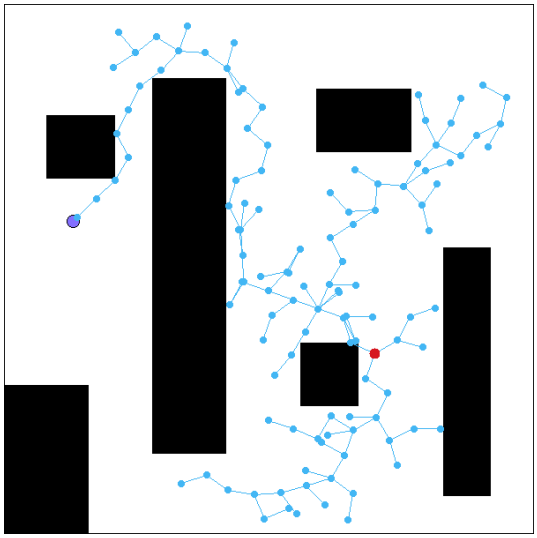


Abbildung 2.6: Beispielhafter RRT eines Agenten (roter Punkt) auf der Suche nach einem Ziel (lila Kreis) auf einer Freifläche.

von einem Agenten (roter Punkt) ein Baum aufgebaut wurde, bis das Ziel (lila Kreis) gefunden wurde. Die Route zum Ziel ergibt sich für den Agenten durch die Iteration über die Vaterknoten im Baum ausgehend vom Blattknoten des Ziels hin zur Wurzel des Baumes, welche der Agent darstellt.

Vorteilhaft ist die Möglichkeit, während der Generierung neuer Knoten Bedingungen zu definieren, die Eigenschaften wie Geschwindigkeit, Beschleunigung oder Kurvenradius usw. berücksichtigen. Ein Nachteil ist jedoch die Tatsache, dass bei der Steuerung mehrerer Agenten und Ansteuerung mehrerer Ziele für jeden Agenten und jedes Ziel jeweils ein eigener Baum generiert werden muss.

2.4 Kollisionserkennung und -vermeidung

Kollisionsvermeidung ist ein weites Forschungsfeld mit einer Vielzahl unterschiedlicher Ansätze und einer hohen Geschwindigkeit, mit der neue Lösungen entwickelt werden.

Im Gegensatz zu Cooperative Pathfinding Problemen [130] wird hier in dieser Arbeit nicht davon ausgegangen, dass Agenten explizit miteinander kooperieren, sondern sich gegenseitig als Hindernis betrachten. Ein zentrales Element ist die Bewegungsvorhersage von dynamischen Hindernissen. Philippsen et al. besagen in [114], dass Vorhersagen Wahrscheinlichkeitsaussagen sind und ihr Nutzen sich daher nur dann offenbart, wenn ein bestimmtes Kollisionsrisiko eingegangen wird. Zudem schlagen die Autoren von [114] ein mehrstufiges Konzept vor, bei dem auf lokaler Ebene Algorithmen zur Kollisionsvermeidung wie der Potential Field Ansatz [67, 68] eingesetzt werden, um letztlich eine Kollision zu verhindern. Darüber hinaus gibt es eine breite Palette von Techniken zur

Bewegungsvorhersage [85]. Die Methoden reichen von einfachen Bewegungsinterpolationen bis hin zu komplexen, lernenden Modellen. Durch die Verwendung von Kalman-Filter können Unsicherheiten in den Vorhersagen besser abgeschätzt werden. Sie finden u.a. Anwendung in autonomen Fahrzeugen [5, 6]. Bayes'sche Filter- und Monte-Carlo-Methoden werden zur Analyse des Bewegungsverhaltens im Straßenverkehr eingesetzt [65, 30]. Die zentrale Aufgabe der Bewegungsvorhersage ist das Lernen und die Vorhersage zukünftiger Trajektorien. In diesem Zusammenhang sind Gauß'sche Prozesse und Bayes'sche Netzwerke wichtige Werkzeuge, die integrale Bestandteile autonomer Agenten beschreiben [44, 137]. Kim et al. weisen darauf hin, dass diese Modelle durch einen hohen Grad an Komplexität zusammen mit einem hohen Anteil an manuellen Parametereinstellungen gekennzeichnet sind [70]. In einem hochdynamischen Umfeld kann diese Eigenschaft ein starker Nachteil sein. Im Gegensatz zu den vorher genannten Methoden werden künstliche neuronale Netze (KNN) oft mit der Fähigkeit assoziiert, eine Intuition für Regeln und Muster zu entwickeln. Seit dem Durchbruch in der Bilderkennung [75] und Sprachverarbeitung [46, 92] dienen sie zunehmend der Vorhersage von Bewegungen. Bisherige Forschungen haben gezeigt, dass sie konventionellen Ansätzen in bestimmten Aspekten deutlich überlegen sind [70, 109]. Erfahrungsbasierte Lernprozesse machen KNN besonders geeignet für die Vorhersage komplexer menschlicher Bewegungsmuster und Gruppendynamik [2, 147].

Bewegungsvorhersagen weisen einen Zeitbezug auf und Pfadplanungsalgorithmen sind in der Regel für Graphen definiert. Dies wirft die Frage auf, wie die Bewegungen der Hindernisse in den Routing-Graphen eines Agenten abgebildet werden können. Zwei beliebte Diskretisierungsalgorithmen zur Erstellung dieser Graphen aus Freiflächen sind Probabilistic Road Map (PRM) [66] und Rapidly Exploring Random Tree (RRT) [82], die häufig zur Diskretisierung einer statischen Umgebung verwendet werden. Frühere Arbeiten haben sich mit der Frage beschäftigt, wie diese Algorithmen für dynamische Umgebungen angepasst werden können. Dabei werden dynamische Hindernisse bereits bei der Erstellung eines Graphen berücksichtigt. Die Arbeit von van den Berg et al. [138] gehört zu den ersten Arbeiten, die den Einsatz deterministischer Planungsalgorithmen mit wahrscheinlichkeitsbasierten Darstellungen der Umgebung untersuchen. Sie haben einer zweidimensionalen, räumlichen PRM eine zeitliche Dimension hinzugefügt. Die Bewegungen der Hindernisse werden in diesem dreidimensionalen Suchraum als Zylinder dargestellt und beschreiben die komplette Bewegungsbahn eines Hindernisses in der Zeit. Es wird ein Graph erstellt, der diese Trajektorien vermeidet. Dann wird der Algorithmus von Dijkstra verwendet, um kürzeste, kollisionsfreie Wege zu finden. Der Zeitbezug einer Bewegung wird jedoch bei ihrer Lösung ignoriert. Van den Berg et al. gehen auch davon aus, dass die Bewegungen bekannt sind, bevor der Graph erstellt wird. Im Gegensatz dazu präsentiert [78] eine neuartige Datenstruktur namens Time-Bounded Lattice, d.h. einen Graphen, in dem die Knoten verschiedene Zeitpunkte repräsentieren. Dieser Graph wird in regelmä-

ßigen Iterationen an die Bewegungsvorhersagen angepasst. [40] schlagen einen ähnlichen Ansatz vor, bei dem sie den Graphen mit Hilfe eines RRT kontinuierlich erweitern. Repetitive Diskretisierungsprozesse können in dynamischen Umgebungen sehr komplex sein. Daher gibt es alternative Datenstrukturen wie Time-Expanded Networks oder Time-Dependent Graphs, in denen wiederholte Diskretisierungsprozesse nicht notwendig sind. Der ursprünglich erstellte Graph wird lediglich um eine Zeitdimension erweitert. Das bedeutet, dass ein Graph nicht neu berechnet werden muss.

Im Verlauf dieser Arbeit werden Techniken vorgestellt, wie Agenten durch vorausschauendes Planen des Pfades Kollisionen mit statischen und dynamischen Hindernissen (d.h., auch mit anderen Agenten) vermeiden können. Deshalb soll im Folgenden zunächst eine Methode namens „Potential Field“ [68] beschrieben werden, mit welcher Kollisionen auf lokaler Ebene vermieden werden können (relevant für Kapitel 3 und 4), bevor in den folgenden Abschnitten auf künstliche neuronale Netze eingegangen wird, welche wiederum in Kapitel 3 zur Abschätzung von Objektbewegungen verwendet werden.

Das Problem der Kollisionserkennung/-vermeidung ist dem Problem der Pfadplanung sehr ähnlich, da ein korrekt geplanter Pfad in der Regel so durch den Raum verlaufe sollte, dass der Agent mit nichts auf seinem Weg kollidiert. Deshalb werden die Probleme der Pfadplanung und Kollisionserkennung häufig zusammen betrachtet bzw. ineinander übersetzt. Gerade in komplexen Szenarien, in denen die Flugbahn von dynamischen Hindernissen nicht oder nur ungenau vorhergesagt werden kann, können Kollisionen allein auf Grundlage der Pfadplanung beruhend nicht komplett ausgeschlossen werden. Deshalb kann es sinnvoll sein, die Pfadplanung und die Kollisionsvermeidung voneinander zu trennen, um die Kollisionsvermeidung auf einer niedrigeren, lokaleren Ebene durchzuführen. Da die im Folgenden dieser Arbeit vorgestellten Techniken auf das (probabilistische) Abschätzen von Hindernisbewegungen ausgelegt sind und diese in die Pfadplanung eines Agenten integrieren, können Kollisionen mit dynamischen Hindernissen nicht gänzlich ausgeschlossen.

Mit der „Potential Field“ Methode [68] können Kollisionen losgelöst von der Pfadplanung, auf lokaler Ebene verhindert werden. Die Methode ist angelehnt an das natürliche Phänomen des Elektromagnetismus, bei welchem sich Teilchen bzw. Objekte gleicher Ladung abstoßen. Dies wird mathematisch abgebildet und auf Kollisionsobjekte (wie zum Beispiel Agenten untereinander bzw. Agenten und dynamische Hindernisse) übertragen, damit sie sich wie Teilchen gleicher Ladung abstoßen und so drohende Kollisionen auf lokaler Ebene verhindern.

Die mathematische Modellierung der Potential Field Methode in ihrer einfachsten Form soll im Folgenden exemplarisch erläutert werden. Dazu werden die folgenden Variablen benötigt:

- $a \in \mathbb{R}^n$ beschreibt die Position des Agenten im Raum,
- $C \subseteq \mathbb{R}^n$ ist die Menge der Positionen der Kollisionsobjekte zu einem

Agenten,

- $z \in \mathbb{R}^n$ ist die Zielposition des Agenten im Raum,
- F_{cr} ist eine Konstante für die abstoßende Kraft,
- F_{ct} ist eine Konstante für die anziehende Kraft,
- $d(a,c)$ ist eine Funktion die die Distanz zwischen den beiden Positionen $a, c \in \mathbb{R}^n$ angibt.

Die resultierende Kraft R, die (virtuell) auf den Agenten wirkt und ihn von Kollisionsobjekten weg zum Ziel hin drückt, berechnet sich wie folgt:

$$R = F_r + F_t \quad ,$$

wobei F_r die summierten abstoßenden Kräfte der Kollisionsobjekte darstellt:

$$F_r = \sum_{n=0}^{n} F_i$$

Die einzelnen abstoßenden Kräfte ausgehend von den Kollisionsobjekten berechnen sich wie folgt:

$$F_i = F_{cr} \cdot \left(\frac{c_i - a}{d(a, c_i)} \right)$$

Zudem wirkt eine Kraft F_t auf den Agent, die eine Anziehung in Richtung eines Ziels bzw. einer Zielposition z ausübt:

$$F_t = F_{ct} \cdot \left(\frac{a - z}{d(a, z)} \right)$$

Der Agent wird durch den auf ihn wirkenden Kraftvektor in Richtung des Ziels bewegt. Dadurch könnte man annehmen, dass die Potential Field Methode selbst als Pfadplanungsalgorithmus zu diesem Ziel verwendet werden kann. Dies ist allerdings nur in sehr einfachen Szenarien der Fall, da ein Agent, der alleinig den Potential Field Ansatz zur Pfadplanung verwendet, sehr leicht in lokalen Minima der Kräfte, d.h., zwischen oder hinter Hindernissen stecken bleibt. Dies ist z.B. der Fall, wenn die abstoßenden Kräfte von Kollisionsobjekten und die anziehende Kraft des Ziels sich gegenseitig aufheben und die resultierende Kraft gleich null ist. Deshalb kommt der Potential Field Ansatz im Rahmen dieser Arbeit nur in Verbindung mit anderen Planungsalgorithmen zum Einsatz.

Um genau zu sein, kommt im Folgenden einer Weiterentwicklung des Potential Field namens Virtual Force Field (VFF) [74] zur Anwendung. Bei dieser befindet sich der Agent im Zentrum eines kreisförmigen Kraftfeldes und die anziehende Kraft wirkt kontinuierlich ins Zentrum dieses Kraftfeldes, sodass

der Agent möglichst mittig gehalten wird. Kollisionsobjekte, wie dynamische/-statische Hindernisse oder andere Agenten, haben die gleiche Ladung wie der Agent selbst und sorgen, wenn sie in einen gewissen Detektionsradius kommen, dafür, dass sich der Agent aus der Mitte des Kraftfeldes im Rahmen des Kraftfeldes bewegen kann und dadurch Kollision zwischen den Kollisionsobjekten verhindert werden, da sie sich gegenseitig abstoßen (vgl. Abbildung 4.4 in Kapitel 4).

2.5 Künstliche Neuronale Netze

Künstliche Neuronale Netze (KNN) können als biologisch inspirierte generische Funktionsapproximatoren [41] angesehen werden. Mit ihnen wird versucht, die Struktur und Funktionsweise des Gehirns vereinfacht zu imitieren und mathematische zu modellieren. Das Herzstück eines künstlichen neuronalen Netzes sind die Neuronen, welche als einfacher Verarbeitungseinheit angesehen werden können. Die Neuronen werden als Knoten in einem Graph veranschaulicht, wo sie durch gewichtete, gerichtete Kanten miteinander verbunden und in der Re-

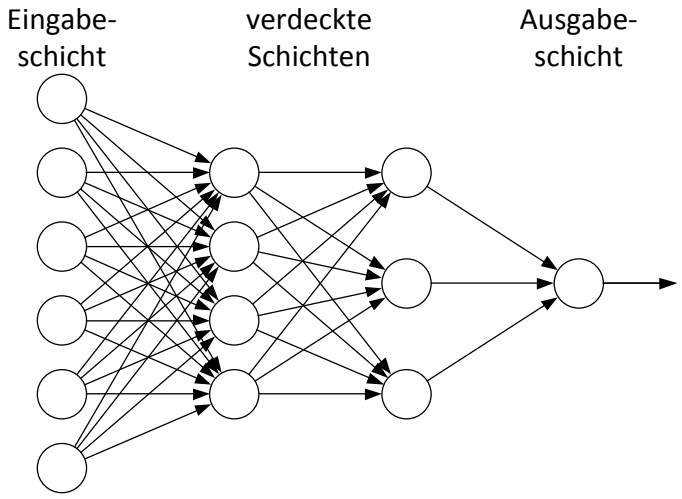


Abbildung 2.7: Beispiel eines Feedforward Netzes mit einer Eingabeschicht, einer Ausgabeschicht und zwei verdeckten Schichten dazwischen. Die Eingaben werden an der Eingabeschicht angelegt und mit den Kantengewichten (nicht eingezeichnet) verrechnet, woraus sich eine Aktivierung der Neuronen ergibt. Auf diese Aktivierung wird eine Ausgabefunktion angewendet. Das Ergebnis bildet die Eingabe der folgenden Schicht bzw. des Gesamtnetzes. Die Neuronen zwischen den Schichten sind in Vorwärtsrichtung vollvermascht. Es existieren keine Rückkanten, Rückkopplungen und es werden keine Schichten übersprungen.

gel in mehreren Schichten zu einem Netzwerk angeordnet sind (vergleichbar mit Abbildung 2.7). Sie verarbeiten die Eingaben, welche sie über ihre eingehenden Kanten erreichen und bilden diese auf einen Ausgabewert ab. Dabei kann zwischen einer Aktivierungsfunktion, welche die Eingaben verarbeitet, und einer Ausgabefunktion, welche auf die Aktivierung angewendet wird, unterschieden werden. Üblicherweise wird die Eingabe eines Neurons i als gewichtete Summe der Ausgänge der vorhergehenden Neuronen (bzw. der generellen Netzeingabe, falls es keine vorhergehenden Neuronen gibt) und der Kantengewichte, über die sie empfangen werden, berechnet und dient als dessen Aktivierung. Hinzu kommt in der Regel noch eine synthetische Eingabe $x_0 = 1$ mit entsprechendem Gewicht als Bias (siehe [45]).

$$z_i = \sum_{j=0}^{n} w_{ij} \cdot x_i + w_0$$

Auf diese Aktivierung wird dann eine Ausgabefunktion φ angewendet, welche häufig eine nichtlineare Sigmoidfunktion ist

$$o_i = \varphi(z_i) = \frac{1}{1 + e^{-c(z_i - T_i)}},\ o_i \in (0,1),$$

wobei T_i eine Reizschwelle (Threshold) und c ein variables Steigungsmaß darstellen. o_i ist damit die Ausgabe des Neurons i. Neben Sigmoidfunktionen existieren einen ganze Reihe weiterer Aufgabefunktionen bzw. sind diese Gegenstand von Untersuchungen. Eine Übersicht verschiedener Ausgabefunktionen ist in [105] gegeben, wobei diese in der Literatur, im Gegensatz zu dieser Arbeit, häufig als Aktivierungsfunktion bezeichnet werden. Neben Sigmoidfunktionen ist z.B. die Rectified Linear Unit (ReLU) Ausgabefunktion weit verbreitet [105]. Der so errechnete Wert o_i dient als aus Ausgabe des Neurons und wird an die Neuronen der nachfolgenden Schicht weitergeleitet und dient diesen als Eingabe bzw. oder als Gesamtausgang des Netzes in der letzten Schicht. Dabei sind die Neuronen zwischen den Schichten im einfachen Falle eines Feedforward Netzwerkes vollvermascht (siehe Abbildung 2.7). Diese Zusammensetzung einfacher Funktionen erlaubt die Approximation generischer Funktionen.

Training anhand von Beispielen

Wenn die Ausgabefunktion differenzierbar ist, wie es bei der genannten Sigmoidfunktion der Fall ist,

$$\varphi'(z_i) = \frac{d\varphi(z_i)}{dz} = c \cdot \varphi(z_i)(1 - \varphi(z_i)) \ ,$$

kann das Netzwerk anhand von Beispielen darauf trainiert werden eine vorgegebene Funktion zu approximieren (bekannt als überwachtes Lernen engl.

Supervised Learning). Dazu muss für eine gewisse Anzahl an Beispielen der gewünschte Ausgabewert t_i zur zugehörigen Eingabe x_i bekannt sein. Daraus lässt sich dann zunächst der Abbildungsfehler berechnen:

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - o_i)^2$$

Der so genannte Backpropagation-Algorithmus [125, 124, 12] propagiert den Fehler zwischen der Ausgabe des Netzwerks für eine bestimmte Eingabe o_i und dem bekannten gewünschten (Beispiel-)Ausgabewert t_i rückwärts durch das Netzwerk und passt die Kantengewichte an, um diesen Fehler zu minimieren. Dabei handelt es sich um ein Gradientenverfahren und die partielle Ableitung der Fehlerfunktion ergibt sich durch die Anwendung der Kettenregel.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Fügt man einen Faktor η für die Stärke der Gewichtsänderung hinzu, ergibt sich folgende Formel zur Berechnung der Gewichtsänderung:

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = -\eta\delta_j o_i$$

Hierbei muss unterschieden werden, ob es sich um ein Neuron in der Ausgabeschicht handelt oder nicht:

$$\delta_j = \begin{cases} \varphi'(z_j)(o_j - t_j) & \textit{falls } j \textit{ ein Ausgabeneuron ist,} \\ \varphi'(z_j)\sum_k \delta_k w_{jk} & \textit{falls } j \textit{ ein Neuron einer verdeckten Schicht ist.} \end{cases}$$

Dabei ist o_j die aktuelle Ausgabe, t_j der gewünschte Ausgabewert und δ_j das Fehlersignal des Neurons j. Danach können die Gewichte wie folgt aktualisiert bzw. neu zugewiesen werden.

$$w_{ij} := w_{ij} + \Delta w_{ij}$$

Das Ziel dieses Prozesses ist es, für ungesehene aber ähnliche Eingabedaten eine Annäherung der Ausgabe an den tatsächlichen Funktionswert zu erhalten. Ein Netz lässt sich durch seine Hyperparameter, wie z. B. die Anzahl der Schichten oder die Anzahl der Neuronen je Schicht, individuell auf das jeweilige Aufgabengebiet anpassen bzw. für dieses optimieren.

Rekurrente Neuronale Netze

Die zuvor beschriebenen Feedforward Netze weisen keine Rückkopplungsverbindungen auf. Für die Bearbeitung von sequentiellen Daten, wie z.B. zwecks Sprachverarbeitung oder die Vorhersage von Zeitreihen ist diese Art von Rück-

verbindungen von einem Neuron zu sich selbst aber von Vorteil. Rekurrente neuronale Netzwerke verarbeiten Eingaben elementweise und halten dabei einen versteckten, internen Zustand welcher implizit Informationen über die Historie der vergangenen Elemente der Sequenz enthält. Dies ermöglicht es, Zusammenhänge innerhalb der Sequenz zu erkennen. Rekurrente neuronale Netze unterscheiden sich von Feedforward Einheiten dadurch, dass sie Rückkopplung zu sich selbst haben und in der Zeit arbeiten, da man die Ausgabe des versteckten, intern Zustandes zu diskreten Zeitschritten interpretieren kann wie die Ausgabe von verschiedenen Neuronen in einem tiefen mehrschichtigen Netzwerk [84]. Man kann ein rekurrentes Netzwerk wie multiple Kopien desselben Netzwerkes auffassen, das Informationen jeweils zu seinem Nachfolger weitergibt. Stellt man sich die Anwendung des rekurrenten Netzwerkes auf jedes Element einer Sequenz vor, kann man anschaulich die Berechnung des Netzes über die Zeit ausrollen (siehe Abbildung 2.8), was einen Hinweis darauf gibt, wie der Backpropagation Algorithmus zum Training rekurrenter Netzwerke verwendet werden kann.

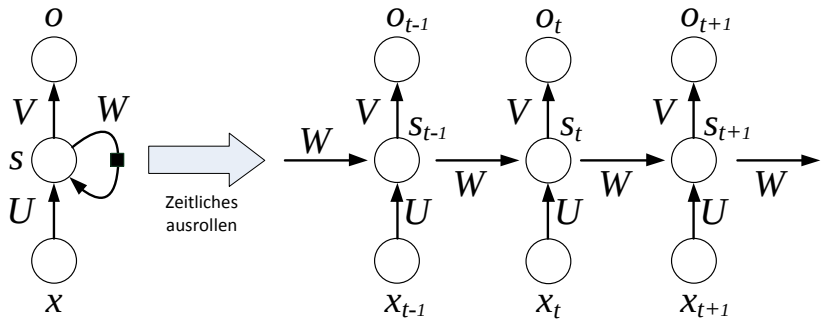


Abbildung 2.8: Beispiel eines rekurrenten neuronalen Netzes und des Ausrollens von dessen Berechnung mehrerer Eingaben über die Zeit. In Anlehnung an [84].

Die Abbildung zeigt ein verdecktes Neuron s mit einem Wert von s_t zum Zeitpunkt t. Mit jedem Forward Pass durch die rekurrenten Einheiten wird dieser interne Zustand aktualisiert, da die rekurrenten Einheiten z.B. eine Sigmoidfunktion auf die sie erreichenden Eingaben anwenden, woraus sich eine Art Speicher ergibt. Das rekurrente neuronale Netz bildet eine Eingabesequenz mit Elementen x_t auf eine Ausgabesequenz mit den Elementen o_t ab, wobei jedes Ausgabeelement o_t von allen vorangegangenen $x_{t'}$ abhängt (mit $t' \leq t$). Dabei kommen bei jedem Zeitschritt die gleichen Parameter (Matrizen U, V, W) zum Einsatz. Alle Parameter (Gewichtsmatrizen) werden während des Trainings angepasst. U stellt dabei die Gewichte der Verbindungen von der vorangegangen Schicht (bzw. Eingabeschicht) zur verdeckten Schicht dar. V sind die Gewichte der Verbindung von der verdeckten zur Ausgabeschicht. W nimmt eine Sonderstellung ein und bezeichnet die Gewichte der rekurrenten Verbin-

dungen von der verdeckten Schicht zurück zur verdeckten Schicht, wodurch das Feedback des Netzes an sich selbst bestimmt wird [45]. Neben der in Abbildung 2.8 gezeigten Architektur, welche für jeden Zeitschritt eine Ausgabe produziert, sind noch viele andere denkbar, wie z.B. eine solche, die erst am Ende einer gesamten Eingabesequenz eine Ausgabe produziert, um z.B. eine Zusammenfassung der Sequenz zu liefern [84].

Ein rekurrentes neuronales Netz simuliert ein dynamisches System in diskreten Zeitschritten. Ein Netz mit einem Eingabevektor x_t , einer Ausgabe y_t und einem versteckten Zustand s_t kann durch

$$s_t = f_s(x_t, s_{t-1}) = \phi_s \left(W^\top s_{t-1} + U^\top x_t \right)$$
$$y_t = f_o(s_t, x_t) = \phi_o \left(V^\top s_t \right),$$

beschrieben werden, wobei f_s und f_o eine Zustandsübergangs- und Ausgabefunktion repräsentieren bzw. ϕ_s und ϕ_o elementweise nicht-lineare Funktionen sind. Es ist üblich, eine sättigende nichtlineare Funktion wie die logistische Sigmoidfunktion oder die hyperbolische Tangensfunktion für ϕ_s zu verwenden [110].

Der zuvor beschriebene Backpropagation-Algorithmus kann direkt auf den Berechnungsgraphen des über die Zeit ausgerollten rekurrenten Netzwerkes in Abbildung 2.8 angewendet werden, um die Ableitung eines Gesamtfehlers in Bezug auf alle Zustände s_t und alle Parameter zu berechnen, da das über die Zeit ausgerollte rekurrente Netz als tiefes Feedforward Netz betrachtet werden kann. Allerdings hat sich gezeigt, dass es für die hier beschriebenen grundlegenden rekurrenten neuronalen Netze schwierig ist, langfristige Zusammenhänge über mehrere Zeitschritte zu lernen, da die Gradienten dazu tendieren, entweder zu verschwinden oder zu explodieren. Zudem ist problematisch, dass auf langfristige Zusammenhänge exponentiell geringere Gewichtung gelegt wird als auf kurzfristige [84]. Lösungsansätze sind die Anwendung von Optimierungsalgorithmen, die den Gradientenvektor auf einem bestimmten Niveau halten, was zwar das erste aber nicht das zweite Problem löst. Ein anderer Ansatz ist die Nutzung sogenannter Gating Units wie in LSTM.

Die Long Short Term Memory (LSTM) Einheiten wurden 1997 von Sepp Hochreiter und Jürgen Schmidhuber in einer Veröffentlichung vorgestellt und entwickelt, um das Problem der verschwindenden Gradienten bei Langzeitabhängigkeiten zu lösen [63]. Die Idee besteht darin, das Netzwerk um ein explizites Gedächtnis zu erweitern. LSTM Zellen zeichnen sich dadurch aus, dass sie mit speziellen *Gating Units* versehen sich, die den Fluss von Informationen durch das Netz steuern. Dies sind ein sogenanntes Input Gate, ein Output Gate und ein Forget Gate. Das Input Gate steuert das Ausmaß, in dem ein neuer Wert in die Zelle fließt, das Forget Gate steuert das Ausmaß, in dem ein Wert in der Zelle verbleibt, und das Ausgangsgatter steuert das Ausmaß, in dem der Wert in der Zelle verwendet wird, um die Ausgangsaktivierung der LSTM-Einheit zu berechnen. Die Aktivierungsfunktion der LSTM-Gatter ist

häufig die logistische Sigmoidfunktion. Wenn die Gates geschlossen sind (Aktivierung um Null), gelangen keine irrelevanten Eingänge und kein Rauschen in die Zelle, und der Zustand der Zelle stört den Rest des Netzwerks nicht.

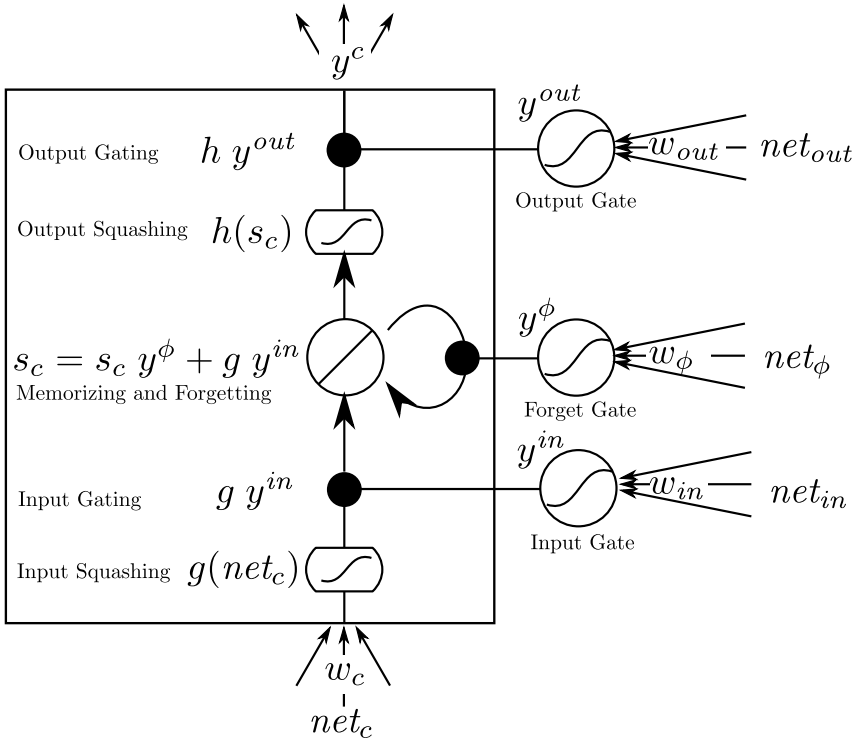


Abbildung 2.9: Beispiel einer LSTM Zelle. Quelle: [43]

Abbildung 2.9 zeigt eine LSTM Zelle. Jede Zelle besteht aus einer linearen Einheit mit einem internen Zustand s_c und einer Feedback-Verbindung zu sich selbst. Diese Einheit dient als interner Speicher der Zelle und ist dafür verantwortlich, die Abhängigkeiten zwischen den Elementen in der Eingabefolge zu verfolgen. Hinzu kommt das Input Gate, das s_c vor unwichtigen Eingaben schützt. Das Output Gate schützt andere Einheiten davor, dass sie durch irrelevante Informationen gestört werden, die in s_c gespeichert sein könnten. Diese drei Einheiten zusammen bilden die ursprüngliche Speicherzelle des LSTM. [63] Ein Problem einer ursprünglichen LSTM Einheit ist, dass der interne Zustand der Zelle s_c linear anwächst, bis hin zur Sättigung der Ausgabefunktion h. Das hat zur Folge, dass die Steigung von h komplett verschwindet und eingehende Fehler blockiert. Damit entspricht die Ausgabe der Zelle der Aktivierung des Output Gates, und das LSTM degeneriert zu einer normalen RNN Zelle. Um das zu verhindern, schlagen Gers et al. [43] ein Forget Gate vor, mit dem die Zellen lernen können, den internen Zustand zurückzusetzen, wenn die gespeicherten Informationen veraltet und damit unbrauchbar erscheinen. Mit Zurücksetzen ist aber nicht nur der sofortige Reset auf null gemeint, sondern auch graduelle Resets, die langsam verblassenden Zellzuständen entsprechen.

Der Zellenzustand s_c, wird dementsprechend auf der Grundlage seines aktuellen Zustands und drei Eingangsquellen aktualisiert: net_c ist die Eingabe zur Zelle selbst, während net_{in} und net_{out} Eingänge für das Input und Output Gate sind. Die Gewichte der Gates, die während des Trainings erlernt werden müssen, bestimmen die Funktionsweise der Gates. Mit dem zusätzlichen Forget Gate sind die erweiterten LSTMs in der Lage, sogar mit Eingabesequenzen zu arbeiten, die nicht a priori in Teilsequenzen segmentiert wurden und kein explizit definiertes Ende besitzen, an welchem der internen Zustand zurückgesetzt werden könnte.

2.6 Reinforcement Learning

Reinforcement Learning (dt: bestärkendes Lernen) bezeichnet ein maschinelles Lernparadigma, bei welchem es darum geht, dass ein Agent durch die Interaktion mit seiner Umgebung lernt, eine ihm gestellte Aufgabe selbständig zu lösen. Die Aufgabe besteht darin, Situationen bzw. Zustände auf Aktionen abzubilden und ein dafür erhaltenes numerisches Belohnungssignal zu maximieren. Der lernende Agent bekommt dabei nicht vorgegeben, welche Aktion in welchem Zustand aufzuführen ist, sondern muss durch Ausprobieren herausfinden, welche Aktion die größte Belohnung bringt, d.h. er kennt die Belohnungsfunktion nicht, sondern kann sie nur abtasten. In den interessanten und herausfordernden Fällen des Reinforcement Learning wirken sich Aktionen nicht nur auf die unmittelbare Belohnung aus, sondern auch auf den nächsten Zustand und damit auf alle nachfolgenden Belohnungen. Diese beiden Merkmale - Trial-and-Error-Suche und verzögerte Belohnung - sind die beiden wichtigsten Unterscheidungsmerkmale des Reinforcement Learning [134]. Reinforcement Learning unterscheidet sich von Supervised Learning in der Art, dass beim *Supervised Learning* in der Regel ein mit Labeln bzw. Bezeichnern versehener Trainingsdatensatz vorliegt, d.h. von einem erfahrenen externen Lehrer vorgegeben. Dies sind häufig Klassifikationsaufgaben, bei denen eine Kategorie bestimmt werden muss, zu der die vorgelegten Eingabedaten gehören. Das Ziel dieser Art des Lernens besteht darin, zu extrapolieren bzw. generalisieren, um so auch Daten richtig zu klassifizieren, die nicht im Trainingsdatensatz enthalten waren. Bezogen auf Problemstellungen des Reinforcement Learnings würde dies bedeuten, für alle Situationen, in denen ein Agent handeln muss, korrekte und repräsentative Handlungsvorgaben zu erstellen. Dies kann zum einen sehr aufwendig sein aber ist vor allem in unbekannten Terrain, wo eigenständiges Lernen vermutlich besonders nützlich wäre, nicht möglich. Ein Agent soll in der Lage sein aus seiner eigenen Erfahrung durch die Interaktion mit der Umwelt zu lernen [134]. Deshalb besteht die Aufgabe eines Entwicklers beim Reinforcement Learning, neben dem Reinforcement Learning Algorithmus selbst, darin, den Zustand der Umgebung, den der Agent wahrnimmt, zu beschreiben bzw. diesen zugänglich zu machen. Aber vor allem muss das von ihm erhaltene Belohnungssignal in der Art geformt sein, dass der Agent auch wirklich das vom

Entwickler intendierte Verhalten lernt. Reinforcement Learning unterscheidet sich aber auch vom sogenannten *Unsupervised Learning*, wobei es in der Regel darum geht versteckte Strukturen bzw. Cluster in nicht markierten (unlabeled) Daten zu finden. Obwohl Reinforcement Learning nicht auf Beispielen von korrektem Verhalten beruht, wie es beim Supervised Learning der Fall ist, fällt Reinforcement Learning auch nicht in das Unsupervised Learning Paradigma. Das Aufdecken von Strukturen in der gesammelten Erfahrung eines Agenten kann beim Reinforcement Learning nützlich sein, reicht aber allein noch nicht, um das Lernproblem beim Reinforcement Learning, das Maximieren eines Belohnungssignals, zu lösen. Deshalb kann Reinforcement Learning als drittes Machine Learning Paradigma neben Supervised und Unsupervised (und ggf. noch weiteren Paradigmen) angesehen werden. [134] Ein weiterer Aspekt beim Reinforcement Learning, der aber bei anderen Paradigmen in dieser Form nicht auftritt, ist die Abwägung zwischen *Exploration* und *Exploitation*. Wenn ein Agent am Anfang des Lernprozesses steht, hat er in der Regel kein Wissen darüber, welche Aktion in einem gewissen Zustand die meiste Belohnung für ihn bringt. Es bleibt ihm also nichts anderes übrig, als den Zustands- und Aktionsraum zu explorieren. Kommt er allerdings erneut in einen Zustand, in dem er sich bereits einmal befunden hat, steht er vor dem Dilemma die gleiche Aktion erneut auszuführen (deren unmittelbare Belohnung er kennt) oder eine neue auszuprobieren, die möglicherweise eine höheren direkten oder zukünftige (über die erreichten Folgezustände) Belohnung verspricht aber auch schlechter als bisher bekannt ausgehen kann. Der Agent muss seine bisherige Erfahrung ausnutzen, um Belohnung zu sammeln aber auch neue Aktionen ausprobieren, um potenziell bessere Verhaltensstrategien zu finden. In einer stochastischen Umgebung muss er unter Umständen eine Aktion in einem Zustand mehrfach ausprobieren, um eine verlässliche Abschätzung über die zu erwartende Belohnung zu erhalten. Für das Exploration-Exploitation-Dilemma wurden viele Strategien vorgeschlagen und es stellt einen eigenen Forschungszweig innerhalb des Reinforcement Learnings dar.

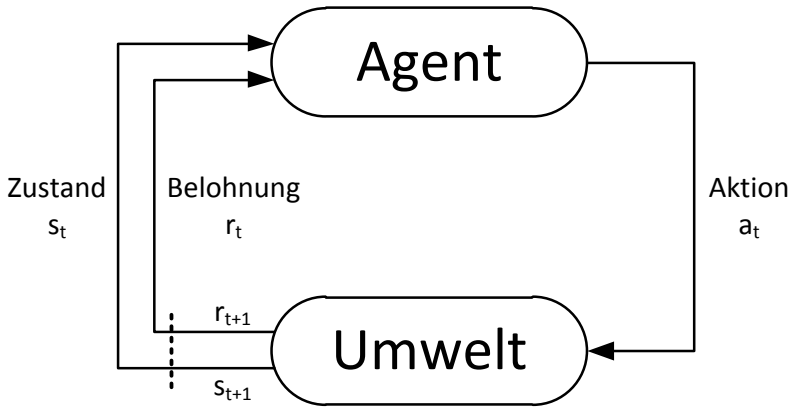


Abbildung 2.10: Beispiel des allgemeinen Reinforcement Learning Kreislaufs

Reinforcement Learning Probleme werden gewöhnlich als Markov-Entscheidungsproblem (engl.: Markov Decision Process (MDP)) modelliert. Dabei bezeichnet $\mathcal{S}$ die Menge der Zustände (engl: States) der Umgebung, $\mathcal{A}$ die Menge der Aktionen (engl. Actions), die ein Agent ausführen kann, und $r(s_t, a_t)$ ist die Belohnung (engl. Reward), die er nach der Ausführung der Aktion a_t im Zustand s_t im Zeitschritt t erhält (siehe Abbildung 2.10). Außerdem geht der Prozess in einen neuen Zustand s_{t+1} über, der von der Aktion a_t beeinflusst wird, wobei die Markov-Eigenschaft darin besteht, dass die Wahrscheinlichkeit des Übergangs in den Zustand s_{t+1} nur vom Zustand s_t und der gewählten Aktion a_t abhängt: $\mathcal{P}(s_{t+1}|s_t, a_t)$. Das Ziel ist es, eine Verhaltensstrategie (engl.: Policy) zu finden $\pi : \mathcal{S} \rightarrow \mathcal{A}$ die die akkumulierte Belohnung maximiert $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ vom Zeitschritt t bis zum Simulationshorizont T mit einem Diskontierungsfaktor (Abschwächungsfaktor) $\gamma \in [0,1]$. Der Diskontierungsfaktor beschreibt die Präferenz eines Agenten für aktuelle Belohnungen gegenüber zukünftigen Belohnungen. Ist er nahe 0, werden zukünftige Belohnungen als unbedeutend angesehen. Wird $\gamma = 1$ gewählt, muss sichergestellt sein, dass die Umgebung über einen Endzustand verfügt, welcher vom Agenten garantiert erreicht wird. Ist dies nicht der Fall, werden die Aktions- bzw. Zustandsfolgen unendlich lang, womit auch die akkumulierte Belohnung unendlich wird und sich Aktions-/Zustandssequenzen nicht mehr vergleichen lassen. Alternativ könnte mit einer durchschnittlichen Belohnung pro Zeitschritt gearbeitet werden. Im Allgemeinen wird $\gamma < 1$ gewählt, um das angesprochene Problem zu umgehen [126].

In einer Verallgemeinerung des Markov-Entscheidungsprozesses ist der Zustand nur teilweise beobachtbar (Partially Observable Markov Decision Process (POMDP)), d.h. anstatt die vollständige Zustandsbeschreibung s_t zur Bestimmung der Aktion $a_t = \pi(s_t)$ zu haben, steht dem Agent nur eine Beobachtung (engl. Observation) $o_t \in \mathcal{O}$ (wobei $\mathcal{O}$ der Raum aller möglichen Beobachtungen ist) als Input der Policy $\pi : \mathcal{O} \rightarrow \mathcal{A}$ zur Berechnung der Aktion $a_t = \pi(o_t)$ zur Verfügung. Dabei kann die Beobachtung für jeden Agenten unterschiedlich sein. Es kann noch unterschieden werden, ob die Domäne deterministisch oder probabilistisch ist. Im Folgenden kommen nur deterministische Domänen zum Einsatz, also $\mathcal{P}(s_{t+1}|s_t, a_t) \in \{0, 1\}$.

2.6.1 Deep Learning

Beim Reinforcement Learning werden die Verhaltensstrategie oder Zwischenfunktionen, die zu ihrer Ableitung beitragen, gewöhnlich durch tiefe künstliche neuronale Netze repräsentiert. Künstliche neuronale Netze dienen als biologisch inspirierte Funktionsapproximatoren, die beispielhaft trainiert werden können, um eine Funktion f zu approximieren, indem sie einen Eingangsvektor $x \in \mathbb{R}^n$ auf einen Ausgangsvektor $o \in \mathbb{R}^m$ abbilden, abhängig von den Gewichten der Kanten θ. Ziel beim Training eines neuronalen Netzes ist es, den Fehler zwischen der Ausgabe $o = f(x; \theta)$ des Netzes und der bekannten gewünschten

(Beispiel) Ausgabe t durch entsprechende Anpassung der Gewichte θ zu minimieren. Dies kann mit der Backpropagation-Methode in Kombination mit einem Gradientenabstiegsverfahren erreicht werden. Neuronale Netze können als ein gerichteter Graph von Knoten, Neuronen genannt, die durch gewichtete Kanten miteinander verbunden sind, dargestellt werden. Ein Neuron erhält Eingaben über seine Eingangskanten, berechnet normalerweise die gewichtete Summe der Eingaben, wendet auf diese gewichtete Summe eine nichtlineare Funktion an und leitet seine Ausgabe über seine Ausgangskanten an nachfolgende Neuronen weiter. Die Neuronen sind normalerweise in Schichten angeordnet, wobei Schichten zwischen der Eingangsschicht und der Ausgangsschicht des Netzwerks als verborgene Schichten bezeichnet werden. Netzwerke mit mehreren verborgenen Schichten zwischen Eingabeschicht und Ausgabeschicht werden als tiefe neuronale Netzwerke bezeichnet. Frühe Arbeiten haben gezeigt, dass sich anspruchsvolle Aufgaben nicht mit einem einschichtigen neuronalen Netz, in seiner damaligen Form noch Perzeptron genannt, lösen lassen [103]. Seitdem hat sich das Deep Learning mit tiefen künstlichen neuronalen Netzen als eigene Klasse von Machine-Learning-Algorithmen entwickelt, mit Anwendungen im Supervised bzw. Unsupervised Learning, und wie im Folgenden dargestellt, im Reinforcement Learning [26].

2.6.2 Deep Q-Learning (DQN)

Q-Learning ist ein Ansatz, der nach der Bewertungsfunktion $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ benannt ist. Diese beschreibt die erwartete akkumulierte Belohnung $Q^\pi(s_t, a_t)$ nach dem Ergreifen von Aktion a_t im Zustand s_t und dem Befolgen der Verhaltensstrategie π in allen nachfolgenden Zuständen. Q-Learning orientiert sich rein an den Werten der Zustands-Aktions-Paare und benötigt kein Modell der Umwelt bzw. Information über die Übergangswahrscheinlichkeiten zwischen Zuständen. Das Ziel ist es, eine optimale Bewertungsfunktion Q^* zu finden, die die höchste akkumulierte Belohnung ergibt. Q^* kann durch das Bellman'sche Prinzip angenähert werden, das auf der Intuition beruht, dass für eine optimale Verhaltensstrategie. unabhängig vom Ausgangszustand und der ursprünglichen Entscheidung, alle verbleibenden Entscheidungen eine optimale Verhaltensstrategie in Bezug auf den aus der ersten Entscheidung resultierenden Zustand darstellen müssen ([9]). Ausgehend von einer anfänglichen Schätzung für Q kann sie iterativ über

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

aktualisiert werden, wobei die Lernrate $\alpha \in (0, 1)$ ein zu spezifizierender Parameter ist. Die erlernte Bewertungsfunktion Q konvergiert unter gewissen Bedingungen zu Q^* [142, 141], woraus eine optimale Verhaltensstrategie über $\pi^*(s_t) = \arg\max_a Q(s_t, a)$ abgeleitet werden kann.

Im Deep Q-Learning (DQN) ([99]) wird ein künstliches neuronales Netz ver-

wendet, um die Bewertungsfunktion Q darzustellen, was vor allem bei großen Zustands- und Aktionsräumen Vorteile bietet. Allerdings ist die Konvergenz von Q-Learning unter Verwendung solcher Funktionsapproximatoren nicht mehr gegeben [91]. Um Korrelationen zwischen den Stichproben von Zuständen, Aktionen und erhaltener Belohnung zu minimieren und nicht-stationäre Verteilungen zu mildern, wird ein Speichermechanismus (engl.: Replay Buffer) verwendet ([99]), aus welchem zufällig frühere Zustands-Aktionsübergänge ausgewählt werden, um das neuronale Netz zu trainieren.

2.6.3 Deep Deterministic Policy Gradient (DDPG)

Um die Beschränkung des Q-Learnings, welches nicht direkt auf kontinuierliche Aktionsräume angewendet werden kann, zu überwinden wurde versucht, die Verhaltensstrategie (Policy) direkt mit einer parametrisierten Zielfunktion zu erlernen ([131, 91]). Dazu wird ein hybrider Ansatz namens Deep Deterministic Policy Gradient (DDPG) [91] vorgeschlagen, bei dem ein sogenannter *Actor* die aktuelle Policy $\mu(s|\theta^\mu)$ angibt, welche deterministisch Zustände der Umgebung auf auszuführende Aktionen (mit reellwertigen Zahlenbereich) abbildet. θ^μ bezeichnet dabei die Parameter der Abbildung, was in der Regel die Gewichte eines künstlichen neuronalen Netzes sind. Mit Hilfe gewonnener Erfahrung aus Zustand, gewählter Aktion und erhaltener Belohnung $[s_t,a_t,r_t,s_{t+1},a_{t+1},r_{t+1},\ldots]$ ist es möglich einen sogenannten *Critic* mittels des zuvor beschriebenen Deep Q-Learnings unter Verwendung der Bellman-Gleichung zu trainieren. Die so gewonnene wertbasierte Q-Funktion $Q(s_t, a_t)$ des Critic kann aufgrund ihrer Beschaffenheit nicht direkt verwendet werden, um in einem gewissen Zustand s_t eine reellwertige Aktion a_t zu wählen (allenfalls durch abtasten/ausprobieren diskreter Aktionswerte). Sie liefert dem Actor aber Informationen darüber, in welche Richtung er seine Aktionen bzw. die Policy-Parameter θ^μ anpassen muss, um die erwartete kumulative Belohnung zu maximieren. Damit ist es möglich eine Verhaltensstrategie zu erlernen, welche Zustände, die von einem Agenten wahrgenommen werden, auf auszuführende, reellwertige Aktionen abbildet.

2.6.4 Multi-Agenten-Fall

Sind multiple Akteure anwesend, die auf die Erfüllung eines gewissen Ziels hin trainiert werden sollen, fällt dies in den Forschungsbereich des Multi-Agent Reinforcement Learnings (MARL), wofür verschiedene Ansätze bestehen. Die Szenarien können danach unterschieden werden, ob die Agenten entweder ein Eigeninteresse haben oder zusammenarbeiten sollen, um ein kooperatives Ziel zu erreichen. Ein direktes Vorgehen für den Fall, dass mehrere Agenten in Eigeninteresse handeln sollen, so dass sie nur ihre eigene akkumulierte Belohnung maximieren, ist es, einen gewöhnlichen Reinforcement Learning Ansatzes für den Einzelagentenfall auf jeden Agenten im Multiagentenszenario anzuwen-

den, so dass alle parallel aber für sich alleine lernen. Dieser geradlinige Ansatz birgt das Problem der Nichtstationarität in den Zustandsübergängen. Wenn ein Agent versucht, seine Aktionen in bestimmten Zuständen anzupassen, tun dies andere Agenten ebenfalls, während sie als Teil der Umgebung für den ersten Agenten betrachtet werden. Das macht es schwierig, eine vom beobachteten Zustand abhängige Strategie zu erlernen, weil das Setting die Markov-Eigenschaft nicht mehr erfüllt und der Übergang von einem Zustand zum nächsten nicht mehr rein vom vorangegangenen Zustand abhängt, sondern von der Historie an Erfahrungen, welche die einzeln Agenten gemacht haben.

Egorov behandelt ein Verfolgungs-/Ausweichspiel mittels Reinforcement Learning aber ebenfalls mit Ansätzen, welche eigentlich zum Training eines einzelnen Agenten gedacht sind [29]. Im Szenario gibt es mehrere Verfolger und mehrere Ausweicher. Von jeder Art wird jeweils nur ein Agent durch Deep Q-Learning trainiert, während die Strategie der anderen Agenten fix bleibt. Nach einer Reihe von Iterationen wird die Strategie des lernenden Agenten auf alle anderen Agenten desselben Typs verteilt. Durch diesen Prozess wird die Strategie der zwei Mengen von Agenten im Laufe der Zeit schrittweise verbessert.

Dadurch wird das Problem der Nichtsstationarität abgemildert. Darüber hinaus erscheint es sinnvoll, die Verhaltensstrategie eines Agenten auf mehrere homogene Agenten zu kopieren, da alle gleich sind und dasselbe eigennützige Ziel verfolgen. Diese Beobachtung ist auch für das Schwarmverhalten mehrerer Agenten von Bedeutung, wie in Kapitel 5 und 6 gezeigt wird.

2.7 Zusammenfassung

In diesem Kapitel wurden Grundlagen erläutert und Begrifflichkeiten eingeführt, welche für die folgenden Kapitel dieser Arbeit von Bedeutung sind.

Dies umfasst zunächst die Definition von Graphen und klassischen Pfadplanungsalgorithmen. Graphen sind in nahezu jedem Kapitel dieser Arbeit von Bedeutung und Pfadplanung kommt insbesondere in Kapitel 3 und 4 zum Einsatz. Ebenso wurden intelligente Agenten erläutert und weil sich diese im Rahmen dieser Arbeit in kontinuierlichen zweidimensionalen Flächen bewegen, wurde auf einfache Ansätze zur Diskretisierung dieser Freiflächen eingegangen. Im Zuge dessen wurden auch einfache Strategien zur Kollisionsvermeidung erläutert. Da im Verlauf dieser Arbeit auf fortgeschrittene Techniken zur Kollisionsvermeidung eingegangen wird, welche auf künstlichen neuronalen Netzen basieren, wurde in diesem Kapitel dafür ebenfalls die Grundlage geschaffen. Da im Kapitel 5 und 6 Techniken wie DQN oder DDPG zur Steuerung ganzer Schwärme von Agenten genutzt werden, wurde in diesem Kapitel auf diese Reinforcement Learning Methoden eingegangen.

3 Bewegung eines Agenten in seiner Umgebung

In diesem Kapitel geht es darum, dass sich ein einzelner Agent auf einer zweidimensionalen Freifläche bewegen und dabei unterschiedliche Ziele ansteuern soll. Während seiner Bewegung darf er nicht mit statischen oder sich ebenfalls bewegenden dynamischen Hindernissen kollidieren. In Kapitel 2 wurde mit der Potential Field Methode bereits ein Ansatz vorgestellt, wie Kollisionen auf lokaler Ebene vermieden werden können. In diesem Kapitel soll ein Agent jedoch mit einem eigens gelernten Modell der Umwelt ausgestattet werden, womit sich Kollisionen bereits auf globaler Ebene während der Pfadplanung vermeiden lassen. Nachdem in Abschnitt 3.1 bereits veröffentlichte Ergebnisse dieses Kapitels angesprochen und das Kapitel in Abschnitt 3.2 weiter motiviert wurden, wird in Abschnitt 3.3 auf Techniken der intuitiven Physik eingegangen. Die Techniken beruhen darauf, dass ein Agent mit Hilfe künstlicher neuronaler Netze lernt, physikalische Interaktionen und Bewegungsabläufe von Objekten in seiner Umgebung vorherzusagen. Deshalb wird in Abschnitt 3.3 eine durchgeführte Voruntersuchung vorgestellt, welche zeigt, dass dies generell möglich ist. Darauf aufbauend wird in Abschnitt 3.4 ein Agententyp entwickelt, der mittels eines solchen erlernten Umgebungsmodells ausgehend von seinem aktuellen Zustand baumartig mögliche Folgezustände vorausplant und damit für ihn möglichst günstige Handlungsfolgen findet. Es wird gezeigt, dass ein Umgebungsmodell, welches auf künstlichen neuronalen Netzen basiert neben der Flexibilität vor allem in der Berechnungsgeschwindigkeit Vorteile gegenüber einer exakten Physiksimulation bietet. Der Planungsprozess des Agenten aus Abschnitt 3.4 lässt sich noch effizienter gestalten, da Simulationen der Umgebungen zum Teil redundant ausgeführt werden. Deshalb wird in Abschnitt 3.5 ein darauf aufbauender Ansatz namens Predictive Collision Management Path Planning konzipiert, bei dem Vorhersagen der künstlichen neuronalen Netze, namentlich LSTM Netze, über zeitabhängige Kantenkosten in einen Graph integriert werden. Dadurch können mit einem eigens angepassten Pfadplanungsalgorithmus Pfade in einer Raum-Zeit-Dimension bestimmt werden, die den Kontakt mit Kollisionsobjekten von vornherein vermeiden. Da die in diesem Kapitel angesprochenen Ansätze sich nur auf einen einzelnen Agenten beziehen, behandeln folgende Kapitel Szenarien mit multiplen Agenten bzw. ganze Schwärme aus diesen.

3.1 Vorveröffentlichungen

Die Kerninhalte dieses Kapitels wurden vom Autor bereits in [52] und in [53] publiziert. Dabei umreißt [52] das grundlegende Konzept, Ansätze der intuitiven Physik im Planungsprozess intelligenter Agenten zu verwenden, um so z.B. Hindernissen auszuweichen bzw. dynamische Objekte anzusteuern. In [53] wird dieses Konzept dann konkret umgesetzt und evaluiert. Wie in Abschnitt 1.3 dargestellt, stammt die Idee bzw. das in diesem Kapitel erläuterte Konzept vom Autor, der auch den Hauptanteil an den genannten Veröffentlichungen trägt.

In der Publikation [53] sind bereits die Tabelle 3.1 sowie die Abbildungen 3.1, 3.4, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12 und 3.13 enthalten. Die Abbildungen 3.2, 3.3a und 3.3b, als Vorstudien zur Eignung von Methoden der intuitiven Physik, wurden zuvor noch nicht veröffentlicht und gehen über die genannten Publikationen hinaus.

Die Abbildungen 3.18, 3.21, 3.25, 3.26, 3.30a und 3.30b wurden bereits in [54] veröffentlicht. In der erweiterten Version dieser Veröffentlichung [55] sind die Abbildungen 3.14, 3.15, 3.16, 3.17, 3.19, 3.20, 3.22a, 3.22b, 3.23, 3.24, 3.27, 3.28a, 3.28b, 3.29a, 3.29b, 3.31 und 3.32 enthalten, in welcher auch die Tabellen 3.2, 3.3, 3.4, 3.5, 3.6 und 3.7 zu finden sind.

3.2 Motivation

Künstliche Intelligenz wird unter anderem mit Hilfe intelligenter Agenten [126] beschrieben. Ein intelligenter Agent ist eine autonome Einheit, d.h. Hard- und Software, die nicht unter direkter menschlicher Kontrolle steht, die Sensoren zur Wahrnehmung der Umgebung und Aktoren zur Erreichung eines bestimmten Ziels einsetzt. Es gibt zahlreiche Beispiele für derzeit existierende intelligente Agenten, die sich nicht nur in der Anwendungsdomäne, sondern auch im „Grad“ der Intelligenz unterscheiden. So kann sogar ein Thermostat als ein intelligenter Agent angesehen werden: er verwendet ein Thermometer, um die Umgebung zu messen und steuert die Heizung, um eine bestimmte Temperatur zu erreichen oder zu halten [126]. Ein weiteres Beispiel könnte ein Aktienhandelsagent [111, 3] sein, der die Umwelt anhand historischer Aktienkursentwicklungen wahrnimmt und auf dem Markt agiert, indem er Aktien kauft oder verkauft, unter der Einschränkung ein bestimmtes Budget auszugeben, mit den Zielen, das Risiko zu minimieren und den Ertrag zu maximieren. Ein anspruchsvolleres Beispiel ist ein autonomer mobiler Roboter [31], wie ein Staubsaugroboter, der seine räumliche Umgebung mit Abstandssensoren misst und mit Hilfe eines Antriebsmechanismus agiert, mit dem Ziel die ganze Wohnung zu durchqueren. Es gibt ein breites Spektrum von Anwendungsfällen für intelligente Agenten, die einen räumlichen Kontext einbeziehen, wie selbstfahrende Autos [88], Nicht-Spieler-Charaktere in Computerspielen [79] oder mobile Roboter im Bereich der intelligenten Fabriken [132].

Eine weitere Möglichkeit intelligente Agenten zu unterscheiden besteht darin, sich auf das internes Modell der Umgebung zu konzentrieren, mit dem sie arbeiten. Dieses Modell wird oft als das „Wissen des Agenten darüber, wie die Welt funktioniert“ [126] bezeichnet. Andere Unterscheidungen sind das Ausmaß, in dem ihre Handlungen von früheren Zuständen abhängen, oder die Wahrscheinlichkeit, dass ihre Handlungen zum gewünschten Ergebnis führen. In den einfachsten Anwendungsfällen ist der Agent in der Lage die Umwelt vollständig zu beobachten und seine Aktionen hängen nur vom aktuellen Zustand des Systems ab. Leider kann dies nicht immer angenommen werden. Stattdessen ist die Umgebung oft nur teilweise beobachtbar, die Aktionen des Agenten können von der Geschichte früherer Aktionen abhängen, oder die Aktionen des Agenten können von den Aktionen weiterer Agenten abhängen. Eine Variante, für das Treffen einer intelligenten Entscheidung durch einen Agenten besteht darin, ihn explizit mit einem Modell seiner Umgebung auszustatten. Dieses Modell wird verwendet, um die bevorstehenden Aktionen des Agenten zu simulieren und das jeweilige Ergebnis durch die Maximierung der erhaltenen Belohnung zu bewerten. Je umfangreicher und genauer das Modell ist, desto präziser schätzt der Agent das Ergebnis einer Handlungssequenz ein. Daher ist die Darstellung der Umgebung des Agenten ein entscheidendes Element für seinen Erfolg. Es muss im Vorfeld ein qualitativ hochwertiges Modell erstellt werden, das möglicherweise nicht flexibel auf Veränderungen in der Umgebung reagiert. Darüber hinaus kann das resultierende Modell bzw. Regelwerk komplex sein, was zu rechenintensiven Simulationen von Handlungsabläufen - oft als Pläne bezeichnet - führen kann.

Die Grundidee dieses Kapitels besteht darin, Wissen über die Umwelt (d.h. das Modell) durch ein intuitives Verständnis der Umwelt, ihrer Objekte und deren Beziehungen zu ersetzen. Intuitives Verstehen – oder Intuition – bedeutet, in der Lage zu sein, Einsichten in Fakten, Gesetze und die subjektive Kohärenz von Entscheidungen ohne bewusste Argumentation zu gewinnen. Ein einfaches Beispiel ist die Abschätzung der Flugbahn eines geworfenen Balls. Ein Mensch wird nicht die Flugbahn (Verhältnis von Geschwindigkeit, Ballgewicht, Luftwiderstand, Gravitation und mehr) berechnen, sondern aus Erfahrung eine subjektive kohärente Entscheidung darüber treffen, wo der Ball auf den Boden trifft. Der Schwerpunkt dieses Kapitels liegt daher auf der Untersuchung der Frage, ob das Konzept der Intuition in einen autonom handelnden Agenten integriert werden kann. Ziel ist es, ein vordefiniertes und hart kodiertes Modell durch eine Art erlernte Intuition zu ersetzen. Folglich wird die Simulation und Bewertung von Plänen nicht mehr auf komplexen Berechnungen, sondern auf flexiblen Einschätzungen und Erfahrungen beruhen.

Der Beitrag dieses Kapitels besteht in der Untersuchung der Frage, ob die Verwendung eines approximativen und flexibleren, auf Intuition basierenden Modells anstelle eines physikalisch exaktem Simulationsmodells im Bereich der autonomen Systeme sinnvoll ist. Hierfür wird der konkrete Bereich der intuitiven Physik betrachtet. Es wird eine Intuition darüber erlernt, wie sich

Objekte bewegen werden, d.h. die zukünftige Positionen bestimmter Objekte. Diese Intuition soll dem Agenten als Modell der Umwelt dienen.

3.3 Hintergrund und verwandte Arbeiten

In diesem Abschnitt soll grundlegend eine Domäne vorgestellt werden, in der ein Agent seine zukünftigen Aktionen plant und welche bestehenden Ansätze es für solche Planungsprozesse bereits gibt. Zudem wird auf Methoden der intuitiven Physik eingegangen, mit welchen sich auf der Grundlage von Machine Learning Techniken physikalische Objektinteraktionen vorhersagen lassen. Zudem wird auf eine vom Autor durchgeführte Vorstudie eingegangen, welche die grundsätzliche Verwendbarkeit von intuitiver Physik für den Planungsprozess eines Agenten nachweisen soll.

3.3.1 Simulation und Planung

Dieses Kapitel betrachtet intelligente Agenten, die in einer zweidimensionalen kontinuierlichen Umgebung agieren. Abbildung 3.1 zeigt beispielhaft, wie eine solche Domäne aussehen könnte (in der Draufsicht). Der Agent (orangefarbener Kreis) kann sich in diesem beispielhaften Grundriss frei bewegen mit der Aufgabe gewisse Ziele anzusteuern (grüne Kreise), wobei diese statisch sein können oder sich ebenfalls dynamisch bewegen. Dabei soll die Kollision mit statischen oder dynamischen Hindernissen (graue Rechtecke bzw. Kreise) vermieden werden. Zuvor wurde ein Problem dieser Form unter anderem mit Hilfe von Cross Entropy Open-Loop Planning [144, 10] gelöst. Bei beiden Ansätzen wird dem Agenten ein Simulationsmodell der Umgebung zur Verfügung gestellt, damit er das Ergebnis ausgewählter Aktionen simulieren und die erhaltene Belohnung verschiedener Pläne auswerten kann. Die Simulation und Bewertung mehrerer Pläne wird dazu verwendet, eine Wahrscheinlichkeitsverteilung über Aktionsfolgen zu konstruieren, aus der Pläne mit höherer kumulativer Belohnung mit größerer Wahrscheinlichkeit gezogen werden. Dabei ist zu beachten, dass die Ausführung der geplanten Aktionen und die Simulation und Bewertung von Plänen miteinander verschränkt sind. Die Simulation von Kandidatenplänen ist der teuerste Teil beider Algorithmen, wobei offen ist, woher das Simulationsmodell stammt. Ein solches vorgefertigtes Modell könnte unflexibel gegenüber Veränderungen in der Umgebung sein und die Auswertung kann rechenintensiv sein.

3.3.2 Intuitive Physik

Intuition bedeutet, eine subjektive kohärente Entscheidung ohne den diskursiven Gebrauch des Verstandes, d.h. ohne bewusste Argumentation, zu erreichen. Der Mensch entwickelt während seines Heranwachsens ein intuitives Verständnis für die Beziehung von physischen Objekten. Auf der Grundlage

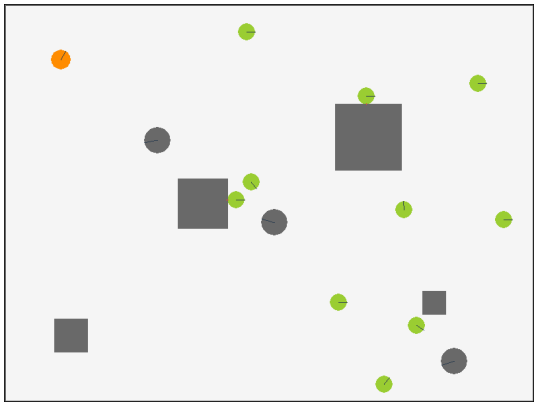


Abbildung 3.1: Draufsicht auf einen zweidimensionalen kontinuierlichen Bereich. Der Agent (orangefarbener Kreis) kann innerhalb des Bereichs Positionen mit reellwertige Koordinaten annehmen, sofern diese nicht von statischen oder dynamischen Hindernissen (graue Flächen) blockiert sind. Zudem existieren Ziele, welche als grüne Kreise dargestellt sind. Die dynamischen Objekte besitzen eine Orientierung im Raum, welche als schwarze Striche eingezeichnet ist.

von Beobachtungen und Erfahrungen entwickelt der Mensch eine Vorstellung von Begriffen wie Schwerkraft, Festigkeit von Objekten, Formerhaltung und Eigendynamik von Objekten [36]. Mit Hilfe dieser Konzepte ist der Mensch in der Lage, das Ergebnis der Kollision bestimmter Objekte ohne Kenntnis der Newtonschen Gesetze oder ohne explizite Berechnungen vorherzusagen. Diese Beobachtung ist die treibende Kraft hinter einem Forschungsgebiet, das intuitive Physik [7] genannt wird. Der Schwerpunkt liegt auf der automatischen Erstellung von Modellen, die in der Lage sind, das Ergebnis von Objektwechselwirkungen auf makroskopischer Ebene allein auf der Grundlage von Beobachtungen abzuschätzen. Obwohl die künstliche Intelligenz noch weiter von einer allgemeingültigen Intelligenz entfernt ist, wird argumentiert, dass ein intuitives Verständnis der physikalischen Objektbeziehungen ein Baustein dafür sein kann [119]. Gewöhnlich werden Methoden des maschinellen Lernens, insbesondere KNN verwendet, um intuitive Physik [80] zu implementieren.

Im Bereich der intuitiven Physik werden künstliche neuronale Netze anstelle von komplexen Modellen eingesetzt, um die physikalische Interaktion von Objekten vorherzusagen. Li et al. stellen einen lernbasierten Ansatz vor, der die Stabilität von aus Holzblöcken gebauten Türmen vorhersagt [89]. Ihr Algorithmus arbeitet nur mit Bildern als Eingabedaten und trainiert ein Modell, um Bilder von Blocktürmen in die Kategorien „stabil“ und „instabil “ einzuordnen. Die Autoren trainieren ihr Modell mit Daten, die aus einer Physiksimulation generiert wurden. Das Ergebnis des Modells wurde dann zur Steuerung eines Roboters verwendet, der Holzblöcke stapelt und die Stabilität zukünftiger Blockplatzierungen vorhersagt.

Fragkiadaki et al. betrachten die Domäne des Billards und schlagen ein objektbasiertes Vorhersagemodell vor, um zukünftige Zustände von Billardkugeln und folglich den Zustand des gesamten Billardtisches vorherzusagen [36]. Der Input des Modells ist ein Stapel von vier Bildern, bestehend aus dem aktuellen und drei vorhergehenden Ansichten auf das vorherzusagende Objekt (Kugel) sowie der im aktuellen Zeitschritt darauf ausgeübten Kraft. Mit dieser Eingabe sagt das Modell die Geschwindigkeit des Objekts für die nächsten 20 Zeitschritte voraus. Dasselbe Modell wird auf alle Objekte angewendet, um den zukünftigen Zustand der Umgebung abzuschätzen. Das Modell wurde an einer Vielzahl von Aufstellungen mit einer unterschiedlichen Anzahl von Kugeln und Billardtischgeometrien trainiert und anschließend an ungesehenen Einstellungen getestet. Nach Ansicht der Autoren könnte es für einen intelligenten Agenten wertvoll sein, mit einem Modell der Umgebungsdynamiken ausgestattet zu sein, um zur Erreichung definierter Ziele Aktionen in unbekannten Umgebungen zu planen und ausführen zu können.

Chang et al. stellen eine Neural Physics Engine [17] vor, ein Framework zum Lernen von Simulatoren der intuitiven Physik, das über eine variable Objektanzahl und verschiedene Szenenkonfigurationen hinweg verallgemeinert. Nach ihrem Ansatz werden Szenen in objektbasierte Darstellungen zerlegt, wobei paarweise Interaktionen zwischen dem fokussierten Objekt und anderen benachbarten Kontextobjekten verfolgt werden. Um die Geschwindigkeiten von Objekten vorherzusagen, werden Long Short-Term Memory (LSTM) Netze [63] verwendet.

In der Arbeit von Agrawal et al. [1] lernt ein Roboter Objekte zu einem Zielort zu verschieben indem er sie anstößt. Die Dynamik der Interaktionen des Roboters wird auf der Basis tiefer neuronaler Netze geschätzt. Diese Schätzungen werden direkt aus Kamerabildern, durch die Verwendung von Convolutional Layern für die Bildverarbeitung [83] im künstlichen neuronalen Netz, abgeleitet.

Wenn auch nicht direkt mit der intuitiven Physik verbunden, haben Ha und Schmidhuber eine Arbeit über Weltmodelle im Allgemeinen [50] veröffentlicht. In dieser Arbeit wird eine komprimierte räumliche und zeitliche Darstellung der Umgebung erlernt, die sogar dazu verwendet werden kann, einen Agenten vollständig innerhalb seines eigenen, von seinem Weltmodell erzeugten, sogenannten halluzinierten Traumes zu trainieren und die erhaltene Strategie wieder in die tatsächliche Umgebung zu übertragen. Auch eine erlernte Intuition über physikalische Beziehungen von Objekten könnte zu diesem Zweck eingesetzt werden.

3.3.3 Voruntersuchung zu intuitiver Physik

Um zu untersuchen, inwieweit sich Methoden der intuitiven Physik eignen, um sie in den Planungsprozess eines intelligenten Agenten zu integrieren, wurde eine Vorstudie durchgeführt, bei der die nichtlineare Bewegungsbahn eines

elastischen Körpers vorhergesagt werden sollte.

Dazu wurde eine Physiksimulation aufgesetzt, bei der in einer begrenzten zweidimensionalen Fläche ein Impuls auf einen elastischen Ball ausgeübt wird, so dass sich diese auf der Fläche bewegte und von den Begrenzungswänden abprallte. Um eine nichtlineare Kurvenbewegung herbeizuführen, wirkte eine Gravitation in y-Richtung, zudem wurde der Ball durch Reibung abgebremst.

Dieser Versuchsaufbau wurde für 100 000 Läufe wiederholt und dabei in festen Zeitschritten die x und y Position des Balls, sowie dessen Geschwindigkeit in x- und y-Richtung aufgezeichnet, so dass sich jeweils ein Tupel (x, y, v_x, v_y) ergibt. Diese aufgezeichneten Daten wurden zum Training eines künstlichen neuronalen Netzes verwendet, welche die Datenpunkte für die letzten 50 Zeitschritte erhält und daraus die Daten für die nächsten 20 Zeitschritte voraussagen sollte. Die Daten wurden zuvor auf Werte zwischen 0 und 1 normiert.

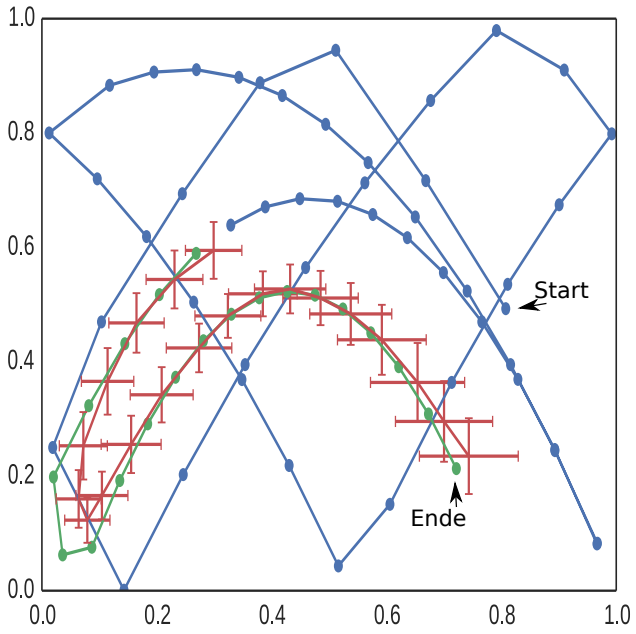


Abbildung 3.2: Beispiel für eine Trajektorie des elastischen Balls in der Voruntersuchung. Zu sehen sind in Blau die 50 Datenpunkte, die das neuronale Netz unter anderem als Eingabe erhalten hat und in Grün die vorherzusagende Fortsetzung der Trajektorie. Die roten Kreuze ergeben sich durch die Vorhersage des neuronalen Netzes unter Anwendung der Dropout Technik.

Bei dem Netz handelte es sich um ein einfaches Feed-Forward Netzwerk mit 5 Neuronenschichten und jeweils 200 Neuronen pro Schicht, abgesehen von der Ausgabeschicht mit 80 Neuronen, und einer ReLU-Aktivierung. Zudem kam eine Technik namens Dropout zum Einsatz. Dropout ist eine Regulari-

sierungtechnik und soll Overfitting vermeiden. Dabei handelt es sich um ein Problem des maschinellen Lernens bei welchen es zu einer Überanpassung des zu bildenden Modells an den Trainingsdatensatz kommt, was zu einer fehlenden Abstraktion bzw. Generalisierung in der Testphase führt. Beim Dropout in künstlichen neuronalen Netzen werden in jedem Trainingsschritt Neuronen bestimmter Schichten mit einer gewissen Wahrscheinlichkeit deaktiviert. Dies führt bei wiederholter Anwendung approximativ zur Realisierung vieler verschiedener Netzwerkarchitekturen.

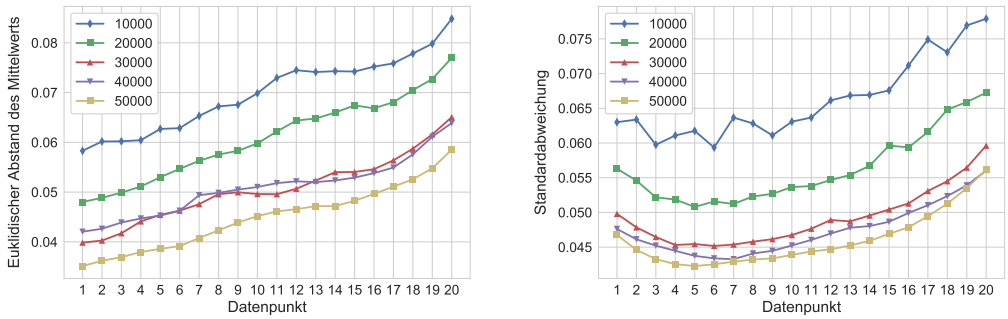
In der hier vorgestellten Vorstudie kommt Dropout aber noch zu einem anderen Zweck zum Einsatz, denn Gal und Ghahramani stellen einen Zusammenhang zwischen neuronalen Netzen und Gaußprozessen über die Dropout-Technik her [42]. Wird Dropout in der Testphase angewendet, erzeugt es eine Verteilung der Ausgabewerte, womit sich laut Gal und Ghahramani eine Aussage über die Sicherheit bzw. Unsicherheit des Modells ableiten lässt.

Die Abbildung 3.2 zeigt eine beispielhafte Bewegungsbahn des elastischen Balls in der Voruntersuchung. Die 50 blauen Datenpunkte verdeutlichen dabei die Eingabedaten, welche das neuronale Netz als Positionen aber auch als Geschwindigkeit in x- und y-Richtung erhalten hat (Beispiel der Trainingsdaten). Die grünen Datenpunkte beschreiben, wie die entsprechende Trajektorie sich fortsetzt und was das künstliche neuronale Netz hätte vorhersagen sollen (Beispiel für die Test-/Vorgabedaten). Die roten Kreuze ergeben sich durch die Vorhersage des neuronalen Netzes unter Anwendung der Dropout Technik, wobei das Zentrum des Kreuzes den Mittelwert der Vorhersagen in x- und y-Richtung markiert und die Ausdehnung des Kreuzes die Standardabweichung der Messdaten darstellt. In den Datenpunkten, welchen nach immer gleichen Zeitintervallen aufgezeichnet wurden, zeigt sich, dass sich die Bewegung verlangsamt, was eine weitere Dynamik herbeiführt. Es ist ersichtlich, dass das Modell durchaus in der Lage ist die Bewegung des elastischen Objektes, welches an den Flächengrenzen abprallt, vorherzusagen. Interessant ist zu bemerken, dass die Unsicherheit des Modells mit weiter in der Zukunft liegenden Vorhersagen zunimmt, was an der größeren Standardabweichung für diese Vorhersagepunkte abzulesen ist.

Abbildung 3.3 zeigt eine Evaluation der Voruntersuchung zur Vorhersage der Objektpositionen für verschiedene Größen des Trainingsdatensatzes.

Dabei wurde für Abbildung 3.3a der Mittelwert der Vorhersagen für den jeweiligen zukünftigen Datenpunkt, das heißt die zukünftige Position des Objektes, berechnet, danach der euklidische Abstand zwischen diesem Mittelwert und der tatsächlichen zukünftigen Position bestimmt und dies für die zu vorhersagenden 20 zukünftigen Objektpositionen im Diagramm abgetragen. Zum einen ist zu erkennen, dass der Vorhersagefehler für weiter in der Zukunft liegende Datenpunkte zunimmt aber auch, dass der Vorhersagefehler mit zunehmender Größe des Trainingsdatensatzes insgesamt abnimmt.

Abbildung 3.3b zeigt die Standardabweichung der Messwerte vom Mittelwert für den jeweiligen zukünftigen Datenpunkt, das heißt, die zukünftige Objekt-



(a) Abweichung des Mittelwerts der Vorhersagen von der tatsächlichen Position des zu vorhersagenden Objektes

(b) Standardabweichung der Messwerte von Mittelwert

Abbildung 3.3: Evaluation der Voruntersuchung für verschiedene Größen des Trainingsdatensatzes

position. Auch hier ist ersichtlich, dass die Schwankungen der Vorhersagewerte mit zunehmender Größe des Trainingsdatensatzes insgesamt abnehmen. Interessant ist, dass die Standardabweichung für weiter in der Zukunft liegende Datenpunkte größer ist, was auf eine größere Unsicherheit des Modells in der Vorhersage hindeutet.

3.4 Intuitiv planende autonome Agenten

Die Idee dieses Abschnitts besteht darin, das Modell, das ein intelligenter autonomer Agent zum Verstehen von Abläufen in seiner Umgebung benutzt, durch eine intuitive Vorstellung zu ersetzen. Dazu soll eine Art Intuition unter Verwendung von Ansätzen aus dem Bereich der intuitiven Physik erlernt werden, wonach dieses intuitive Modell in einen Agenten integriert wird.

Die Idee wird demonstriert anhand des Szenarios eines autonomen mobilen Roboters (Agenten), der sich bewegende Objekte (Ziele) sammeln und währenddessen die Kollision mit dynamischen Hindernissen, statische Hindernissen und statischen Grenzen vermeiden muss. Der Agent erhält eine positive Belohnung sowohl für das Sammeln von Zielen als auch für die Ausführung von Bewegungen. Zudem erhält er eine negative Belohnung für die Kollision mit statischen Objekten und eine hohe Strafe für die Kollision mit dynamischen Hindernissen. Der Agent plant seine zukünftigen Aktionen, um die Belohnung zu maximieren. Die Beiträge des diskutierten Ansatzes bestehen darin, die Bewegung der umgebenden Objekte intuitiv abzuschätzen, anstatt rechenintensive, wenn auch genaue, Simulationen zukünftiger Zustände durchzuführen. Die Kernpunkte des Ansatzes sind:

- Reduzierte Berechnungskomplexität durch intuitive Schätzung der Objektdynamik anstelle der physikalisch exakten Simulation von nichtlinea-

ren Bewegungsmustern elastischer Objekte.

- Verringerte Modellierungskomplexität durch Erlernen einer Intuition anstelle von fest kodierten Regeln.
- Erhöhte Verallgemeinerbarkeit durch die Wiederverwendbarkeit der gelernten Intuition.
- Ermöglichte Anpassungsfähigkeit durch kontinuierliches Lernen.

Das übergeordnete Ziel des Kapitels ist es, weitere Arten von Intuitionen als nur intuitive Physik im Bereich der autonomen Agenten zu fördern.

3.4.1 Problemstellung

Die Genauigkeit der Planung künftiger Schritte durch einen intelligenten Agenten hängt von der Genauigkeit und Komplexität seines internen Modells der Umwelt ab. Sowohl die Modellierung als auch die Auswertung des Modells kann in komplexen oder großen Umgebungen schnell arbeitsintensiv werden. Daher versucht der vorgeschlagene Ansatz die folgenden Herausforderungen anzugehen:

1. Das Modell muss zuvor und möglicherweise manuell erstellt werden. Darüber hinaus kann es sein, dass es im Vorfeld kein ausreichendes Modell der Umgebung gibt.
2. Das Modell könnte unflexibel gegenüber Veränderungen innerhalb der Umgebung sein, so dass es mit der Zeit ungenau werden könnte.
3. Komplexe Szenarien mit mehreren Objekten und Beziehungen können dazu führen, dass die Simulation und Ausführung des Modells rechenintensiv wird.

Im Folgenden wird der Ansatz mit Hilfe eines autonomen mobilen Agenten untersucht.

3.4.2 Planung mit einem exakten Modell und durch Intuition

Wie bereits beschrieben, kann sich der Agent in einem zweidimensionalen Bereich frei bewegen (siehe Abbildung 3.1). Der Bereich hat statische Grenzen und enthält sowohl statische Hindernisse als auch dynamische Hindernisse, die kontinuierlichen nichtlinearen Bewegungsmustern folgen. Auf die genaue Ausgestaltung der Bewegungsmuster wird später eingegangen. Eine Aktion des Agenten besteht darin, sich zunächst an seiner Position um einen bestimmten

Tabelle 3.1: Belohnungsstruktur für die Planung des Agenten

Ereignis	Belohnung
Schritt	+ 1
Einsammeln eines Ziels	+ 5,000
Kollision mit einem dynamischen Hindernis	- 10,000
Kollision mit einem statischen Hindernis	- 5,000

Winkel zu drehen und sich dann für eine vordefinierte Zeitspanne (200 Schritte/Bilder) in diese Richtung zu bewegen. Der Agent muss sich bewegende Ziele sammeln und dabei statische und dynamische Hindernisse vermeiden. Der Agent ist in der Lage, Aktionen durchzuführen, die aus zwei Phasen bestehen: Zunächst führt der Agent an seiner aktuellen Position eine Drehung mit einem bestimmten Winkel aus. Zweitens bewegt sich der Agent für eine festgelegte Zeitspanne in diese Richtung. Während einer Aktion des Agenten (Drehen und Bewegen) folgen die anderen Objekte in der Umgebung ihrem definierten Bewegungsmuster mit physikalisch korrekten Kollisionen und Abprallen. Der Agent muss seine Aktionen in der Form planen, dass er sich um einen bestimmten Winkel dreht und sich dann eine festgelegte Zeitspanne in diese Richtung bewegt. Wenn der Agent mit einem statischen Hindernis kollidiert, stoppt er und die ausgeführte Aktion ist beendet. Der Agent plant seine zukünftigen Aktionen und bewertet verschiedene Pläne anhand einer vorgegebenen Belohnungsstruktur. Diese Belohnungsstruktur ist in Tabelle 3.1 dargestellt.

Die Planungsstrategie des genauen Agenten ist relativ einfach, da er einen Baum vielversprechender zukünftiger Zustände aufspannt, welcher der Breite nach erstellt wird. Zu diesem Zweck wird der Agent zunächst mit einer exakten Simulation des Umfelds ausgestattet, in dem er agiert. Dadurch ist der Agent in der Lage, Handlungen und die Bewegung von Zielen und Hindernissen zu simulieren und das mögliche Ergebnis in Bezug auf die zuvor gegebene Belohnungsstruktur zu bewerten. Konkret probiert der Agent fünf verschiedene Drehrichtungen $\{-60°, -30°, 0°, 30°, 60°\}$ aus und bewegt sich 200 Zeitschritte lang in die jeweilige Richtung. Dabei wird der Zustand des internen Simulationsmodells gespeichert und zwischen jedem Simulationsschritt zurückgesetzt. Nach jeder Aktion wird der nachfolgende Zustand anhand der erhaltenen Belohnung bewertet. Für die beiden vielversprechendsten Folgezustände wird der Planungsprozess wiederholt. Auf diese Weise wird ein Baum von möglichen Aktionen und zukünftigen Zuständen der Umgebung erstellt. Die Planung wird gestoppt, wenn der Baum eine Tiefe von vier erreicht. Jedes Blatt im Baum repräsentiert den Zustand der Umgebung nach vier Aktionen, die der Agent in seiner Simulation ausgeführt hat. Der Weg von der Wurzel zu diesem Blatt stellt die Abfolge der Aktionen (d.h. den Plan) dar, die zu diesem Zustand führt. Daher wird der Plan, der die höchste Belohnung erhalten hat, tatsächlich ausgeführt, woraufhin der Planungsprozess erneut beginnt. Ein Beispiel für einen solchen Baum zukünftiger Agentenpositionen (ohne jegliche Ziele oder

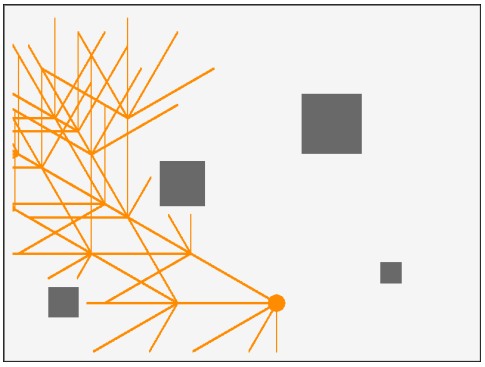


Abbildung 3.4: Skizze des baumsuch-ähnlichen Planungsprozesses des Agenten. Der Agent befindet sich unten in der Mitte und plant seine zukünftigen Positionen, wobei immer die zwei vielversprechendsten zukünftigen Zustände erweitert werden, bis der Baum eine Tiefe von vier erreicht hat (zukünftige Zustände anderer Objekte in der Umgebung sind nicht abgebildet, werden aber auch vorausgeplant).

dynamische Hindernisse) ist in Abbildung 3.4 dargestellt.

Es geht nun darum, das erwähnte physikalisch exakte Simulationsmodell des Agenten zu ersetzen, um die in Abschnitt 3.4.1 genannten Herausforderungen zu überwinden. Es ist arbeitsintensiv in der Erstellung, rechenaufwendig in der Auswertung und unflexibel gegenüber Veränderungen in der Umgebung. Das exakte Modell soll durch eine schnell auswertbare, intuitive Schätzung der Objektdynamik auf der Grundlage eines künstlichen neuronalen Netzes ersetzt werden. Dies erleichtert die automatische Erstellung bzw. das automatische Lernen der Umgebungsdynamik und macht den Agenten anpassungsfähig an Veränderungen in der Umgebung, wenn der Agent weiter lernen und sein intuitives Modell verändern darf. Dies bedeutet, dass das Modell, wie es in der Agentenstruktur in Abbildung 2.1 dargestellt ist, zunächst eine physikalisch exakte Simulation der Welt ist, welche durch ein künstliches neuronales Netz ersetzt wird, das dem Konzept folgt, eine Intuition über die Welt zu lernen. Die baumartige Suche und die Planung bleiben gleich. Obwohl es unter Umständen bessere Planungsstrategien gibt, konzentriert sich der Ansatz mehr auf den Vergleich zwischen einem exakten Simulationsmodell und einer auf Intuition basierenden Schätzung der Umwelt durch den Einsatz von Techniken des maschinellen Lernens, wie z.B. künstliche neuronale Netze.

3.4.3 Lernen einer physischen Intuition

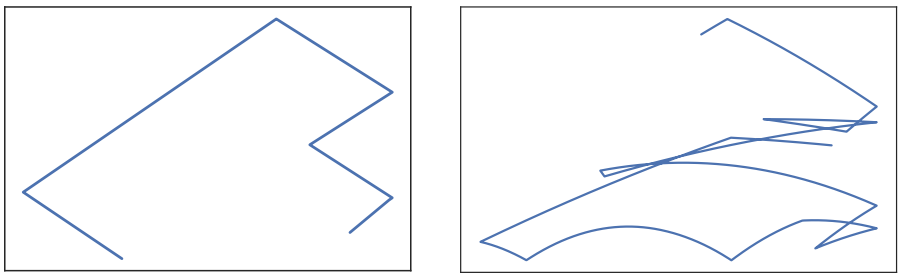
Intuition bedeutet, wie bereits erwähnt, ohne diskursive Argumentation nur auf der Grundlage von Erfahrungen und Beobachtungen zu einer Entscheidung zu kommen. Im Kontext der intuitiven Physik bedeutet es, das Ergebnis physikalischer Prozesse auf makroskopischer Ebene abzuschätzen. Neuere Ergebnisse

auf diesem Gebiet zeigen, dass sich künstliche neuronale Netze für diesen Zweck eignen, da sie generische Funktionsapproximatoren sind. Im Kontext dieser Arbeit agiert ein autonomer Agent innerhalb einer Domäne, die physikalische Beziehungen zwischen den Objekten hat. Anstatt das physikalische Modell für die Planung seiner Aktionen genau zu simulieren, schätzt ein künstliches neuronales Netz die Bewegungspfade der Ziele und der dynamischen Hindernisse.

Zum Zweck des Lernens und der Vorhersage der Objektbewegungen wurde ein KNN mit LSTM-Einheiten verwendet, da sich solche Netzwerke besonders gut für die Zeitreihenvorhersage eignen [63]. LSTM-Einheiten sind so konzipiert, dass sie sowohl kurzfristige als auch langfristige Beziehungen in Zeitreihen erkennen, was die Vorhersage der Zeitreihen verbessert.

Das Netzwerk soll zur Vorhersage der Trajektorien der den Agenten umgebenden Objekte verwendet werden. Es wurde vor dem eigentlichen Planungsprozess des Agenten trainiert. Dazu wurden die Trajektorien von sich bewegenden, einzusammelnden Zielen sowie dynamischen Hindernissen aufgezeichnet, während ein Dummy-Agent zufällige Aktionen in dem zuvor beschriebenen Bereich durchführte. Die aufgezeichneten Daten bestehen aus den Koordinaten x und y sowie der Geschwindigkeit in Richtung x und y für jedes Objekt in der Szene zu diskreten Zeitschritten. Die Aufzeichnung der Daten und das Training des Netzes erfolgte analog dem Vorgehen aus der Voruntersuchung in Abschnitt 3.3.3.

Abbildung 3.5a zeigt die Bewegungsbahn eines Ziels in der Domäne im Verlauf von 5000 Zeitschritten. Die Bewegung der Ziele ist linear, außer dass sie abprallen, wenn sie mit Grenzen oder anderen Objekten kollidieren, was ihre Bewegung ebenfalls verlangsamt. Abbildung 3.5b zeigt die Trajektorie eines dynamischen Hindernisses im Verlauf von 5000 Zeitschritten. Sie werden von einer Kraft beeinflusst, die Nichtlinearitäten in ihre Bewegung einführt. Dynamische Hindernisse haben ebenso wie die Ziele eine Art Elastizität, die sie nach einer Kollision abprallen lässt, was sie ebenfalls verlangsamt und die Nichtli-



(a) Beispielhafte Trajektorie eines Ziels

(b) Beispielhafte Trajektorie eines Hindernisses

Abbildung 3.5: Beispielhafte Trajektorien eines anzusteuernden, beweglichen Ziels und eines beweglichen Hindernisses über 5000 Zeitschritte.

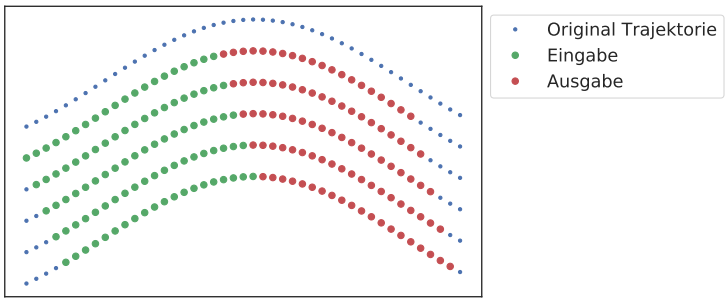


Abbildung 3.6: Überlappendes Schneiden der aufgezeichneten Daten zum Training des neuronalen Netzes.

nearität in ihrer Bewegung erhöht. Um die Objekte in ständiger Bewegung zu halten, wird nach einer bestimmten Anzahl von Zeitschritten ein Impuls auf sie ausgeübt.

Das Modell wurde der Einfachheit und Geradlinigkeit halber auf den explizit gesammelten Daten trainiert, obwohl einige verwandte Arbeiten auf Bildern der Szene beruhen, was aber zu komplexeren neuronalen Netzstrukturen führt. Da eine Aktion des Agenten aus 200 Einzelschritten der Simulation besteht, wurde das Modell darauf trainiert, die Koordinaten x und y sowie die Geschwindigkeit in Richtung x und y eines Objekts für die nächsten 200 Zeitschritte vorherzusagen, wobei diese Informationen für die letzten 200 Zeitschritte gegeben werden. Um ausreichende Trainingsdaten für das neuronale Netz bereitzustellen, wurden die aufgezeichneten Datenpunkte der aufeinanderfolgenden Zeitschritte, wie in Abbildung 3.6 schematisch dargestellt, überlappend geschnitten. Für das Training wurden die Werte der Koordinaten auf das Intervall $[0, 1]$ normiert. Die Geschwindigkeitswerte wurden unter Verwendung der minimalen und maximalen beobachteten Werte im Trainingsdatensatz auf den Bereich $[-1, 1]$ skaliert.

Die Architektur des Netzes selbst ist einfach gehalten. Es benötigt 800 Eingabewerte, die die x und y Koordinaten sowie die Geschwindigkeit in x und y Richtung eines Objektes für die letzten 200 Zeitschritte repräsentieren. Die Eingabe wird dann durch eine Schicht mit 10 LSTM-Einheiten und eine vollständig verbundene Neuronenschicht mit linearer Aktivierung geleitet. Eine Dropout-Schicht deaktiviert 10% der Neuronen bei jeder Aktualisierung während der Trainingszeit, um Overfitting [133] zu verhindern Die Ausgabe des Netzes sind wieder 800 Werte, die die gleichen Merkmale wie die Eingabewerte darstellen, aber für die kommenden 200 Zeitschritte. Abbildung 3.7 skizziert die Architektur des Netzwerks.

Da die Ziele des Agenten und die dynamischen Hindernisse unterschiedlichen Bewegungsmustern folgen, wurden zwei verschiedene LSTM-Netzwerke trainiert und für die Vorhersage ihrer Bewegungen verwendet. Beide Netzwerke weisen die gleiche Architektur auf. Dabei ist zu beachten, dass die Vorhersage für jedes Objekt in der Szene einzeln durchgeführt wurde. Dies hält die Größe

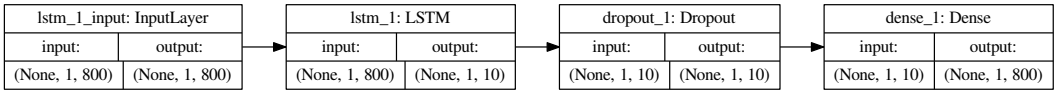


Abbildung 3.7: Visualisierung der Netzwerkarchitektur durch das verwendete Keras-Framework. [19]

des Netzwerks klein und erleichtert das Training und die Auswertung des Modells, mit dem Nachteil, dass Kollisionen zwischen den Objekten in der Szene nicht korrekt modelliert werden. Dennoch enthält das intuitive Modell Informationen über Kollisionen zwischen dynamischen und statischen Objekten, da diese Kollisionen und das Abprallen von Objekten in den Trainingsdaten enthalten ist.

3.4.4 Kombinieren von Planung und der Intuition der Umgebung

Die zuvor beschriebene erlernte Intuition über die Umwelt wurde in einen Agenten eingesetzt, der die ebenfalls zuvor beschriebene baumartige Planung durchführt. Abbildung. 3.8 veranschaulicht beispielhaft den Planungsprozess des Agenten. Die orangefarbenen Geraden sind die Schätzung seiner eigenen Position während der Ausführung einer bestimmten Aktion. Dargestellt sind auch die Eingabedaten und die Vorhersage der neuronalen Netze für die Ziele und dynamischen Hindernisse. Der Agent kann das neuronale Netz für eine Vorhersage der Objektpositionen während seiner nächsten Aktionen abfragen. Wenn der Abstand zwischen den zukünftigen Positionen des Agenten und den vorhergesagten Hindernispositionen unter einem definierten Schwellenwert liegt, wird dies als Kollision interpretiert. Dasselbe gilt für bewegliche Ziele. Der Agent spannt einen Baum möglicher Folgezustände auf, indem er sich an seiner aktuellen Position jeweils mit einem Winkel aus der Menge der Werte $[-60°, -30°, 0°, 30°, 60°]$ dreht und dann für 200 Zeitschritte in diese Richtung bewegt. Dabei simuliert er sowohl seine Bewegung auf der Geraden aber nutzt auch das künstliche neuronale Netz, um die Positionen der beweglichen Objekte um ihn herum im jeweiligen Zeitschritt vorherzusagen. Die Ereignisse auf diesem Pfad, d.h. Kollisionen mit Zielen oder Hindernissen, werden entsprechend der Belohnungsstruktur aus Tabelle 3.1 bewertet. Dabei kann die Bewegung vorzeitig beendet werden, wenn der Agent mit einem statischen Hindernis kollidiert. Die zwei erfolgversprechendsten Pfade werden vom Agenten herangezogen, um sich an deren Ende erneut jeweils mit einem Winkel aus der Menge $\{-60°, -30°, 0°, 30°, 60°\}$ zu drehen und für 200 Zeitschritte in die entsprechende Richtung zu bewegen, was zu einer weiteren Ebene im Baum führt. Dies wird bis zu einer Planungstiefe von vier Aktionen fortgesetzt, woraus sich ein entsprechender Baum ergibt. Die Pfade im Baum werden anhand der auf ihnen vorhergesagten Ereignisse und anhand der Belohnungsstruktur aus Tabelle 3.1 bewertet. Der Pfad im Baum, d.h. die Folge aus vier Aktio-

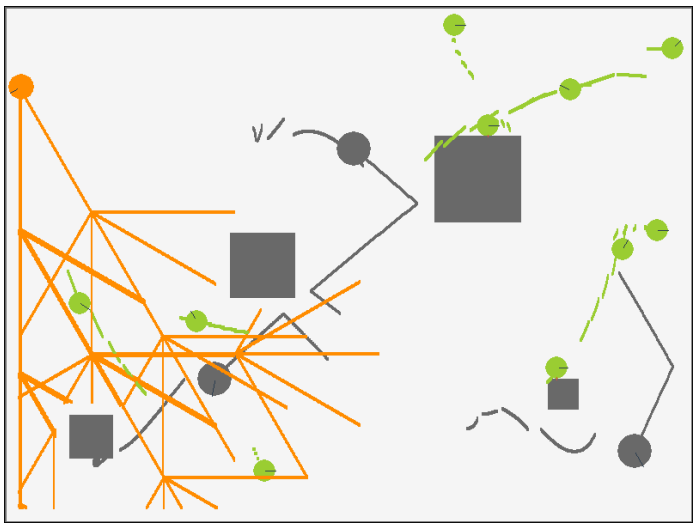


Abbildung 3.8: Überblick über die Domäne mit einer Illustration des Planungsprozesses des Agenten mit dem neuronalen Netz.

nen, mit der höchsten kumulativen Belohnung wird ausgewählt, um ihn in der tatsächlichen Umgebung auszuführen. Wenn der Agent an der Endposition angekommen ist, beginnt der Planungsprozess von neuem. Der Agent, der diese beschriebene Strategie verfolgt und über ein künstliches neuronales Netz zur Vorhersage von Objektpositionen verfügt, wird im Folgenden als „Neuraler Planer“ bezeichnet.

3.4.5 Evaluation

Um die Leistung eines Agenten, der ein exaktes Simulationsmodell verwendet, mit der eines Agenten zu vergleichen, der seine Umgebung mit einem künstlichen neuronalen Netz abschätzt, wurde ein Experiment mit einer zunehmenden Anzahl von dynamischen Hindernissen in der Umgebung durchgeführt. Dieses soll den Zielkonflikt zwischen der Effizienz und Genauigkeit der Modelle aufzeigen, während sie für die Planung verwendet werden.

Vier Agenten mit unterschiedlichen Modellen der Umgebung wurden miteinander verglichen. Jeder Agent führte 1.000 Aktionen aus (was 250 Plänen entspricht). Es wurde gemessen, wie hoch die kumulative Belohnung über diese 1.000 Aktionen war. Um diesen Belohnungswert und den Ablauf selbst besser zu verstehen, wurde auch aufgezeichnet, wie viele Ziele jeder Agent einsammelte (die kontinuierlich an zufälligen Stellen im Raum wieder auftauchten), wie oft jeder Agent mit einem der dynamischen Hindernisse kollidierte und wie oft sie mit einem statischen Hindernis kollidierten. Zum Leistungsvergleich wurde die Zeit gemessen, die jeder Agent für die Durchführung von 1.000 Aktionen benötigte, da jeder einzelne von ihnen auf derselben Hardware ohne weitere störende Arbeitsbelastung auf dem Rechner ausgeführt wurde. Die Agenten bzw. ihre auf ihrem Weltmodell basierenden Planungsstrategien wurden wie

folgt entworfen:

Random Planner (RP) Dieser Agent hat kein Modell der Welt und ist eigentlich nicht wirklich planend, da er zufällige Aktionen in der Domäne ausführt. Er wird als Baseline zum Vergleich verwendet.

Static Obstacle Planner (SOP) Dieser Agent führt die zuvor beschriebene Planung in der Art einer Baumsuche der Breite nach durch, bis eine Baumtiefe von vier erreicht ist. Danach wird der Plan ausgeführt und die Planung beginnt wieder von diesem Zustand aus. Das Ziel dieses Agenten besteht nur darin, statischen Hindernissen (Mauern) auszuweichen, da er keine Informationen über bewegliche Ziele oder dynamische Hindernisse hat. Mit der bekannten Position des Agenten und den Positionen der statischen Hindernisse erstellt er Pläne, die die Kollision mit diesen entsprechend der Belohnungsstruktur vermeiden.

Exact Planner (EP) Diesem Agenten steht eine exakte physikalische Simulation seiner Umgebung zur Verfügung. Er führt die zuvor beschriebene Baumsuche durch und jede von ihm geplante Aktion wird genau in der Umgebung simuliert. Dies umfasst sowohl die Bewegung des Agenten selbst, als auch die Bewegung aller dynamischen Hindernisse sowie deren Kollisionen miteinander und mit den statischen Hindernissen bzw. Grenzen.

Neural Planner (NP) Dieser Agent ist eine Erweiterung des Planers für statische Hindernisse. Darüber hinaus hat er aber auch eine Vorstellung von den dynamischen Objekten in der Umgebung. Er wurde mit einem künstlichen neuronalen Netz ausgestattet, um die Bewegung seiner Ziele und der dynamischen Hindernisse intuitiv abschätzen zu können. Das Netz wurde vor dem Experiment an aufgezeichneten Daten trainiert, aber es ist auch denkbar, dass das Lernen der Intuition während des Experiments weitergeht oder an diesem Punkt gerade erst begonnen wird.

Das Experiment wurde für jeden Typ Agenten und eine gegebene Anzahl dynamischer Hindernisse 50-mal mit verschiedenen Initialisierungs-Seeds (was bedeutet, dass zufällige Ereignisse wie die Anfangsposition der Objekte über die Läufe variieren) wiederholt. Zugrunde lag die Abbildung 3.8 gegebene Fläche mit den darin enthaltenen statischen Hindernissen. Ein Lauf der 50 Wiederholungen bestand darin, dass ein einzelner Agent des jeweiligen Typs mit dem ihm zur Verfügung stehenden Planungsmodell 1.000 Aktionen in der in Abbildung 3.8 dargestellten Umgebung ausführte. Dabei wurden verschiedene Messwerte erhoben, wie z.B. die kumulative Belohnung, die sich aus den während der Ausführung dieser 1.000 Aktionen auftretenden Ereignissen mit der in Tabelle 3.1 gegebenen Belohnungsstruktur ergibt. Um diese kumulative Belohnung besser zu verstehen, wurden zudem Messwerte für jeden einzelnen Ereignistyp

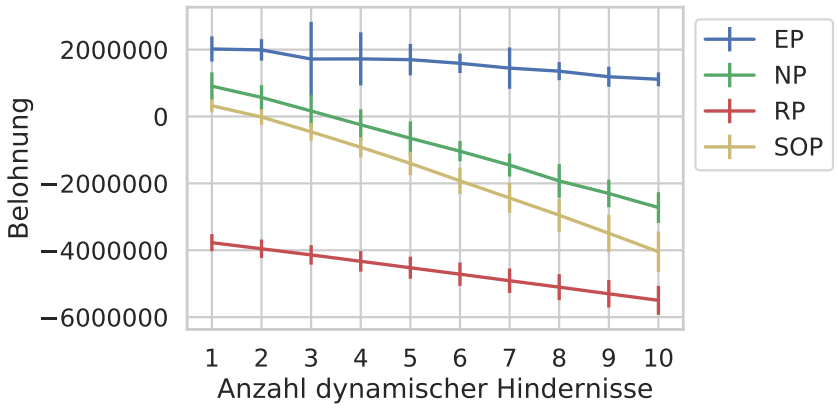


Abbildung 3.9: Die kumulative Belohnung für vier verschiedene Planer über 1.000 Aktionen mit einer unterschiedlichen Anzahl an dynamischen Hindernissen in der Umgebung.

erhoben, sowie die verstrichene „wall-clock time“, die verging während der jeweilige Agent die 1.000 Aktionen plante und ausführte. In den Abbildungen 3.9 bis 3.13 sind die Mittelwerte über die 50 Läufe zusammen mit der Standardabweichung der einzelnen Messwerte zu sehen. In jedem Szenario gab es 10 bewegliche Ziele (grüne Kreise in Abbildung 3.8), die nach dem Einsammeln, das heißt nachdem eine gewisse Distanz zwischen Agent und Ziel unterschritten wurde, zunächst verschwanden aber unverzüglich an einer anderen Position in der Umgebung wieder auftauchten. Die Anzahl der dynamischen Hindernisse (graue Kreise in Abbildung 3.8) in der Umgebung wurde in jedem Experiment im Bereich von 1 bis 10 variiert.

Wie in Abbildung 3.9 dargestellt, sammelte der Agent mit dem komplexesten und genauesten Modell seiner Umgebung, der Exact Planner, die höchste kumulative Belohnung über 1.000 Aktionen, da er den Ausgang einer Aktion (ob er mit einem Ziel oder einem Hindernis kollidieren wird) und den zukünftigen Zustand der Welt genau vorhersagen kann. Es folgt der Agent mit einer intuitiven Vorstellung von der Welt, die auf neuronalen Netzen (Neural Planner) basiert. Am schlechtesten schneiden die Agenten ab, die nur statische Hindernisse berücksichtigen oder zufällig handeln. Aufgrund der Belohnungsstruktur erhalten der Neuronale Planer, der Zufallsplaner und der Planer von statischen Hindernissen nach 1.000 Aktionen eine negative kumulative Belohnung, da Kollisionen, insbesondere mit dynamischen Hindernissen, eine hohe Strafe mit sich bringen.

Um die Ergebnisse weiter zu verstehen, werden im Folgenden die Komponenten der Belohnung weiter analysiert, siehe Abbildung 3.10, 3.11, und 3.12.

Abbildung 3.10 zeigt die Anzahl der gesammelten Ziele über 1.000 Agentenaktionen und legt nahe, dass die Abschätzung des Agenten mit den neuronalen Netzen über die Bewegung der Ziele recht gut ist. Er kollidiert leider immer noch mit dynamischen Hindernissen im Verlauf von 1.000 Aktionen, vermutlich

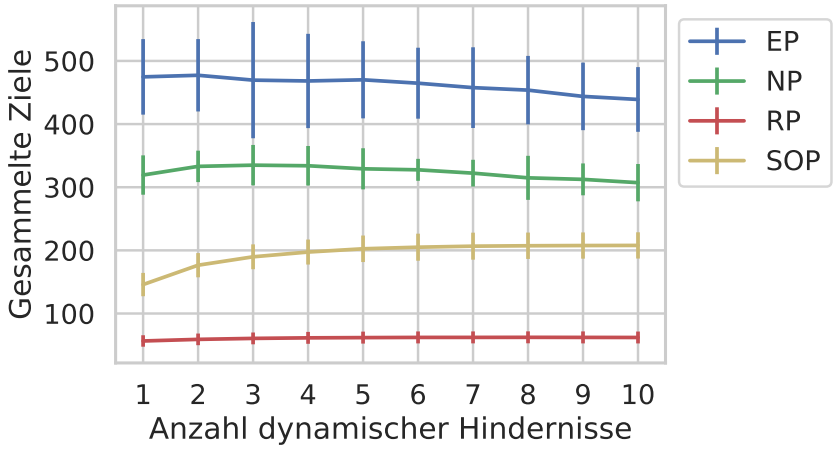


Abbildung 3.10: Die kumulativ gesammelten Ziele der Agenten über 1.000 Aktionen mit einer unterschiedlichen Anzahl von dynamischen Hindernissen in der Umgebung.

weil sie sich viel schneller bewegen als die Ziele und einem komplexeren Bewegungsmuster folgen. Gelegentlich bewegen sich dynamische Hindernisse so schnell in Richtung des Agenten, dass der keine Chance hat, der Kollision auszuweichen. Auch aufgrund der Belohnungsstruktur wird der Agent manchmal die Kollision mit einem dynamischen Hindernis akzeptieren, wenn er anschließend mehrere Ziele sammeln kann. Dies erklärt, warum sogar der Agent mit dem exakten Modell gelegentlich mit dynamischen Hindernissen kollidiert, wie in Abbildung 3.11 gezeigt.

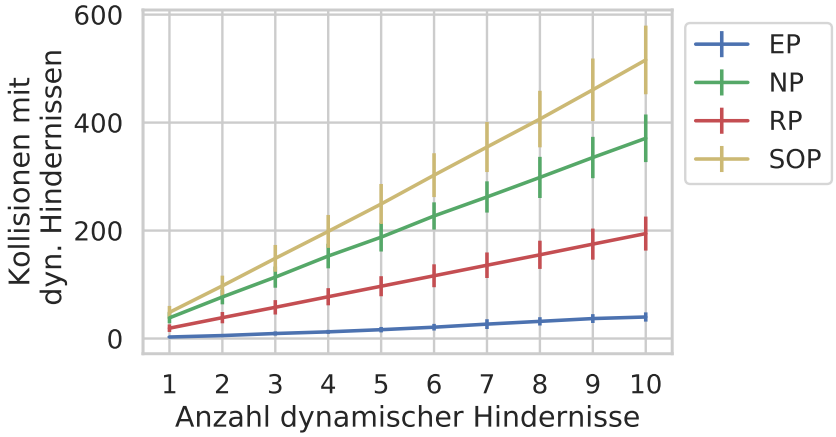


Abbildung 3.11: Die kumulative Anzahl dynamischer Hindernisse, mit denen die Agenten bei der Ausführung von 1.000 Aktionen mit unterschiedlichen Anzahlen an dynamischen Hindernissen in der Umgebung kollidierten.

Abbildung 3.11 zeigt auch, dass der Neural Planner häufiger mit dynamischen Hindernissen kollidiert als der zufällig handelnde Agent, was zunächst nicht intuitiv ist. Dies lässt sich durch die Tatsache erklären, dass der zufällig handelnde Agent oft mit statischen Hindernissen kollidiert (wodurch seine Aktionen gestoppt werden), was in Abbildung 3.12 dargestellt ist.

Daher steht er öfter in Ecken oder an Wänden als der neuronale Planer, ohne mit dynamischen Hindernissen auch nur entfernt in Berührung zu kommen. Der Grund, warum der Planer, der nur statische Hindernisse betrachtet, manchmal immer noch auf sie trifft (wie in Abbildung 3.12 dargestellt), liegt sowohl an der festen Schrittlänge und dem eingeschränkten Drehwinkel des Agenten als auch am festen Planungshorizont. Manchmal landet der Agent nach der Ausführung eines Plans in einer Ecke und hat keine Chance, die Kollision im nächsten Schritt zu vermeiden.

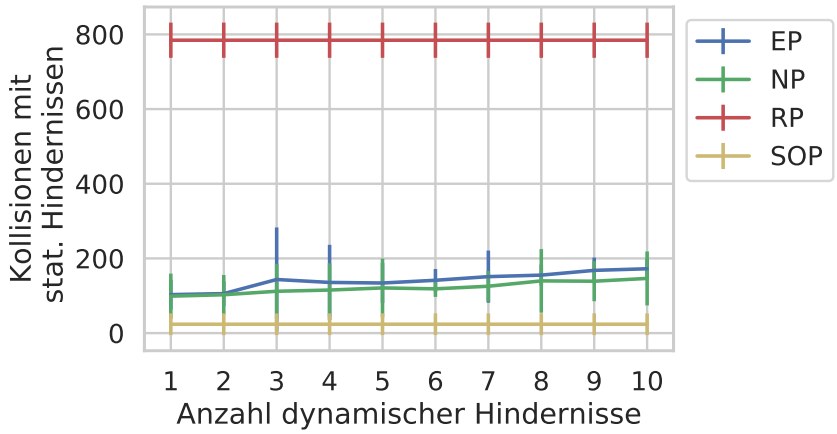


Abbildung 3.12: Die kumulative Anzahl von statischen Hindernissen, die von den Agenten über 1.000 Aktionen mit einer variierenden Anzahl von dynamischen Hindernissen in der Umgebung getroffen wurden.

Abbildung 3.13 stellt die verstrichene „wall-clock time“ dar, die jeder Agent für die Planung und Ausführung von 1.000 Aktionen benötigte. Man kann sehen, dass der Agent, der neuronale Netze für die intuitive Schätzung seiner Umgebung (Neural Planner) verwendet, weniger als ein Viertel der Zeit benötigt, die der Agent mit dem exakten Weltmodell (Exact Planner) benötigte. Dennoch erzielte er vergleichsweise gute Ergebnisse, wie in den vorhergehenden Abbildungen ersichtlich ist. Dieses Verhältnis spricht umso mehr für den Neural Planner, je mehr dynamische Hindernisse beteiligt sind, da die Laufzeit des Exact Planner mit der Anzahl der Hindernisse wesentlich schneller zunimmt als die Laufzeit des Neural Planners. Dies liegt daran, dass die physikalisch exakte Bewegungssimulation einer zunehmenden Anzahl von Objekten schneller an Komplexität zunimmt als die wiederholte Abfrage eines neuronales Netzes, um Objektbewegungen bzw. Objektpositionen zu bestimmten

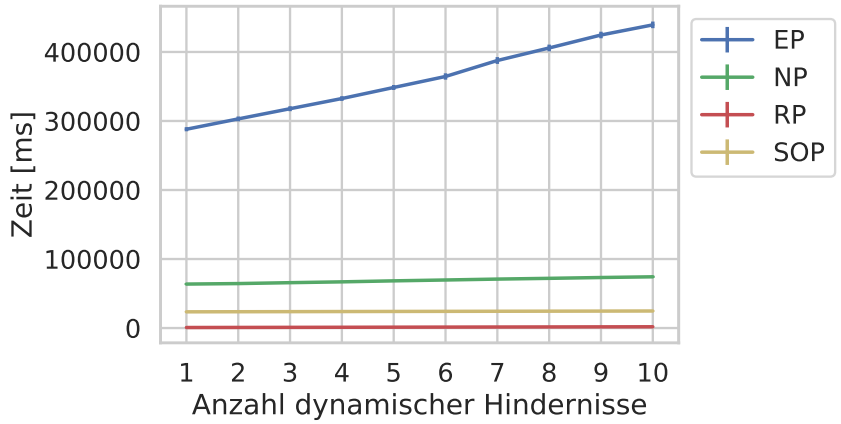


Abbildung 3.13: Zeit in Millisekunden, die verging, bis der jeweilige Agent 1.000 Aktionen mit einer unterschiedlichen Anzahl dynamischer Hindernisse in der Umgebung plante und ausführte.

Zeitpunkten abzuschätzen. Es ist zu beachten, dass das Training des neuronalen Netzes einige Zeit in Anspruch genommen hat. Dies kann aber im Vorfeld erfolgen und verkürzt die Laufzeit des Neural Planners im Vergleich zur Laufzeit des Exact Planners im beschriebenen Umfang. Die Laufzeit des Neural Planners liegt nicht weit über der des Static Obstacle Planners, obwohl der Static Obstacle Planner ebenfalls im Neural Planner enthalten ist. Der Zufallsagent, der eigentlich gar keine organisierte Planung durchführt, kann zum Vergleich herangezogen werden, wie lange die reine Ausführung von 1.000 Aktionen dauert.

3.4.6 Diskussion und Zusammenfassung

In diesem Kapitel kam das Konzept der intuitiven Physik zum Einsatz, um einen intelligenten Agenten mit einer intuitiven Vorstellung der Dynamiken seiner Umgebung auszustatten. Es wurde gezeigt, dass es generell möglich ist einen Agenten mit solch einem erlernten Modell auszustatten, der es verwendet zukünftige Zustände der Umgebung vorauszuplanen und mit seinen Aktionen abzugleichen. Dieser Agent wurde mit einem Agenten verglichen, der über eine physikalisch exakte Simulation seiner Umgebung verfügt. Durch das Erlernen einer Intuition wird die Modellierungskomplexität reduziert, während die Verallgemeinerbarkeit und Anpassungsfähigkeit an sich ändernde Umgebungen zunimmt. Die betrachtete Domäne war ein zweidimensionaler Grundriss, in dem sich der Agent frei bewegen konnte. Das Ziel war es, sich bewegende Ziele zu sammeln und dabei die Kollision mit statischen und dynamischen Hindernisse zu vermeiden. Der Agent mit Intuition zeigt vergleichsweise gute Ergebnisse, während er nur einen Bruchteil der Zeit für seine Planung benötigt, verglichen mit dem Agenten, der seine Aktionen mit einem exakten Modell der Umgebung simulierte. Die Leistung des Agenten mit Intuition könnte wahrscheinlich

weiter verbessert werden, wenn seine Vorhersage auf die Interaktion aller Objekte anstelle der auf einzelne Objekte fokussierten Vorhersage ausgeweitet werden würde. Die Konstruktion eines möglichst detailgetreuen und exakten erlernten Modells bietet Raum für weitere Forschung. Sie stand nicht im Fokus dieses Kapitels, da es hier vorrangig um die Darstellung der Machbarkeit geht. Es bleibt zu untersuchen wie sich dies auf die Komplexität des zugrunde liegenden neuronalen Netzes sowie dessen Trainings- und Vorhersagezeit auswirkt. Ein weiteres Konzept, das einbezogen werden sollte, ist Unsicherheit. Im betrachteten Szenario war die Welt für den Agenten ohne jedes Rauschen vollständig beobachtbar. Dies könnte auf teilweise beobachtbare Umgebungen ausgedehnt werden oder auf Umgebungen, in denen sich umliegende Objekte nicht immer wie erwartet verhalten bzw. Agentenaktionen nicht immer zu den erwarteten Ergebnissen führen. Im nachfolgend vorgestellten Ansatz, wird unter anderem die Unsicherheit im Modell des Agenten behandelt. Dies wird dadurch erreicht, dass für zu vorhersagende, zukünftige Objektpositionen eine Wahrscheinlichkeitsverteilung über mögliche Zonen ausgegeben wird.

3.5 Kollisionsmanagement durch zeit- und risikoabhängige Pfadplanung

In diesem Abschnitt soll nun der Ansatz, mit Hilfe von künstlichen neuronalen Netzen Bewegungen vorherzusagen, in die Graph-Repräsentation einer Freifläche integriert werden, um darauf mit klassischen Algorithmen kürzeste Wege zu suchen, welche aber zugleich Kollisionen mit dynamischen Hindernissen von vornherein vermeiden.

Autonome Agenten wie selbstfahrende Autos oder Paketroboter müssen mögliche Kollisionen mit Hindernissen erkennen und vermeiden, um sich in ihrer Umgebung erfolgreich bewegen zu können. Der Mensch hat gelernt, Bewegungen intuitiv vorherzusagen und Hindernissen vorausschauend auszuweichen. Die Aufgabe der Kollisionsvermeidung kann in eine globale und eine lokale Ebene unterteilt werden. Auf der globalen Ebene sucht der Agent nach einer Lösung für das Pfadplanungsproblem, unter der Annahme, dass eine Graphrepräsentation einer Freifläche bereits gegeben ist. Statische Hindernisse können bereits bei der Diskretisierung berücksichtigt werden, so dass sie bei der späteren Pfadplanung kein Problem mehr darstellen. Bei der Ausführung der gefundenen Lösung kann der Agent auf dynamische Hindernisse stoßen. Das Verhindern einer Kollision wird in der Literatur in der Regel als Kollisionsvermeidung bezeichnet, wobei hier nur die unmittelbare Umgebung und nicht der gesamte Graph relevant ist. Das Ausweichverhalten wird auf lokaler Ebene durchgeführt, wobei Potential Field Ansätze [68] als gängige Beispiele in der Literatur [101] zu finden sind. Es kann für den Agenten sinnvoll sein, die Anwendung lokaler Verfahren vollständig zu vermeiden. Wenn die implementierte Lösung ineffizient oder unzuverlässig ist, sollte sie nur dann eingesetzt

werden, wenn es absolut notwendig ist.

Im Hinblick auf die globale Ebene wird deshalb hier ein Ansatz namens *Predictive Collision Management Path Planning* (PCMP) vorgeschlagen. Das Ziel dieses Abschnittes ist die Integration von Bewegungsprognosen eines LSTM-Modells in die graphenbasierten Planungsentscheidungen eines Agenten. Einerseits soll sich der Agent auf dem kürzesten Weg bewegen und andererseits sollen Kollisionsszenarien so weit wie möglich vermieden werden. Dieser Kompromiss beschreibt das zentrale Thema bzw. die Kernidee des PCMP-Ansatzes. Der Nutzen der Bewegungsvorhersage wird nur dann erzielt, wenn das Risiko einer Kollision in Kauf genommen wird. Dementsprechend müssen sichere Verfahren zur Kollisionsvermeidung auf lokaler Ebene implementiert werden [114], wie der bereits angesprochene Potential Field [68] bzw. Virtual Force Field [74] Ansatz. PCMP kann als Teil eines mehrstufigen Kollisionsvermeidungskonzepts angesehen werden, da es (1) Bewegungen dynamischer Hindernisse mit LSTM-Modellen vorhersagt, (2) diese Bewegungsvorhersagen in den Routing-Graphen integriert und (3) eine zeit- und risikoabhängige Wegplanung durchführt. Auf der lokalen Ebene wird weiterhin die Verwendung von Lösungen zur Kollisionsvermeidung vorgeschlagen, die eine unvermeidliche Kollision verhindern.

Während klassische kürzeste Wege Probleme in einer räumlichen Dimension definiert werden, werden Bewegungsvorhersagen als zeitabhängig definiert. Die zeitliche Dimension ist der Rahmen, in dem PCMP implementiert wird.

PCMP kann zwischen Bewegungsvorhersage mittels KNN, zeitabhängigen Graphen und deterministischen Pfadplanungsalgorithmen eingeordnet werden. Die einzelnen Komponenten werden in der Regel isoliert voneinander betrachtet. Hier werden die verschiedenen Teile jedoch zu einer neuen, prädiktiven und graphenbasierten Vermeidungslösung kombiniert. Die zeitliche Dimension ist die Verbindung zwischen den Komponenten. Sie wird in verwandten Arbeiten oft vernachlässigt, da ihre Relevanz nur bei einem umfassenden Konzept wie PCMP auftritt. Vorhersagen sind Wahrscheinlichkeitsmessungen, die bei der Pfadplanung [114] ein Kollisionsrisiko darstellen. Dieser Gedanke wird in der PCMP konsequent weitergeführt. Das Konzept der zeitabhängigen Risikofunktionen zur Kollisionsvermeidung ist in der bisherigen Forschung wenig beachtet worden. Es scheint in der Literatur keine Arbeiten zur Integration von LSTM-Bewegungsvorhersagen in zeit- und risikoabhängige Wegplanungsalgorithmen auf der Grundlage zeitabhängiger Graphen zu geben. Insbesondere durch die Konzentration auf die zeitliche Dimension und die Bewertung von Kollisionsrisiken trägt PCMP dazu bei, diese Lücke zu schließen.

3.5.1 Szenario

Agent und Hindernisse bewegen sich in einer 2D-Karte, die auf allen Seiten durch Wände begrenzt ist, während der Agent auf seinem Weg mit dynamischen Hindernissen konfrontiert wird. Dies könnte ein Agent in einem Compu-

terspiel sein, der sich auf einer begrenzten, zweidimensionalen Fläche bewegt und welchem bewegliche Hindernisse entgegenkommen, die einem vorhersagbaren, zuvor gesehenen Bewegungsmuster folgen. Sehr abstrakt kann es auch als Roboter, der sich in einer Industriehalle bewegt und mit anderen Robotern konfrontiert ist, gesehen werden. Der Graph, d.h. eine diskrete Darstellung des Raums, wird als gegeben angenommen. Dabei soll es sich um einen ungerichteten Graphen handeln, dessen Knoten durch ihre (x,y)-Koordinaten dargestellt werden. Potenzielle Algorithmen zur Diskretisierung eines Raums sind PRM, RRT oder SGNG. Eine wichtige Eigenschaft dieser Algorithmen ist, dass sie zu einem Graphen mit zufälliger Struktur [66] führen.

Zeitabhängige Bewegungsvorhersagen werden mit zeitabhängigen Kostenfunktionen der Kanten des Graphen modelliert. Der Agent verwendet einen modifizierten A*-Algorithmus für die Pfadplanung und durchquert bei der Ausführung seines Plans Knoten über Kanten, während sich Hindernisse unabhängig vom Graphen bewegen. Der Agent kann seine Route neu planen, wenn er einen Knoten erreicht. Wenn er sich für die Verwendung einer Kante entscheidet, kann er nicht umkehren, außerdem hat der Agent nicht die Möglichkeit, an einem Knoten zu warten. Der Agent beobachtet die (x,y)-Positionen des Hindernisses in regelmäßigen Zeitabständen. Es wird davon ausgegangen, dass sich Agent und Hindernisse mit konstanter Geschwindigkeit bewegen.

In der Implementierung wird angenommen, dass eine *Kollision* auftritt, wenn der Agent eine Kante benutzt, die gleichzeitig von einem Hindernis geschnitten wird.

In der Literatur gibt es ein reges Forschungsgebiet zur Vorhersage menschlicher Bewegungen. In dieser Arbeit werden nur einfache Bewegungsmuster betrachtet, um den rechnerischen Aufwand und die Komplexität der Modelle zu begrenzen. Die vorgestellten Ansätze lassen sich jedoch leicht auf komplexe Bewegungsmuster anwenden. Insbesondere gibt es zwei verschiedene Muster. Ein Hindernis bewegt sich linear auf ein zufällig ausgewähltes Ziel zu. Wenn die Zielkoordinaten erreicht sind, wird wieder ein zufälliges Ziel ausgewählt. Eine Richtungsänderung wird umso wahrscheinlicher, je näher ein Hindernis an einer Wand liegt. Besonders an den Rändern der Karte nimmt die Unsicherheit künftiger Bewegungen zu. Eine zweite Art von Hindernissen oszilliert zwischen zwei Punkten auf der Karte auf einer parabolischen Bahn.

3.5.2 Bewegungsvorhersage von dynamischen Hindernissen

Das Vorhersagemodell soll die zukünftige Position von Hindernissen vorhersagen, wobei auf die bereits eingeführten LSTM-Netze als Modell zurückgegriffen wird. Grundsätzlich kann zwischen Regressions- und Klassifikationsmodellen unterschieden werden, wobei der Unterschied in der Struktur und dem Informationsgehalt der Ergebnisse liegt. Dementsprechend unterscheidet sich auch die Integration von Vorhersagen in den Planungsalgorithmus des Agenten. Wäh-

rend das Regressionsmodell so einfach und effizient wie möglich gehalten wird, soll das komplexere Klassifizierungsmodell eine leistungsfähigere Kollisionskontrolle ermöglichen.

Ein Modell könnte so trainiert werden, dass es nur den nächsten Zeitschritt vorhersagt, und weitere Vorhersagen bauen Schritt für Schritt auf früheren Ergebnissen auf [53]. Dies hat den Nachteil, dass sich der Fehler in der Vorhersage mit fortschreitender Iteration immer weiter vergrößert. Bei PCMP wird hingegen für jeden vorauszusagenden Zeitschritt ein separates Modell trainiert. Dieser Ansatz verursacht einen höheren Trainingsaufwand und ist hinsichtlich der Länge der Vorhersagefolge weniger flexibel. Ungenauigkeiten werden jedoch nicht über die Zeit transportiert, was zu genaueren Vorhersagewerten für weiter in der Zukunft liegenden Zeitpunkten führt [70]. Dies leitet zu der Frage, für wie viele Zeitschritte in der Zukunft die Bewegungen der Hindernisse vorhergesagt werden sollen. Die Wahl des Zeithorizonts lässt sich durch die Struktur des Graphen begründen, da der Agent Planungsentscheidungen nur an Knoten treffen kann. Um zu entscheiden, ob eine zukünftig genutzte Kante zu einer Kollision führt, muss der Vorhersagehorizont größer sein als die Nutzungszeit der längsten Kante. In den Szenarien dieser Arbeit entspricht die Nutzungszeit einer Kante ihrer Länge. Da in der beispielhaft verwendeten Diskretisierung die maximale Nutzungszeit 3 Zeiteinheiten beträgt, wurde ein Vorhersagehorizont von $t = 4$ gewählt.

Abbildung 3.14 zeigt die Struktur des Vorhersageproblems für ein Hindernis, welche für multiple Hindernisse in der Umgebung des Agenten jeweils einzeln Anwendung findet. Der Agent beobachtet die Positionen der Hindernisse in seiner Umgebung in regelmäßigen Abständen, mit der letzten Beobachtung bei $t = 0$ an der Position (x_0, y_0). Für die Vorhersage werden die letzten 16-Beobachtungen $t = 0$ bis $t = -15$ verwendet. Der Agent soll in der Lage sein, Vorhersagen auf der Grundlage einer variablen Anzahl von Beobachtungsdaten zu treffen. Technisch wird diese Eigenschaft durch Auffüllen und Maskieren der LSTM Eingabedaten realisiert. Die Trainingsdaten wurden durch die Simulation und Aufzeichnung der Trajektorien von Hindernisse erzeugt. Für jeden Zeitschritt $t \in 1,...,4$ wird ein separates Modell V_t trainiert. Der Trainingsdatensatz besteht aus 200.000 Sequenzen variabler Länge. Die 16×2 Eingabe-

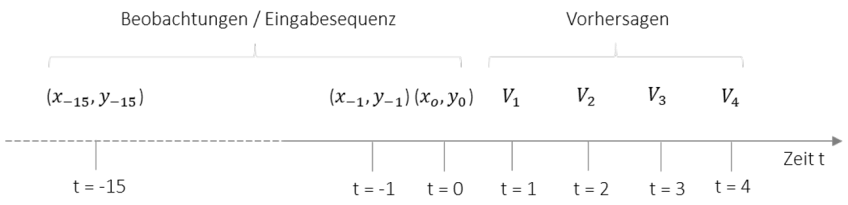


Abbildung 3.14: Struktur des Vorhersageproblems. Basierend auf den letzten 16 Beobachtungen werden die nächsten 4 Zeitschritte vorhergesagt.

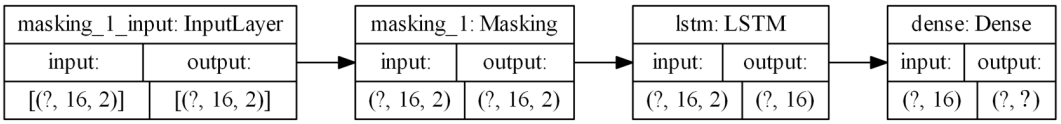


Abbildung 3.15: Grundstruktur des Vorhersagemodells.

matrix enthält in jeder Zeile die beobachtete (x_t, y_t) Position des Hindernisses zum entsprechenden Zeitpunkt t, welche auf das Intervall [0,1] normiert werden. Abbildung 3.15 zeigt die Architektur des Modells. Die Eingabedaten fließen zunächst durch eine Maskierungsschicht, so dass die Auffüllwerte für unvollständige Sequenzen beim Training ignoriert werden. Dann durchlaufen sie eine Schicht von 16 LSTM-Zellen mit ReLU als Aktivierungsfunktion.

Die Ausgabeschicht ist mit der vorhergehenden LSTM-Schicht vollständig verbunden und konvertiert das Ergebnis in das gewünschte Format. Je nach Art des Problems (Regression/Klassifikation) unterscheidet sich die Ausgabeschicht hinsichtlich der Neuronenanzahl und der Aktivierungsfunktion.

3.5.2.1 LSTM-Regressionsmodell

Beim Regressionsmodell stellt eine Vorhersage konkrete Koordinaten auf der Karte dar, an denen das Hindernis zu entsprechenden Zeitpunkten zu erwarten ist. Abbildung 3.16 zeigt die Architektur des Modells und ein Beispiel. Für jeden Zeitschritt $t \in 1,...,4$ wird die zukünftige Position (x_t, y_t) des Hindernisses vorhergesagt. Die Ausgabeschicht des KNN muss an dieses Ergebnisformat angepasst werden. Konkret liefern zwei Knoten ein $2D$-Ergebnis. Mit Hilfe einer linearen Aktivierungsfunktion können die Ergebnisse als Koordinaten interpretiert werden. Die Vorhersagen von vier getrennten Modellen werden zu einer Sequenz kombiniert, die als zukünftige Trajektorie des Hindernisses angenommen wird.

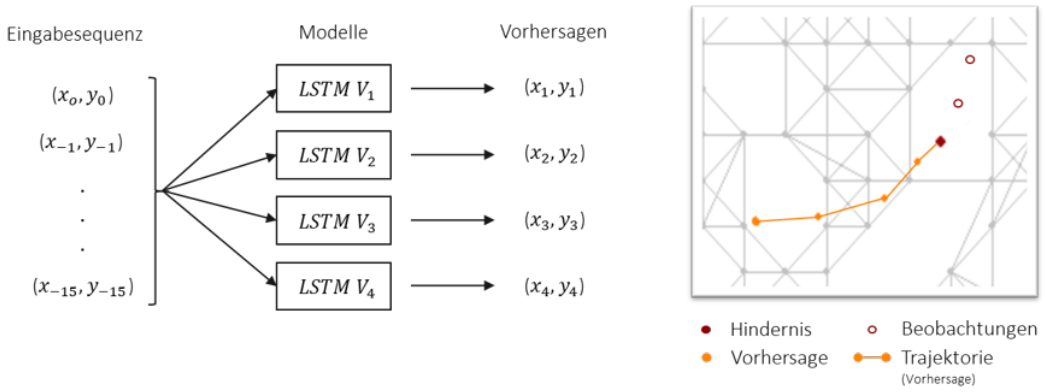


Abbildung 3.16: Die Architektur des Regressionsmodells und die Bewegungsprognose eines Hindernisses für 4 zukünftige Punkte. Es wird eine parabolische Bewegung vorhergesagt.

3.5.2.2 LSTM-Klassifikationsmodell

Ziel des Klassifikationsmodells ist es, Wahrscheinlichkeitsaussagen über zukünftige Bewegungen zu treffen. Eine erste Idee, wie Bewegungen zu klassifizieren sind, liefert die Literatur zu Belegungsgittern [70], die typischerweise zur Darstellung der Positionen von statischen und dynamischen Hindernissen verwendet werden. Die Gitterzellen können als Kategorien eines Klassifikationsproblems interpretiert werden. Indem für jeden Zeitschritt ein eigenes Belegungsgitter erstellt wird, können zukünftige Positionen dargestellt werden.

Eine Kernidee von PCMP ist die relative Definition von Bewegungen. Die zukünftige Position eines Hindernisses (x_t, y_t) kann im Verhältnis zur aktuellen Position des Hindernisses (x_0, y_0) über $\vec{p_t} = (x_t - x_0, y_t - y_0)$ beschrieben werden.

Zunächst wird die Reichweite bestimmt, die ein Hindernis im jeweiligen Zeitschritt maximal erreichen kann. Dieser Bereich wird gleichmäßig in quadratische Zellen unterteilt, mit der aktuellen Position des Hindernisses als Mittelpunkt. Die einzelnen Zellen werden relativ zu diesem Bezugspunkt definiert, d.h. eine Zelle beschreibt eine Menge zukünftiger Positionen $\vec{p_t}$. Je weiter in der Zukunft die Vorhersage liegt, desto größer ist der erreichbare Bereich des Hindernisses. Die relative Struktur schafft universelle Kategorien für das Klassifikationsproblem, die unabhängig von der Position des Hindernisses und der Diskretisierung des Raums durch den Agenten sind.

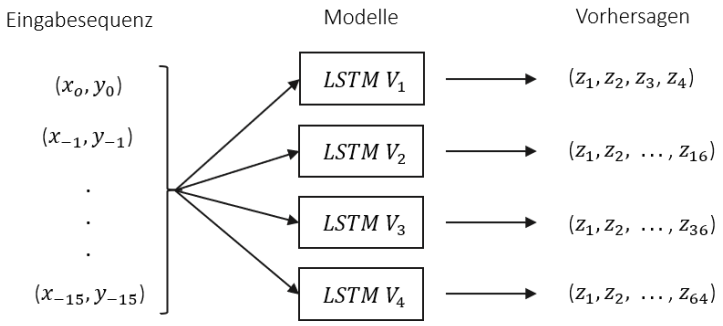


Abbildung 3.17: Bewegungsvorhersage über Belegungsgitter. Verschiedene Zeitpunkte werden mit verschiedenen Modellen vorhergesagt V_t. Die Gitterzellen z repräsentieren die Kategorien des Klassifizierungsproblems.

Für jeden Zeitschritt $t \in 1,...,4$ wird ein Gitter mit $(2t)^2$-Zellen konstruiert. Die Granularität des Gitters kann entsprechend der Komplexität der Bewegungsmuster und der gewünschten Vorhersagegenauigkeit angepasst werden. Abbildung 3.17 zeigt die Architektur des Modells. Die Ergebnisvektoren sind je nach Vorhersagezeitpunkt t unterschiedlich lang, wobei die Zellen z sind Kategorien für die zukünftigen Positionen eines Hindernisses darstellen. Als Aktivierungsfunktion wird *Softmax* verwendet. Da die Werte des Ergebnis-

vektors im Intervall [0,1] liegen, können sie als Wahrscheinlichkeitsaussagen interpretiert werden.

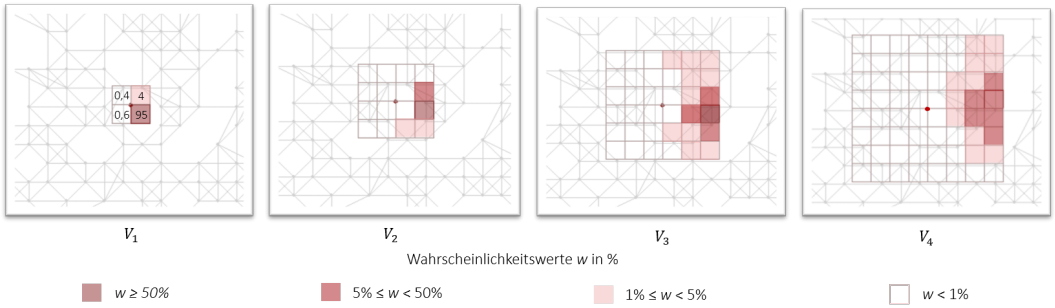


Abbildung 3.18: Klassifikationsergebnisse für vier Zeitpunkte $t \in 1,...,4$. Je dunkler die Farbe ist, desto größer ist die Wahrscheinlichkeit, dass sich das Hindernis zum entsprechenden Zeitpunkt in dieser Zelle befindet.

Im Gegensatz zum Regressionsmodell werden keine konkreten Koordinaten vorhergesagt, sondern Wahrscheinlichkeiten dafür, dass sich das Hindernis zum gegebenen Zeitpunkt innerhalb der Zelle befindet. In Abbildung 3.18 sagt das Modell V_1 voraus, dass sich das Hindernis zum Zeitpunkt $t = 1$ mit einer Wahrscheinlichkeit von 95% im dunkelroten Quadranten befinden wird. Ein Ergebnisvektor der Klassifikation bezieht sich auf die gesamte zugängliche Region eines Hindernisses und enthält daher Informationen über alle möglichen Trajektorien. Die wahrscheinlichste Bewegungsbahn erhält man, wenn man die dunkelsten Zellen verbindet. Die Modelle können verschiedene Trajektorien, deren Sicherheit aber auch die Unsicherheiten der Modelle selbst abbilden. Je unsicherer die Vorhersage, desto gleichmäßiger sind die Wahrscheinlichkeiten auf die Zellen verteilt. Eine typische Eigenschaft des Modells kann man an den Ergebnissen von V_4 in Abbildung 3.18 erkennen. Der Übergang zwischen Zellen mit hohen und niedrigen Werten ist tendenziell fließend. Die dunkelroten Zellen in der Abbildung besitzen keine ungefärbte Zelle als direkte Nachbarn. Die Vorhersagen sind Momentaufnahmen der jeweiligen Zeitpunkte.

3.5.3 Integration der Vorhersagen in zeitabhängige Graphen

Die räumliche Umgebung des Agenten wird durch einen Graph repräsentiert welcher nun mit den Vorhersagen des Bewegungsmodells verbunden werden soll. Diese Vorhersagen haben einen Zeitbezug. Obwohl die Modelle für diskrete Zeitschritte definiert sind, können Bewegungen kontinuierlich in Raum und Zeit interpretiert werden indem eine Trajektorie als kontinuierliche Verbindung zwischen den einzelnen Vorhersagen angenommen wird. Es soll nun ein zeitabhängiger Graph erstellt werden, der Bewegungsvorhersagen in einer

kontinuierlichen Raum-Zeit-Dimension darstellt. Mit Hilfe von Bewegungsvorhersagen kann der Agent 4 Zeitschritte in die Zukunft planen. Innerhalb dieser 4 Zeitschritte steht der Agent vor einem zeitabhängigen Planungsproblem. Nach [35] können zukünftige Änderungen mit zeitabhängigen Kantenkosten modelliert werden. Die Vorhersageergebnisse des Regressions- und Klassifikationsmodells unterscheiden sich in ihrem Format signifikant. Daher wird ihre Integration in den Graphen auf unterschiedliche Weise gelöst.

3.5.3.1 Zeitabhängige Funktionen für Kantenkosten

Mit Hilfe der Wegplanung sucht der Agent im Graphen eine optimale Route, welche durch minimale Pfadkosten charakterisiert ist Dementsprechend hängen die Eigenschaften der gefundenen Lösung von den Kantenkosten ab, welche im Folgenden durch eine zeitabhängige Kostenfunktion ausgedrückt werden sollen. Beim klassischen Kürzeste-Weg-Problem ist es das Ziel, die Entfernung oder Reisezeit zwischen Start- und Zielknoten zu minimieren. In dem hier vorgestellten Szenario soll der Agent immer den kürzesten Weg nehmen aber in Abwägung zum Kollisionsrisiko mit einem dynamischen Hindernis. Wird auf dem geplanten Pfad eine Kollision vorhergesagt, soll der Agent das Kollisionsrisiko gegen alternative Lösungen abwägen und entsprechend seiner Risikoaversion entscheiden. Der Agent verwendet den Pfadplanungsalgorithmus, um die Summe der Kantengewichte zu minimieren. Der angestrebte Kompromiss wird an den Kanten mit zeitabhängigen Kostenfunktionen $c_{v,w}(t) = l + r \cdot k(t)$ abgebildet.

Die Komponenten der Kostenfunktion für eine Kante (v,w) sind die Länge einer Kante l (zeitunabhängiger Skalar, berechnet durch den euklidischen Abstand zweier Knoten), ein Risikoparameter zur Bewertung des Kollisionsrisikos r (zur Kontrolle der Risikoaversion des Agenten) und eine zeitabhängige Kollisionsfunktion $k(t)$ (repräsentiert das Kollisionsrisiko einer Kante mit Hindernissen zum Zeitpunkt t).

Die folgenden Abschnitte befassen sich mit der Frage, wie die Ergebnisvektoren der Vorhersagemodelle in zeitabhängige Kollisionsfunktionen k_t transformiert werden können.

3.5.3.2 Graph-Integration der Regressionsergebnisse

Das Regressionsmodell erlaubt es dem Agenten die zukünftigen Positionen eines bzw. mehrerer Hindernisse abzuschätzen. Vereinfacht kann angenommen werden, dass sich ein Hindernis auf einer direkten und linearen Bahn zur nächsten vorhergesagten Position bewegt. Eine Trajektorie enthält daher sowohl räumlich als auch zeitlich kontinuierliche Informationen. Wenn man bedenkt, dass Kollisionen als Schnitte von Hindernistrajektorien mit Kanten definiert werden, während diese verwendet werden, ergibt sich ein intuitiver Lösungsansatz. Die Benutzung einer Kante führt dann zu einer Kollision, wenn sie zum Zeitpunkt der Benutzung von der vorhergesagten Trajektorie eines Hindernis-

ses geschnitten wird. Als Ergebnis erhält man für jeden Zeitschritt eine Menge von Kanten, die in diesem Zeitschritt zu einer Kollision mit dem Hindernis führen.

Diese Menge wird für jedes Hindernis in der Umgebung des Agenten berechnet. Es handelt sich um ein binäres Entscheidungskriterium, da eine Kante entweder zu einer Kollision führt oder nicht. Daher macht es innerhalb eines Zeitschrittes keinen Unterschied, ob eine Kante von einem oder mehreren Hindernissen geschnitten wird. Für jeden Zeitschritt ergibt sich die Gesamtzahl der Kollisionskanten K als Vereinigung der Kollisionskanten aller k Hindernisse:

$$K = \bigcup_{i=1}^{k} K_i$$

Die Kollisionszeiten können in die zeitabhängigen Kollisionsfunktionen der Kanten integriert werden. Sie unterscheiden mit dem dahinter liegenden Regressionsmodell nur zwischen einer Kollisionswahrscheinlichkeit von 0% oder 100%. Daraus ergibt sich eine zeitabhängige Schrittfunktion. Die in Abbildung 3.19 gezeigte Schrittfunktion führt zu einer Kollision zwischen $t = 2$ und $t = 3$. Im Gegensatz zu Vorhersagen sind die erzeugten Kollisionsfunktionen zeitkontinuierlich.

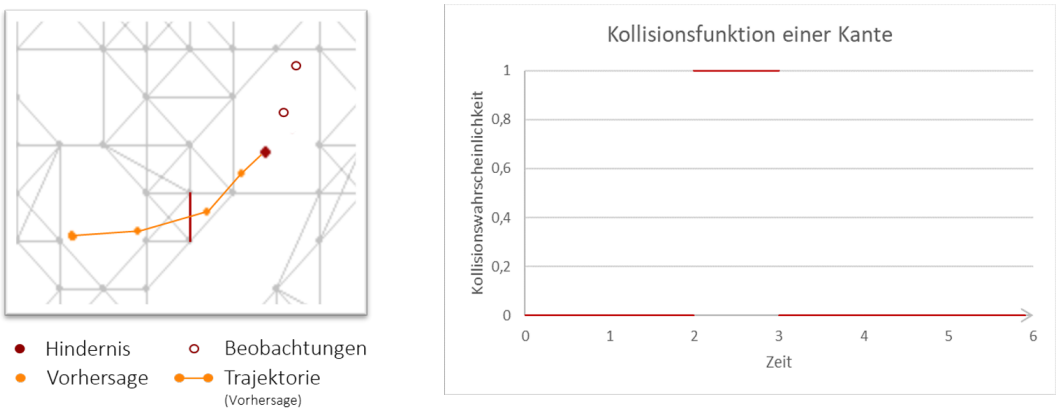


Abbildung 3.19: Integration von Regressionsergebnissen in eine zeitabhängige Kollisionsfunktion. Die rot markierte Kante wird durch das Hindernis im Zeitintervall [2,3] geschnitten.

3.5.3.3 Graph-Integration der Klassifikationsergebnisse

Das Klassifikationsmodell bietet ein Belegungsgitter für jeden vorhergesagten Zeitschritt. Die Übersetzung der Zellwerte in den Graph stellt im Vergleich zu den Regressionsergebnissen ein wesentlich komplexeres Problem dar. Während bisher exakte Koordinaten des Hindernisses vorhergesagt wurden, ist die Genauigkeit hier auf ganze Zellen beschränkt. Im Gegensatz zum Regressionsmodell wird keine konkrete Trajektorie vorhergesagt, sondern der Ergebnisvek-

tor enthält Informationen über eine Vielzahl möglicher Trajektorien. Dennoch ist es weiterhin möglich, zeitkontinuierliche Kollisionsfunktionen für einzelne Kanten abzuleiten. Das Problem wird zunächst für einen konkreten Zeitpunkt dargestellt. Zunächst werden jeder Kante die Zellen zugeordnet, die von der Kante durchlaufen werden. Dabei ist es wichtig, dass mehrere Zellen zu einer Kante gehören können. Mit Hilfe einer Rechenregel werden die Zellenwerte der Kollisionswahrscheinlichkeit der Kante zum entsprechenden Zeitpunkt zugeordnet. An diesem Punkt sind verschiedene Zuordnungsvorschriften denkbar.

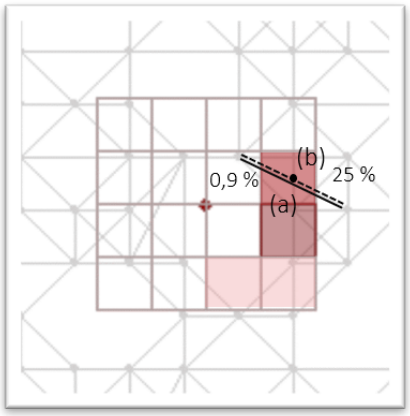


Abbildung 3.20: Zuordnung von verschiedenen Zellen zu einer Kante. Der durchgezogenen Kante (a) sind die Kollisionswahrscheinlichkeiten 0,9% und 25% zugeordnet. Die gestrichelten Kanten (b) verbinden die gleichen Endknoten wie (a).

Im Gegensatz zum Regressionsmodell werden im Klassifikationsmodell allen zugänglichen Kanten Kollisionswahrscheinlichkeiten zugeordnet. Damit der Agent unterschiedliche Attraktivität der Pfade durch die Region beurteilen kann, müssen die Kanten feingranular unterscheidbar sein. Das Gewicht einer Kante könnte berechnet werden, indem man die maximale Zellwahrscheinlichkeit der zugeordneten Zellen auswählt. Diese Berechnungsvorschrift erschwert es dem Agenten jedoch, zwischen Kanten zu unterscheiden, da mehreren Kanten die gleiche Kollisionswahrscheinlichkeit zugewiesen wird. Eine Kante mit Zellwerten [25%, 0,9%] hätte die gleiche Kollisionswahrscheinlichkeit wie eine Kante mit $[25\%, 25\%]$. Die Intuition, dass der hohe Kollisionswert einer Kante automatisch mit einem konservativen Vermeidungsverhalten verbunden ist, ist irreführend, da so Kanten unnötigerweise vermieten werden bzw. nicht zwischen Kanten unterschieden werden kann, die die gleiche Zelle teilen. Das Problem könnte mit der Durchschnittsfunktion gelöst werden. Das Kollisionsrisiko von langen Kanten wird jedoch systematisch unterschätzt. Angenommen, ein Agent hat die Wahl zwischen einem langen *(a)* und zwei kurzen Kanten *(b)* (siehe Abbildung 3.20). Beide Optionen führen zum gleichen Ziel und

gehen durch die gleichen Zellen. Wenn der Agent die beiden Einzelkanten verwendet, ist die Kollisionswahrscheinlichkeit des Pfades gleich der Summe der beiden Zellen (25,9%). Wenn das Kollisionsrisiko der langen Kante gleich dem durchschnittlichen oder maximalen Zellenwert ist, dann ist das Kollisionsrisiko unterschätzt (12,95%). Der Agent würde lange Kanten bevorzugen. Die genaue Lösung des Problems könnte deshalb anhand der Proportionalverteilung berechnet werden. Sie ist jedoch relativ aufwändig zu berechnen, weshalb zur Vereinfachung die folgende Gleichung verwendet wird, bei der jedoch Ungenauigkeiten auftreten können. Die Kollisionswahrscheinlichkeit einer Kante zum Zeitpunkt k_t ist gleich der Summe der n zugeordneten Zellwerte $z_{i,t}$:

$$k_{t,h} = \sum_{i=1}^{n} z_{i,t} \text{ für Hindernis } h$$

Die Kollisionswahrscheinlichkeit hängt von konkreten Zellwerten und der Anzahl der durchlaufenen Zellen ab und beträgt z.B. in Abbildung 3.20 25.9%. Je mehr Zellen von einer Kante durchquert werden, desto mehr Zellwerte werden addiert, was potenziell die Wahrscheinlichkeitsaussage für eine Kollision erhöht. Dies steht im Einklang mit der Länge der Kanten, welche sie durch verschiedene Zonen/Zellen führt. Im Gegensatz zum Regressionsproblem handelt es sich um Wahrscheinlichkeiten anstatt binärer Aussagen und es macht einen großen Unterschied, ob eine Kante von einem oder mehreren m Hindernissen geschnitten werden kann. Der Agent berechnet zunächst das Kollisionsrisiko für jede Kante, jeden Zeitschritt t und jedes Hindernis h einzeln $k_{t,h}$. Die Gesamtkollisionswahrscheinlichkeit für einen Zeitpunkt t wird berechnet mittels:

$$k_{t,total} = 1 - \prod_{i=1}^{m}(1 - k_{t,i})$$

Das Produkt $\prod_{i=1}^{m}(1-k_{t,i})$ beschreibt die Wahrscheinlichkeit, dass kein Hindernis die Kante schneidet. Die Wahrscheinlichkeit, dass mindestens ein Hindernis mit der Kante kollidiert, ergibt sich aus der Gegenwahrscheinlichkeit davon. Zusammenfassend lässt sich daraus eine zeitabhängige Treppenfunktion für das Kollisionsrisiko ableiten.

Abbildung 3.21 zeigt die zeitkontinuierlichen Risikofunktionen für zwei Beispiele. Die rot gestrichelte Kante hat in der Zeit zwischen $t = 0$ und $t = 2$ ein Kollisionsrisiko von 0%, weil sie nicht innerhalb von zwei Zeitschritten für das Hindernis erreichbar ist. Im Intervall [2,3] jedoch, hat die Kante ein Kollisionsrisiko von 65%. Für jede Kante liegt die Kollisionsfunktion für $t > 4$ bei 0%, da der Agent einen Zeithorizont in die Zukunft von 4 Zeitschritten hat. An dieser Stelle ist anzumerken, dass die Genauigkeit des Modells unter anderem von der Granularität des Gitters abhängt. Je nach Bedarf kann das Verfahren in Bezug auf Genauigkeit und Komplexität angepasst werden.

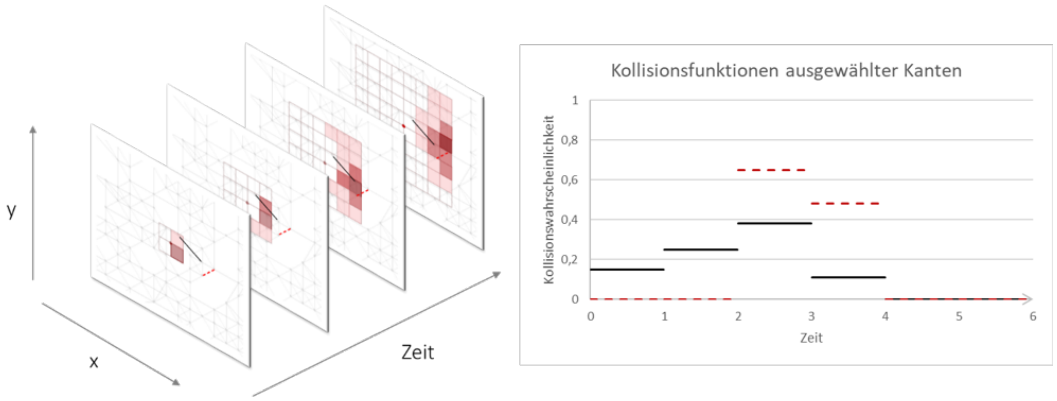


Abbildung 3.21: Integration von Klassifikationsergebnissen in zeitabhängige Kollisionsfunktionen. Der rote und der schwarze Funktionsgraph stellen das zeitabhängige Kollisionsrisiko für zwei spezifische Kanten dar.

3.5.4 Zeit- und risikoabhängige Pfadplanung

Bislang wurden die zukünftigen Bewegungen der Hindernisse vorausgesagt und diese Vorhersagen in einen zeitabhängigen Graph integriert. Die Suche nach dem optimalen Pfad in dieser Raum-Zeit-Darstellung erfolgt durch einen Pfadplanungsalgorithmus, der im Folgenden vorgestellt wird.

3.5.4.1 Zeitabhängiger A* Algorithmus

Der Agent verwendet einen zeitabhängigen A*-Algorithmus auf der Basis des Ansatzes von Zhao et al. [148], der in wesentlichen Punkten für die Bewertung von Kollisionsszenarien angepasst wurde (siehe Algorithmus 2). Dabei kommen folgende Bezeichner zum Einsatz: Startknoten s; Zielknoten d; Planungszeitpunkt t_0; Kostenfunktion $g(v)$ für Weglänge und Kollisionsrisiko; zeitraumabhängige Kostenfunktion $c_{v,w}(t(v))$ für jede Kante $(v,w) \in E$; euklidischer Abstand als Heuristik $h(v)$; Zeitfunktion $t(v)$ berechnet den Ankunftszeitpunkt am Startknoten v einer Kante.

Zentrales Element des Algorithmus sind die zeitabhängigen Kantenkosten, welche bereits von Zhao et al. angedacht waren. In [148] entsprechen die Pfadkosten zu einem Knoten v der Zeit, bis der Knoten erreicht wird. In ihrer Version des A* Algorithmus entspricht der Zeitpunkt der Nutzung einer Kante den bisherigen angefallenen Pfadkosten bis zu ihrem Startknoten. In der vorliegenden Arbeit bestehen die Pfadkosten jedoch nicht rein aus der Länge der im Pfad enthaltenen Kanten (bzw. aus der Zeit, die benötigt wird die Kanten zu traversieren). In dieser Arbeit setzen sich die Kosten einer Kante aus ihrer Länge und einem weiteren Term zusammen, der beziffert wie wahrscheinlich es ist, dass die Kante im Zeitraum ihrer Nutzung durch den Agenten von einem Hindernis geschnitten wird. Daraus folgt, dass die Pfadkosten bis zu einem Knoten $g(v)$ und der Nutzungszeitpunkt einer von v ausgehenden

Algorithmus 2 Angepasster A* Algorithmus auf Basis von [148], welcher um eine Zeitfunktion $t(v)$ ergänzt wurde, da die Pfadkosten bis zu einem Knoten $g(v)$ und die benötige Zeit bis zu seinem Erreichen $t(v)$ verschieden sind.

Require: gewichteter Graph $G = (V, E, c)$
Startknoten s
Zielknoten d
Planungszeitpunkt t_0
Kostenfunktion $g(v)$ für die Pfadlänge
Zeitraumabh. Kostenfunk. $c_{v,w}(t(v))$ für jede Kante $(v,w) \in E$
Heuristik $h(v)$: euklidische Distanz
Zeitfunktion $t(v)$ berechnet den Ankunftszeitpunkt an einem Knoten v

1: *status(s)* $\leftarrow$ *marked*
2: $g(s) = 0$
3: $t(s) = t_0$
4: **for each** $v \in N$ **and** $v \neq s$ **do**
5: | *status(v)* $\leftarrow$ *unmarked*
6: **end for**
7: **Sei** v ein markierter Knoten mit kleinstem $g(v) + h(v)$
8: **if** $v = d$ **then**
9: | **Goto** 23
10: **end if**
11: **for each** *edge* $e(v,w)$ **do**
12: | **if** *status*(w) = *unmarked* **then**
13: | | *status*$(w) \leftarrow$ *marked*
14: | | $g(w) \leftarrow g(v) + c_{v,w}(t(v))$
15: | | $p(w) \leftarrow v$
16: | **else if** *status*$(w) =$ *marked* **and** $g(w) > g(v) + c_{v,w}(t(v))$ **then**
17: | | $g(w) \leftarrow g(v) + c_{v,w}(t(v))$
18: | | $p(w) \leftarrow v$
19: | **end if**
20: **end for**
21: *status*$(v) \leftarrow$ *finished*
22: **Goto** 7
23: **Return.:** $g(d)$ *# Kosten zum Ziel*
$p(s,d) = (d, p(d), p(p(d)), \ldots, s)$ *# Pfad zum Ziel*

Kante auseinanderfallen und mit verschiedenen Funktionen berechnet werden müssen. Aus diesem Grund muss dem Agent im Planungsprozess eine Funktion $t(v)$ zur Verfügung gestellt werden, die die Zeit bis zum Erreichen von v berechnet, zu dem eine von v ausgehende Kante verwendet werden soll. Diese Unabhängigkeit von $g(v)$ und $t(v)$ ist der Hauptunterschied zu [148]. Der nächste Abschnitt befasst sich ausführlich mit der Berechnung des Nutzungszeitpunktes einer Kante über die Zeitfunktion $t(v)$.

Bisher betrachtete Kollisionsfunktionen bilden Kollisionswahrscheinlichkeiten zu verschiedenen Zeitpunkten ab. Im Folgenden werden sie als zeitpunktabhängige Kollisionsfunktionen bezeichnet. Wie in [25] beschrieben, hängen die Kosten einer Kante von dem Zeitraum ab, in dem die Kante verwendet wird. Ein Zeitraum ist definiert als ein Intervall über die Zeit mit einem Start und einem Ende. Zeitpunktabhängige Kollisionsfunktionen $k_{zeitpunkt}(t)$ müssen dementsprechend in zeitraumabhängige Kollisionsfunktionen $k_{zeitraum}(t)$ umgewandelt werden.

3.5.4.2 Nutzungszeitpunkt einer Kante

Grundsätzlich ergibt sich im Rahmen der Pfadplanung die Ankunftszeit an einer Kante indem die Nutzungszeiten aller vorhergehenden Kanten zu einem Startwert addiert werden. Unter der Annahme, dass die Kantenlänge der Nutzungsdauer entspricht, da sich der Agent mit konstanter Geschwindigkeit bewegt, und dass Kanten eine feste Länge haben, wird die Ankunftszeit des Agenten an einer Kante berechnet, indem die Längen l aller vorherigen Kanten n und die Startzeit t_0 über $t(v) = t_0 + \sum_{i=1}^{n} l_i$ aufaddiert werden.

Die Prognosen der Hindernisbewegungen beziehen sich auf die beobachteten Daten und damit auf den letzten Beobachtungszeitpunkt $t = 0$. Der Agent ist so angelegt, dass er nur an Knotenpunkten Planungsentscheidungen treffen kann, wobei der Beobachtungszeitpunkt der Hindernispositionen davon losgelöst ist. Die Hindernisse werden von Sensoren in konstanten Zeitintervallen erfasst. Das bedeutet, dass die Planungszeit t_0 und die Beobachtungszeiten in der Regel unterschiedlich sind. Die Obergrenze dieser Diskrepanz kann maximal einem Beobachtungsintervall entsprechen.

Mit dem Startwert t_0 kann der Planungszeitpunkt mit dem Referenzpunkt $t = 0$ der Hindernisvorhersagen synchronisiert werden.

3.5.4.3 Zeitraumabhängige Kollisionsfunktion

Die bisher vorgestellten Kollisionsfunktionen stellen das Risiko zu verschiedenen Zeitpunkten dar. Wenn man bedenkt, dass das Kollisionsrisiko eines Agenten bei Verwendung einer Kante von einem Zeitraum und nicht von einem Zeitpunkt abhängt, wird deutlich, dass die aufgestellten Kollisionsfunktionen noch nicht die gewünschte Eigenschaft haben. Benötigt werden Kostenfunktionen, die in Abhängigkeit vom Nutzungszeitpunkt t die Kosten (Kollisionswahrscheinlichkeiten) über die Nutzungsdauer d abbilden.

Zeitraumabhängige Kollisionsfunktion im Regressionsmodell

Die angestrebten zeitraumabhängigen Kollisionskosten kann man sich als ein Fenster, das über der zeitabhängigen Kollisionsfunktion geschoben wird, vorstellen. Die Fenstergröße entspricht der Nutzungsdauer der Kante. Für die Berechnung des Kollisionsrisikos in Abhängigkeit von der Zeitperiode sind unterschiedliche Abbildungsregeln denkbar. Die Auswahl der Abbildungsregel hat Einfluss auf das Ausweichverhalten des Agenten. Der Agent weiß über das ihm zur Verfügung stehende Modell nur, dass die Kollision zu irgendeinem Zeitpunkt in einem Intervall auftreten wird. In diesem Teil der vorliegenden Arbeit wird eine Kollision angenommen, wenn sich Nutzungsdauer und Kollisionsintervall überlappen. Das zeitraumabhängige Kollisionsrisiko mit dem Regressionsmodell entspricht dem Maximalwert der zeitpunktabhängigen Kollisionsfunktion über die Nutzungsdauer d:

$$k_{zeitraum}(t) = \max(k_{zeitpunkt}([t : t + d])$$

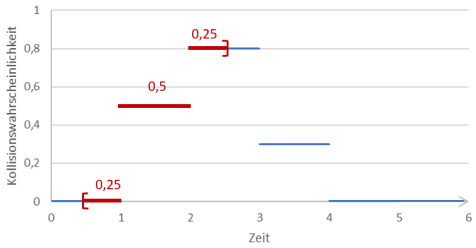
Zeitraumabhängige Kollisionsfunktion im Klassifikationsmodell

Im Gegensatz zum Regressionsmodell sind bei der Klassifikation ausgehend von der zuletzt beobachteten Position des Hindernisses alle relativ zur Hindernisposition erreichbaren Kanten mit Risikowerten versehen und diese Werte sind im Gegensatz zum Regressionsmodell auch nicht binär 0% oder 100%. Die bisher entwickelte Intuition, dass die Wahl des Maximalwertes der zeitpunktabhängigen Kollisionsfunktion zu risikoaversen Planungsentscheidungen führt, funktioniert mit dem Klassifikationsmodell nicht. Die Qualität der Planungsentscheidungen des Agenten hängt von der Fähigkeit ab, zwischen verschiedenen Pfaden unterscheiden zu können (siehe Abschnitt 3.5.3.3). Dasselbe Argument kann auf die Zeitdimension angewandt werden. Damit der Agent in der Lage ist, die Attraktivität einer Kante zu verschiedenen Zeitpunkten zu beurteilen, sollten möglichst wenige Zeitpunkte das gleiche Risikoniveau aufweisen. Eine Stufenfunktion ist entsprechend nicht geeignet. Die Abbildungsvorschrift soll hier die Treppenstruktur bestmöglich auflösen. Der gewichtete Mittelwert über die einzelnen Risikostufen im Nutzungszeitfenster erfüllt die gewünschte Eigenschaft. In Abbildung 3.22a ist die zeitpunktabhängige Kollisionsfunktion einer Kante mit einer Nutzungsdauerzeit von $d = 2$ dargestellt. Für den Startzeitpunkt von $t = 0.5$ ist der Nutzungszeitraum rot markiert. Die Anteile der Risikoniveaus an der Nutzungsdauer sind ebenfalls dargestellt.

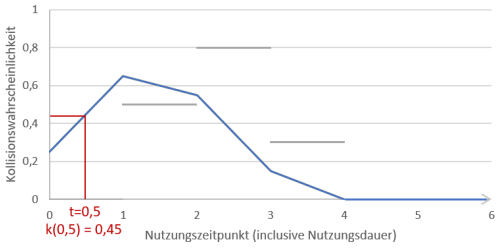
Der gewichtete Durchschnitt differenziert das Kollisionsrisiko einer Kante über die Zeitdimension. Die zeitraumabhängige Kollisionsfunktion ist gegeben durch

$$k_{zeitraum}(t) = \sum_{i=0}^{d} p(t+i) \cdot k_{zeitpunkt}(t+i) \ .$$

Die Funktion $p(t)$ berechnet die Anteile der Risikoniveaus an der Nutzungs-



(a) Kollisionsfunktion einer Kante



(b) Zeitraumabhängige Kollisionsfunktion einer Kante, indem die gewichtete Summe der Kollisionswahrscheinlichkeit über die Nutzungsdauer berechnet wird

Abbildung 3.22: Übersetzung der zeitpunktabhängigen Kollisionsfunktion in eine zeitraumabhängige Kollisionsfunktion für das Klassifikationsmodell.

dauer (siehe Abbildung 3.22b). Wenn ein Agent die angezeigte Kante zum Zeitpunkt $t = 0{,}5$ für 2 Zeiteinheiten verwendet, geht er ein Kollisionsrisiko von $k_{zeitraum}(t) = 0{,}25 \cdot 0 + 0{,}5 \cdot 0{,}5 + 0{,}25 \cdot 0{,}8 = 45\%$ ein. Zum Verwendungszeitpunkt $t = 1$ (für eine Dauer $d = 2$) hat die Kante ein Kollisionsrisiko von 65%.

3.5.5 Risikoparameter r

Das Ziel von PCMP ist die intelligente Vermeidung lokaler Kollisionsszenarien, wenn ein Umweg lohnenswert ist. Die optimale Lösung ist ein Kompromiss zwischen Kollisionsgefahr und kürzestem Weg. Bislang ging es vorrangig um die Kollisionswahrscheinlichkeiten der Kanten. Der zeitabhängige A* Algorithmus sucht den Weg mit minimalen Kosten. In PCMP setzen sich die Kosten einer Kante e zusammen aus deren Länge l_e und dem über die Nutzungsdauer vorhergesagtem Kollisionsrisiko $k_e(t) \in [0, 1]$. Die Funktion zur Vorhersage des Kollisionsrisikos nimmt als Argument den Zeitpunkt entgegen, an dem die Nutzung der Kanten durch den Agenten beginnt. Die Nutzungsdauer ergibt sich aus ihrer Länge unter der Voraussetzung, dass sich der Agent mit konstanter Geschwindigkeit bewegt. Um zu steuern, wie stark das vorhergesagte Kollisionsrisiko ins Gewicht fällt und damit die Risikoaversion eines Agenten beeinflussen zu können, wird ein Parameter $r \in \mathbb{R}_{\geq 0}$ eingeführt. Damit ergeben sich die Kosten einer Kante e des Graphen aus:

$$c_e(t) = l_e + r \cdot k_e(t) \ .$$

Über den Risikoparameter r können auch kleine Kollisionsrisiken in beliebiger Höhe gewichtet werden, was die Kosten einer Kante beeinflusst. Je größer der Risikoparameter, desto größer sind die Kantenkosten. Und je höher die

Kantenkosten sind, desto stärker ist der Anreiz für den Agenten, eine Kante mit einem Kollisionsrisiko zu vermeiden. Mit dem Risikoparameter r kann der Kompromiss zwischen Weglänge und Kollisionsrisiko beliebig in beide Richtungen verschoben werden. Mit dem Parameterwert $r = 0$, werden die Vorhersagen von dynamischen Hindernissen ignoriert und die Pfadplanung degeneriert zu einer gewöhnlichen Pfadplanung mittels des A* Algorithmus, die sich rein auf die Länge der Kanten stützt.

An diesem Punkt ist die Entwicklung von PCMP abgeschlossen. Der Agent ist in der Lage, die Bewegungen von Hindernissen vorherzusagen, sie in einen Graphen zu integrieren und zeit- und risikoabhängige Vermeidungsentscheidungen zu treffen. Im Folgenden soll die steuernde Wirkung des Risikoparameters r weiter skizziert sowie eine grafische Intuition für die Funktionsweise von PCMP entwickelt werden.

Interpretation des Risikoparameters: Die Steuerungswirkung und Interpretation des Risikoparameters wird in einen Mikro- und einen Makroeffekt unterteilt. Der Mikroeffekt beschreibt die Wirkung des Risikoparameters r in Bezug auf die Risikobewertung einer Kante. Der Makroeffekt ergibt sich daraus, dass im Klassifikationsmodell ganzen Regionen Kollisionswahrscheinlichkeiten zugeordnet werden.

Im Regressionsansatz kann der Risikoparameter r wie folgt interpretiert werden: Unter der Annahme, dass der bisher beste Weg mit Sicherheit zu einer Kollision führt (binäre Entscheidungen im Regressionsansatz) und es einen alternativen Weg mit zusätzlichen Kosten von $k < r$ gibt, entscheidet sich der Agent für den Umweg. Eine Risikosteuerung im eigentlichen Sinne findet nicht statt, da der Agent nur binär zwischen den Kollisionskanten unterscheidet. Die Kosten eines Pfades ist die Summe der Kosten der in ihm enthaltenen Kanten. Die Kosten einer Kante setzen sich wiederum aus deren Länge und dem mit dem Risikoparameter r gewichteten, vorhergesagten Kollisionsrisiko zusammen. Im Regressionsansatz besteht das Kollisionsrisiko einer Kante aus einer binären Entscheidung, ob die Kante im Zeitintervall ihrer Nutzung durch den Agenten von der vorhergesagten, zukünftigen Trajektorie eines Hindernisses geschnitten wird und ist dementsprechend entweder 0% oder 100%. Nimmt man an, dass ein Wert von $r = 100$ eingestellt ist und sich auf der kürzesten, rein auf den Kantenlängen basierenden Route ein Kante befindet, die zu einer Kollision führt, so verteuert sich diese Route um 100, weil über die eine Kollisionskante r einmal in die Pfadkosten einfließt (analog, mehrfach bei multiplen Kollisionskanten im Pfad). Dementsprechend wird der Agent eine alternative Route wählen, welche die Kollisionskante vermeidet, sofern die zusätzlichen Kosten dieser Alternativroute < 100 sind. Je größer der Risikoparameter r ist, desto länger sind die Umwege, die der Agent in Kauf nimmt, um eine Kollision zu vermeiden.

Im Klassifikationsmodell werden den Kanten Kollisionswahrscheinlichkeiten zugeordnet. Nimmt man an, dass es genau zwei mögliche Wege zum Zielpunkt

gibt, sich der Agent auf der attraktivsten Route befindet und ein Kollisionsrisiko von 1% vorausgesagt wird. Die alternative Route verursacht 8 zusätzliche Kosteneinheiten. Der Agent hat einen Parameterwert von $r = 1000$. Das Kollisionsrisiko von 1% erhöht die Kosten für die ursprüngliche Route um 10. Dann wird der Agent den Umweg wählen, um das Risiko zu vermeiden. Er zeigt ein risikoaverses Verhalten. Der Parameterwert von $r = 100$ resultiert in einem risikoaffineren Agenten. In diesem Fall muss das Kollisionsrisiko des aktuellen Wegs mindestens 8% betragen, damit der Agent sich für den Umweg entscheidet. Abstrahiert man von einzelnen Kanten zu einer Makroperspektive, erhält man im Klassifikationsmodell einen weiteren Steuerungseffekt von r. Im Gegensatz zum Regressionsmodell werden alle dem Agenten zugänglichen Kanten mit Risikowerten belegt. Die Vorhersagen teilen die Umgebung in Zellen mit hohem und niedrigem Risiko auf. Typischerweise ist der Übergang zwischen diesen Bereichen fließend (siehe Abbildung 3.18). Aus der in den Vorhersagen beschriebenen Eigenschaft folgt, dass der Agent einen größeren Bogen um Regionen mit einem hohen Kollisionsrisiko macht. Je höher der Risikoparameter ist, desto größer ist der Sicherheitsabstand zwischen dem geplanten Weg und den zukünftigen Bewegungen des Hindernisses. Es ist ersichtlich, dass der Wert des Risikoparameters r in seiner aktuellen Ausgestaltung in Abhängigkeit von den während des Routings des Agenten auftretenden Pfadlängen gewählt werden muss, um dessen Entscheidungen sinnvoll zu beeinflussen.

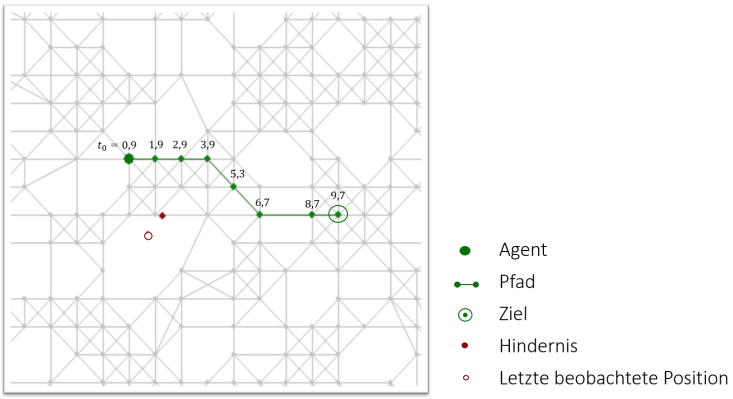


Abbildung 3.23: Beispiel für eine konkrete Planungsentscheidung.

Ein konkretes Planungsproblem ist in Abbildung 3.23 dargestellt. Es zeigt einen Agenten, der sich in der Nähe eines Hindernisses befindet. Der Agent muss bei seiner Planungsentscheidung berücksichtigen, dass sich das Hindernis (roter Punkt) seit der letzten Beobachtung (roter Kreis) weiterbewegt hat. Seit der letzten Beobachtung sind 0,9 Zeitschritte vergangen ($t_0 = 0{,}9$). Die Nutzungszeitpunkte der Kanten werden an den Knoten angegeben. Der Nutzungszeitpunkt einer Kante ist entscheidend für die Berechnung der Kollisionskosten

und des Ausweichverhaltens. Die nächsten Abschnitte zeigen die verschiedenen Modelle in Aktion.

Ausweichverhalten im Regressionsmodell Abbildung 3.24 zeigt das Ausweichverhalten des Agenten für das Regressionsmodell mit einem Risikoparameter von $r = 100$.

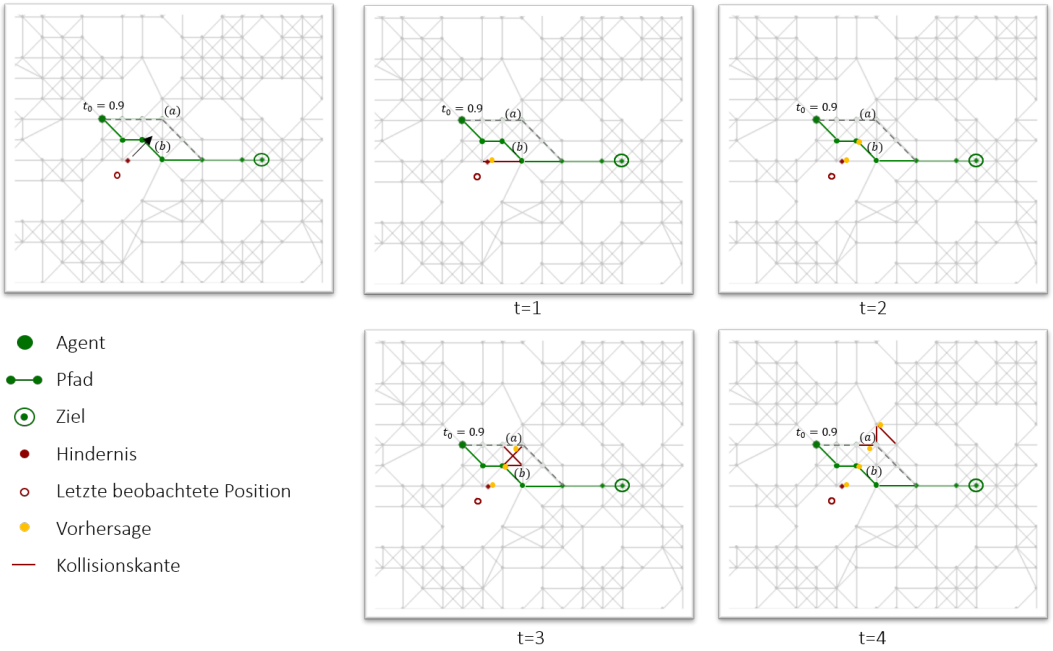


Abbildung 3.24: Ausweichverhalten eines Agenten mit Regressionsmodell und $r = 100$. Momentaufnahmen der Kantengewichte zu vier verschiedenen, aufeinanderfolgenden Zeitpunkten.

Der Agent sagt eine Kollision auf seinem vorherigen (die Kante (a) enthaltenden) Weg voraus und weicht dieser aus. Es ist zu erkennen, dass der Agent das Hindernis knapp hinterläuft. Das bedeutet, dass die in der Alternativroute enthaltene Kante (b) ebenfalls von der Trajektorie des Hindernisses geschnitten wird, das Hindernis diese Kante aber bereits passiert hat, wenn der Agent sie nutzt. Es fällt auf, dass die neue Lösung die gleiche Länge wie die alte hat. Damit kann der Agent einem Kollisionsszenario ausweichen, ohne einen Umweg zu nehmen. Um das Ausweichverhalten des Agenten verstehen zu können, werden verschiedene Momentaufnahmen des Graphen zu verschiedenen Zeitpunkten dargestellt. Die zukünftigen Positionen der Hindernisse werden als Koordinaten vorhergesagt und Kanten, die zu einer potenziellen Kollision führen, werden im entsprechenden Zeitschritt rot markiert. Sie verursachen zusätzliche Kosten von $r = 100$. Die Kante (a) führt zu einer Kollision im Zeitintervall $[3, 4]$ und die Kante (b) zwischen $[1, 2]$. Beide Wege schneiden die

vorhergesagte Bewegungsbahn des Hindernisses. Der Unterschied liegt im Nutzungszeitpunkt der Kanten, welchen der Agent im Planungsprozess berechnet bzw. die Ankunftszeit am Startknoten einer Kante. Die Kante (a) würde durch den Agenten im Intervall [2,9, 3,9] genutzt. Der Agent sagt voraus, dass die Kante in diesem Intervall zu einer Kollision führt und vermeidet sie daraufhin, weil es eine alternative Route gibt. Im Gegensatz dazu verwendet der Agent in der neuen Lösung die Kante (b) im Intervall [2,3, 3,7]. Durch die zeitabhängige Kostenfunktion der Kanten weiß der Agent implizit, dass das Hindernis die Kante bereits vor mindestens 0,3 Zeiteinheiten passiert hat. In dem gezeigten Beispiel sind die zentralen Eigenschaften des Regressionsmodells zu sehen. Der Agent belegt nur die Kanten mit Kollisionskosten, die durch das Hindernis geschnitten werden. Der Agent unterscheidet nicht zwischen Kanten, die nahe der vorhergesagten Bewegungsbahn liegen, und Kanten, die weit davon entfernt sind. Aus diesem binären Entscheidungskriterium folgt das typische Vermeidungsverhalten des Agenten. Die Umwege zur Vermeidung einer Kollision sind so minimal wie möglich.

Ausweichverhalten im Klassifikationsmodell Mit dem Klassifikationsmodell kann der Agent nicht nur einzelne Kanten unterscheiden, sondern es lässt sich auch die Länge des Umweges steuern. Daher wird das Vermeidungsverhalten für einen risikoaffinen ($r = 100$) und einen risikoaversen ($r = 1000$) Agenten betrachtet. Abbildung 3.25 zeigt das Ergebnis für $r = 100$.

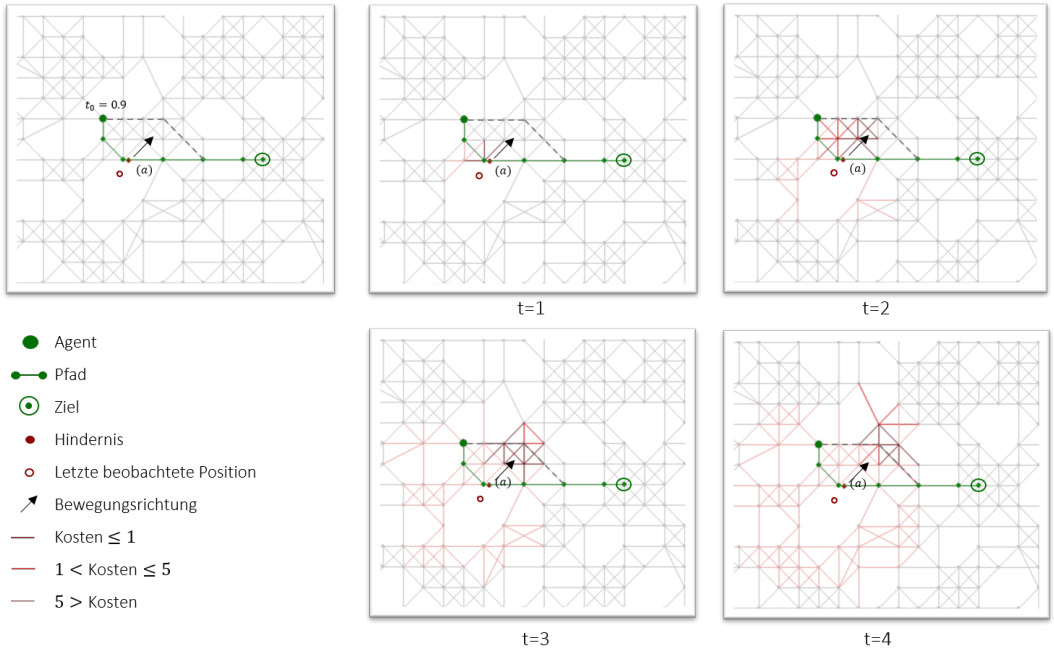


Abbildung 3.25: Ausweichverhalten eines Agenten mit Klassifikationsmodell und $r = 100$. Momentaufnahmen der Kantengewichte zu vier verschiedenen, aufeinanderfolgenden Zeitpunkten.

Im Klassifikationsmodell wird allen zugänglichen Kanten ein Kollisionsrisiko zugeordnet. Entsprechend verursachen alle Kanten zusätzliche Kollisionskosten. Die unterschiedlichen Kosten der Kanten werden durch Farbschattierungen angezeigt. Je dunkler die Kante, desto höher die Kosten. Der zukünftige Bewegungsverlauf des Hindernisses ist am Farbverlauf leicht zu erkennen, da der Risikobereich (dunkle Kanten) sich mit der Zeit über den Graphen bewegt. Die Kante (a) hat ein hohes Kollisionsrisiko im Zeitschritt $t = 1$ und ein geringes Risiko in $t = 4$. Entscheidend für den Agenten sind die Kosten einer Kante zum Zeitpunkt der Nutzung. Ähnlich wie beim Regressionsmodell hinterläuft der Agent das Hindernis relativ eng, d.h., er nutzt Kanten, nachdem das Hindernis diese passiert hat. Daraus resultiert ein etwas länger, neuer Weg aber der Agent vermeidet dunkle Kanten.

Es lohnt sich für den Agenten nicht, einen noch größeren Umweg zu machen, um das Restrisiko der nun genutzten Kanten zu vermeiden. Das Beispiel zeigt das Gleichgewicht zwischen Kollisionsrisiko und Pfadlänge.

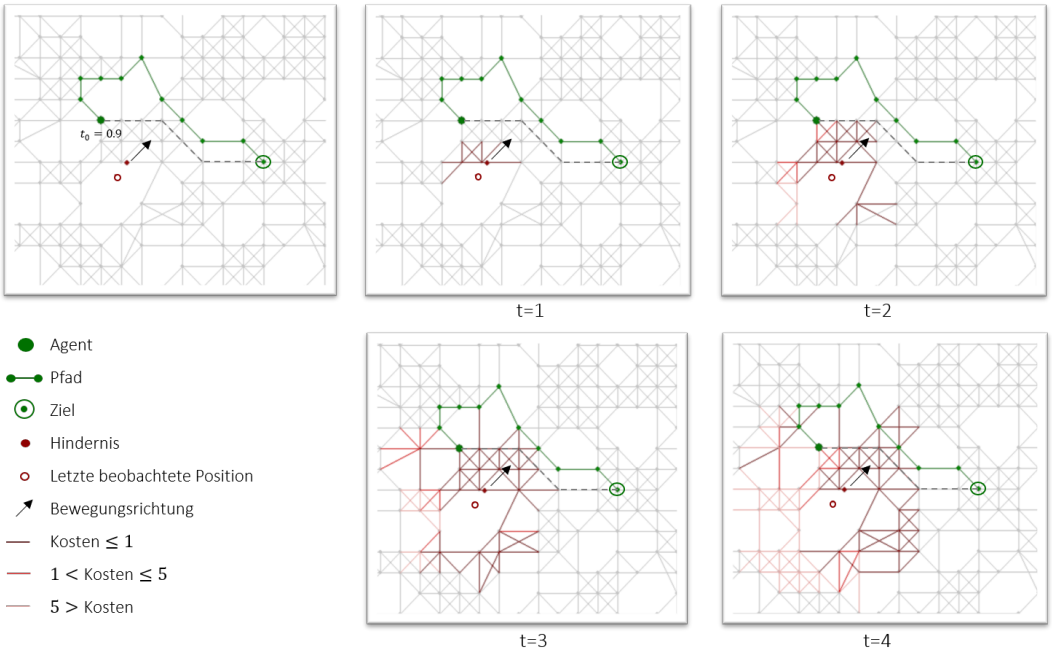


Abbildung 3.26: Ausweichverhalten eines Agenten mit Klassifikationsmodell und $r = 1000$. Momentaufnahmen der Kantengewichte zu vier verschiedenen, aufeinanderfolgenden Zeitpunkten.

Mit $r = 1000$ hat ein Agent eine viel kleinere Toleranzgrenze hinsichtlich des Kollisionsrisikos und wird damit risikoaverser. Abbildung 3.26 zeigt das Ergebnis für $r = 1000$. Je höher r eingestellt ist, umso mehr verteuern sich die Kanten, für die ein Kollisionsrisiko vorhergesagt wird und der Agent wird eher Kanten wählen die (rein auf der Grundlage ihrer Länge) zu einem längeren Weg führen aber kein Kollisionsrisiko aufweisen.

Auf den ersten Blick ist zu erkennen, dass ein großer Teil der erreichbaren Kanten hohe Kosten verursacht. Die vorhergesagten Kollisionswahrscheinlichkeiten sind die gleichen wie im vorherigen Bild (Abbildung 3.25). Allerdings hat sich die Risikobeurteilung geändert. Eine Kante mit einer Kollisionswahrscheinlichkeit von 1% und $r = 1000$ verursacht die gleichen Kosten wie ein Risiko von 10% mit $r = 100$. Dementsprechend ist der Risikobereich (dunkle Kanten) viel größer als zuvor. Die Ausdehnung des Risikobereichs kann durch den Risikoparameter r (multiplikativer Faktor) gesteuert werden. Infolgedessen weicht der Agent dem Hindernis in einem großen Bogen aus. Diese Beobachtung wurde zuvor als Makroeffekt eingeführt. Je größer der Risikoparameter, desto risikoscheuer ist der Agent und desto weiter weicht er einem Hindernis aus.

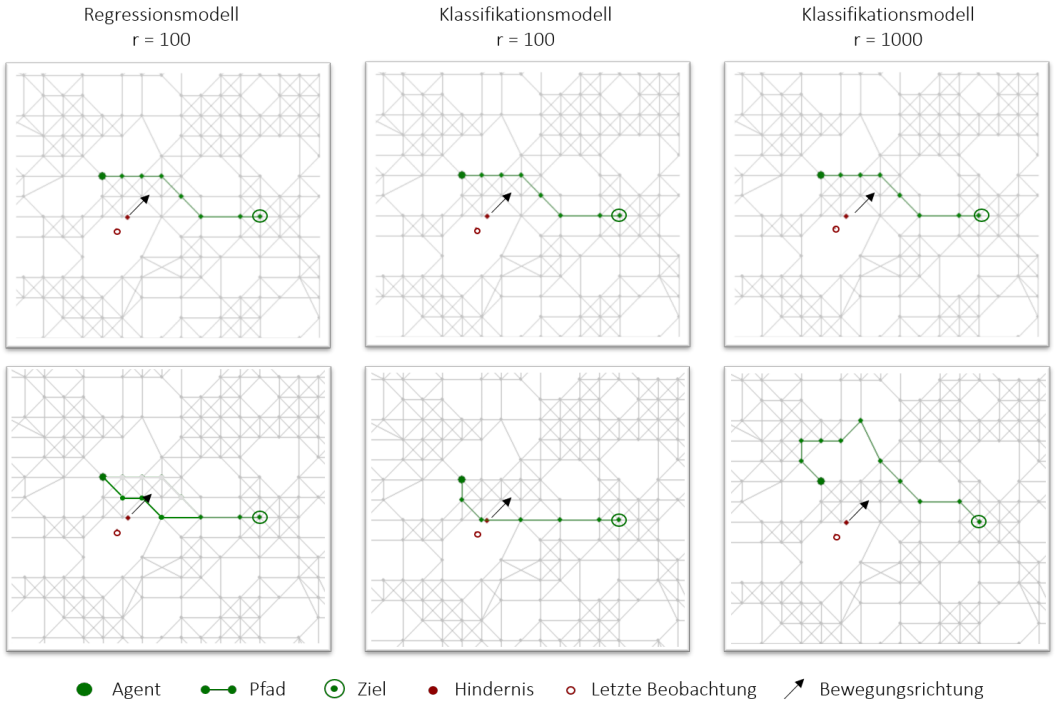


Abbildung 3.27: Ergebnisse des PCMP in verschiedenen Varianten für eine konkrete Planungsentscheidung.

Vergleich: Abbildung 3.27 zeigt die verschiedenen Ergebnisse nebeneinander. Die verschiedenen Modelle verursachen ein unterschiedliches Kollisionsverhalten. Mit dem Regressionsmodell weicht der Agent einem Hindernis recht knapp aus, da die Kanten binär mit einem Risiko behaftet sind oder nicht. Mit dem Klassifikationsmodell ist der Abstand zwischen direktem bzw. kürzestem Pfad und Ausweichen des Hindernisses kontrollierbar. Die konkrete Ausweichentscheidung des Agenten stellt immer ein Gleichgewicht zwischen

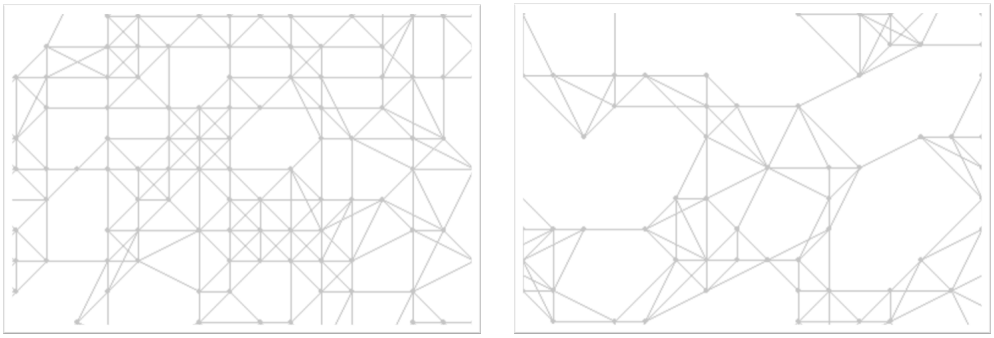
dem Kollisionsrisiko und dem kürzesten Weg dar. Mit dem Risikoparameter kann der Agent einen der beiden Faktoren priorisieren.

3.5.6 Evaluation

Die Evaluation vergleicht die Kollisionsvermeidungsstrategie mit verschiedenen Bewegungsmodellen für unterschiedliche Einstellungen des Risikoparameters r sowie für verschiedene Anwendungsszenarien.

Das Vermeidungsverhalten des Agenten stellt einen Kompromiss zwischen Kollisionsrisiko und Vermeidungskosten dar. Dementsprechend sind die Anzahl der vermiedenen Kollisionen und die Länge des zusätzlichen Umweges die wichtigsten Messwerte zur Beurteilung von PCMP. Um sicherzustellen, dass die Ergebnisse der Experimente das Ausweichverhalten der spezifischen Lösung widerspiegeln und unabhängig von den konkreten Planungsaufgaben des Agenten sind, soll eine ausreichend große Anzahl von Pfadplanungsoperationen in Betracht gezogen werden. So bestehen die Experimente aus 1000 zufällig platzierten Zielen für den Agenten. Wenn der Zielknoten erreicht ist, wird automatisch ein neues Ziel gesetzt.

Der zurückgelegte Weg besteht aus der Gesamtlänge der verwendeten Kanten. Für jede Kante wird berechnet, ob eine Kollision stattgefunden hat. Beide gemessenen Werte hängen vom implementierten Modell und vom Risikoparameter des Agenten ab. Neben dem Risikoparameter spielen auch die Feinheit der Diskretisierung, welche die Anzahl und Länge der Alternativrouten beeinflusst und die Anzahl der dynamischen Hindernisse eine wichtige Rolle.



(a) Dichter Graph (b) Dünn besetzter Graph

Abbildung 3.28: Graphen mit unterschiedlicher Feinheit.

Abbildung 3.28 zeigt Ausschnitte von zwei Graphen unterschiedlicher Feinheit. Konkret wird der PCMP in drei verschiedenen Szenarien evaluiert:

1. Szenario (a): 8 Hindernisse und Diskretisierung nach Abbildung 3.28a
2. Szenario (b): 8 Hindernisse und Diskretisierung nach Abbildung 3.28b
3. Szenario (c): 16 Hindernisse und Diskretisierung nach Abbildung 3.28a

Im Hauptteil der Evaluation kommt Szenario (a) zur Anwendung. Die Abschnitte 3.5.6.1 und 3.5.6.2 stützen sich auf dieses und stellen detailliert die Ergebnisse für das Regressionsmodell bzw. Klassifikationsmodell dar. In Abschnitt 3.5.6.3 werden die Ergebnisse für Szenario (b) und (c) vorgestellt.

Eine Simulation über 1000 Pfade für einen Agenten ohne Kollisionsvermeidungsstrategie (entspricht $r = 0$) liefert Referenzwerte für die Pfadlänge (rein auf Grundlage der Kantenlängen) und der aufgetreten Kollisionen mit Hindernissen auf dem kürzesten Pfad, die mit anderen Einstellungen/Szenarien verglichen werden. In Szenario (a) mit 8 Hindernissen legt der Agent eine Gesamtstrecke von **17117** Längeneinheiten zurück. Der erwartete Wert für einen Agenten ohne Kollisionsvermeidungsstrategie ist **184** Kollisionen. Der Agent muss immer die gleichen 1000-Ziele erreichen.

3.5.6.1 Ergebnisse des Regressionsmodells im Szenario (a)

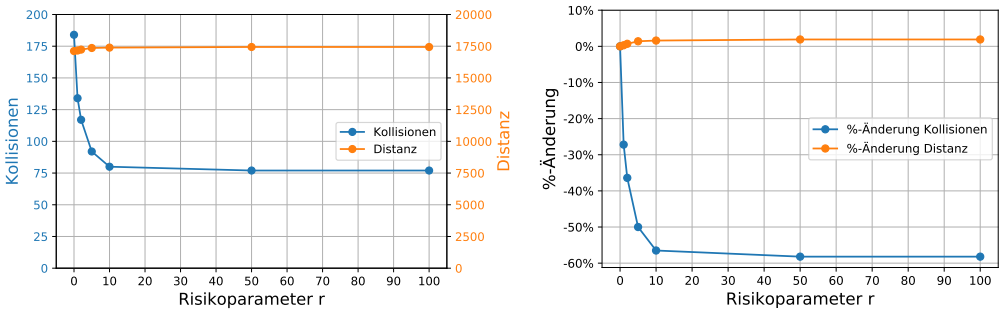
Tabelle 3.2 zeigt die Ergebnisse für PCMP mit einem Regressionsmodell, mit variiertem Risikoparameter. Es fällt auf, dass der Risikoparameter und die Anzahl der Kollisionen negativ korreliert sind. Die Korrelation zwischen dem Risikoparameter und der Länge des Umweges ist positiv.

Tabelle 3.2: Ergebnisse der Kollisionssteuerung mit dem Regressionsmodell im Szenario (a) in Abhängigkeit vom Risikoparameter r. Distanz und Umweg werden in generischen Längeneinheiten (LE) angegeben.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	184	0	0,0%	17117	0	0,0%
1	134	50	-27,2%	17160	43	+0,3%
2	117	67	-36,4%	17232	115	+0,7%
5	92	92	-50,0%	17360	243	+1,4%
10	80	104	-56,5%	17389	272	+1,6%
50	77	107	-58,2%	17439	322	+1,9%
100	77	107	-58,2%	17439	322	+1,9%

Die Tabelle zeigt, dass der Agent bei einem Risikoparameter von $r = 1$, 50 von 184 Kollisionen vermeidet. In dem gegebenen Graph kann der Agent insgesamt 27,2% der Kollisionen durch einen Umweg von 0,3% vermeiden. Ab einem Wert von $r = 50$ ist kein zusätzlicher Kontrolleffekt im Ausweichverhalten des Agenten erkennbar. Im gegebenen Szenario kann der Agent fast 60% der Kollisionen vermeiden. Diese Reduktion erreicht er mit einem Gesamtumweg von etwa 2 %. Abbildung 3.29a zeigt die Anzahl der Kollisionen und die zurückgelegte Strecke in absoluten Zahlen. In Abbildung 3.29b sind die prozentualen Abweichungen der gemessenen Werte von den Referenzwerten bei $r = 0$ dargestellt.

Die Ergebnisse lassen sich durch die Struktur des Graphen und den gewählten Integrationsansatz erklären (siehe auch Abschnitt 3.5.3.2). Das Regressi-



(a) Kollisionen und zurückgelegte Distanz - absolute Werte

(b) Kollisionen und zurückgelegte Distanz - relative Änderungen zum Referenzwert $r = 0$

Abbildung 3.29: Kollisionssteuerung mit dem Regressionsmodell in Abhängigkeit vom Risikoparameter r.

onsmodell bestraft betroffene Kanten sehr selektiv (d.h. einzelne Kanten und nicht ganze Zonen, wie im Klassifikationsansatz) und durch den hohen Vernetzungsgrad der Knoten resultieren viele alternative Wege zu einem Zielknoten, welche von der Risikobestrafung häufig unberührt bleiben. Der in diesem Abschnitt verwendete Graph hat eine maximale Kantenlänge von 3 Längeneinheiten. Es macht keinen wirklichen Unterschied, ob der Risikoparameter r einen Wert von 10 oder 100 hat, wenn aufgrund des Vernetzungsgrads ein alternativer Weg mit 5 zusätzlichen Längeneinheiten gefunden werden kann, der die Kollisionskante vermeidet. Dementsprechend ist der Einfluss des Risikoparameters begrenzt. Ein weiter Einflussfaktor auf die Ergebnisse ist die Qualität des Vorhersagemodells, was in der Prädiktion der Hindernistrajektorien nicht immer korrekt ist.

3.5.6.2 Ergebnisse des Klassifikationsmodells im Szenario (a)

Die Tabelle 3.3 stellt die Ergebnisse des Klassifikationsmodells für verschiedene Risikobewertungen des Agenten durch den Risikoparameter r dar. Aufgrund der Makrowirkung des Risikoparameters (beschrieben im Abschnitt Ausweichverhalten im Klassifikationsmodell) ist zu erwarten, dass das Kollisionsvermeidungsverhalten über den Parameterwert differenzierter gesteuert werden kann.

Grundsätzlich kann folgende Korrelation in den Ergebnissen für $r \in [0{,}500]$ beobachtet werden: Je größer die Risikoaversion des Agenten ist, desto mehr Kollisionen werden vermieden und desto länger sind die Umwege. Ein risikoaffiner Agent ($r = 1$) wird nur dann vom kürzesten Weg abweichen, wenn die Wahrscheinlichkeit einer Kollision sehr hoch ist und die Alternativlösung nur einen minimalen Umweg bedeutet. Dieser Agent vermeidet etwa 25% der Kollisionen mit einem Umweg von 0,5%. Er schätzt ein Kollisionsrisiko von 100% in der gleichen Weise ein, wie ein risikoaverser Agent ($r = 100$) ein Kollisionsrisiko von 1% schätzt. Der risikoaverse Agent vermeidet etwa 93% der

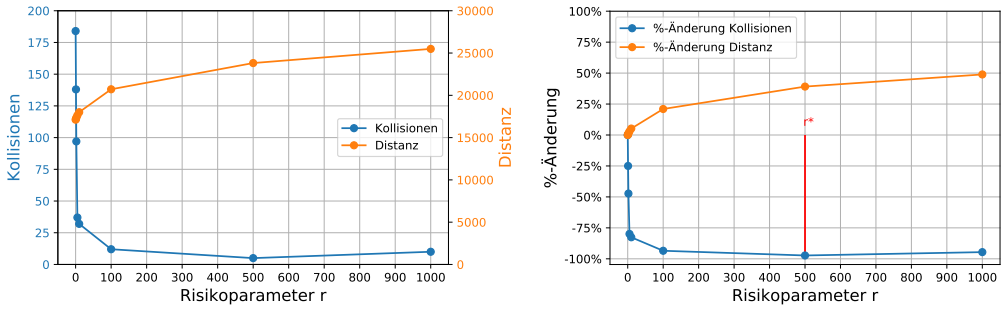
Tabelle 3.3: Ergebnisse der Kollisionssteuerung mit dem Klassifikationsmodell im Szenario (a) in Abhängigkeit vom Risikoparameter r. Distanz und Umweg werden in generischen Längeneinheiten (LE) angegeben.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	184	0	0,0%	17117	0	0,0%
1	138	46	-25,0%	17202	85	+0,5%
2	97	87	-47,3%	17346	229	+1,3%
5	37	147	-79,9%	17638	521	+3,0%
10	32	152	-82,6%	18005	888	+5,2%
100	12	172	-93,5%	20714	3597	+21,0%
500	5	179	-97,3%	23815	6698	+39,1%
1000	10	174	-94,6%	25489	8372	+48,9%

Kollisionsszenarien und nimmt einen Umweg von 21% in Kauf. Abbildung 3.30 zeigt die Simulationsergebnisse der Klassifikationslösung.

Mit Hilfe des Risikoparameters kann die Anzahl der Kollisionsszenarien auf ein sehr kleines Maß reduziert werden. Je größer der Risikoparameter, desto größer ist der Bogen um ein Hindernis. Dieser hängt von der Gewissheit der Vorhersage ab, da sie die Größe der Region bestimmt, zu der die Kosten hinzugerechnet werden. Die Ergebnisse zeigen, dass Kollisionen nicht zu 100% vermieden werden können, was sich durch den begrenzten Planungshorizont erklären lässt.

Die Ergebnisse zeigen auch, dass die Anzahl der Kollisionen für $r = 1000$ im Vergleich zu $r = 500$ leicht ansteigt. Aufgrund des Makroeffektes vermeidet der Agent zunehmend Hindernisse (siehe Abschnitt Ausweichverhalten im Klassifikationsmodell). Auf diesen langen Umwegen begegnet der Agent Hin-



(a) Kollisionen und zurückgelegte Distanz - absolute Werte

(b) Kollisionen und zurückgelegte Distanz - relative Änderungen zum Referenzwert $r = 0$

Abbildung 3.30: Kollisionssteuerung mit dem Klassifikationsmodell in Abhängigkeit vom Risikoparameter r.

dernissen, denen er nicht begegnet wäre, wenn er den ersten Hindernissen nicht ausgewichen wäre. Die in Abbildung 3.30b dargestellten Messwerte für die relative Anzahl vermiedener Kollisionen haben ein Minimum bei r^* Im Intervall $[0, r^*]$ findet der Agent die bestmöglichen Kombinationen von Umweg und Kollisionsrisiko. Zusammenfassend lässt sich sagen, dass der Agent über das Klassifikationsmodell bis zu 97,3% der Kollisionsszenarien vermeiden kann.

3.5.6.3 Alternative Szenarien

Im Folgenden soll PCMP in Szenarien untersucht werden, bei denen der vorgegebene Graph dünner besetzt, das heißt nicht so feingranular ist oder die Anzahl der dynamischen Hindernisse variiert.

Szenario (b) - Dünn besetzter Graph: Die Diskretisierung des Raums beeinflusst die Fähigkeit des Agenten, Kollisionen zu vermeiden. Bei einer gröberen Diskretisierung hat der Agent bei seinen Planungsentscheidungen weniger Alternativen. Es ist daher zu erwarten, dass die Umwege dadurch tendenziell länger werden. Tabelle 3.4 und Tabelle 3.5 zeigen die Ergebnisse für Szenario (b). Die Referenzwerte in diesem Szenario sind **249** Kollisionen auf dem direkten Weg (ohne Ausweichverhalten) von **17866** Längeneinheiten (LE).

Tabelle 3.4: Regression PCMP für Szenario (b): Dünn besetzter Graph.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	249	0	0,0%	17866	0	0,0%
1	200	49	-19,7%	17915	49	+0,3%
5	124	125	-50,2%	18182	316	+1,8%
10	106	143	-57,4%	18343	477	+2,7%
100	113	136	-54,6%	18416	550	+3,1%

Tabelle 3.5: Klassifikation PCMP für Szenario (b): Dünn besetzter Graph.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	249	0	0,0%	17866	0	0,0%
1	173	76	-30,5%	17950	84	+0,5%
10	37	212	-85,1%	18942	1076	+6,0%
100	20	229	-92,0%	22539	4673	+26,2%
500	13	236	-94,8%	27271	9405	+52,6%
1000	13	236	-94,8%	29919	12053	+67,5%

Die Ergebnisse zeigen, dass PCMP sowohl für eine grobe als auch für eine feine Diskretisierung funktioniert. Dies lässt vermuten, dass die Funktionalität von PCMP unabhängig vom konkreten Graph zu sein scheint. Die Struktur

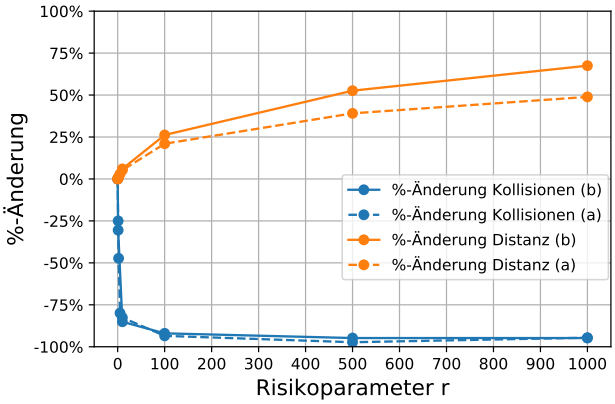


Abbildung 3.31: Vergleich zwischen grober (b) und feiner Diskretisierung (a) im Klassifikationsmodell - prozentuale Änderung an Kollisionen und zurückgelegter Distanz.

des Graphen beeinflusst jedoch das Verhältnis zwischen vermiedenen Kollisionen und dem damit einhergehenden Umweg. Die Ergebnisse in Tabelle 3.5 zeigen eine deutliche Zunahme des Umweges im Vergleich zu Szenario (a) bei gleichbleibendem prozentualen Anteil an Kollisionen.

Der Effekt unterschiedlicher Diskretisierungen ist in Abbildung 3.31 für das Klassifikationsmodell dargestellt. Sie zeigt, dass der prozentuale Umweg für $r = 1000$ in Szenario (b) um 18,6% länger ist als in Szenario (a) (siehe Tabelle 3.3: 48,9% Umweg im Vergleich zu Tabelle 3.5). Dadurch wird ungefähr die gleiche Anzahl von Kollisionen vermieden ((a): 94,6% und (b): 94,8%). Dieser Unterschied ist auf die Feinheit des Graphen zurückzuführen. Mit einem feinen Graphen hat der Agent viele Möglichkeiten, ein Kollisionsrisiko mit einem kleinen Umweg zu vermeiden. In Szenario (b) hat der Agent weniger Möglichkeiten, sich im Raum zu bewegen. Daher führen alternative Routen tendenziell zu längeren Umwegen.

Szenario (c) - Viele Hindernisse In diesem Szenario sieht sich der Agent mit einer Vielzahl von Hindernissen konfrontiert. Konkret gibt es 16 Hindernisse, die sich bewegen (im Vergleich zu 8 Hindernissen vorher). Tabelle 3.6 und Tabelle 3.7 zeigen die Ergebnisse. Die Referenzwerte in diesem Szenario sind **372** Kollisionen auf dem direkten Weg (ohne Ausweichverhalten) von **17117** Längeneinheiten (LE).

Es wird deutlich, dass PCMP auch in Umgebungen mit vielen Hindernissen einsetzbar ist. Der Agent kann erfolgreich zwischen Hindernissen navigieren und risikobasierte Ausweichentscheidungen treffen. Die Ergebnisse für das Regressionsmodell stimmen mit den anderen Szenarien überein und unterstreichen die Wirksamkeit dieses Ansatzes. Abbildung 3.32 zeigt die Auswirkung der Zunahme an Hindernissen auf die gemessenen Werte. Es ist zu erkennen, dass der Agent einem ähnlichen Prozentsatz von Hindernissen ausweicht wie

Tabelle 3.6: Ergebnisse PCMP mit Regressionsmodell für Szenario (c): Viele Hindernisse.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	364	8	-2,2%	17117	0	0,0%
1	241	131	-35,2%	17204	87	+0,5%
5	179	193	-51,9%	17565	448	+2,6%
10	162	210	-56,5%	17658	541	+3,2%
100	163	209	-56,2%	17719	602	+3,5%

Tabelle 3.7: Ergebnisse PCMP mit Klassifikationsmodell für Szenario (c): Viele Hindernisse.

Risiko-parameter r	Kollisionen	Vermiedene Kollisionen	%-Änderung Kollisionen	Zurückgelegte Distanz (LE)	Umweg (LE)	%-Änderung Distanz
0	364	8	-2,2%	17117	0	0,0%
1	236	136	-36,6%	17262	145	+0,8%
10	56	316	-84,9%	18936	1819	+10,6%
100	28	344	-92,5%	25186	8069	+47,1%
500	34	338	-90,9%	36142	19025	+111,1%
1000	33	339	-91,1%	40277	23160	+135,3%

in Szenario (a), dabei aber einen größeren Umweg nehmen muss.

Der starke prozentuale Anstieg der Umwege ist auf die größere Anzahl von Hindernissen zurückzuführen. Logischerweise muss der Agent mehr potenzielle Kollisionen vermeiden und daher größere Umwege in Kauf nehmen. Wenn sich der Agent in einer Umgebung mit vielen Hindernissen bewegt, läuft er Gefahr, in einer Endlosschleife stecken zu bleiben. Dies kann passieren, wenn der Agent zwischen den Laufbahnen mehrerer Hindernisse gefangen ist, die in einer pa-

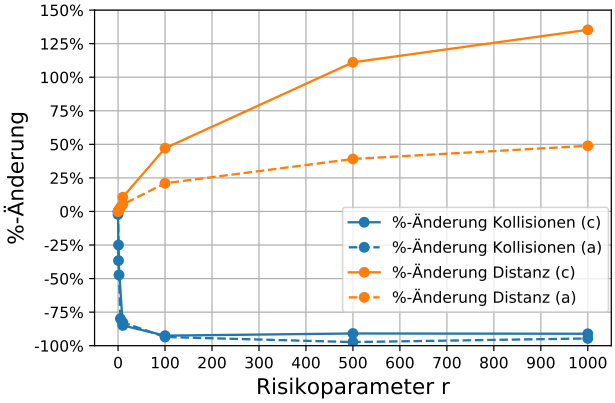


Abbildung 3.32: Vergleich der Ergebnisse des Klassifikationsmodells mit 8 Szenario (a) und 16 Szenario (c) Hindernissen - prozentuale Änderung an Kollisionen und zurückgelegter Distanz.

rabolischen Form oszillieren. Wenn der Agent eine solche Sackgasse feststellt, erhöht er seine Risikopräferenz, um die Situation lokal aufzulösen. Auf diese Weise erhält das Kollisionsrisiko eines Pfades weniger Gewicht und der Agent kommt seinem Ziel näher.

3.5.7 Diskussion und Zusammenfassung

Es wurde ein Kollisionskontrollkonzept für die vorausschauende Pfadplanung namens PCMP vorgestellt. Die Zeitdimension ist das Bindeglied zwischen Bewegungsprognosen und zeitabhängigen Pfadplanungsentscheidungen. In diesem Zusammenhang wird der Rolle der Zeit in der Literatur nur wenig Aufmerksamkeit geschenkt. Vorhersagen zu Kollisionsobjekten sind Wahrscheinlichkeitsaussagen, die bei der Pfadplanung ein Kollisionsrisiko darstellen. Die Betrachtung zeitabhängiger Risikofunktionen ist in früheren Arbeiten oft vernachlässigt worden. Aus dieser Lücke ergibt sich die Notwendigkeit, Relevanz und der Beitrag des hier entwickelten Ansatzes. Durch die Verwendung von LSTM-Modellen lernt der Agent eine Intuition für die Bewegungsmuster dynamischer Hindernisse in seiner Umgebung. Die Vorhersagen zeitabhängiger, zukünftiger Positionen werden als Trajektorien (Regressionsmodell) und Belegungsgitter (Klassifikationsmodell) formuliert und in einen Graphen integriert, um sie für einen Agenten nutzbar zu machen. Die Modelle unterscheiden sich hinsichtlich Informationsgehalt und Komplexität. Das zentrale Konzept zur Integration der Bewegungsprognosen in Graphen sind die zeitabhängigen Kantenkosten. Ein hierfür modifizierter A*-Algorithmus ermöglicht die Suche nach optimalen Pfaden in der Raum-Zeit-Dimension. Die Lösung des Pfadplanungsalgorithmus stellt einen Kompromiss zwischen Kollisionsrisiko und Pfadlänge dar, die über einen Risikoparameter gesteuert werden kann. Einerseits soll sich der Agent auf dem kürzesten Weg bewegen, andererseits sollen Kollisionsszenarien bestmöglich vermieden werden. Die Ergebnisse zeigen, dass PCMP diese Entscheidung erfolgreich in den Planungsprozess des Agenten integriert. PCMP eignet sich daher als prädiktiver Kollisionsmanagement-Algorithmus für die zeit- und risikoabhängige Wegplanung.

3.6 Zusammenfassung und Ausblick

In diesem Kapitel wurden zwei neuartige Ansätze vorgestellt, welche die Vorhersagen aus gelernten Umgebungsmodellen verwenden, die auf künstlichen neuronalen Netzen basieren, um dynamischen Hindernissen auszuweichen oder sich bewegende Objekte gezielt anzusteuern.

Mit dem ersten Verfahren wurde vorgeschlagen, dass ein Agent mit Hilfe eines erlernten Modells explizit zukünftige Zustände der Umwelt baumartig und diskret durchsucht, diese auf ihre Nützlichkeit hin bewertet und dann die erfolgversprechendste Sequenz von Aktionen zur Ausführung auswählt. Dies

geschieht mit einem vordefinierten Zeithorizont, der die Länge der auszuführenden Sequenz bestimmt und ebenfalls mit Schritten in einer festen, vordefinierten Länge. Verwandte Arbeiten zu Planungsansätzen wie [144, 10] lassen offen woher das Modell zur Vorausplanung von Folgezuständen stammt. Deshalb wurde hier die Integration eines erlernten Modells in den Planungsprozess vorgeschlagen, da es die in Abschnitt 3.4.1 genannten Herausforderungen eines exakten, physikalischen Modells überwindet, weil es vergleichsweise leicht zu erstellen und flexibel in der Anpassung an veränderte bzw. zuvor unbekannte Umgebungsbedingungen ist. Die Evaluationsergebnisse in Abschnitt 3.4.5 haben gezeigt, dass die Performance eines Agenten, der auf einem erlernten Umgebungsmodell basiert, mit dem eines Agenten, der über ein exaktes physikalisches Umgebungsmodell verfügt, mithalten kann aber sich signifikante Verbesserungen in der Berechnungszeit, die für die Erstellung der Pläne benötigt wird, ergeben.

Weil das baumartige Simulieren von Folgezuständen rechenintensiv ist, da über die Teiläste des Baumes hinweg redundante Berechnungen durchgeführt werden, wurde mit dem zweiten Ansatz vorgeschlagen die Vorhersagen des erlernten Bewegungsmodells direkt in einen Graphen zu integrieren, damit auf diesem mit klassischen Pfadplanungsalgorithmen eine kürzeste Route gesucht werden kann, welche aber auch gleich dynamischen Hindernissen ausweicht. Die Integration erfolgt dabei über zeitabhängige Kantenkosten, welche dann höher gesetzt werden, wenn die vorhergesagte Trajektorie eines Hindernisses die betreffende Kante schneidet bzw. sich die Kante in einer Zone befindet, für die das Hindernis vorhergesagt wird. Darüber hinaus wurde in diesem Kapitel eine Erweiterung klassischer Pfadplanungsalgorithmen wie des Dijkstra oder A* Algorithmus vorgeschlagen, damit die Pfadplanung eines Agenten auf diesem mit Informationen über Kollisionsrisiken angereicherten Graphen geschehen kann.

Beide vorgeschlagenen Ansätze können dahingehend weiterentwickelt werden, indem man sie unter realistischeren physikalischen Bedingungen erprobt. Die Agenten in den hier gezeigten Simulationen verfügten über keine Masse oder Trägheit und unterlagen keinen Kurvenradien, die eingehalten werden müssen. Diese physikalischen Beschränkungen ließen sich sehr einfach in den baumartigen Planungsprozess des ersten Ansatzes integrieren. Beim zweiten Ansatz müssen entweder die auf dem Graph gefundenen Pfade nachbearbeitet werden oder die physikalischen Beschränkungen der Agenten, die sich auf dem Graph bewegen, bereits bei dessen Erstellung einbezogen werden.

Die in diesem Kapitel trainierten Bewegungsmodelle auf Basis von künstlichen neuronalen Netzen dienen rein zur Demonstration der Machbarkeit und lassen sich ohne Zweifel im Hinblick auf Genauigkeit und unterstütze Bewegungsmuster verbessern. Ein Beispiel ist die Architektur der Bewegungsvorhersage. In diesem Kapitel wird die Bewegung jedes Objekts einzeln durch ein gemeinsames Modell vorhergesagt. Dies geschieht entweder in aufeinander aufbauenden Vorhersagen wie beim ersten Ansatz oder in mit einem eigenen

Modell für jeden Zeitschritt wie im zweiten Ansatz. Es sind andere Architekturen für die künstlichen neuronalen Netze zur Vorhersage der zukünftigen Positionen denkbar. Es könnte ein LSTM-Netz darauf trainiert werden sowohl die Koordinaten der zukünftigen Objektposition sowie den exakten Zeitpunkt zu dem sich das Objekt an den Koordinaten befinden wird vorherzusagen. Zudem könnte das Modell alle beweglichen Hindernisse und deren Interaktion vorhersagen. Die Einführung unterschiedlicher Geschwindigkeiten, komplexerer LSTM-Modelle und feinerer Belegungsgitter stellen weitere Möglichkeiten zur Erweiterung dar. Die Erstellung von Bewegungsmodellen stellt einen eigenen Forschungszweig [93, 2, 146, 70] dar und wird hier nicht weiter verfolgt, da es voranging um die Integration der Vorhersagen in den Planungsprozess eines Agenten ging.

Um den Regressionsansatz im PCMP so einfach wie möglich zu halten, wurden Unsicherheiten bei den Vorhersagen nicht berücksichtigt, wie dies z.B. bei der Voruntersuchung zur intuitiver Physik in Abschnitt 3.3.3 zu sehen ist. Das Ausweichverhalten lässt sich sehr viel differenzierter steuern, wenn Kanten mit Kollisionswahrscheinlichkeiten behaftet sind. Der Umgang mit Unsicherheit stellt eine zentrale Forschungsrichtung in der Literatur zu KNN dar [69]. Einige der vorgestellten Methoden sind geeignet, Regressionsergebnisse in wahrscheinlichkeitsbasierte Aussagen umzuwandeln [118]. Dadurch könnte die Unsicherheit der Bewegungsprognosen im Planungsprozess explizit berücksichtigt werden. Eine solche Erweiterung würde zum einen die Steuerungsmöglichkeiten des Regressionsmodells erhöhen und zum anderen die Abhängigkeit von der Qualität des LSTM-Modells verringern.

Es wurde davon ausgegangen, dass sich der Agent in jedem Zeitschritt bewegt. In vielen Situationen ist jedoch das Warten eine interessante Alternative, um Kollisionen zu vermeiden. Der Suchalgorithmus müsste um die zusätzliche Warte-Aktion als Handlungsalternative erweitert werden.

Die Erweiterung auf multiple Agenten ist ein weiterer Ansatzpunkt für weitere Arbeiten. Darauf wird im folgenden Kapitel eingegangen, sowie auf eine Methode mit der eine sich dynamisch verändernde Freifläche zu einem Graphen diskretisiert werden kann.

4 Bewegung multipler Agenten

In diesem Kapitel wird ein ganzheitlicher Ansatz entwickelt, welcher aus der dynamischen Diskretisierung einer Freifläche zu einem Graphen und der Pfadplanung auf diesem besteht. Der resultierende Graph kann parallel von multiplen Agenten genutzt werden, um Ziele auf der Freifläche anzusteuern. Deshalb wird darüber hinaus ausgearbeitet, wie sich eine bestehende Kollisionsvermeidungsstrategie in den vorgeschlagenen Ansatz integrieren lässt. Im Abschnitt 4.1 wird zunächst auf die Vorveröffentlichung eingegangen, auf welcher dieses Kapitel basiert, und im Abschnitt 4.2 wird die Thematik weiter motiviert.

Der Fokus liegt auf der Diskretisierung einer sich dynamisch veränderten Fläche, und dementsprechend wurde eine Technik gewählt, welche mit solchen Veränderungen über die Zeit umgehen kann, namentlich Stable Growing Neural Gas. Diese wird unter dem Punkt 4.3, Verwandte Arbeiten, eingeführt. Im Abschnitt 4.4 wird der Ansatz insgesamt erläutert und welche Vorteile sich aus der Diskretisierung mittels Stable Growing Neural Gas und dem Zusammenspiel mit klassischen Pfadplanungsalgorithmen wie dem Dijkstra oder A* Algorithmus ergeben. Zudem wird auf die Kollisionsvermeidung mittels eines Potential Field Ansatzes eingegangen. Das Kapitel schließt mit einer Evaluation in Abschnitt 4.5 und wird in 4.6 noch einmal zusammengefasst.

Die Erfindungshöhe des Ansatzes besteht vor allem in der geschickten Auswahl geeigneter Methoden zur Lösung der Teilprobleme der Diskretisierung, der Pfadplanung und der Kollisionsvermeidung, aus deren Zusammenspiel sich Synergien ergeben. Es wird in diesem Kapitel ein Gesamtsystem entwickelt, das im Hinblick auf die Anpassungsfähigkeit an Veränderungen der begehbaren Fläche oder die Unterstützung multipler Agenten verwandten Arbeiten überlegen ist.

4.1 Vorveröffentlichungen

Die Kerninhalte dieses Kapitels wurden bereits in [56] veröffentlicht. Die Publikation basiert, wie im Abschnitt 1.3 beschrieben, auf einer Bachelorarbeit, wobei die Problemstellung vom Autor dieser Arbeit ausgeschrieben wurde und mögliche Methoden zur Lösung von Teilproblemen zwischen dem bearbeitenden Studenten der Bachelorarbeit, Manuel Zierl, und dem Autor diskutiert wurden. Unterstützt wurde dies durch Sebastian Feld als Zweitbetreuer. Der Hauptanteil an der Publikation liegt beim Autor dieser Arbeit, insbesondere im Hinblick auf den experimentellen Vergleich mit verwandten Arbeiten auf diesem Gebiet. Die Inhalte und Ergebnisse der Publikationen sind größtenteils

in dieses Kapitel eingeflossen und bilden dessen Kerninhalt. Die damit verbundenen Abbildungen 4.2, 4.3, 4.4, 4.6, 4.7, 4.8a, 4.8b, 4.9, 4.10, 4.11, 4.12 und 4.13 sind bereits in [56] enthalten.

4.2 Motivation

Die autonome Pfadplanung ist ein wichtiges Thema in der Erforschung künstlicher Intelligenz. Aus evolutionsbiologischer Sicht besteht die Vermutung, dass das Gehirn bei Lebewesen u.a. aus der Notwendigkeit entstanden ist, sich zu bewegen [86]. Es ist daher logisch, dass eine Intention der künstlichen Intelligenz darin besteht, zielgerichtete und sinnvolle Bewegungen automatisch, d.h. computergesteuert, durchzuführen. Eine sinnvolle Bewegung ist nur möglich, wenn sie vorher geplant wurde, was zum Problem der Pfadplanung führt.

In diesem Kapitel wird eine Problemdomäne betrachtet, bei der sich multiple autonome Agenten frei in einem zweidimensionalen Grundriss bewegen können. Das Ziel der Agenten ist es, zu vorgegebenen Zielen zu navigieren. Es wird angenommen, dass ein Agent sich entweder innerhalb des Grundrisses positionieren kann oder er zumindest seine Startposition kennt. Es wird zudem zugelassen, dass sich der begehbare Bereich dynamisch verändern kann. Diese Änderungen können durch neu zur Verfügung gestellte Bereiche oder durch Blockaden entstehen, die den zuvor begehbaren Bereich abtrennen. Ein Agent soll sich an diese Änderungen anpassen können. Schließlich soll ein Agent weder mit statischen Hindernissen noch mit dynamischen Hindernissen, wie anderen Agenten, kollidieren.

Im Folgenden wird die Problemdomäne in drei Teilprobleme aufgeteilt: a) Diskretisierung der kontinuierlichen Domäne in eine Graphstruktur, um die Berechnung zu erleichtern b) Pfadplanung auf der zuvor erzeugten Graphstruktur und c) Einsatz einer geeigneten Kollisionsvermeidungsstrategie.

Die in diesem Kapitel vorgestellte Lösung *Neural Gas Dynamic Path Planning* (NGDPP) kombiniert bestehende Methoden für die einzelnen Teilaufgaben, um die beschriebene Problemdomäne anzugehen. Dazu werden die Vorteile aufgezeigt, die sich aus der Kombination der gewählten Methoden ergeben. Darüber hinaus werden Verbesserungen für die gewählte Methode zur Diskretisierung des kontinuierlichen Bereichs, namentlich Stable Growing Neural Gas (SGNG), vorgeschlagen.

4.3 Verwandte Arbeiten

Es gibt mehrere bestehende Ansätze, die zumeist nur eines der genannten Teilprobleme behandeln. Im Kapitel 2 wurde bereits auf Ansätze zur Kollisionsvermeidung eingegangen. Beim sogenannten Potential Field [68] bzw. dem darauf aufbauenden Virtual Force Field [74] wird in einem gewissen Radius um die Agenten herum ein Kraftfeld simuliert, so dass sie wie gleich geladene

Teilchen voneinander oder von Hindernissen abgestoßen werden. Zudem wurde in Kapitel 2 bereits auf einfache und fortgeschrittenere Ansätze zur Diskretisierung von Freiflächen eingegangen. Diese haben aber bei dem dynamischen Multiagentenszenario, welches in diesem Kapitel betrachtet werden soll, gewisse Nachteile. So hat RRT [82] den Nachteil, dass für jeden Agenten und jedes anzusteuernde Ziel ein neuer Baum konstruiert werden muss, der zum Ziel führt. PRM [66] wiederum hat den Nachteil, dass es sich nicht an dynamisch verändernde Umgebung anpassen kann, sondern die Diskretisierung bei jeder Änderung komplett neu vorgenommen werden muss. Deshalb baut dieses Kapitel auf einen Ansatz namens Stable Growing Neural Gas auf, welcher sich kontinuierlich an Veränderungen im Raum anpasst und auf dessen resultierenden Graph multiple Agenten geroutet werden können. Dafür sollen im Folgenden die vorhandenen Vorarbeiten anderer Autoren vorgestellt werden.

4.3.1 Self Organizing Feature Map

Neural Gas basiert auf Self-Organizing (Feature) Maps (SOM) [72]. Diese sind eine spezielle Art künstlicher neuronaler Netze und fallen in die Kategorie der Unsupervised Learning Methoden. Sie sind eine Methode der Dimensionsreduktion, da sie dazu verwendet werden können höherdimensionale Eingabedaten auf eine niedriger dimensionierte (in der Regel ein- oder zweidimensional), diskrete Repräsentation abzubilden (genannt Map). Dementsprechend können SOM zur Visualisierung und Strukturierung höherdimensionaler Daten und damit dem Data-Mining dienen. SOM wurden 1982 von Teuvo Kohonen vorgeschlagen und beruhen auf der Beobachtung, dass mit einem einfachen Netzwerk adaptiver physikalischer Elemente (im Sinne von künstlichen Neuronen), Signale aus einem primären Ereignisraum automatisch auf einen Satz von Ausgangsreaktionen abgebildet werden können, so dass die Reaktionen dieselbe topologische Ordnung erhalten wie die der primären Ereignisse. Das soll bedeuten, es wurde ein Prinzip entdeckt, das die automatische Bildung topologisch korrekter Karten von Merkmalen beobachtbarer Ereignisse erleichtert [72]. Diese Art von Netzwerkstruktur wird als verwandt mit den Feedforward-Netzwerken betrachtet, da sie als Graph mit durch Kanten verbundene Knoten visualisiert werden. Allerdings unterscheiden sich die Netzwerke grundlegend in Anordnung und Motivation der zu lösenden Aufgabe. Bereits SOM eignen sich zur Diskretisierung von Freiflächen, in dem die freien Bereiche den primären Signal- bzw. Ereignisraum darstellen und sich die Knoten bzw. Neuronen in der SOM Freifläche ausrichten. Der die Freifläche abdeckende Graph aus Knoten und Kanten kann dann zur Pfadplanung im Raum verwendet werden. Allerdings sind SOM in ihrer ursprünglichen Form relativ unflexibel, so dass die Knoten in einem rechteckigem oder hexagonalem Gitter angeordnet sind (vorstellbar wie ein Fischernetz, dass sich in der Freifläche aufspannt). Dabei sind aber die Anzahl der Knoten und Kanten vorbestimmt und es kommt leicht zu Abbildungsfehlern bzw. Anomalien, so dass sich z.B. das gedachte

Fischernetz in sich verdreht. SOM können in ihrer ursprünglichen Form auch nicht mit mehreren Freiflächen umgehen, die keine Verbindung untereinander haben, so dass sich im resultierenden Graph fälschlicherweise Kanten zwischen den eigentlich getrennten Räumen befinden. Deshalb wird im Folgenden auf Weiterentwicklungen der SOM eingegangen.

4.3.2 Neural Gas

Die Autoren des Neural Gas Ansatzes Martinetz und Schulten [95] bezeichnen ihn als Vektorquantisierungstechnik unter anderem zur Datenkompression. Eine der ältesten Methoden dieser Art ist das Voronoi Diagramm [139]. Bei diesem wird ein Raum $A \in \mathbb{R}^n$ anhand vordefinierter Referenzvektoren $w_i \in \mathbb{R}^n, i = 1, \cdots, N$ so aufgeteilt, dass ein jeder Punkt $v \in A$ durch den am besten passenden Referenzvektor w_j repräsentiert wird. Um diesen zu bestimmen kommt in der Regel die euklidische Distanz zum Einsatz $\|v - w_j\|_2$, welche minimiert werden soll. Dies führt dazu, dass der Raum A in eine Anzahl von Unterräumen

$$A_j = \{v \in A \mid \|v - w_j\|_2 \leq \|v - w_i\|_2 \; \forall i\},$$

die so genannten Voronoi Regionen. Diese Regionen bzw. deren durch Kanten verbundene Mittelpunkte (die ursprünglichen Referenzvektoren w_i) könne bereits als Diskretisierung des Raumes zur Pfadplanung verwendet werden. Allerdings sind die Referenzvektoren entweder händisch oder zufällig gewählt, wodurch keine gute bzw. gleichmäßige Abdeckung des Raums gewährleistet ist. Zudem ist die Herangehensweise nicht für dynamische Änderungen des Raumes ausgelegt, wodurch die Aufteilung des Raumes bei jeder Änderung komplett neu berechnet werden muss. Eine Möglichkeit, um die Referenzvektoren automatisiert zu wählen stellen bereits Self-Organising Feature Maps dar, allerdings mit den zuvor genannten Nachteilen. Deshalb schlagen Martinez und Schulten den darauf aufbauenden Neural Gas Algorithmus [95] vor, bei dem es sich dementsprechend ebenfalls um einen Unsupervised Learning Ansatz handelt. Die Bezeichnung „Gas“ bezieht sich darauf, dass sich bei der Visualisierung des Algorithmus, das heißt, bei der Anordnung der Knoten im Raum, diese gleichmäßig in diesem verteilen, analog einem Gas auf molekularer Ebene. Die ausgebildeten Kanten zwischen den Knoten müssen dabei nicht zwingend ein reguläres Raster aus Quadraten oder Hexagonen bilden, sondern können beliebige Graph-Strukturen annehmen. Im Vergleich zur SOM kann dieser zudem disjunkt sein, das heißt, es kann mehrere nicht miteinander verbundene Freiflächen geben.

4.3.3 Growing Neural Gas

Ein Nachteil am Neural Gas Algorithmus ist, dass eine vordefinierte Anzahl an Iterationen durchlaufen bzw. eine vordefinierte Menge an Knoten ausgebildet wird. Dadurch wird wie bei SOM eine gewisse Menge an Vorwissen über die

Beschaffenheit des zu diskretisierenden Raumes benötigt, um die Parameter entsprechend einstellen zu können. Eine relativ komplexe Freifläche, das heißt, mit vielen Einbuchtungen oder Hindernissen, kann mehr Knoten und Kanten für eine hinreichend gute Repräsentation benötigen, als das bei einfach aufgebauten Freiflächen der Fall ist. In statischen Räumen können die Parameter des Neural Gas entsprechend angepasst werden, dass sich eine zufriedenstellende Graph-Repräsentation des Raumes ergibt. Wenn aber dynamische Veränderungen auftreten, können die Parameter nur statisch auf eine Konfiguration eingestellt werden bzw. muss die Diskretisierung bei jeder Veränderung des Raumes erneut komplett durchgeführt werden. Aus diesem Grund erweiterte Bernd Fritzke den Neural Gas Algorithmus weiter zum Growing Neural Gas (GNG) [39]. Der Algorithmus ist so aufgebaut, dass kontinuierlich neue Knoten ausgebildet werden. Es kann aber auch dazu kommen, dass Kanten entfernt werden, die ein gewisses Alter überschritten haben, und mit ihnen Knoten, wenn diese über keine weitere Kante verfügen. Das Alter einer Kante wird dann zurückgesetzt, wenn ein mit ihr verbundener Knoten nächstgelegen zu einem zufällig erzeugten Signal aus dem ursprünglichen Ereignisraum (d.h. aus der zu diskretisierenden, frei begehbaren Fläche) liegt. Dadurch, dass kontinuierlich Knoten ausgebildet werden, wird die diskretisierte Abbildung, das heißt, der Graph, immer präziser je länger der Algorithmus in einem unveränderten statischen Raum ausgeführt wird. Zudem erhält der Algorithmus dadurch die Fähigkeit, mit sich verändernden Räumen umgehen zu können. Kommen neue begehbarer Bereich hinzu, kann der GNG direkt neue Knoten ausbilden. Fallen Bereiche weg, werden in diesen keine Signale aus dem sogenannten Ereignisraum mehr erzeugt, womit zunächst Kanten entfernt werden, die ihr Maximalalter überschreiten und im Verlauf mit ihnen Knoten, die über keine Kanten mehr verfügen. Um zu verhindern, dass der Algorithmus eine unbegrenzte Anzahl an Knoten produziert, was z.B. die Pfadplanung auf dem resultierenden Graphen sehr ineffizient macht, wurde unter anderem eine maximale Anzahl an Knoten als Abbruchkriterium für den Algorithmus vorgeschlagen. Es existieren Arbeiten, die explizit den GNG Algorithmus für die Diskretisierung von Karten mit frei begehbaren Flächen verwenden. Dellinger et al. verwenden den Algorithmus, um in statischen, virtuellen Spielkarten Wegpunktgraphen zu erzeugen, auf welchen sich simulierte Agenten mittels Pfadplanungsalgorithmen bewegen [24].

4.3.4 Stable Growing Neural Gas

Das Einführen einer Deckelung der maximalen Anzahl an Knoten in den GNG Algorithmus ermöglicht es grundsätzlich, den Algorithmus für eine unbekannte Zeit auszuführen und so auch veränderliche Räume fortlaufend zu diskretisieren. Allerdings kann es bei einem sich immerfort verändernden Raum schwierig sein eine maximale Anzahl an Knoten zu bestimmen, die eine hinreichend gute Diskretisierung für das Routing garantiert. Damit steht der Algorithmus aber

vor einem ähnlichen Problem, wie es schon bei den SOM mit der festen Anzahl an Iterationen oder beim NG mit der maximalen Anzahl an Knoten der Fall war. Deshalb erweiterten Tencé et al. den Algorithmus wiederum weiter zum sogenannten Stable Growing Neural Gas (SGNG), um dieses Defizit zu beseitigen [136]. Um das Problem zu beseitigen wurde eine globale Variable $\text{error}_{\text{max}}$ eingeführt, die den Fehler angibt, den ein Knoten mindestens besitzen muss, damit ein neuer Knoten generiert wird. Den Fehler eines Knotens gab es bereits im Growing Neural Gas Algorithmus und er hängt im Groben damit zusammen, wie weit ein Knoten von den fortlaufend erzeugten Signalen aus dem ursprünglichen Ereignisraum entfernt ist. Auch dort wurde er bereits zur Entscheidung herangezogen, ob neue Knoten gebildet werden sollen. Beim SGNG Algorithmus kommt nun noch das Kriterium des $\text{error}_{\text{max}}$ hinzu, was in die Entscheidung einfließt, ob ein neuer Knoten gebildet wird. Damit ist es möglich die Präzision der Diskretisierung, das heißt, des Graphen, einzustellen ohne dabei vorab eine Aussage über die Anzahl der zu erzeugenden Knoten oder der durchlaufenen Iterationen treffen zu müssen. Dies beschreibt auch den Begriff „Stable“ im Namen des Algorithmus. Er kann fortwährend ausgeführt werden, um auch veränderliche Räume zu diskretisieren und hält dabei eine vorab eingestellte Dichte an Knoten und Kanten, ohne unerwünscht viele oder zu wenig Knoten und Kanten zu erzeugen. Dementsprechend kann eine „Präzision“ der Diskretisierung voreingestellt werden.

Die Abbildung 4.1 zeigt die beispielhafte Diskretisierung einer Freifläche mittels SGNG. Zu erkennen ist der resultierende Graph, bei welchem sich die Knoten gleichmäßig auf der begehbaren Fläche verteilt haben.

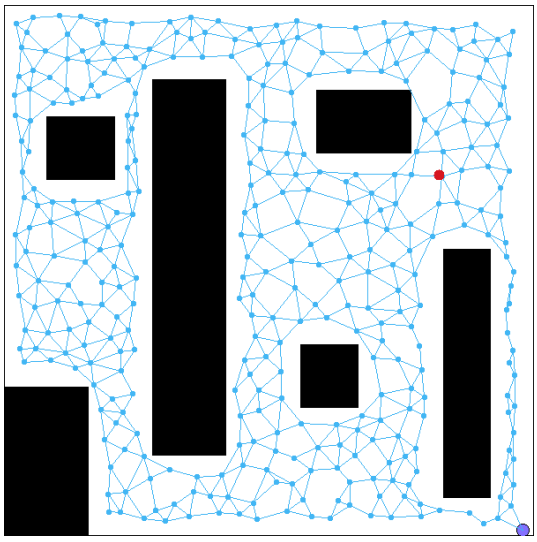


Abbildung 4.1: Beispielhafte Diskretisierung einer Freifläche mittels SGNG und einem Agenten (roter Punkt), welcher auf dem resultierenden Graphen zu einem Ziel (lila Kreis) geroutet wird.

4.4 Dynamische Pfadplanung mittels neuronalem Gas

Der in diesem Kapitel vorgestellte Algorithmus – Neural Gas Dynamic Path Planning (NGDPP) – versucht, das Problem der Pfadplanung auf sehr allgemeine Weise zu lösen, indem er auf dynamische Räume angewendet wird, und zwar sowohl in Form von räumlichen Veränderungen als auch von dynamischen Hindernissen. Mehrere Agenten sind mit ihm in der Lage, sich jedem Ziel innerhalb eines kontinuierlichen Raums zu nähern, ohne mit Wänden, Kollisionsobjekten oder untereinander zu kollidieren.

NGDPP verwendet einen sogenannten Signalraum, der die frei begehbare Fläche repräsentiert. Aus diesem Signalraum werden Stichproben an Koordinaten gezogen, wie sie beim Growing Neural Gas Algorithmus Anwendung finden und für einen begehbaren Punkt im Raum sprechen. Nicht begehbare Flächen des Raumes erzeugen keine Signale. Die Fläche kann in Form einer Matrix bzw. Bitmap dargestellt werden, bei der einzelne Pixel entweder frei oder belegt sind. Um die kontinuierliche Eigenschaft des Raumes beizubehalten kann die Bitmap so groß oder so fein wie nötig sein. Die Darstellung des Signalraumes ist allerdings nicht an eine Bitmap gebunden, so lange sich aus ihr Signale erzeugen lassen, die der spätere Algorithmus als Teil der Freifläche interpretiert.

Grundsätzlich ist der Algorithmus in drei parallel laufende Teile unterteilt. Diese sind Diskretisierung, Pfadplanung und Kollisionsvermeidung.

4.4.1 Diskretisierung durch Stable Growing Neural Gas

Die Diskretisierung erzeugt einen Wegpunktgraph aus dem Signalraum. Der Graph besteht aus Knoten und Kanten, wobei die Knoten die Positionen im Raum und die Kanten die kollisionsfreien Übergänge zwischen ihnen darstellen. Da sporadische Änderungen im Raum auftreten können, ist es notwendig, eine Methode zur Diskretisierung zu wählen, die auch dynamisch eine Darstellung dieses Raumes mitentwickeln kann. Für diese dynamische Diskretisierung wird der Algorithmus des Stable Growing Neural Gas (SGNG) [136] verwendet, bei dem es sich um eine iterative unsupervised Lerntechnik handelt, welche die zugrundeliegende Topologie eines n-dimensionalen Raums in quantisierter Form lernt.

Das Hauptmerkmal des hier verwendeten SGNG ist, dass es sich mit dem sich verändernden Raum kontinuierlich weiterentwickelt. Darüber hinaus kann der Algorithmus kontinuierlich ausgeführt werden, während er in Bezug auf die Anzahl der Knoten zu einem stabilen Zustand konvergiert.

Sporadische Änderungen des Raumes werden im Algorithmus als Modifikationen in der Raummatrix, die den Signalraum bildet, verstanden, und es wird davon ausgegangen, dass diese Änderungen einen relativ großen Teil der Fläche betreffen. Darüber hinaus sollten die Änderungen nicht zu dynamisch

auftreten, da das neuronale Gas aufgrund seiner Latenzzeit zur Behebung fehlerhafter Kanten oder Knoten eine gewisse Zeit benötigt (vergleichbar zu Abbildung 4.2a). Daher müssen kleine oder schnelle dynamische Änderungen mit Hilfe der Kollisionsvermeidung auf andere Weise behandelt werden, wie später erläutert wird.

Algorithmus 3 NGDPP

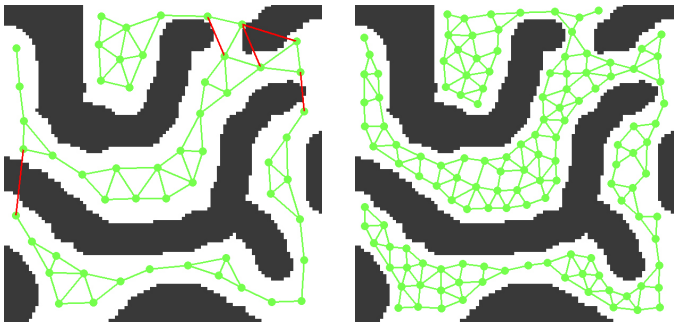
Require: *graph* ← GRAPH(*nodes* ← [], *edges* ← [])
area ← LOADINITIALAREA()
agents ← LOADAGENTS()
collisionObjs ← LOADCOLLISIONOBJS()

1: *nodes* ← $[n_1, n_2]$
2: $i \leftarrow 0$
3: **while** *True* **do**
4: $i \leftarrow i + 1$
5: *# Zufällige Position s im Raum auswählen*
6: $s \leftarrow$ GETSIGNAL(*area*)
7: *# Nächste zwei Knoten zu s*
8: $n_1, n_2 \leftarrow$ NEARESTNODES(s, *graph*)
9: $n_1.error \leftarrow n_1.error +$ DISTANCE(s, n_1)^2
10: *# n_1 und dessen Nachbarn in Richtung s bewegen*
11: ATTRACTNODES(s, n_1)
12: *# Neue Kante zwischen n_1 und n_2 falls nicht vorhanden*
13: NEWEDGECONNECTION(n_1, n_2)
14: *# Kanten gemäß Abschnitt 4.4.1.2 entfernen*
15: AGESELECTION(i)
16: *# Heuristische Erzeugung neuer Knoten*
17: SPLITNODES()
18: DECREASEALLERRORS()
19: **for each** *agent* **in** *agents* **do**
20: *agent*.STEP(*area, collisionObjs, graph*)
21: **end for**
22: **end while**

Der SGNG-Algorithmus wird kontinuierlich ausgeführt, um eine topologische Darstellung aufzubauen, die sich dynamisch mit dem Raum entwickeln kann. Wenn ein bestimmter Bereich des Raumes unzugänglich wird, werden dort keine Signale mehr erzeugt. Dies wiederum bedeutet, dass die Knoten an dieser Stelle vom Algorithmus nicht mehr selektiert werden und somit das „Alter“ der verbundenen Kanten zunimmt, bis sie schließlich entfernt werden. Wenn alle Kanten eines Knotens entfernt wurden, wird auch der Knoten entfernt. Wird dem Raum ein neuer Bereich hinzugefügt, werden auch dort Signale erzeugt und Knoten in ihre Richtung gezogen. Aufgrund der in der Regel größeren Abstände zwischen den bestehenden Knoten und dem Signal nimmt die

„Fehler“ Eigenschaft des Knotens zu, so dass sich mit größerer Wahrscheinlichkeit neue Knoten bilden, die den neu geschaffenen Raum ausfüllen. Der High-Level-Pseudocode von NGDPP wird in Algorithmus 3 gezeigt. Der Agent verwendet den erstellten Wegpunkt-Graphen in seiner STEP()-Methode, auf welche im Abschnitt 4.4.3 näher eingegangen wird. Die Variable i zählt die Durchläufe der Hauptschleife und dient

Die Diskretisierung der Umgebung mit SGNG ist möglicherweise nicht optimal. Dieser Fehler hängt auch stark von einer „maxError“-Variablen ab, die die Erzeugung neuer Knoten steuert und mit der die Genauigkeit des SGNG kontrolliert werden kann. Ein Vergleich von zwei verschiedenen Einstellungen dieser „maxError“ Variablen ist in Abbildung 4.2 zu sehen.



(a) • (b) •

Abbildung 4.2: Im Vergleich der SGNG mit niedriger (a) und hoher Präzision (b) fällt auf, dass im ersten Fall falsche (nicht kollisionsfreie) Kanten gebildet werden können (rot eingefärbt).

Um es den Agenten zu ermöglichen, zu jedem beliebigen Ziel im Raum zu navigieren, wird vorgeschlagen, die Ziele als Knoten in den Graphen zu integrieren. Zielknoten müssen nicht zu Beginn definiert werden, sondern können zur Laufzeit eingefügt werden. Dies ist besonders nützlich, wenn zu Beginn nicht alle Ziele bekannt sind. Im Gegensatz zu einem normalen Knoten hat ein Zielknoten die folgenden drei Eigenschaften:

1. Direkt nachdem ein Zielknoten erstellt wurde muss dieser mit dem Knoten verbunden werden, der die kürzeste Entfernung zu ihm hat. Andernfalls würde er sofort gelöscht.

2. Wenn der Zielknoten nur eine Kante hat und diese Kante das „maxAge“ überschreitet, muss seine Variable „Alter“ um mindestens eins reduziert werden, um die Kante zu erhalten.

3. Ziele müssen statisch sein. Wenn der Algorithmus versucht, einen Zielknoten zu verschieben, weil er der nächste Knoten zu einem Signal (oder der topologische Nachbar eines solchen Signals) ist, ist dies nicht zulässig und wird daher nicht ausgeführt.

Aus diesen Eigenschaften folgt, dass es für den SGNG-Algorithmus nicht möglich ist, einen Zielknoten zu entfernen. Sie müssen anderweitig entfernt werden, z.B. wenn ein Agent sein Ziel erreicht hat.

4.4.1.1 Problem des nächsten Nachbarn

Ein großer Teil des Berechnungsaufwandes des SGNG wird durch die Notwendigkeit verursacht, in jeder Iteration die Knoten mit der kürzesten Entfernung zu einer bestimmten Position zu finden. Diese Suche könnte einfach dadurch erfolgen, dass man den Abstand zwischen jedem Knoten im Graphen und der Position berechnet und dann die Knoten entsprechend sortiert. Dies ist jedoch recht ineffizient, insbesondere bei einer hohen Anzahl von Knoten. Deshalb soll diese Suche optimiert werden. Aufgrund des hochdynamischen Charakters von SGNG, bei dem sich die Knoten bei jeder Iteration im Raum bewegen, kommen fortgeschrittene raumaufteilende Datenstrukturen wie k-d Bäume [11] nicht in Betracht, da diese Datenstrukturen in jeder Iteration aufwendig neu aufgebaut oder aktualisiert werden müssten.

Es wird deshalb ein einfacherer Ansatz verwendet, welcher den Raum in welchem der Graphen aufgespannt wird, in Gitterzellen unterteilt. Jeder Knoten wird in einem zweidimensionalen Array an der Position gespeichert, an der er sich innerhalb dieses Gitters befindet. Ein Knoten, der sich an der Position (x, y) befindet, wird dann in dem Array an der Position $g_{i,j}$ mit $(i, j) = (\lfloor (x/a) \rfloor, \lfloor (y/a) \rfloor)$ gespeichert, wobei a die Genauigkeit des Gitters angibt und $\lfloor \ldots \rfloor$ das Ergebnis auf ganze Zahlen abrundet. Die Genauigkeit muss zu Beginn definiert werden und darf während der Laufzeit des Algorithmus nicht verändert werden. Werden alle Knoten nach diesem Schema abgespeichert, kann es genutzt werden, um den nächsten Knoten im Graphen effizienter zu finden. Immer wenn ein Knoten verschoben wird, muss auch seine Position innerhalb des Arrays überprüft und eventuell aktualisiert werden. Dies ist kein Problem, da die Besprechungsvorschrift für $g_{i,j}$ sehr schnell ausgewertet werden kann.

4.4.1.2 Sortierte Liste für Kanten

Ein weiterer Faktor, der den Rechenaufwand des Algorithmus erhöht, ist die Berechnung des Kantenalters. Im Prinzip erfordert der SGNG-Algorithmus, dass jede Kante eine „Alter“ Eigenschaft erhält, die bei jeder Iteration um eins erhöht wird. Zusätzlich muss für jede Kante geprüft werden, ob sie ein „maxAge“ überschritten hat. Deshalb wurden in [56] die folgenden Änderungen vorgeschlagen:

Jede Kante erhält einen „Geburt“-Zeitpunkt anstelle eines „Alters“. Diese gibt den Iterationsschritt des Algorithmus an, in dem die Kante erzeugt wurde. Die Kanten werden nun in einer Liste gespeichert. Die älteste Kante, deren „Geburt“ die niedrigste ist, steht an erster Stelle. Sobald eine neue Kante erstellt wird, hat sie ein Alter von Null („Geburt“ = aktueller Iterationsschritt)

und wird einfach am Ende der Liste hinzugefügt. Die resultierende Liste ist natürlich sortiert, da der Iterationszähler streng monoton ansteigend ist. Für die Löschung veralteter Kanten muss der Algorithmus nicht mehr in jeder Iteration alle Kanten prüfen, sondern kann einfach die erste Kante in der Liste inspizieren. Die Subtraktion ihrer „Geburt“ vom aktuellen Iterationsschritt ergibt ihr Alter. Wenn dieser Wert das „maxAge“ überschreitet, wird sie aus der Liste und aus dem Graphen gelöscht. Dann muss auch die folgende Kante dahingehend überprüft werden, ob es sich um das gleiche Alter handelt. Wenn sie das Höchstalter nicht überschritten hat, kann berechnet werden, wann sie dieses zum frühesten Zeitpunkt überschreitet, indem der „Geburtswert“ von der aktuellen Iterationszahl subtrahiert wird. Durch die Speicherung dieses Wertes kann ermittelt werden, für wie viele Iterationen es nicht notwendig ist überhaupt zu prüfen, ob eine Kante zu alt ist. Der hier vorgestellte Algorithmus erspart sich also nicht nur die Iteration über alle Kanten, sondern muss auch die Kanten für mehrere Iterationen der Hauptschleife überhaupt nicht mehr überprüfen.

Es kommt in SGNG auch vor, dass das Alter der Kanten auf null zurückgesetzt werden soll, wenn ein mit ihr verbundener Knoten durch ein Signal in seiner Nähe aktiviert wird. In diesem Fall wird die Kante aus der Liste entfernt, egal an welcher Position sie sich befindet, ihre „Geburt“ auf die aktuelle Iterationszahl gesetzt und sie am Ende der Liste hinzugefügt.

4.4.2 Pfadplanung mittels A* Algorithmus

Parallel zum Wegpunktgraph muss ein Pfad für den bzw. die Agenten berechnet werden. Aufgrund der dynamischen Eigenschaft des Graphen sind geplante Pfade nicht unbedingt über die Zeit gültig. Dieses Problem kann durch eine kontinuierliche Neuberechnung des Pfades eines Agenten gelöst werden. Bei mehreren Agenten muss jeder Agent seinen eigenen Pfad parallel mit Hilfe des global verfügbaren Wegpunktgraphen berechnen.

Für die Pfadplanung kommt der A*-Algorithmus [62] zum Einsatz, da der gebildete Wegpunktgraph im euklidischen Raum liegt und somit heuristische Informationen leicht extrahiert werden können, was einen Leistungsvorteil gegenüber dem Dijkstra-Algorithmus [28] bietet. Für den A*-Algorithmus ist es notwendig, die Kosten der Kanten zu kennen. Im vorliegenden Fall werden diese durch den Abstand zwischen den Knoten im Raum beschrieben. Die Kosten einer Kante zwischen zwei Knoten kann mit Hilfe des euklidischen Abstands berechnet werden. Da diese Berechnung auf alle Kanten angewendet werden muss, die bei der Pfadplanung untersucht werden, macht sie einen großen Teil des Berechnungsaufwands der Pfadplanung aus.

Der den Graphen aufspannende SGNG-Algorithmus hat eine entscheidende Eigenschaft, die sich der NGDPP zunutze machen kann, um sich diese Berechnungen zu sparen. Der SGNG-Algorithmus ist so konzipiert, dass die Knoten in einen Zustand konvergieren, in dem sie gleichmäßig im Raum verteilt sind,

ähnlich wie ein Gas auf molekularer Ebene. Dies zieht nach sich, dass sich die zwischen den Knoten gebildeten Kanten auf ziemlich genau die gleiche Länge ausgleichen. Diese Eigenschaft ergibt sich aus der Tatsache, dass die erzeugten Signale gleichmäßig im Raum verteilt sind und die Knoten entsprechend erzeugt oder verschoben werden.

Die Tatsache, dass alle Kantenlängen nahezu gleich lang sind (vgl. Abbildung 4.5), macht die Berechnung ihrer exakten Länge für die Pfadplanung überflüssig. Es genügt, die Kantenkosten gleich eins zu setzen. Der Leistungsvorteil, der sich aus dieser Methodik zusammen mit der Tatsache ergibt, dass keine Verschlechterung des Pfades auftritt, ist einer der Beiträge dieses Kapitels und wird in Abschnitt 4.5 weiter ausgewertet.

4.4.3 Kollisionsvermeidung durch den Potential Field Ansatz

Ein Kollisionsobjekt, d.h. ein Objekt, das sich im Raum bewegt, verändert den ursprünglichen Raum nicht direkt. Natürlich kann der Agent den Bereich, in dem sich ein Kollisionsobjekt befindet, nicht betreten, obwohl dieser Bereich Teil des Raums ist. Im Prinzip könnte das Kollisionsobjekt auch als eine Veränderung des Raums definiert werden. Eine solche Interpretation ist jedoch aufgrund der folgenden Eigenschaften eines Kollisionsobjekts nicht sinnvoll.

Ein Kollisionsobjekt verändert seine Größe nicht, ist meist in Bewegung und in der Regel viel kleiner als die in diesem Kapitel angenommenen sporadischen Veränderungen, die im Raum auftreten können. Da das Neural Gas einige Zeit braucht, um sich an Veränderungen anzupassen, ist es nicht praktikabel, es direkt auf kleine, sich schnell bewegende Objekte anzuwenden. Der wichtigere Grund ist aber, dass der Graph eine global gültige Diskretisierung des Raumes darstellen soll, die von multiplen Agenten parallel genutzt werden kann. Geht man von einem Gebiet aus, in dem sich zwei Agenten unabhängig voneinander bewegen sollen, so ist der andere Agent aus der Sicht eines Agenten nichts anderes als ein Kollisionsobjekt, das sich relativ unvorhersehbar bewegt. Würde man das Neural Gas direkt anwenden, um Kollisionsobjekte zu vermeiden, wäre es notwendig, für jeden einzelnen Agenten einen separaten SGNG-Graphen zu erstellen, da der Graph nicht mehr global gültig wäre. Ein global gültiger Graph hat jedoch den großen Vorteil, dass er von einer zentralen Verarbeitungseinheit berechnet werden kann, während er von mehreren Agenten gleichzeitig genutzt wird.

Daher wird für Kollisionsobjekte die Verwendung des Potential-Field-Ansatzes [73] vorgeschlagen. Um mehrere Objekte im gleichen Raum ohne Kollisionen zu bewegen, wird das Phänomen des Elektromagnetismus der Natur nachempfunden, bei welchem zwei Teilchen mit gleicher Ladung gegenseitig abstoßen. Dieses Konzept wird auf Kollisionsobjekte angewendet. Objekte im Raum werden im Prinzip wie gleich geladene Teilchen behandelt, so dass sie sich im Falle einer drohenden Kollision voneinander abstoßen und so eine Kol-

lision verhindert wird.

Der Potential-Field-Ansatz allein reicht allerdings nicht für die Pfadplanung aus. Er ist jedoch sehr wirksam, um Kollisionen zwischen Objekten im lokalen Raum zu vermeiden. Deshalb wird er beim NGDPP-Algorithmus verwendet. Um das „Potential Field“ anzuwenden und es einem Agenten zu ermöglichen, seinen geplanten Weg lokal zu verlassen, wird der Basisagent, der nur durch seine Position definiert wurde, zu einem „Kraftfeldagenten“ erweitert, der durch die folgenden drei Eigenschaften definiert ist.

1. Die *Position des Kraftfeldes*. Das Kraftfeld selbst ist als Kreis definiert, innerhalb dessen das „Potential Field“ berechnet wird und in dessen Zentrum sich der Agent befindet.

2. Der *Wahrnehmungsradius* ist der Radius des Kraftfeldes, ab welchem Kollisionsobjekte detektiert werden und Auswirkungen auf den Agenten im Zentrum des Kraftfelds haben.

3. Die *Position des Agenten* wird relative zur Position des Kraftfeldes definiert, also zum Mittelpunkt des Wahrnehmungskreises.

Der Agent ist von einem Kraftfeld umgeben, innerhalb dessen Kollisionsobjekte detektiert werden. Der Radius des Kraftfelds sollte deutlich größer gewählt werden als der Radius des Agenten selbst. Infolgedessen hat der Agent keine absolute Position im Raum mehr, sondern erhält relative Koordinaten zum Zentrum seines Kraftfeldes. Anstatt den Agenten also direkt entlang seines Pfades zu bewegen, wird nun das Zentrum des Kraftfelds entlang des Pfades bewegt, wobei sich der Agent normalerweise in der Mitte des Kraftfeldes $(x = 0, y = 0)$ befindet. Steht eine Kollision mit einem anderen Agenten (bzw. einem statischen Hindernis) unmittelbar bevor, wenn das Kollisionsobjekt also in den Kraftfeldradius des Agenten eintritt, wird der Agent von diesem abgestoßen und kann sich innerhalb seines Kraftfeldes bewegen (siehe Abbildung 4.4). Da auf den Agenten ständig eine anziehende Kraft zur Mitte des Kraftfeldes hin ausgeübt wird, kehrt der Agent in seine relative Nullposition zurück, wenn sich kein Kollisionsobjekt mehr innerhalb seines Erfassungsradius befindet. Ein weiterer Einblick in die Bewegung des Agenten wird im Algorithmus 4 gegeben. Der gesamte NGDPP-Prozess ist in Abbildung 4.3 zusammengefasst.

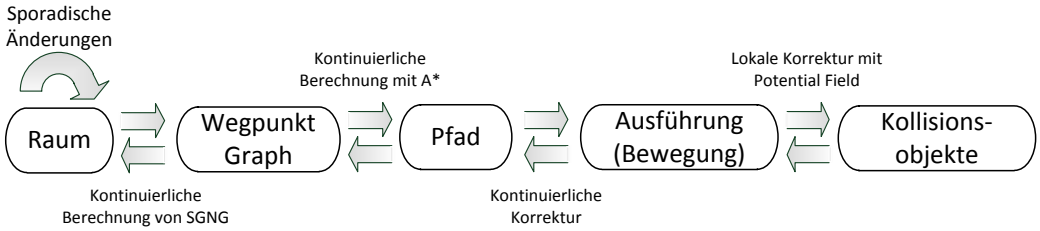


Abbildung 4.3: Struktur des NGDPP

(a) • (b) •

Abbildung 4.4: Ein Agent (rot) weicht einem Kollisionsobjekt (blau) innerhalb seines Kraftfeldes aus. Nachdem das Kollisionsobjekt den Detektionsradius verlassen hat, kehrt der Agent zum Zentrum zurück.

Algorithmus 4 agent.STEP( )

Require: *graph* ← GRAPH(*nodes* ← [], *edges* ← [])
area ← LOADINITIALAREA()
collisionObjs ← LOADCOLLISIONOBJS()
target ∈ *graph.nodes*

1: **function** STEP(*area, collisionObjs, graph*)()
2: | *# Agenten gemäß Abschnitt 4.4.3 bewegen*
3: | MOVEAGENTINFORCEFIELD(*collisionObjs*)
4: | *# Nächsten Knoten zu s im Graph bestimmen*
5: | *n* ← NEARESTNODE(*agent.position, graph*)
6: | *path* ← PATHPLANNING(*n, target*)
7: | *# Einen Schritt entlang des Pfades bewegen*
8: | MOVEFORCEFIELD(*path*)
9: **end function**

4.5 Evaluation

In diesem Abschnitt wird die Beobachtung über die gleichmäßig verteilten Kantenlängen und andere Eigenschaften von NGDPP, wie auftretende Kollisionen, empirisch bewertet. Außerdem wird NGDPP mit verwandten Ansätzen verglichen.

Zunächst soll mit Abbildung 4.5 eine beispielhafte Diskretisierung einer Freifläche durch NGDPP gezeigt werden. Ersichtlich sind die gleichmäßig im Raum verteilten Knoten des Graphen, was zu einer Homogenität in den Längen der Kanten führt. Die Agenten, die verschiedene Ziele ansteuern, sind mit einem Virtual Force Field umgeben und stoßen sich untereinander ab. Sie können sich in einem gewissen Ausmaß aus dem Zentrum des Kraftfeldes bewegen, um Kollisionen miteinander zu verhindern.

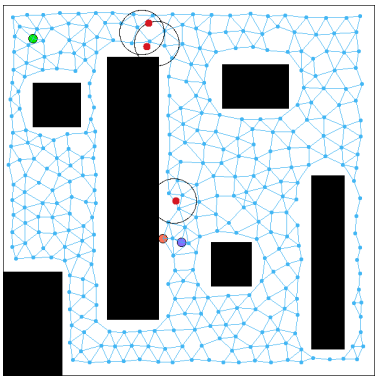


Abbildung 4.5: Beispielhafte Diskretisierung einer Freifläche mittels NGDPP zusammen mit multiplen Agenten, die zu verschiedenen Zielen geroutet werden. Sie sind mit einem Virtual Force Field umgeben und stoßen sich in einem gewissen Radius von einander ab, um Kollisionen zu vermeiden.

4.5.1 Vereinfachung für die Verwendung des A* Algorithmus

Im Abschnitt 4.4.2 wurde die Annahme getroffen, dass der SGNG-Algorithmus einen Graphen mit gleich langen Kanten erzeugt. Dies vereinfacht die Pfadplanung, da die tatsächlichen Kantenlängen nicht berechnet werden müssen, was einen Leistungsvorteil ergibt. In Abbildung 4.6 sind sowohl die Mittelwerte als auch die Varianzen der Kantenlängen über 100.000 Iterationen dargestellt. Zu Beginn ist ein starker Ausschlag in der Varianz zu erkennen, der aber schnell gegen Null konvergiert. Auch der Mittelwert konvergiert nach einigen tausend Iterationen des Algorithmus gegen einen bestimmten Wert. Der genaue Wert gegen welchen die Kantenlängen konvergieren hängt maßgeblich von der Einstellung des „maxError“ Parameters ab, der die Feinheit der Diskretisierung

steuert. Der genau Wert der Kantenlängen soll aber gerade ersetzt werden, indem alle Kantenlängen als gleich angenommen werden, d.h. z.B. als eins, um die Ausführung des A* Algorithmus zu beschleunigen.

Die obige Annahme wurde verwendet, um die Leistung von vier verschiedenen Pfadplanungsalgorithmen zu vergleichen. Abbildung 4.7 zeigt, dass die „vereinfachten“ Verfahren, welche die Annahme nutzen, die anderen in Bezug auf die Leistung übertreffen. Darüber hinaus wurde für mehr als 1 Million Iterationen überprüft, ob die Verfahren gegenüber der Vereinfachung eine andere Pfadlänge berechnen. Dies könnte theoretisch durch ungenaue Kantenlängen, aber auch durch die heuristische Arbeitsweise der A* geschehen. Es wurde jedoch keine einzige Abweichung in der Pfadlänge gefunden. Dadurch wird nicht nur die Leistung verbessert, sondern es ist auch sehr wahrscheinlich, dass der berechnete Pfad seine Qualität behält.

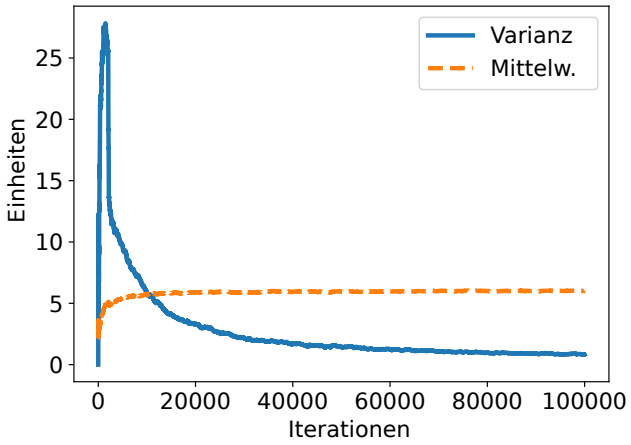


Abbildung 4.6: Mittelwert und Varianz aller tatsächlichen Kantenlängen über mehr als 100.000 Iterationen von NGDPP.

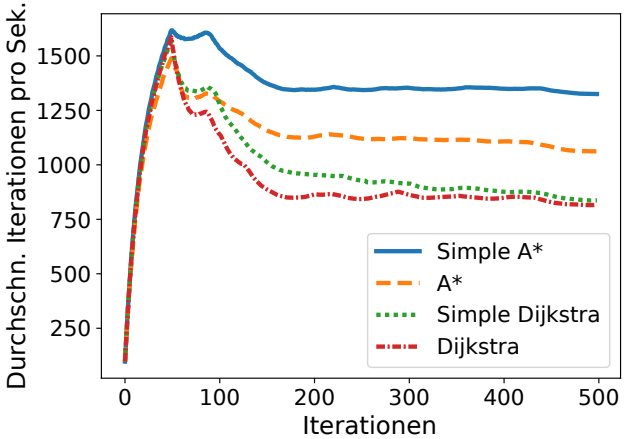
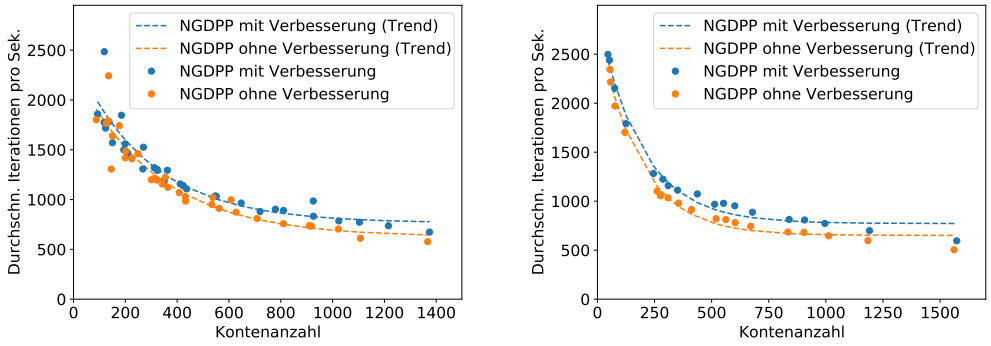


Abbildung 4.7: Leistungsvergleich von Pfadfindungsalgorithmen mit und ohne die angenommene Vereinfachung.

4.5.2 Nächste-Nachbarn-Suche und Kantenbehandlung von NGDPP

An dieser Stelle werden die in Abschnitt 4.4.1.1 und 4.4.1.2 betrachteten Leistungsverbesserungen bewertet. Ziel ist es, herauszufinden, ob der theoretische Vorteil der Konzepte empirisch nachgewiesen werden kann. Das Problem des „Nächsten Nachbarn“ bezieht sich auf die Tatsache, dass es sehr oft notwendig ist, den Knoten des Graphen mit dem geringsten Abstand zu einer bestimmten Position im Raum zu finden. Eine einfache Methode wäre, den Abstand zu allen Knoten zu berechnen und dann denjenigen auszuwählen, der den geringsten Abstand hat. Im Abschnitt 4.4.1.1 wurde jedoch eine Verbesserung vorgestellt. Abbildung 4.8a zeigt die durchschnittlichen Iterationen pro Sekunde für eine unterschiedliche Anzahl von Knoten im Graph. Die Trendlinie ergibt sich, indem bezogen auf die gleichfarbigen Datenpunkte ein exponentieller Zerfallsprozess angepasst wurde. Aus der resultierenden Trendlinie ist ersichtlich, dass das neu eingeführte Verfahren eine, wenn auch geringe, Verbesserung der Leistungsfähigkeit des Algorithmus darstellt, was insbesondere bei einer größeren Anzahl von Knoten und damit höherer Genauigkeit des Wegpunktgraphen der Fall ist.

Da der Algorithmus einen Mechanismus benötigt, um das Alter von Kanten zu bestimmen und sie zu löschen, wenn sie ein bestimmtes Alter überschreiten, wurde in Abschnitt 4.4.1.2 eine Leistungsverbesserung vorgeschlagen, die die Kanten in einer nach deren Alter sortierten Liste speichert. Abbildung 4.8b zeigt die durchschnittlichen Iterationen pro Sekunde für eine unterschiedliche Anzahl von Knoten im Graph. Die Trendlinie, die sich auf die gleichfarbigen Daten bezieht, ergibt sich aus einem exponentiellen Zerfallsprozess, dessen Parameter an die Datenpunkte angepasst wurde. Auf diese Weise wurde empirisch gezeigt, dass die Anwendung der Verbesserung zu einem Leistungsvorteil führt, da die Varianten des NGDPP mit der jeweiligen Verbesserung mehr Iterationen pro Sekunde erreichen.



(a) Ergebnisse für die Verbesserung der Nächste-Nachbarn-Suche.

(b) Ergebnisse mit einer sortierten Liste der Kanten.

Abbildung 4.8: Vorgeschlagene Leistungsverbesserungen

4.5.3 Kollisionen

NGDPP ist ein heuristischer Algorithmus und Kollisionen können sowohl zwischen Agenten als auch zwischen Agenten und ihrer Umgebung auftreten.

Wie im Abschnitt 4.4.1 erwähnt, ist die Diskretisierung der Umgebung mit SGNG nicht perfekt, da die Genauigkeit an Parameter des Algorithmus gebunden ist und auch aufgrund der Tatsache, dass die Diskretisierung sich iterativ an Veränderungen der Umgebung anpasst. Aus diesem Grund wird in diesem Experiment die Anzahl der falschen Kanten, d.h. Kanten, die sich mit der Umgebung schneiden (siehe Abbildung 4.2), nach einer Änderung der Karte über die weitere Ausführung von NGDPP gemessen. Das Experiment zeigt, dass der Anteil der falschen Kanten (oder Knoten) nach einer Änderung der Karte mit fortlaufender Ausführung von NGDPP signifikant abnimmt (Abbildung 4.9).

Abbildung 4.10 zeigt die Auswertung von auftretenden Kollisionen zwischen

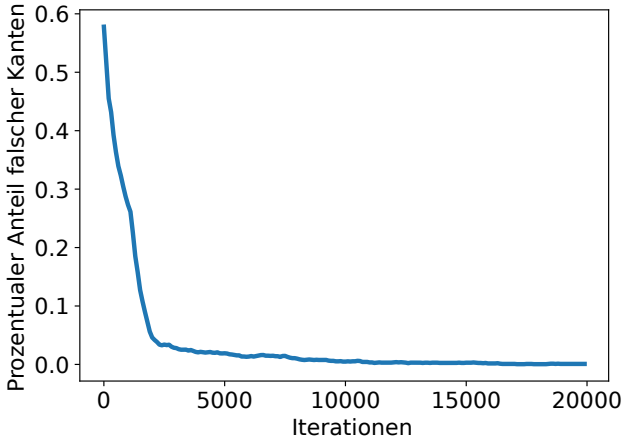


Abbildung 4.9: Prozentsatz der falschen Kanten über die Ausführung von NGDPP hinweg (Iterationen der Hauptschleife).

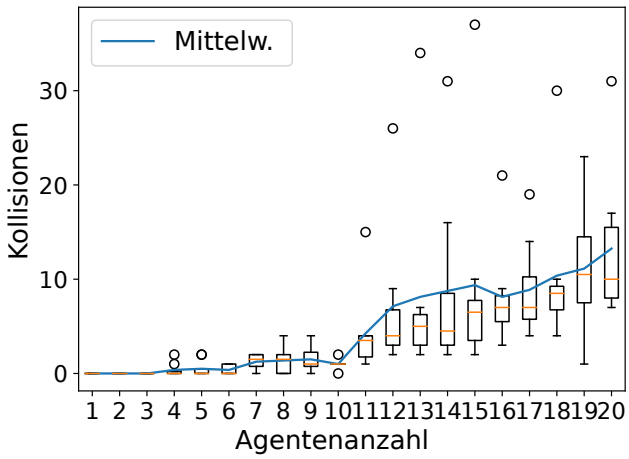


Abbildung 4.10: Auswertung der Kollision mehrerer Agenten.

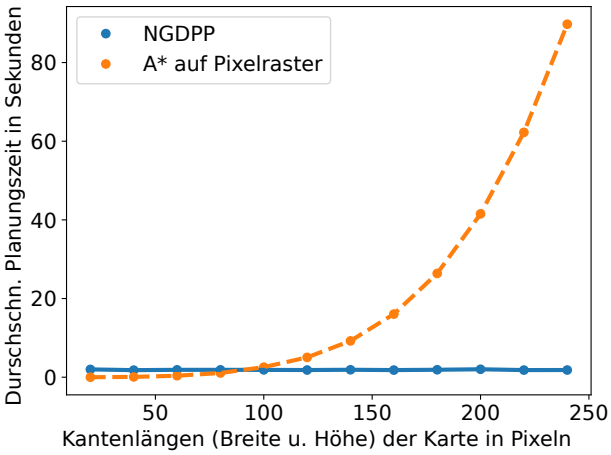


Abbildung 4.11: Leistungsvergleich von NGDPP zu A* Routing auf einem Graphen, der sich aus der Interpretation jedes Pixels des Grundrisses als Knoten ergibt.

Agenten. Im Experiment sammelte eine Anzahl von 1-20 Agenten zufällige Ziele über 100.000 Iterationen in einem freien Raum. Da Kollisionen probabilistisch sind, wurde das Experiment für jede Anzahl von Agenten achtmal wiederholt. Die Kurve zeigt die durchschnittliche Anzahl von Kollisionen pro Agentenanzahl. Es ist zu erkennen, dass eine größere Anzahl von Agenten häufiger zu Kollisionen führt.

4.5.4 Vergleich mit anderen Methoden

In Abbildung 4.11 wird der NGDPP-Algorithmus mit einem anderen, sehr einfachen Wegfindungsverfahren verglichen, um seine Effizienz zu zeigen. Das für den Vergleich herangezogene Verfahren verwendet den A*-Algorithmus zur Pfadfindung, der zugrundeliegende Graph ergibt sich jedoch aus der Tatsache, dass jedes begehbare Pixel des Grundrisses in einen Knoten mit Kanten zwischen benachbarten Knoten umgewandelt wird. Bei größeren Räumen nimmt die Laufzeit dieser Methode quadratisch zu, während die Laufzeit des NGDPP-Algorithmus unverändert bleibt. Dies liegt daran, dass sich ohne Anpassung der Parameter des NDGPP-Algorithmus die Anzahl der erzeugten Knoten und Kanten selbst bei höher aufgelösten Karten nicht ändert und damit auch seine Laufzeit konstant bleibt.

Vergleich mit Probabilistic Roadmaps

In Abschnitt 4.3 wurde der PRM-Ansatz mit dem Kritikpunkt erwähnt, dass die Probabilistic Roadmap nach jeder Veränderung des Raumes komplett neu aufgebaut werden muss. Während dieser Rekonstruktion des Wegpunktgraphen kann kein Routing ausgeführt werden. Im Vergleich dazu passt sich der

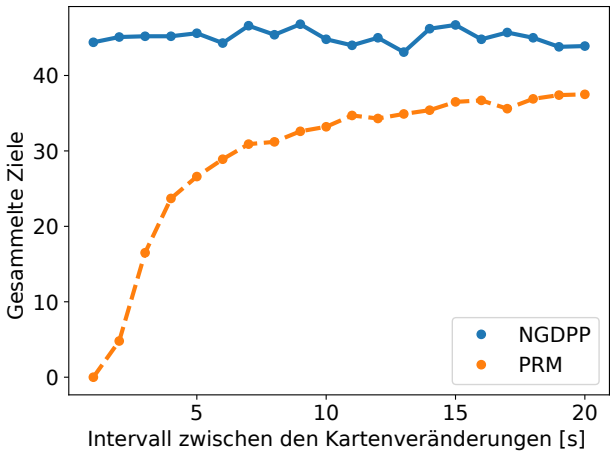


Abbildung 4.12: Leistungsvergleich von NGDPP mit einer einfachen PRM-Implementierung (adaptiert von [127]).

hier vorgestellt NGDPP-Ansatz dynamisch an Änderungen an und erlaubt kontinuierlich Routing-Abfragen (mit der Einschränkung, dass Kanten bei der Anpassung von Änderungen mit Hindernissen kollidieren könnten oder mit ungeeigneten Parametereinstellungen, wie in Abschnitt 4.4.1 erwähnt).

Im Experiment wurden beide Ansätze für die Navigation eines einzelnen Agenten verwendet, der Ziele sammeln musste, die im Verlauf von 5 Minuten an zufälligen Positionen auf der Karte auftauchten. In diesen 5 Minuten traten Veränderungen der Karte mit unterschiedlichen Geschwindigkeiten auf. Die Veränderungen hatten den Charakter, dass Teile der Karte zugänglich und wieder blockiert wurden. Das Experiment wurde für jeden Algorithmus und jede Kartenänderungsrate 10 Mal wiederholt. Der Mittelwert der Ergebnisse ist in Abbildung 4.12 eingezeichnet.

Die Ergebnisse zeigen, dass der Agent, der NGDPP verwendet, im Verlauf von 5 Minuten eine nahezu konstante Menge an Zielen sammelt, da die Anpassung des Routing-Graphen an die Kartenänderungen und die Wegplanung des Agenten verzahnt ablaufen. Auf der anderen Seite muss der Agent, der den PRM-Ansatz verwendet, auf die Fertigstellung des Wegpunktgraphen warten, bevor die Navigation beginnen kann. Wenn das Intervall zwischen Kartenänderungen gering ist, z.B. 2 Sekunden, nimmt die Erstellung des Graphen die meiste Zeit in Anspruch, so dass der Agent keine Zeit hat, sich zu bewegen, bevor die nächste Änderung eintritt und der Graph wiederum neu erstellt werden muss.

Vergleich mit Rapidly Exploring Random Trees

Wie in Abschnitt 4.3 erwähnt, besteht ein Problem bei RRT (siehe Abschnitt 2.3.2) darin, dass für jeden Agenten ein eigener Graph (Baum) erstellt werden muss. Die Überlegung war hier, dass mit zunehmender Anzahl von Agenten

NGDPP der RRT-Methode überlegen sein sollte. Dies beruht auf der Tatsache, dass bei NGDPP alle Agenten den gleichen Wegpunktgraph verwenden können. Um diese These empirisch zu verifizieren, wurde das folgende Experiment durchgeführt.

Die beiden Algorithmen wurden verwendet, um eine Anzahl von 1 - 10 Agenten zufällig platzierte Ziele (aber für beide die gleichen) für 3 Minuten in 10 verschiedenen Umgebungen sammeln zu lassen. Die Ergebnisse dieses Experiments sind in Abbildung 4.13 zu sehen.

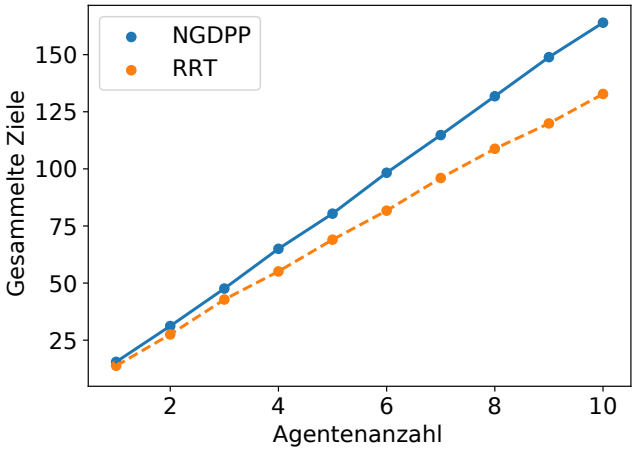


Abbildung 4.13: Leistungsvergleich von NGDPP mit einer einfachen RRT-Implementierung (adaptiert von [127]).

Das Experiment zeigt, dass das NGDPP dem RRT-Ansatz überlegen ist. Agenten, die NGDPP verwenden, sammeln mehr Ziele in der gleichen Zeit, was auf die höhere Planungszeit von RRT zurückzuführen ist, während der NGDPP-Wegpunktgraph kontinuierlich von allen Agenten parallel verwendet werden kann. Dieser Unterschied nimmt, wie bereits angenommen, mit einer höheren Anzahl von Agenten zu.

4.6 Diskussion und Zusammenfassung

In diesem Kapitel wurde vom Autor eine neue ganzheitliche Methode zur Pfadplanung in dynamischen Umgebungen entwickelt und vorgestellt: der NGDPP-Algorithmus. Die in der Motivation dieses Kapitels im Abschnitt 4.2 erwähnten Teilprobleme der Problemdomäne, Diskretisierung, Pfadplanung und Kollisionsvermeidung, wurden separat behandelt. Die dynamische Diskretisierung des Raumes wurde mit dem SGNG-Algorithmus gelöst. Der resultierende Wegpunktgraph kann dann vom A*-Algorithmus zur Planung eines gültigen Pfades verwendet werden. Während der Ausführung wird dieser Pfad mit der „Potential Field“-Methode lokal angepasst, so dass Kollisionen mit dynamischen Hindernissen vermieden werden. Im Abschnitt 4.4 wird erläutert, wie

diese Methoden zusammenwirken, um den neuen NGDPP-Algorithmus zu bilden, und welche Synergien durch die Kombination der Methoden entstehen. Bei der Auswertung ergeben die vorgeschlagenen Änderungen, die an den Methoden für die Teilprobleme vorgenommen werden können, um rechnerische Vorteile zu erzielen geringfügige Leistungsverbesserungen. Insgesamt zeigt der Algorithmus im Vergleich zu verwandten Arbeiten wie PRM oder RRT eine gute Leistung mit Leistungsvorteilen in gewissen Anforderungsbereichen, wie häufigen Kartenänderungen oder der Unterstützung multipler Agenten.

Die meisten realen Agenten können keine plötzlichen Drehungen z.B. in einem 90°-Winkel ausführen. Solche Agenten wären wahrscheinlich nicht in der Lage, die meisten vom NGDPP-Algorithmus geplanten Wege auszuführen. Eine mögliche Lösung hierfür ist die Glättung des Pfades mit Splines [15] bzw. durch den De-Casteljau-Algorithmus [4]. Inwieweit dies auf einen durch den NGDPP-Algorithmus berechneten Pfad angewendet werden kann, ist zu prüfen.

Es ist davon auszugehen, dass der Algorithmus ohne weiteres für dreidimensionale Räume angepasst werden kann, da alle verwendeten Methoden, d.h. sowohl das Neural Gas, das Potential Field, als auch die Verbesserungsvorschläge, in einer dreidimensionalen Anwendung möglich wären. Der Algorithmus könnte dann z.B. für die Bewegungsplanung von fliegenden Drohnen verwendet werden. Die Anwendbarkeit vom NGDPP und seine Leistungsfähigkeit in dreidimensionalen Räumen ist jedoch noch nicht getestet worden und lassen Raum für weitere Untersuchungen.

5 Steuerung eines Schwarms

Nachdem in den vorangegangen Kapiteln untersucht wurde, wie einzelne oder auch mehrere Agenten zu individuellen Zielen gesteuert werden können, soll in diesem Kapitel betrachtet werden, ob eine Gruppe an Agenten als Einheit gelenkt werden kann. Dazu wird ein Szenario entwickelt, in welchem die einzelnen Individuen mittels Reinforcement Learning darauf trainiert werden einem virtuellen Futterpunkt zu folgen. Das ausgewählte Trainingsverfahren musste insbesondere in der Belohnungsfunktion und den den Agenten zur Verfügung stehenden Aktionen und Beobachtungen der Umwelt an das zu untersuchende Szenario angepasst werden. Die Agenten werden dafür belohnt ihren Abstand zum Zielpunkt zu minimieren. Sie erhalten als Beobachtung Informationen über den Zielpunkt und ihre nächsten Nachbarn, sofern sie sich in Beobachtungsreichweite befinden. Als Aktion können sie aus einer Menge vordefinierter Drehwinkel wählen, während sie sich mit konstanter Geschwindigkeit fortbewegen. In der Evaluation wird gezeigt, dass die Agenten im entwickelten Szenario lernen, dem Zielpunkt zu folgen auch, wenn dieser zeitweise aus ihrer Wahrnehmung verschwindet, indem sie sich in ihren Nachbarn orientieren. Darüber hinaus wird gezeigt, dass die Agenten ohne explizit darauf trainiert zu werden, lernen, Kollision untereinander zu vermeiden. Weil die Agenten bei Kollisionen verlangsamt werden, ist das Vermeiden von diesen ihrem Ziel förderlich, dem virtuellen Zielpunkt zu folgen. Eine weitere Erkenntnis der in diesem Kapitel durchgeführten Untersuchung ist, dass die gelernten Verhaltensweisen ähnliche Eigenschaften wie ein etablierter Ansatz zur algorithmischen Erzeugung von Schwarmverhalten aufweisen. Die Technik des Reinforcement Learnings wurde bereits in den Grundlagen in Kapitel 2 erläutert, wird aber im Abschnitt 3.3 noch einmal aufgegriffen und im Hinblick auf Schwärme von Agenten betrachtet. Nachdem im Abschnitt 5.1 die dem Kapitel zugrundeliegende Vorveröffentlichung dargestellt und das Kapitel im Abschnitt 5.2 weiter motiviert wurde. Der Ansatz selbst, mit den damit einhergehenden Trainingsmethoden wird in Abschnitt 5.4 erläutert, im Abschnitt 5.5 evaluiert und in 5.6 diskutiert.

5.1 Vorveröffentlichungen

Die Kerninhalte dieses Kapitels wurden vom Autor bereits in [61] publiziert. Diese ist aus einer Bachelorarbeit hervorgegangen, welche aber wiederum stark auf den Publikationen [60] und [59] des Autors basiert. Die Fragestellung der Bachelorarbeit wurde vom Autor dieser Arbeit aufgeworfen, denn sie ergibt sich generisch aus den beiden vorweg genannten Publikationen, da sie das dort

untersuchte Setting herumdreht. In der Bachelorarbeit selbst wurde auf die Simulationsumgebung des Autors und dem von ihm gewählten Trainingsansatz zurückgegriffen. Bei der Bachelorarbeit standen der Autor und Fabian Ritz als Betreuer zur Seite. Zudem ist die Evaluation stark an die vom Autor entwickelte Methodik zur Messung von Schwarmverhalten angelehnt. Damit bildet dieses Kapitel bzw. die dazugehörige Veröffentlichung ein Bindeglied zwischen anderen Arbeiten des Autors, nämlich, Untersuchungen zur kollisionsfreie Wegfindung einzelner oder multipler Agenten hin zur Steuerung von Schwärmen als Ganzes und der Analyse emergenten Verhaltens von Agenten, die mittels Reinforcement Learning auf ein gewisses Ziel hin trainiert wurden.

Die Erstellung der Publikation wurde hauptsächlich vom Autor vorangetrieben, Paula Wikidal, Fabian Ritz, Thomy Phan und Thomas Gabor lieferten Beiträge. Die Abbildungen 5.1a und 5.1b sowie die Tabelle 5.3 wurden zur besseren Darstellung und Verständlichkeit für diese Kapitel neu erstellt. Die Abbildungen 5.2, 5.3, 5.4, 5.5 und 5.6 sowie die Tabellen 5.1 und 5.2 sind bereits in [61] enthalten.

5.2 Motivation

In diesem Kapitel wird nun dazu übergegangen einen Schwarm autonomer Agenten als eine Einheit zu steuern. Das biologische Äquivalent zu dieser Aufgabe wären Verfolgungs- oder Nahrungssuche-Szenarien, in denen mehrere Individuen im Schwarm eine Beute verfolgen. Schwarmverhalten ist ein in der Natur weit verbreitetes Phänomen, bei dem sich mehrere Entitäten zusammenschließen, um Raubtieren auszuweichen, Futter zu suchen oder kollektive Entscheidungen zu treffen, wie z.B. die Auswahl eines neuen Nestes. Es steht zur Debatte, ob dieses Verhalten aus reinem Eigeninteresse auf der Ebene des Individuums in der Evolution entstanden sein könnte, z.B. ob Fische sich zusammenschließen, um ihre individuellen Überlebenschancen auf Kosten des Lebens einiger anderer Fische zu erhöhen, oder ob Vögel sich in Scharen zusammenschließen, um schneller eine Nahrungsquelle zu finden, obwohl sie die Nahrung anschließend miteinander teilen müssen.

Ein bedeutender Ansatz zur Simulation von Schwarmverhalten verwendet einen festen Regelsatz um dieses zu anzuregen. Zudem gibt es Ansätze, die mit Hilfe von Reinforcement Learning versuchen Schwärme zu erzeugen. Diese verwenden allerdings in der Regel spezielle Belohnungsfunktionen, um Schwärme mit einem expliziten Optimierungsziel zu erzwingen oder arbeiten auf diskreten Welt Modellen vergleichbar zu Rastergrafiken bzw. einem Schachbrett (engl. Gridworlds). Diese Ansätze können zwar zu entstehenden Schwärmen führen, erfordern jedoch detaillierte Vorkenntnisse über das Anwendungsgebiet. Darüber hinaus liefern diese Ansätze keine weiteren Erkenntnisse über das Schwarmverhalten, da das Schwärmen künstlich erzwungen wird. In diesem Kapitel wird ein Ansatz vorgestellt bei welchem mehrere autonome Einheiten einem sich bewegenden Zielobjekt in einer kontinuierlichen und nur teilweise

beobachtbaren Domäne folgen müssen. Mit Hilfe von Multi-Agent Reinforcement Learning wird die Entstehung von Schwärmen bei einer gedachten Nahrungssuche untersucht. Alle Entitäten handeln aus Eigeninteresse, so dass es keinen direkten Anreiz zur Zusammenarbeit gibt. Die Auswertung zeigt einfaches kooperatives Verhalten, d.h. die Individuen lernen Schwärme zu bilden und so dem Zielobjekt zu folgen, auch wenn es für die meisten Individuen außer Sichtweite ist.

Die Simulation dieses Phänomens ist für eine Vielzahl von Bereichen interessant, in denen Schwarmverhalten verstanden werden muss, wie z.B. bei der Planung der Lage von Notausgängen oder in Rettungsszenarien, in denen mehrere Helfer dezentral Opfer finden und retten müssen (siehe z.B. [106]). Das Verständnis der Entstehung von Schwärmen, die sich aus eigennützigen Zielen und teilweiser Beobachtbarkeit ergeben, kann zu neuen Wegen zur Lösung realer Probleme führen, wie z.B. Verkehrseffizienz, bei der mehrere Fahrer durch eine Stadt navigieren, oder Kommunikationsnetzwerke, bei denen verschiedene Netzwerkbenutzer eine gemeinsame Netzwerkressource effizient gemeinsam nutzen müssen. In all diesen Situationen gibt es nur eigennützige Einheiten mit begrenzten Informationen über den globalen Zustand.

5.3 Hintergrund und verwandte Arbeiten

Nachfolgend wird zuerst das Multi-Agent Reinforcement Learning weiter formalisiert, bevor auf bestehende Ansätze zur algorithmischen Erzeugung von Schwarmformationen eingegangen wird.

5.3.1 Multi-Agent Reinforcement Learning

Das Problem zur Steuerung multipler Agenten bzw. *Multi-Agent Reinforcement Learning (MARL)* Probleme können als stochastisches Spiel $M = \langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{P}, (\mathcal{R}_e)_{1 \leq e \leq N}, \mathcal{Z}, \mathcal{O} \rangle$ formuliert werden, dabei ist $\mathcal{D} = \{1,....,N\}$ eine Menge von Agenten, $\mathcal{S}$ eine Menge von Zuständen s und $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$ die Menge der *gemeinsamen Aktionen* a. $\mathcal{P}(s'|s,a)$ bezeichnet die Übergangswahrscheinlichkeit, in einen Zustand s' zu gelangen, wenn im Zustand s die gemeinsame Aktion a ausgeführt wird. Wird im Zustand s eine gemeinsame Aktion a ausgeführt, erhält jeder Agent $e \in \mathcal{D}$ eine skalare Belohnung $r_e = \mathcal{R}_e(s,a)$. $\mathcal{Z}$ ist eine Menge von lokalen Beobachtungen o_e für jeden Agenten $e \in \mathcal{D}$ und $\mathcal{O}(s) = o' = \langle o'_1,...,o'_N \rangle \in \mathcal{Z}^N$ die gemeinsame Beobachtungsfunktion ist, die den Zustand s der Umgebung in eine lokale Beobachtung o_e für jeden Agenten $e \in \mathcal{D}$ übersetzt.

Das Ziel jedes Agenten e ist es, eine *beste Antwort* Strategie $\pi_e : \mathcal{Z} \rightarrow \mathcal{A}_e$ (auf die Aktionen der anderen Agenten) zu finden, die seine erwartete Belohnung maximiert. π_e kann anhand der Q-Funktion bewertet werden $Q_e^{\pi}(s,a) = \mathbf{E}_{a=\pi(s)}[G|s,a]$, wobei $\pi = \langle \pi_1, ..., \pi_N \rangle$ die gemeinsame Strategie aller Agenten

ist, welcher gefolgt wird, und $G_t = \sum_{k=1}^{\infty} \gamma^{k-1} \mathcal{R}_e(s_k, a_k)$ die kumulierte Belohnung mit Diskontierungsfaktor $\gamma \in [0,1)$ ist. Zu beachten ist, dass Q_e immer von der aktuellen Strategie anderer Agenten abhängt.

Die besten Antworten eines Agenten auf die Aktionen der Anderen lassen sich mit Reinforcement Learning finden, wobei π_e mit Erfahrungstupeln $\langle o_e, a_e, r_e, o'_e \rangle$ gelernt wird, die aus der Interaktion der Agenten mit der Umgebung gewonnen werden. *Q-Learning* ist ein beliebter Ansatz für RL, bei dem $\hat{Q}_e(o_e,a_e) \approx Q_e^{\pi}$ mit der folgenden Aktualisierungsregel angenähert wird [141]:

$$\hat{Q}_e(o_e,a_e) \leftarrow (1-\alpha) \cdot \hat{Q}_e(o_e,a_e) + \alpha \cdot y \tag{5.1}$$

wobei $y = r_e + \gamma \cdot \max_{a'_e} \hat{Q}_e(o'_e,a'_e)$ und $\alpha \in [0,1]$ die Lernrate ist.

RL auf dem aktuellen Stand der Technik wird mit Deep Learning unter Verwendung von *Deep Q-Networks (DQN)* implementiert und wurde auf Multiagentensysteme in einer Vielzahl von Bereichen angewendet ([100, 87, 49]).

5.3.2 Emergentes Schwärmen

In einer grundlegenden Arbeit zur Schwarmsimulation formulierte Craig Reynolds 1987 [121] drei Verhaltensregeln für autonom agierende Einheiten (von Reynolds in Anlehnung an Birds *Boids* genannt) zur Bildung eines Schwarms:

Kohäsion Steuerung zur zentralen Position unter den benachbarten Boids (Massenschwerpunkt).

Separation Entfernung von anderen Boids weg, um einen Mindestabstand einzuhalten.

Ausrichtung Anpassen der eigenen Ausrichtung an die der benachbarten Boids.

Wenn jedes Individuum diesen Regeln folgt, entsteht eine Schwarmformation. Die Regeln basieren ausschließlich auf lokalen Informationen. Jeder Boid benötigt nur die Position und Bewegungsrichtung seiner nächsten Nachbarn, nicht aber einen Überblick über den gesamten Schwarm. Dieser Ansatz ist in hohem Maße skalierbar, denn selbst wenn die Anzahl der Boids zunimmt, bleibt die Informationsmenge zur Bestimmung der Bewegungsrichtung jedes Boids gleich. Jede dieser drei Regeln kann als eine Kraft verstanden werden, die die Boids in eine bestimmte Richtung lenkt.

In der Folge entwickelte Reynolds Algorithmen, um natürliches kollektives Verhalten in Situationen zu erzeugen, in denen Individuen einem gemeinsamen Anführer folgen oder Hindernissen ausweichen [122]. Dies wird durch eine Kombination aus der oben beschriebenen *Separation* und dem so genannten *Ankunftsverhalten* erreicht. Im Wesentlichen wird eine kontinuierliche Kraft zwischen der aktuellen Position eines Boids und seiner Zielposition angewendet. Diese Kraft lenkt die Bewegungsrichtung des Boids um, bis er schließlich

am Ziel ankommt. Eine zusätzliche Kraft fügt eine verlangsamende Wirkung hinzu. Diese Kraft wird dazu verwendet, den Boid bei der Annäherung an das Ziel abzubremsen, damit er sich nicht über das Ziel hinaus bewegt. Damit ein Boid einem Führer folgen kann, wird die Zielposition mit einem gewissen Abstand hinter der tatsächlichen Position des Führers definiert. Um dem Anführer nicht in die Quere zu kommen, werden die Boids auch seitlich abgelenkt, wenn sie sich vor dem Anführer befinden.

Um die statischen Regeldefinitionen zu überwinden, nutzten Morihiro et al. [102] Multi-Agent Reinforcement Learning, insbesondere Q-Learning, um Agenten auf die Einhaltung dieser Regeln zu trainieren. Anstelle von festen Regeln können die Agenten zwischen vier verschiedenen Aktionen wählen, um ihre Bewegungsrichtung zu ändern. In ihrem Modell lernen die Agenten iterativ, während in jedem Zeitschritt ein Agent i nur einen anderen Agenten j berücksichtigt. Der Agent i erhält den euklidischen Abstand zu j als Beobachtung und kann zwischen vier auszuführenden Aktionen wählen. Mit einer Aktion bewegt sich Agent i auf den Agenten j zu, vom Agenten j weg oder parallel zum Agenten j entweder in die gleiche oder in die entgegengesetzte Richtung. Die Belohnung, die der Agent i für eine Handlung erhält, hängt von dem zuvor erwähnten Abstand zum Agenten j ab und ist so gestaltet, dass sie intuitiv der Kohäsions- und Separationsregel entspricht. In dieser Hinsicht erhält der Agent i eine positive Belohnung, wenn er so agiert, dass sein Abstand zu j innerhalb vordefinierter Grenzen bleibt. Morihiro et al. erweitern ihr Szenario insofern, dass periodisch ein Räuber erscheint, den die Agenten entsprechend ihres Wahrnehmungsbereichs sehen. Dann ergibt die Belohnungsfunktion eine positive Belohnung für das Einhalten eines bestimmten Abstands zu den nächsten Nachbarn und für das Entfernen vom Räuber. Während die Agenten in den Experimenten eine starke Schwarmkohäsion entwickelten, wurde dies hauptsächlich durch die Gestaltung der Belohnung analog zu Reynolds Verhaltensregeln erreicht. Darüber hinaus wurden keine Szenarien in Betracht gezogen, in denen der Schwarm in eine bestimmte Richtung gelenkt werden soll, ohne vor einem Räuber zu fliehen.

Dieses Kapitel bzw. die dazugehörige Veröffentlichung [61] erweitert die Publikation [60] des Autors und untersucht, ob Schwarmformationen gelenkt werden können, z.B. um sich in eine bestimmte Richtung zu bewegen oder einem Ziel zu folgen, wobei ausschließlich positive Belohnungen im Multi-Agent Reinforcement Learning verwendet werden, d.h. es geht rein um die positive Verstärkung und nicht um Bestrafung von Verhalten. Das biologische Äquivalent zu dieser Aufgabe wären Verfolgungs- oder Nahrungssuche-Szenarien, in denen mehrere Individuen eine Beute verfolgen. Während es in diskreten Bereichen wie Gitterwelten, z.B. [48], [149] und [117], reichlich Arbeit an ähnlichen Aufgaben gibt, wird Multi-Agent Reinforcement Learning nur selten in kontinuierlichen Szenarien eingesetzt. Und während Hüttenrauch et al. [64] eine eingehende Untersuchung von End-to-End-RL-Feature-Einbettungen als Zustandsrepräsentationen in einer kontinuierlichen Multi-Agenten-Domäne

bieten, konzentrieren sie sich auf die Verbesserung der Leistungsfähigkeit der Verhaltensstrategie, analysieren aber nicht weiter, ob die entstehenden Strategien erfolgreiche Problemlösung mit (natürlichem) Schwärmen kombinieren.

5.4 Futtersuchschwärme

In diesem Kapitel wird untersucht, inwieweit sich multiple autonome Agenten darauf trainieren lassen einem Zielobjekt zu folgen, um so als ein Schwarm gesteuert zu werden. Analog des natürlichen Vorbilds der Futtersuche soll jeder Agent dabei egoistisch handeln, so dass sich unter Umständen Schwarmverhalten emergent ergibt. Das Lernproblem wird daher so definiert, dass die Agenten nur indirekte Vorteile dadurch haben, das Handeln anderen Agenten in ihre Entscheidungsfindung einzubeziehen. Deshalb soll die Belohnung eines Agenten nicht direkt von der Interaktion mit den anderen Agenten abhängen, so wie es bei Morihiro et al. [102] der Fall war. Stattdessen wird die Belohnungsfunktion dazu verwendet, ihnen das Ziel zu geben, einen virtuellen Punkt, im Folgenden auch als Zielobjekt bezeichnet, auf die bestmögliche Weise zu verfolgen. Die Umgebungsbedingungen sowie die Handlungs- und Beobachtungsmöglichkeiten der Agenten werden im Folgenden näher beschrieben. Die wichtigsten Daten sind in der Tabelle 5.1 zusammengefasst.

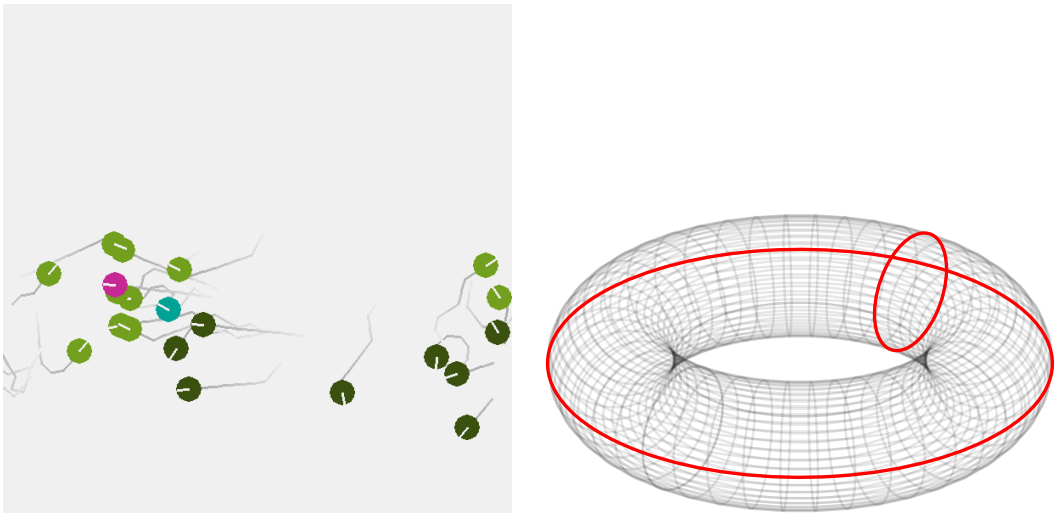
Tabelle 5.1: Die wichtigsten Parameter der Domäne.

Feldgröße	60x60
Anzahl der Agenten	25
Maximal wahrnehmbare benachbarte Agenten	8
Radius eines Agenten	1
Bewegungsdistanz eines Agenten pro Schritt	1.0
Bewegungsdistanz des Zielobjekts pro Schritt	0.95
Geschwindigkeitsreduzierung im Falle einer Kollision	50%

5.4.1 Domäne

Die Agenten bewegen sich in einer quadratischen, zweidimensionalen Welt und besitzen eine vordefinierten Radius. Die Position jedes Agenten ist durch seine x- und y-Koordinaten definiert. Dies sind reelle Zahlen, so dass der Zustandsraum kontinuierlich ist. Zu Beginn jeder Episode werden alle Agenten und auch das Zielobjekt mit zufälligen Positionen und Orientierungen innerhalb der Welt initialisiert. Danach können sie sich in beliebige Richtungen bewegen, indem sie sich zu jedem Zeitschritt für einen bestimmten Drehwinkel entscheiden. Sie bewegen sich mit konstanter Geschwindigkeit in ihre Blickrichtung. Der Raum hat die Eigenschaft eines Torus. Das bedeutet, dass ein Agent, der die quadratische Visualisierung nach rechts verlässt, sofort wieder von links in den

Raum eintritt (dasselbe gilt für die Gegenrichtung bzw. oben und unten). Das primäre Ziel der Agenten ist es, dem Zielobjekt so nah wie möglich zu folgen. Dieses bewegt sich zufällig. Seine Bewegungsrichtung wird in Intervallen variabler Länge im Bereich von 1 bis 15 Schritten geändert. Die Aktion, die nach dieser Zeit ausgeführt wird, ist ebenfalls zufällig und entspricht den Bewegungsoptionen, die den Agenten ebenfalls zur Verfügung stehen und die im Abschnitt Aktionen näher beschrieben werden. Um die Aufgabe der verfolgenden Agenten etwas zu vereinfachen, ist ihre Bewegungsgeschwindigkeit etwas geringer. Da das Zielobjekt eher ein virtueller Punkt als ein Teil der physischen Welt ist, werden Kollisionen mit ihm nicht behandelt. Die Domäne ist beispielhaft in Abbildung 5.1 dargestellt.



(a) Beispielbild der Domäne.

(b) Torus, der die verbundenen Kanten der Domäne symbolisiert

Abbildung 5.1: (a) zeigt eine beispielhafte Visualisierung der Domäne, in welcher sich Agenten (grüne Kreise) bewegen. Der virtuelle Zielpunkt ist als pinker Kreis dargestellt. Agenten, die den Sichtkontakt zu diesem verloren haben, werden dunkler dargestellt. Der Kreis in Türkis ist der einzelne lernende Agent, dessen erlernte Verhaltensstrategie von allen genutzt wird. Der Schweif zeigt die letzten zehn Schritte der Objekte und dient nur zur Visualisierung in einem Bild. In (b) ist die Torus-Eigenschaft zu sehen, bei welchem gegenüberliegende Kanten miteinander verbunden sind. Die Koordinaten werden aber nicht wie in der Skizze verzerrt, sondern die Eigenschaft wird durch eine einfache Modulo Rechnung auf den Koordinaten erzeugt.

5.4.2 Aktionen

Der Aktionsraum A ist diskret und umfasst fünf verschiedene Drehwinkel, zwischen denen die Agenten als Aktion wählen können: $\{-90°, -45°, 0°, +45°, +90°\}$. Innerhalb eines Zeitschrittes wählen alle Agenten eine der fünf Aktionen aus und drehen sich in die entsprechende Richtung. Wenn sie mit einem anderen Agenten kollidieren, d.h. der Abstand zu dem anderen Agenten kleiner ist als die addierten Radien beider Agenten, bewegen sie sich nur mit halber Geschwindigkeit vorwärts. Um die Kollisionshandhabung realistischer zu gestalten, wird der Agent nur dann abgebremst, wenn er zum Zeitpunkt der Kollision in Richtung des anderen Agenten gedreht ist. Das bedeutet, dass in Situationen, in denen ein Agent einen anderen von hinten trifft, nur der hintere der beiden Agenten abgebremst wird. Die Agenten können ihre Geschwindigkeit nicht direkt ändern. Sie können jedoch indirekt ihre Geschwindigkeit in einer bestimmten Richtung durch Zickzack Fahren verlangsamen und dadurch noch einen gewissen Einfluss darauf ausüben.

5.4.3 Beobachtungsraum

Jeder Agent erhält Informationen über das Zielobjekt, sich selbst und die ihn umgebenden Agenten. Die Daten beschreiben die Entfernung und Richtung in Abhängigkeit von der Position und Blickrichtung der beobachteten Einheiten und ihrer globalen Orientierung, d.h. der Richtung, in die sie sich gerade bewegen. Wie im Abschnitt Emergentes Schwärmen erwähnt, können sich Schwärme nur mit lokalen Informationen der Individuen bilden und damit als ein teilweise beobachtbares Lernproblem modelliert werden. Daher enthält jede Beobachtung $o \in O$ nur einen Teil der Information über den Gesamtzustand der Umwelt, welcher zuvor als $s \in S$ bezeichnet wurde. Diese Zuordnung von s zu o wird durch das Beobachtungsmodell ξ vorgenommen. In dem hier definierten Szenario besteht dieses darin, den Wahrnehmungsbereich eines Agenten auf seine nächsten 8 Nachbarn zu begrenzen. Das Zielobjekt ist für den Agenten unsichtbar, wenn es sich nicht unter den nächsten Nachbarn befindet. Außerdem kann ein Agent nur zwischen Entfernungen unterscheiden, die unter 30 Einheiten, d.h. der halben Breite des Spielfelds, liegen. Wenn ein anderer Agent, der Teil der Beobachtung ist, weiter entfernt ist als dieser Wert, wird der Abstand zu ihm in der Beobachtung auf den maximalen Wert von 30 gesetzt.

Für jede beobachtbare Entität e erhält ein Agent ein 3-Tupel, das den euklidischen Abstand zwischen der Entität und dem Agenten, den Winkel, den sich der Agent drehen müsste, um der beobachteten Entität zugewandt zu sein, und die absolute Orientierung der Entität in der Umgebung: $(dist_e, direction_e, orientation_e)$. Da es sich bei der Umgebung um einen Torus handelt, werden die Entfernungen auch um die Kanten des visualisierten Quadrats herum berechnet, wobei die kürzere Entfernung für die Beobachtung

herangezogen wird (mit der dementsprechenden Ausrichtung $direction_e$). Die absolute Orientierung einer Entität wird in Grad [0°,360°) gemessen, wobei die Ausrichtung nach Osten 0° entspricht und der Winkel gegen den Uhrzeigersinn gemessen wird. Der Winkel, den ein Agent drehen müsste, um sich einer anderen Entität zuzuwenden, wird in Grad im Bereich von $(-180°,180°]$ gemessen.

Alle drei Informationen über Abstand, Richtungswinkel (um den sich ein Agent drehen müsste, um zum beobachteten Objekt zu schauen) und der globale Blickwinkel des beobachteten Objekts werden normiert und in einer zweidimensionalen Matrix zusammengefasst. Dementsprechend erhält ein Agent die folgende Beobachtung für das Ziel, sich selbst und die n nächstgelegenen Nachbaragenten, in der die n Nachbarn nach ihrer Entfernung geordnet sind.

$$\begin{bmatrix} dist_{target} & direction_{target} & orientation_{target} \\ 0 & 0 & orientation_{self} \\ dist_{neighbor_1} & direction_{neighbor_1} & orientation_{neighbor_1} \\ dist_{neighbor_2} & direction_{neighbor_2} & orientation_{neighbor_2} \\ & \vdots & \\ dist_{neighbor_n} & direction_{neighbor_n} & orientation_{neighbor_n} \end{bmatrix}$$

Die erste Zeile ist für das Zielobjekt reserviert, gefolgt von den Daten über die eigene Position, wobei nur die aktuelle Blickrichtung von Interesse ist. Entfernung und Winkel werden daher auf null gesetzt. Die restlichen Zeilen beschreiben die Entfernung, Richtungswinkel und Orientierung der anderen Agenten innerhalb des Wahrnehmungsbereichs. Die Architektur des neuronalen Netzes erfordert, dass die Beobachtungen eine feste Länge haben. Daher wird der Mangel an Information über die Position des Zielobjekts (oder weit entfernter Nachbar) modelliert, indem der entsprechende Teil der Matrix durch (-1) Einträge ersetzt wird. Dadurch gewählte Vorgehen kann das Spielfeld später skaliert werden, ohne den Wertebereich zu verändern. Zudem kann die Anzahl der Agenten erhöht werden, ohne dass sich die Länge des Beobachtungsvektors eines einzelnen Agenten ändert.

5.4.4 Belohnungen

Die Belohnung, die ein Agent $e \in \mathcal{D}$ in jedem Zeitschritt erhält, hängt linear von seiner Entfernung zum Zielobjekt ab. Genauer wird sie aus der negativ beobachteten Entfernung zum Zielobjekt t berechnet, zu der die maximal wahrnehmbare Entfernung zu den anderen Agenten bzw. dem Zielobjekt addiert wird. Das Ergebnis wird dann durch die Feldbreite geteilt. Die Belohnungsfunktion ist definiert durch:

$$r_e = \frac{-distance(e,t) + max.\ obs.\ distance}{field\ width} \in [0;0{,}5]$$

Bei $distance(e, t)$ wird der Wert verwendet, den der Agent über seine Beobachtung erhält. Dieser Wert ist im konkreten Szenario auf 30 begrenzt, was auch der „*max. observable distance*“ entspricht. Mit einer Feldbreite von 60 liegt die Belohnung pro Schritt immer im Intervall $[0; 0{,}5]$. $r_e = 0$, falls die Distanz zwischen dem Ziel t und dem Agenten e nach dessen Aktion ≥ 30 ist, wobei $0 \leq distance(e, t) \leq 30$. Dementsprechend ist $r_e = 0{,}5$, falls die Distanz zwischen dem Ziel t und dem Agenten e nach dessen Aktion 0 ist

Weitere Belohnungsfunktionen, die in Betracht gezogen wurden, waren die Berechnung der Belohnung aus der quadrierten Entfernung zum Zielobjekt oder die Verwendung der Änderung der absoluten Entfernung im Vergleich zum vorherigen Schritt. Bei letzterem erhielten die Agenten eine negative Belohnung, wenn ihre letzte Aktion sie weiter vom Zielobjekt entfernt hat, und eine positive Belohnung, wenn sie seit dem letzten Schritt näher gekommen sind. Die Belohnung mit der Differenz der Entfernung beschleunigte zunächst den Lernerfolg, aber die endgültigen Lernergebnisse waren bei den Ansätzen mit absoluter Entfernung als Belohnungsfunktion etwas besser. Es gab jedoch keine größeren Unterschiede. Die Entscheidung, die oben beschriebene Belohnungsfunktion zu verwenden, beruhte daher weitgehend auf ihrer Einfachheit.

Tabelle 5.2: Auswahl der Hyperparameter für DQN.

Parameter	Wert
Lernschritte	1200000
Schritte pro Episode	1000
Neuronales Netzwerk	
Verborgene Schichten	2
Neuronen pro Schicht	64
Aktivierungsfunktion	ReLU
Lernrate	0.003
Exploration	
Funktion	ϵ-Decay
Anfangswert	1.0
Endwert	0.01
Halbwertszeiten	8
Diskontierungsfaktor	0.999
Optimierer	Adam
Replay Buffer Größe	100000
Batch Größe	512

5.4.5 Training

Die Agenten werden mit Hilfe von Deep Q-Learning (DQN) (siehe Abschnitt 2.6.2) trainiert. DQN (und seine Erweiterungen) hat den Vorteil, dass es sich um einen vielseitigen, gut akzeptierten Algorithmus handelt, der auch in Reinforcement Learning Bibliotheken, wie z.B. *TensorForce* (siehe [77]), angeboten wird. Multi-Agent Reinforcement Learning mit Single-Agent-Algorithmen ist ein anerkannter Ansatz, der insbesondere in Anwendungsbereichen eingesetzt werden kann, in denen Garantien der Optimalität von untergeordneter Bedeutung sind (siehe [96]). Dies ist in dem hier untersuchten Schwarmszenario der Fall. Aufgrund der Homogenität der Agenten erscheint es plausibel, dass sie alle nach der gleichen Strategie agieren. Auch in der Natur folgen die Mitglieder eines Schwarms Verhaltensmustern, die sich kaum voneinander unterscheiden. Dies führt zu Formationen, die nach außen hin als eine Einheit erscheinen.

Darüber hinaus hat jeder Agent sein eigenes egoistisches Ziel, so nah wie möglich an der virtuellen Nahrungsquelle zu bleiben. Aus diesem Grund ist das Lernverfahren so gestaltet, dass jeweils nur ein Agent lernt und seine erlernte Strategie dann zur Steuerung aller Agenten eingesetzt wird. Es wird also nur ein einziges neuronales Netz trainiert, von dem alle Agenten ihre Aktionen ableiten. Dieser Ansatz wird *Parameter Sharing* genannt. Der Ansatz, nur einen Agenten einer homogenen Gruppe lernen zu lassen, wurde auch von [29, 60, 61] verwendet, wo er zu guten Lernergebnissen führte.

Tabelle 5.2 zeigt die Hyperparameter, die trotz der nicht-stationären Umgebung, in der sich die Agenten befinden, das stabilste Lernen erzielten. Sie sind das Ergebnis verschiedener Testläufe, für die die Parameterkonfiguration gewählt wurde, die die besten Ergebnisse erbrachte (siehe Simulationen und Ergebnisse). Im Szenario des lernenden Fischschwarms bedeutet das gemeinsame neuronale Netz, dass jede Änderung im Verhalten des lernenden Agenten zu einer Änderung im Verhalten der anderen Agenten führt. Daher muss ein Gleichgewicht zwischen Stabilität und Anpassungsfähigkeit in Bezug auf die sich ändernden Umstände gefunden werden. Aus diesem Grund wurde eine Explorationsrate gewählt, die mit der Zeit abnimmt. In Testläufen hat sich eine exponentielle Abnahme am günstigsten erwiesen.

5.5 Simulationen und Ergebnisse

Zunächst werden im Folgenden die Verhaltensmuster erörtert, die durch das Lernen in dem gegebenen Szenario entstanden sind. Abbildung 5.2 vermittelt einen Eindruck der erlernten Bewegung der Agenten. Die Position des Zielobjekts hat einen signifikanten Einfluss auf das Verhalten der Agenten. Kommt es in das Blickfeld eines Agenten, dreht sich dieser in dessen Richtung und bewegt sich hinter ihm her. Das bedeutet, dass die Bewegungsrichtung der Agenten in diesem Bereich die gleiche ist und die Dichte der Agenten dort auch deutlich höher ist als in anderen Bereichen der Umgebung. Dies allein deu-

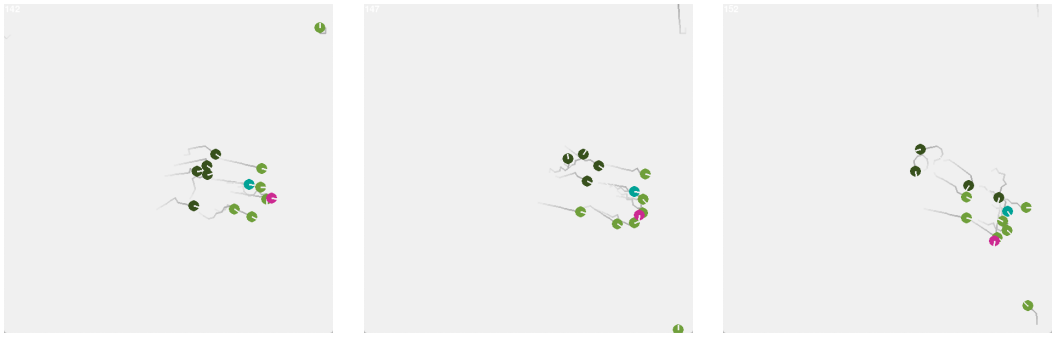


Abbildung 5.2: Typisches Verhalten von Agenten ohne Sichtkontakt zum Zielobjekt in Analogie zur Abbildung 5.1a. Die Agenten werden durch Kreise dargestellt, deren Radius den Grenzen ihres Kollisionsbereichs entspricht. Die Bewegungsrichtung wird durch eine Markierung angezeigt, die von der Mitte des Agenten in die aktuelle Blickrichtung zeigt. Der lernende Agent hat eine türkise Farbe, alle anderen Agenten sind grün gefärbt. Das Zielobjekt wird visuell von den Agenten durch seine rosa Farbe unterschieden. Wenn Agenten den Sichtkontakt zum Zielobjekt verloren haben, sind sie dunkler gefärbt. Darüber hinaus haben die Agenten eine Spur, die ihre Bewegung in den letzten 10 Schritten anzeigt. Diese Spur dient nur zur besseren Darstellung der Zeitkomponente in den Standbildern und hat keinen Einfluss auf den Verlauf des Experiments.

tet auf eine Schwarmformation hin. Es fällt auf, dass die Agenten bestimmte Abstände voneinander einhalten, obwohl Kollisionen nicht per se verhindert werden. Allerdings verlangsamen sie die Geschwindigkeit der Bewegung. Bei einem kurzzeitigen Verlust des Sichtkontakts mit dem Zielobjekt orientieren sich die Agenten an anderen Agenten in ihrer Nachbarschaft. Dies zeigt sich daran, dass sich im Schwarm um das Zielobjekt immer Agenten befinden, die derzeit keine Sicht auf das Zielobjekt haben. Der Schwarmzusammenhalt funktioniert daher in den meisten Fällen recht gut. Es gibt jedoch immer einige Agenten, die den Kontakt zum Schwarm verlieren. Das Verhalten in dieser Situation ist inkonsistent. Meistens bewegen sie sich geradeaus, aber hin und wieder passen sie ihre Bewegungen an einen ihrer Nachbarn an. Wenn die erlernten Strategien auf eine skalierte Version des Szenarios mit mehr Agenten und einem größeren Spielfeld übertragen werden, werden noch mehr Verhaltensweisen sichtbar. Hier bilden die Agenten oft separate Untergruppen, wenn sie vom Hauptschwarm getrennt sind.

Die Position der Nachbaragenten hat einen gewissen Einfluss auf die Entscheidungen des Einzelnen. Wenn sich beispielsweise zu viele Agenten in der Nähe des Zielobjekts befinden, verlieren die dahinter befindlichen Agenten oft den Sichtkontakt zum Zielobjekt. Dennoch gelingt es ihnen in der Regel, den Kontakt mit dem Schwarm über mehrere Schritte hinweg nicht zu verlieren.

5.5.1 Statische regelbasierte Agenten zum Vergleich

Um das erlernte Verhalten besser bewerten zu können, ist es hilfreich, die gesammelten Daten mit den Ergebnissen anderer Verhaltensstrategien zu vergleichen. Zu diesem Zweck werden die folgenden drei regelbasierten Algorithmen betrachtet. Diese wurden neben dem Reinforcement Learning implementiert und dienen dem Vergleich zum erlernten Verhalten bzw. zu dessen Einschätzung.

Simple Wenn das Zielobjekt für einen Agenten sichtbar ist, dreht sich dieser in die Richtung des Ziels, andernfalls bewegt er sich geradeaus in seine aktuelle Blickrichtung.

Boids In Analogie zu den in Abschnitt 5.3.2 vorgestellten Boids-Regeln führt ein Agent Aktionen aus, die aus einer gewichteten Kombination der Prinzipien Ausrichtung, Kohäsion und Separation mit einer zusätzlichen Kraft in Richtung des Zielobjekts berechnet werden.

Random Ein Agent führt zufällige Aktionen aus.

Da der Simple-Algorithmus völlig ohne Informationen über die anderen Agenten in der Umgebung arbeitet, dient er als Bezugsgröße dafür, inwieweit die Agenten ihr Verhalten an den anderen Agenten orientieren. Obwohl Optimierungen dieses regelbasierten Algorithmus noch möglich wären, legen signifikante Verbesserungen gegenüber diesen einfachen Algorithmus nahe, dass lernende Agenten Informationen über die Beobachtung von Nachbarn in ihr Verhalten einbeziehen.

Der Boids-Algorithmus dient als anerkannte Schwarmsimulationsmethode zur Beurteilung, inwieweit das erlernte Verhalten einem natürlichen Schwarm ähnelt. Um die Vergleichbarkeit zu gewährleisten, basiert dieser Algorithmus nur auf den Daten, die auch dem Lernagenten zur Verfügung stehen. Die Kraft, die den Agenten auf das Zielobjekt lenkt, wird daher nur dann in die Berechnung der Bewegungsrichtung einbezogen, wenn sich das Ziel unter den nächsten Nachbarn des Agenten befindet. Diese Kraft wird dann, wie in Abschnitt 5.3.2 beschrieben, mit der Separationskraft kombiniert, die dafür sorgt, dass Abstände zwischen den Agenten eingehalten werden. Die Gewichtung der abstoßenden Kraft hängt quadratisch von der Entfernung zu den Nachbarn ab. Dadurch ergeben sich konstante Abstände zwischen den Agenten, und es kommt zu wenigen Kollisionen.

Wenn kein Sichtkontakt mit dem Ziel besteht, werden die Regeln der Kohäsion, Ausrichtung und Separation angewendet. Die Kraft in Richtung des Ziels wird also durch eine Kombination aus Bewegung in Richtung der mittleren Position aller Nachbarn (Kohäsion) und Ausrichtung auf ihre Sichtlinie (Ausrichtung) ersetzt. Die Kohäsion wird höher gewichtet als die Ausrichtung der Blickrichtung. Die Separationskraft ist ebenfalls quadratisch, aber insgesamt

ist sie stärker als zuvor. Das bedeutet, dass die Abstände zwischen den Agenten größer werden. Dies wirkt sich negativ auf die Schwarmkohäsion aus, hat aber den Vorteil, dass das Zielobjekt nach dem Verlust des Sichtkontakts mit größerer Wahrscheinlichkeit in den Sichtbereich eines benachbarten Agenten zurückkehrt. Die Mitglieder getrennter Untergruppen können sich so leichter wieder dem Hauptschwarm anschließen. Die genaue Gewichtung der einzelnen Kräfte kann der Tabelle 5.3 entnommen werden.

	Ziel	Kohäsion	Ausrichtung	Separation
Sichtkontakt	20	0	0	$(\sum_{i=1}^{n} d_i^2)/300$
Kein Sichtkontakt	0	15	5	$(\sum_{i=1}^{n} d_i^2)/50$

Tabelle 5.3: Die ausgewählten Gewichte der Kräfte im Boids-Algorithmus mit $d_i \in [0, 0.5]$ als beobachtete Entfernung des Agenten zu seinem Nachbarn i und $n = 8$ als Anzahl der wahrnehmbaren Nachbarn.

5.5.2 Erfolg bei der Verfolgung des Ziels

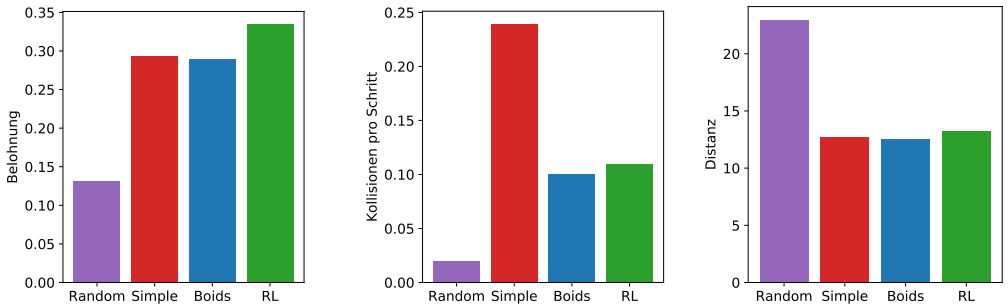
Bei allen untersuchten Algorithmen, mit Ausnahme des Zufallsalgorithmus, ist erkennbar, dass die Agenten versuchen, das Zielobjekt zu verfolgen. Sie unterscheiden sich jedoch in der Art und Weise, wie sich die Agenten in verschiedenen Situationen verhalten, und wie der resultierende Schwarm strukturiert ist. Die Schwarmkohäsion ist im Boids-Algorithmus am stabilsten. Dies bedeutet jedoch nicht unbedingt, dass die Verfolgung des Zielobjekts hier besonders gut funktioniert. Um die verschiedenen Strategien unter diesem Aspekt zu vergleichen, kann man sich die durchschnittliche Belohnung ansehen, der durch die Verfolgung der verschiedenen Strategien erzielt werden konnte. Die Belohnung, welche die Agenten erhalten, hängt linear von ihrer Entfernung zum Zielobjekt ab. Daher können hohe Belohnungen als direkter Indikator für die erfolgreiche Verfolgung des Zielobjekts verwendet werden. Die genauen Werte sind in Abbildung 5.3a dargestellt. Dort ist zu sehen, dass alle Algorithmen deutlich bessere Ergebnisse erzielen als zufälliges Verhalten. Die höchsten Werte werden jedoch durch das mittels Reinforcement Learning erlernte Verhalten erzielt. Daran schließen sich die beiden anderen Strategien an, die trotz ihrer sehr unterschiedlichen Ansätze in dieser Hinsicht etwa gleich gut sind.

5.5.3 Abstände und Kollisionen zwischen Agenten

Einer der Gründe für die höhere Belohnung des Reinforcement Learners ist, dass die Agenten gelernt haben, Kollisionen mit ihren Nachbarn zu vermeiden. Die Häufigkeit der Kollisionen für die verschiedenen Algorithmen ist in Abbildung 5.3b dargestellt. Kollisionen führen dazu, dass sich der „auffahrende“

Agent im nächsten Schritt mit halber Geschwindigkeit weiterbewegt. Häufige Kollisionen erschweren daher die Verfolgung des Zielobjekts erheblich, da die durchschnittliche Geschwindigkeit des Agenten nicht mehr ausreicht, um Schritt zu halten. Verhaltensweisen, bei denen die Agenten ihren Abstand zueinander nicht einhalten, sind daher viel weniger erfolgreich. Bei der „Simple“ Strategie führt die mangelnde Koordination zwischen den Agenten dazu, dass die Agenten beim Verfolgen des Zielobjekts eine Reihe hinter dem Zielobjekt bilden. Da sie sich etwas schneller bewegen als das Zielobjekt, kollidieren die Agenten oft, was sie verlangsamt. Dann verlieren sie den Sichtkontakt mit dem Ziel. Meistens führen diese Situationen dazu, dass die Agenten den Kontakt mit dem Schwarm vollständig verlieren. Dies ist einer der Hauptgründe, warum der Simple-Algorithmus insgesamt weniger Belohnung erhält als der mittels Reinforcement Learning trainierte Agent. Aufgrund der abstoßenden Kraft im Boids-Algorithmus ist die Anzahl der Kollisionen bei dieser Strategie deutlich geringer. Das erlernte Verhalten hat in diesem Bereich sehr ähnliche Werte. Dies zeigt, dass auch eine Form der Kollisionsvermeidung gelernt wurde. Allerdings ist der Boids-Algorithmus bei der Kollisionsvermeidung noch erfolgreicher.

Vergleicht man den durchschnittlichen Abstand zwischen allen Agenten für die verschiedenen Verhaltensstrategien (Abbildung 5.3c), so scheinen diese zunächst ähnliche Abstände zu haben. Ein genauerer Blick auf die Häufigkeitsverteilung des Abstands zum nächsten Nachbarn, die in der Abbildung 5.4 dargestellt ist, zeigt jedoch deutliche Unterschiede: Die Abstände zum nächsten Agenten, die beim Boids-Algorithmus um den Wert 3 relativ konstant bleiben, haben bei den anderen Verhaltensweisen eine größere Varianz. Darüber hinaus ist zu erkennen, dass der Simple-Algorithmus auch einige Werte unter 2 aufweist. Da die Agenten selbst einen Radius von 1 haben, treten Kollisionen in Schritten auf, bei denen der Abstand zum nächsten Nachbarn weniger als 2 beträgt. Bei Agenten, die durch Reinforcement Learning trainiert wurden, werden im Kollisionsbereich kaum Abstände aufgezeichnet. Das beobachtete



(a) Durchschnittliche Belohnungen

(b) Kollisionen

(c) Durchschnittlicher Abstand zwischen allen Agenten

Abbildung 5.3: Vergleich der Ergebnisse der verschiedenen Algorithmen.

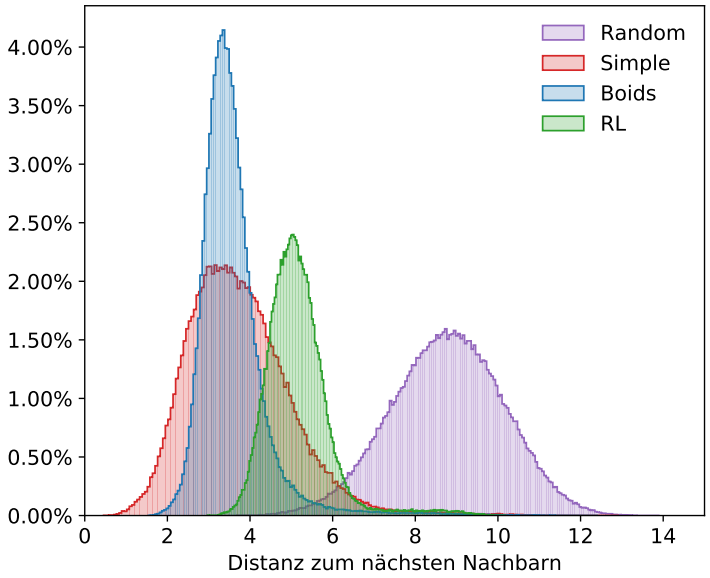


Abbildung 5.4: Entfernung zum nächsten Agenten.

Verhalten der Agenten zur Vermeidung von Kollisionen spiegelt sich daher auch in diesem Diagramm wider.

5.5.4 Verlust des Sichtkontakts mit dem Ziel

Eine weitere Möglichkeit, die Informationen über die anderen Agenten zu nutzen, besteht darin, die Position des Zielobjekts aus ihrer Bewegungsrichtung abzuleiten. Wie in Abbildung 5.2 dargestellt, wurde dieses Verhalten von den Agenten mittels Reinforcement Learning gelernt. Die Agenten, die ihr Verhalten durch Reinforcement Learning gelernt haben, verlieren den Sichtkontakt mit dem Zielobjekt nur in vergleichsweise großen Entfernungen und auch den Kontakt mit dem Rest des Schwarms verlieren sie seltener. Dies ist in der Abbildung 5.5 dargestellt. Sie zeigt die Entfernung der Agenten in den Schritten nachdem sie den Sichtkontakt zum Zielobjekt verloren haben. Die Entfernungen wurden in den nächsten Schritten 20 gemessen, nachdem das Zielobjekt den Wahrnehmungsbereichs eines Agenten verlassen hatte, unabhängig davon, ob der Sichtkontakt in den folgenden Schritten wiederhergestellt wurde oder nicht. Mit dem erlernten Verhalten nimmt die durchschnittliche Entfernung zum Ziel nach dem Verlust des Sichtkontakts nach von etwa 11 Schritten wieder ab. Das bedeutet, dass sich die Agenten von diesem Zeitpunkt an in der Regel wieder näher an das Zielobjekt herantasten. Die Agenten der beiden anderen Algorithmen hingegen entfernen sich im Laufe der Zeit in der Regel immer weiter vom Zielobjekt. Beim Boids-Algorithmus ist die durchschnittliche Entfernung, ab der sie den Sichtkontakt verlieren, etwas größer als beim Simple Algorithmus, aber keiner von beiden ist in der Lage, sich dem Zielobjekt wieder zu nähern und verliert somit den Kontakt zum Schwarm.

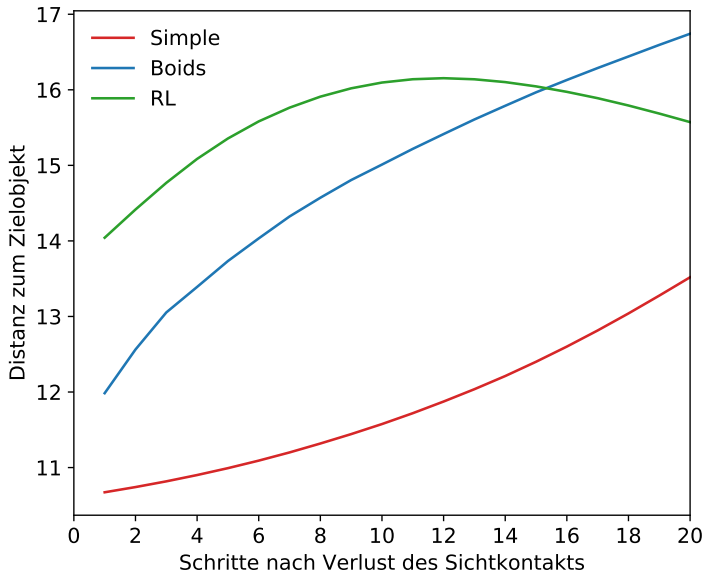


Abbildung 5.5: Entwicklung der Entfernung zum Zielobjekt nach Verlust des Sichtkontakts.

Es gibt verschiedene Gründe, warum der Simple- und der Boids-Algorithmus weniger erfolgreich sind als das erlernte Verhalten. Beim Simple-Algorithmus werden die Bewegungsrichtungen und Positionen der anderen Agenten nicht in die Entscheidungen einbezogen. Im Falle des Sichtkontaktverlustes bewegt sich der Agent geradeaus. Dies ist auch einer der Gründe, warum nur die nächsten 20 Schritte in Betracht gezogen wurden, da die Umgebung torusförmig ist, könnten die Agenten schließlich wieder auf das Ziel zulaufen. Da das Zielobjekt jedoch in der Zwischenzeit in der Regel die Bewegungsrichtung geändert hat, führt die geradlinige Bewegung des Simple Algorithmus nicht dazu, dass der Sichtkontakt wiederhergestellt wird.

Das umgekehrte Problem tritt beim Boids-Algorithmus auf. Hier ist das Verhalten stark an das der Nachbarn angepasst. Es kommt daher fast nie vor, dass einzelne Agenten vom Schwarm getrennt werden. Stattdessen werden immer Gruppen von mehreren Agenten gleichzeitig getrennt, die dann zusammenbleiben und dem Hauptschwarm in einiger Entfernung folgen. Dies geschieht, weil zumeist einer der abgetrennten Agenten noch weitere Agenten in seinem Wahrnehmungsbereich hat, die dem Ziel folgen. Das Zentrum der als Gruppe abgetrennten Agenten, das im Boids-Algorithmus die Bewegungsrichtung bestimmt, verschiebt sich langsam in Richtung des Hauptschwarms, welcher wiederum dem Ziel folgt. In der Regel reicht dies jedoch nicht aus, um die beiden Gruppen wieder zusammenzuführen. Die Parameter des Boids-Algorithmus wurden so gewählt, dass die Separation nach Zielverlust höher gewichtet wird, um die Wahrscheinlichkeit zu erhöhen, dass das Zielobjekt nach Verlust des Kontakts wieder in den Wahrnehmungsradius eintritt. Das

erlernte Verhalten mittels Reinforcement Learning ist hier jedoch erfolgreicher. Der Reinforcement Learner ist nicht auf eine feste Gewichtung der Positionen und Blickrichtungen der Nachbarn beschränkt, sondern kann diese je nach Situation anpassen.

5.5.5 Einfluss von Nachbarn

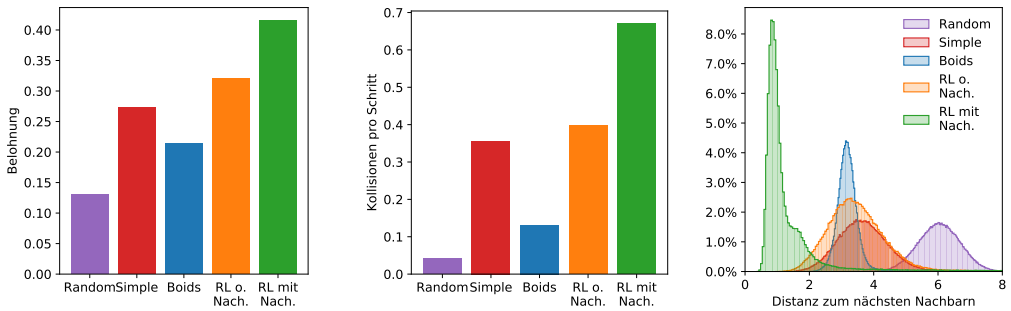
In diesem Abschnitt soll mit einem weiteren Experiment nachgewiesen werden, dass die Agenten mittels Reinforcement Learning gelernt haben sich an ihren Nachbarn zu orientieren, um das Ziel zu finden auch wenn es sich nicht in ihrem Sichtbereich befindet. Deshalb werden in diesem Abschnitt Agenten ohne Informationen über ihre Nachbarn auf die Verfolgung des Ziels trainiert, um ihren Erfolg mit den bisherigen Agenten zu vergleichen, die über diese Informationen verfügen. Da Kollisionen zwischen den Agenten Nachteile mit sich bringen und sich ohne Informationen über benachbarte Agenten nicht vermeiden lassen, wurde das Szenario zunächst so geändert, dass Kollisionen zwischen den Agenten nicht mehr behandelt werden. Das bedeutet, dass sie sich überlappen können, ohne ihre Bewegungsgeschwindigkeit zu verringern. Strategien, die die Positionen der einen Agenten umgebenden Nachbarn nicht berücksichtigen, werden nicht automatisch benachteiligt. In diesem modifizierten Szenario ist es daher möglich, die erlernten Strategien direkt mit und ohne Informationen über die umgebenden Agenten zu vergleichen. Insbesondere können die positiven Auswirkungen der Berücksichtigung von Nachbarschaftspositionen aufgezeigt werden. Für den Vergleich wurden zwei Gruppen von Agenten trainiert, von denen sich eine im ursprünglichen Szenario befindet, in dem ein Agent über Daten zu seinen 8 nächsten Nachbarn verfügt. Die andere Gruppe hingegen hat nur Informationen über die eigene Blickrichtung und die Position des Zielobjekts. Obwohl diese Agenten ihre Nachbarn nicht sehen können, ist das Zielobjekt für sie nur dann sichtbar, wenn es sich unter den 8 nächsten Nachbarn befindet. Dadurch wird sichergestellt, dass die beiden Gruppen vergleichbar bleiben. Für eine bessere Übertragbarkeit der Hyperparameter des neuronalen Netzes wurden die Dimensionen der Beobachtung bei dieser Variante nicht verändert. Die Zeilen der Matrix, die ursprünglich für die Positionen der anderen Agenten reserviert waren, sind mit (-1)-Einträgen gefüllt, um einen Mangel an Information darzustellen. Dies wird also genauso gehandhabt wie die Situation, in der sich das Zielobjekt außerhalb des Wahrnehmungsbereichs des Agenten befindet.

Beim Vergleich der Belohnungen in Abbildung 5.6a, die von den Agenten mit und ohne Kenntnis ihrer Nachbarn erreicht werden, wird deutlich, dass die Orientierung am Nachbarn einen Vorteil bringt. Die Belohnungen der entsprechenden Agenten sind am höchsten. Auch die Anzahl der potentiellen Kollisionen zwischen den Agenten (die gezählt, aber nicht bestraft wurden) ist viel höher (Abbildung 5.6b), da sie keinen Nachteil durch die Nähe zu ihren Nachbarn haben. Der Abstand zum nächsten benachbarten Agenten (Abbildung

5.6c) ist ebenfalls am geringsten.

Das Verhalten, das die Agenten lernen, die ihre Nachbarn nicht wahrnehmen können, ist im Wesentlichen dasselbe wie beim Simple Algorithmus. Das bedeutet, dass die Agenten dem Zielobjekt folgen, wenn sie es sehen. In Situationen, in denen sich das Ziel außerhalb ihres Wahrnehmungsbereichs befindet, bewegen sie sich geradeaus. Dabei berücksichtigen sie auch die Tatsache, dass sich das Zielobjekt bewegt, d.h. sie steuern einen Punkt an, der etwas vor der aktuellen Position des Zielobjekts liegt. Das bedeutet, dass sie bei der Verfolgung des Zielobjekts sogar etwas erfolgreicher sind als die Agenten, die durch den Simple Algorithmus gesteuert werden.

Die explizit in den Boids-Algorithmus integrierte Kollisionsvermeidung wirkt sich negativ auf den Erfolg dieses Algorithmus in Bezug auf die erhaltene Belohnung aus. Dementsprechend erzielt diese Strategie die schlechtesten Ergebnisse in Bezug auf die Belohnungen nach dem Zufallsagenten, wie die Abbildung 5.6a zeigt.



(a) Durchschnittliche Belohnungen

(b) Kollisionen (unbehandelt)

(c) Entfernung zum nächsten Agenten

Abbildung 5.6: Vergleich von trainierten Agenten mit und ohne Kenntnis der Bewegungen ihrer Nachbarn mit den regelbasierten Algorithmen in einer Umgebung ohne Kollisionsbehandlung.

5.6 Diskussion und Zusammenfassung

In diesem Kapitel wurde ein Szenario vorgestellt, in dem Agenten in einer kontinuierlichen, zweidimensionalen, teilweise beobachtbaren Umgebung mittels Multi-Agent Reinforcement Learning trainiert wurden, mit dem Ziel, ihre Entfernung zu einem sich bewegenden Ziel zu minimieren. Jeder Agent hat ein Eigeninteresse daran, das Ziel zu erreichen und es wurde gezeigt, dass unter den Agenten Schwarmverhalten auftritt, obwohl sie durch die Belohnung im Reinforcement Learning keinen expliziten Anreiz zur Zusammenarbeit erhalten. Sie haben gelernt, sich in Situationen, in denen sie das Zielobjekt aus den Augen verloren haben, aneinander zu orientieren, um aufzuholen, ohne explizit auf dieses Verhalten trainiert worden zu sein. Darüber hinaus haben sie in einer

Umgebung, in der Kollisionen implizit durch das Abbremsen der kollidierenden Agenten bestraft werden, gelernt, einen gewissen Abstand zueinander zu halten. Dieses Verhalten ist analog zu den von Reynolds bei Tieren beschriebenen Boids-Regeln. Deshalb wurde das erlernte Verhalten mit der direkten Umsetzung der Boids-Regeln verglichen, die ähnliche Messungen in Bezug auf Kollisionen und Abstände zwischen den Agenten zeigten. Trotz der Abstraktheit der Boids-Regeln führen sie zu einem ähnlichen Verhalten wie das hier vorgenommene zielorientierte Training. Auf diese Weise generiert das Kapitel wertvolle Erkenntnisse über die Entstehung von Schwarmverhalten, ohne dieses künstlich zu erzwingen. Zu diesen Erkenntnissen gehört auch eine mögliche Absicht hinter der Schwarmbildung: Das Schwarmverhalten hier ergibt sich aus einer vorgegebenen Zielvorgabe für die Agenten. Wenn es mit dem von Boids übereinstimmt, kann argumentiert werden, dass Boids-Schwarmverhalten, wie es in der Natur beobachtet wird, sich entwickelt haben könnte, um ähnliche Ziele zu erreichen wie die, die den hier trainierten Agenten gegeben waren. Die Fähigkeit, diese Art von Schwarmverhalten aus reinem Eigeninteresse zu erzeugen, kann beispielsweise Anwendungen in Evakuierungsszenarien haben, in denen sich Menschen kollektiv zu einem Notausgang bewegen, oder in Rettungsszenarien, in denen mehrere Einheiten Opfer finden und retten müssen.

6 Schwarmbildung durch Reinforcement Learning

Nachdem im vorangegangenen Kapitel Reinforcement Learning verwendet wurde, um eine Vielzahl von Agenten darauf zu trainieren einem Zielpunkt zu folgen und das erlernte Verhalten dabei gewisse Schwarmeigenschaften zeigte, wird das Szenario in diesem Kapitel herum gedreht. Die Agenten werden darauf trainiert, einem sich auf sie zu bewegenden Angreifer auszuweichen. Genauer werden sie für jeden Zeitschritt den sie überleben belohnt, da eine Kollision mit dem Angreifer ihr Dasein beendet. Man kann also bei dem Angreifer auch von einem intelligenten Hindernis sprechen, wobei er zusätzlich die Eigenschaft besitzt, dass er von multiplen Zielen in seiner Umgebung abgelenkt wird. Dies hat zur Folge, dass die Agenten einen Schwarm bilden obwohl sie eigentlich egoistisch versuchen ihr Leben zu verlängern. Dieses Schwarmverhalten wird daraufhin im Sinne der Spieltheorie auf Nash-Gleichgewichte untersucht, womit sich eine Beziehung zu menschlichem Verhalten bzw. Verhalten, wie es in der Ökonomie an den Tag gelegt wird, herstellen lässt. Der Hintergrund zur Spieltheorie, Nash-Gleichgewichten und verwandten sozialen Dilemmata in Multiagentensystemen wird im Abschnitt 6.3 vermittelt, nach den Vorveröffentlichungen in Abschnitt 6.1 und der Motivation in Abschnitt 6.2. Der Ansatz und das genaue Szenario werden in Abschnitt 6.4 erläutert. In Abschnitt 6.5 wird das Schwarmverhalten detailliert, unter anderem auf vorhandene Nash-Gleichgewichte, untersucht. Abschnitt 6.6 fast das Kapitel zusammen.

6.1 Vorveröffentlichungen

In diesem Kapitel werden unter anderem Erkenntnisse zu der Fragestellung präsentiert, ob egoistische Agenten, die mittels Reinforcement Learning darauf trainiert werden im Beisein eines Räubers möglichst lange zu überleben, zu ihrem Selbstschutz emergent einen Schwarm bilden würden. Die positive Beantwortung dieser Frage wurde bereits in Veröffentlichung [60] publiziert, wobei die Fragestellung selbst von Thomy Phan während einer Unterhaltung unter Kollegen aufgeworfen wurde. Die Ausarbeitung des Konzepts zur Untersuchung der Fragestellung, die Implementierung und die Evaluation wurden vom Autor alleinig durchgeführt, wobei Thomy Phan, Thomas Gabor und Lenz Belzner beratend zur Seite standen und zur anschließenden Veröffentlichung beitrugen, an der aber auch der Autor den Hauptteil der Arbeit übernahm. Zudem sind in

dieses Kapitel die Inhalte der Publikation [59] eingeflossen. Diese schloss sich direkt an die vorangegangene Publikation an und enthält weitere Experimente, die das emergente Schwarmverhalten der eigentlich egoistisch handelnden Agenten untersucht. Die Fragestellung nach Nash-Gleichgewichten in Schwärmen kam bei der Diskussion der Ergebnisse der vorangegangenen Publikation auf. Daraufhin wurden hauptsächlich vom Autor das bestehende Setting als Normalformspiel im Sinne der Spieltheorie formuliert und entsprechende Experimente aufgesetzt. Auch die Erstellung der Veröffentlichung wurde federführend von ihm übernommen. Der Anteil der übrigen Koautoren wurde bereits in Abschnitt 1.3 erläutert. Die Tabelle 6.1 sowie die Abbildungen 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10 und 6.11 waren bereits in [60] enthalten. Aus [59] stammen die Tabelle 6.2 und die Abbildungen 6.12, 6.13 und 6.14.

6.2 Motivation

Nachdem im vorangegangen Kapitel untersucht wurde, wie eine Gruppe von Agenten mittels Reinforcement Learning gesteuert werden kann und dabei Analogien zu natürlichem Schwarmverhalten festgestellt wurden, soll in diesem Kapitel Schwarmverhalten in Multiagentensystemen weiter das Verhalten der Agenten untereinander untersucht und analysiert werden.

In der Natur ist bei vielen Arten Schwarmverhalten zu beobachtet. Ein bekanntes Beispiel sind Fischschwärme, bei der mehrere Fische Aktionen gemeinsam koordinieren und unter anderem aus sozialen Gründen nahe beieinander bleiben. Das bedeutet, dass ein einzelner Fisch seine Richtung in Bezug auf ihm nahe gelegene Fische ausrichtet, wobei ein gewisser Zusammenhalt gehalten wird und trotzdem Kollisionen mit anderen Individuen vermieden werden.

Das Schwarmverhalten existiert jedoch nicht nur als Selbstzweck. In der Natur profitiert ein Schwarmfisch in mehrfacher Hinsicht vom Schwarmverhalten: Der Schwarm erhöht die Effizienz der Fortbewegung im Wasser oder die Paarungschancen. Der Schwarm steigert zudem den Erfolg bei der Nahrungssuche, da die gemeinschaftliche Wahrnehmung der eines einzelnen Individuums überlegen ist. Dasselbe gilt für die Erkennung von Raubfischen. Darüber hinaus sinkt die Wahrscheinlichkeit, gefangen zu werden, für ein Individuum im Hinblick auf bestimmte Raubtierverhaltensweisen.

Craig Reynolds zeigt in [121], dass die algorithmische Umsetzung von drei Regeln der Ausrichtung, Kohäsion und Separation zu natürlich wirkendem Schwarmverhalten führt, während ein Individuum (von ihm als Boid bezeichnet) nur lokales Wissen über Nachbarn in seiner lokalen Umgebung benötigt. Um diese statischen Schwarmregeln zu überwinden, wurde in [102] Reinforcement Learning dafür verwendet, ein Individuum darauf zu trainieren die oben genannten Regeln zu befolgen, um so einen Schwarm zu bilden. Dies geschah, indem die Belohnungsfunktion entsprechend der Entfernungen zwischen den Individuen geformt wurde und ihre Handlungen darauf beschränkt wurden, sich einem anderen Individuum anzunähern bzw. davon zu entfernen oder sich

parallel bzw. in die entgegengesetzte Richtung zu bewegen.

In diesem Kapitel wird der Fall untersucht, dass ein Individuum (unter vielen) sein Verhalten auf das Ziel optimiert unter Anwesenheit eines Räubers so lange wie möglich zu überleben, wobei sich der Räuber von mehreren Beutetieren in seiner Umgebung verwirren/ablenken lässt. Es wird gezeigt, dass dieses einfache Ziel in einem Multi-Agent Reinforcement Learning Setting zu emergenten Schwarmverhalten (ähnlich wie bei Boids) führt, ohne dass das Schwarmverhalten explizit erzwungen werden muss. Der Ansatz wird angelehnt an das egoistische Verhalten und der Inspiration an Fischschwärmen im Folgenden als *SELFish* bezeichnet. Dabei wird das Reinforcement Learning lediglich als ein Werkzeug zur Untersuchung der Optimalität des Verhaltens verwenden, nicht exakt analog zur Natur (wo Schwärme gemeinhin genetisch bedingt sind nicht durch individuelles Training entstehen).

Zudem werden in diesem Kapitel Nash-Gleichgewichte in den sich durch Reinforcement Learning ergebenden Multi-Agenten-Schwärmen untersucht. Experimente zeigen, dass das teilweise beobachtbare Szenario skaliert werden und die gelernte Verhaltensstrategie ohne weiteres Training in diesem veränderten Szenarien einsetzbar ist, ohne dass die Performance der Strategie erkennbar einbricht. Zur weiteren Untersuchung wird das Kriterium für die Homogenität der Agenten gelockert, d.h. es gibt Agenten, die die Schwarmbildung verfolgen und andere, die ihre Nachbarn nicht in ihre Entscheidung einbeziehen. Dabei zeigt sich, dass manche Schwarmkonfigurationen unter bestimmten Bedingungen Nash-Gleichgewichte bilden und, dass selbst wenn Schwärmen in einer bestimmten Situation nicht die strikt überlegene Strategie sein mag, ein Abweichen vom Schwarmverhalten für einen einzelnen Agenten noch schlimmer ist. Die Entscheidung, zu Schwärmen oder vom Schwarm abzuweichen, stellt deshalb ein soziales Dilemma dar.

6.3 Hintergrund und verwandte Arbeiten

Wie im vorangegangen Kapitel kommt auch in diesem Kapitel Multi-Agent Reinforcement Learning mit der in Abschnitt 5.3.1 gegebenen Definition zur Anwendung.

In MARL muss jeder Agent e eine lokale Verhaltensstrategie π_e finden, die optimal auf die Verhaltensstrategien der anderen Agenten abgestimmt ist. Wenn die anderen Agenten ihr Verhalten ändern, muss sich auch Agent e anpassen, da seine bisherige Strategie möglicherweise suboptimal geworden ist. Der einfachste Ansatz für MARL ist die Verwendung von Reinforcement Learning Algorithmen für Einzelagenten, wie Q-Learning, und deren Skalierung auf mehrere Agenten [135, 87]. In homogenen Umgebungen können Verhaltensstrategien gemeinsam genutzt werden, indem man effektiv nur eine lokale Strategie π_e lernt und diese auf alle Agenten repliziert. Dies kann den Lernprozess beschleunigen, da Erfahrungen unterschiedlicher Agenten während des Trainings ausgetauscht bzw. gemeinsam genutzt werden können [135, 33]. Während es

viele andere Ansätze für MARL in Spielen gibt, die globale Informationen in den Trainingsprozess einbeziehen [33, 94, 34, 120], konzentriert sich diese Arbeit auf die Anwendung von Single-Agent Reinforcement Learning Algorithmen auf Spiele mittels gemeinsamer Nutzung einer Verhaltensstrategie (Policy-Sharing).

6.3.1 Schwarmverhalten

Auch dieses Kapitel baut auf den im vorherigen Kapitel (siehe 5.3.2) beschriebenen Ansätzen zur algorithmischen Erzeugung von Schwarmverhalten auf. Dabei handelt es sich insbesondere um die drei von Craig Reynolds 1987 beschriebenen Schwarmregeln [121] und das Training von Agenten durch Multi-Agent Reinforcement Learning auf die Einhaltung dieser Regel durch Morihiro et al. [102]. Morihiro et al. führen in ihrem Szenario einen Räuber ein und belohnen die Agenten dafür, einen gewissen Abstand zu diesem zu halten. Allerdings, während die zuvor erwähnten Ansätze zu einem Schwarmverhalten führen, vernachlässigen sie die vorteilhaften Eigenschaften, die das Schwarmverhalten für die Individuen haben kann. Einer dieser Vorteile könnte eine erhöhte Überlebenswahrscheinlichkeit in Gegenwart eines Räubers sein, da diese durch die schiere Menge möglicher Ziele abgelenkt werden könnten. Es stellt sich die Frage, ob Schwarmverhalten mittels Reinforcement Learning in einem Szenario mit solchen Eigenschaften auftritt, in dem die Agenten lediglich versuchen, ihre Überlebenszeit zu maximieren. Im Gegensatz zu Morihiro et al. [102] wird in diesem Kapitel ein als *SELFish* bezeichneter Ansatz betrachtet, in welchem die Agenten mittels Reinforcement Learning ausschließlich auf das Ziel des Überlebens trainiert werden, ohne explizit Schwarmverhalten zu erzwingen. Zudem wird bei SELFish unter anderem ein kontinuierlicher Aktionsraum der Agenten untersucht, im Unterschied zu Morihiro et al. [102].

6.3.2 Nash-Gleichgewichte

Die Spieltheorie betrachtet strategische Interaktionen innerhalb einer Gruppe von Individuen. Dabei beeinflussen die Handlungen jedes Einzelnen das Ergebnis, und die Individuen sind sich dieser Tatsache bewusst. Darüber hinaus werden die teilnehmenden Individuen als rational betrachtet. Dies bedeutet, dass sie klar definierte Ziele innerhalb der möglichen Ergebnisse der Interaktionen haben und dass sie die beste verfügbare Strategie zur Verfolgung ihrer Ziele anwenden. In der Regel sind die Spielregeln und die Rationalität allen bekannt.

Grundsätzlich gibt es zwei verschiedene Formen der Repräsentation: Man spricht von der Normalform eines Spiels, wenn die Spieler gleichzeitig eine Strategie wählen, ohne die Entscheidungen der anderen zu kennen. Demgegenüber steht die Extensivform eines Spiels, bei der einige Spieler wissen, was andere Spieler während des Spiels getan haben. In vielen Situationen ist keine

Kommunikation zwischen den Spielern möglich oder erwünscht, weshalb im Folgenden nur die Normalform eines Spiels (auch strategisches Spiels genannt) beschrieben wird. Ein Normalformspiel $G = (N,\{A_i\}_{i\in N}, \{u_i\}_{i\in N})$ besteht aus einer Menge von Spielern N, einer Menge von Aktionen A_i für jeden Spieler i und einer Auszahlungsfunktion $u_i : A \to R$ für jeden Spieler i. Das Aktionsprofil $a = (a_1, ..., a_n)$ ist ein Tupel von Aktionen, eine für jeden Spieler, auch Strategieprofil genannt. a_{-i} ist ein Strategieprofil ohne die Aktion des Spielers i. Alle möglichen Tupel von Aktionen werden auch als Raum von Aktionsprofilen $A = (a_1,...,a_n) : a_i \in A_i, i = 1,...,n$ bezeichnet.

Ein Nash-Gleichgewicht (NE) ist ein Strategieprofil, so dass jede Strategie eine *beste Antwort* auf alle anderen Strategien ist. Eine beste Antwort ist die Reaktion auf eine Aktion, die die Auszahlung maximiert.

Formal ausgedrückt ist ein Ergebnis $a^* = (a_1^*,...,a_n^*)$ ein Nash-Gleichgewicht, wenn für jeden Spieler i folgendes gilt: $u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*)\ \forall a_i \in A_i$.

Die interessanteste Eigenschaft von NEs ist, dass sie selbsterhaltend sind: kein Spieler hat einen Anreiz, einseitig von seiner gewählten Strategie abzuweichen.

Zu erwähnen ist noch die kooperative Spieltheorie. Bei der kooperativen Spieltheorie handelt es sich um einen Ansatz, der von den Aktionen einzelner Agenten abstrahiert. Stattdessen wird die Formung von Koalitionen aus multiplen Agenten untersucht, welche die Erreichung der Ziele der einzelnen Agenten bzw. eines gemeinsamen Ziels unterstützen [18]. Die Bildung von Koalitionen beruht auf der Kommunikation bzw. auf dem Einhalten von bindenden Absprachen zwischen den Agenten [16]. Die Untersuchung von Schwarmbildung mittels kooperativer Spieltheorie ist interessant, weil der Schwarm als große Koalition aus den Agenten angesehen werden kann. Allerdings steht die Untersuchung von MARL mittels kooperativer Spieltheorie noch am Anfang und Arbeiten dazu sind eher theoretischer Natur [129] oder es wird gerade eben auf MARL verwiesen, um die Limitationen von nichtkooperativer Spieltheorie zu überwinden und das Verhalten von Gruppen von Agenten vorherzusagen [113]. Dass MARL im Hinblick auf kooperative Spieltheorie sehr wenig untersucht ist dürfte auch damit zusammenhängen, dass die Kommunikation bzw. das Verhandeln lernender Agenten selbst Gegenstand aktiver Forschungen sind [33, 128].

Deshalb soll im Folgenden das durch MARL erzeugte Schwarmverhalten in Analogie zu verwandten Arbeiten mittels nichtkooperativer Spieltheorie untersucht werden [81, 87, 104, 13, 20]. Im Hinblick auf die Schwarmumgebung in diesem Kapitel entsprechen die selbst-interessierten Agenten damit den nichtkooperativen Spielern eines Normalformspiels, was sich auch darin begründet, dass die Agenten nicht aktiv miteinander kommunizieren oder verhandeln. $\text{SELFish}_{\text{DQN}}$ und TurnAway, die im Abschnitt 6.5 erklärt werden, entsprechen den Strategien aus denen einen Spieler respektive ein Agent wählen kann. Die Belohnung eines Agenten, die mit der Überlebenszeit eines Individuums korreliert, entspricht der Auszahlung eines bestimmten Strategieprofils. Mit der

Definition einer solchen Schwarmumgebung als Normalformspiel ist es möglich NEs mit einem gewöhnlichem Löser für Nash-Gleichgewichte zu finden. Dazu wurde im Folgenden der populäre Gambit Solver [97] verwendet.

Özgüler und Yıldız [108] untersuchten ebenfalls Nash-Gleichgewichte in Multi-Agenten-Schwärmen. Dazu modellierten sie den Prozess der Futtersuche durch mehrere Agenten als ein nichtkooperatives N-Spieler-Spiel. Sie nahmen an, dass jeder Agent seinen individuellen Gesamtaufwand in einem Zeitintervall durch die Kontrolle seiner Geschwindigkeit minimieren will. Indem sie eine nichtlineare Differentialgleichung in Bezug auf die Positionen der Agenten aufstellen und diese Gleichung lösen, zeigen sie, dass das Spiel ein Nash-Gleichgewicht hat.

6.3.3 Soziale Dilemmata in Multi-Agenten-Systemen

In vielen Multi-Agenten-Systemen müssen die Agenten zusammenarbeiten, um ihren eigenen Nutzen zu maximieren. In [23] analysieren die Autoren soziale Dilemmata in Multiagentensystemen, in denen Reinforcement Learning Algorithmen mit Nash-Gleichgewichten konfrontiert werden, die kurzfristig eine rationale optimale Lösung bilden, aber bei wiederholter Interaktion nicht optimal sind. Sie schlagen heuristische Prinzipien zur Verbesserung der Kooperation und zur Überwindung solcher Kurzschluss-Nash-Gleichgewichtsstrategien vor. Ihre Experimente beschränken sich jedoch auf ein Gefangenendilemma-Szenario mit drei Agenten und vier Aktionen. In jüngerer Zeit haben Leibo et al. gezeigt, dass sich das erlernte Verhalten von Agenten in Multiagentensystemen in Abhängigkeit von Umweltfaktoren verändert [87]. Sie zeigen experimentell, wie Konflikte aus dem Wettbewerb um gemeinsame Ressourcen entstehen können und wie die sequentielle Natur sozialer Dilemmata der realen Welt die Zusammenarbeit beeinflusst. Während die Komplexität ihrer Umgebung mit der Umgebung in diesem Kapitel vergleichbar ist, beschränken sich ihre Experimente auf zwei Agenten. Nichtsdestotrotz haben die in diesem Kapitel vermuteten und weiter untersuchten Nash-Gleichgewichte viele Merkmale mit den von Leibo et al. [87] eingeführten sequentiellen sozialen Dilemmata gemeinsam. Das erlernte Schwarmverhalten ist eine Strategie, die sich über mehrere Aktionen erstreckt, und es hat sich experimentell gezeigt, dass es nicht die optimale Strategie für das Kollektiv der Agenten ist. Da das Schwarmverhalten jedoch eindeutig irgendeine Form der Koordination erfordert, wird die Analyse, ob das Schwärmen als Defekt im Sinne von [87] angesehen werden kann oder nicht, für weitere Analysen offen gelassen.

6.4 Emergentes Schwarmverhalten durch Reinforcement Learning

Um zu untersuchen, ob das Ziel, in der Gegenwart eines Räubers zu überleben, zu einem Schwarmverhalten aus Eigeninteresse führt, wurde ein Modell erstellt, das ein solches Verhalten ermöglicht bzw. begünstigt. Das Simulationsmodell ist dem aus dem letzten Kapitel ähnlich. Im Folgenden sollen die Eigenschaften der Umgebung noch einmal erläutert und es soll auf Unterschiede eingegangen werden. Es folgt eine Beschreibung des Aktions- und Beobachtungsraumes sowie der Belohnungsstruktur, die zum Training der Agenten verwendet wurde.

6.4.1 Domäne

Die Agenten, die in diesem Szenario die Beute sind, können sich frei in einem kontinuierlichen zweidimensionalen Raum bewegen, der als ein Quadrat mit vordefinierten Kantenlängen visualisiert wird (siehe Abbildung 6.1). Ein Agent selbst wird als Kreis mit einer Fläche repräsentiert, die wesentlich kleiner ist als der Raum, in dem er sich bewegt.

In der Umgebung gibt es keinerlei Hindernisse, denn auch die Agenten kollidieren nicht miteinander. Um die freie Bewegung der Agenten zu ermöglichen, hat der Raum auch hier die Eigenschaft eines Torus. Wenn Agenten oder der Räuber die Feldgrenzen überschreiten, tauchen sie unmittelbar auf der gegenüberliegenden Seite wieder auf. Die Position eines Agenten im Raum wird durch dessen x- und y-Koordinaten beschrieben.

Zusammen mit den Agenten befindet sich ein Räuber in der Umwelt. Der

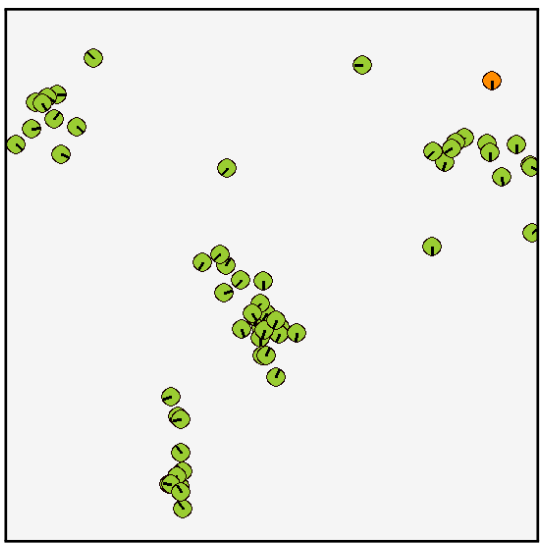


Abbildung 6.1: Beispiel der Umgebung mit 60 Agenten (grün) und einem Räuber (orange).

Räuber wird ebenfalls als Kreis repräsentiert. Das Ziel des Räubers ist es, die Agenten zu fangen, indem es sich zu ihrer Position bewegt. Sobald der Räuber mit einem Agenten kollidiert, wird die Ausführung des betreffenden Agenten beendet und sofort ein neuer Agent an einer zufälligen Position erzeugt, um die Anzahl der Agenten im System konstant zu halten. Eine Kollision zwischen einem Agent und dem Räuber geschieht dann, wenn ihr Abstand geringer als die Summe ihrer Radien ist. Befinden sich mehrere Agenten in einer bestimmten Entfernung um den Räuber herum, wählt er zufällig einen als Ziel und folgt diesem (andernfalls bewegt er sich in die Richtung des nächstgelegenen Agenten). Das bedeutet, dass der Räuber durch mehrere Agenten in seiner Nähe abgelenkt werden kann. Daher kann es für einen Agenten vorteilhaft sein, sich auf andere Agenten zuzubewegen, da es den Räuber ablenkt, was für Schwarmverhalten wesentlich ist. Um jedoch zu verhindern, dass der Räuber sein Ziel ständig wechselt, verfolgt er ein ausgewähltes Ziel eine gewisse Zeit lang, bevor ein neues Ziel unter den in der Nähe befindlichen Agenten ausgewählt wird. Standardmäßig bewegen sich die Agenten und der Räuber mit der gleichen Geschwindigkeit. Dies wurde gewählt, um Agenten zu ermöglichen in die Situation zu kommen, dass der Räuber von ihrer Fährte abgelenkt wird, ohne dass sie direkt gefangen werden. Es würde den Agenten allerdings ermöglichen, sich in die entgegengesetzte Richtung des Räubers zu drehen und sich wegzubewegen, ohne dass dieser eine Chance hat, sie einzuholen (aufgrund der Torus Eigenschaft). Aus diesem Grund beschleunigt der Räuber gelegentlich für kurze Zeit, was einen Sprung nach vorne simuliert, um die Beute, der er folgt, einzuholen. Die Verhaltensstrategie des Räubers ist statisch und ändert sich im Laufe der Zeit nicht.

6.4.2 Ziel eines Agenten

Das Ziel eines Agenten ist es, nicht mit dem Räuber zu kollidieren. Dafür erhält er eine Belohnung von $+\mathbf{1}$ für jeden Zeitschritt, den er überlebt, und **-1000** für die Kollision mit dem Räuber, was sein Leben beendet. Mit dieser Belohnungsstruktur kann das Ziel eines Agenten als „so lange wie möglich zu überleben“ angesehen werden. Da es keine Hindernisse in der Umgebung gibt und die Agenten nicht miteinander kollidieren, gibt es keine anderen Belohnungen/Strafen.

6.4.3 Aktionsraum

Der Aktionsraum der Agenten besteht nur aus dem Winkel, den sie sich in jedem Zeitschritt drehen wollen. Die Bewegungsgeschwindigkeit der Agenten ist konstant und kann von ihnen nicht verändert werden.

Die Aktion a, die den Drehwinkel darstellt, der vom Agenten aus diskreten Schritten oder aus einem kontinuierlichen Intervall heraus gewählt werden kann, hängt vom später verwendeten Reinforcement Learning Algorithmus ab.

Der Agent kann im Falle von DQN (siehe Abschnitt 2.6.2) als Aktion einen von fünf diskreten Gradwerten $\{-90°, -45°, 0°, +45°, +90°\}$ wählen. Im Falle von DDPG kann jeder beliebige reellwertige Drehwinkel gewählt werden.

Der Räuber kann nur begrenzt reellwertige Drehungen $\{x \in \mathbb{R} \mid -45° \leq x \leq 45°\}$ bei jedem Schritt durchführen. Die hat das Ziel den Agenten eine höhere Manövrierfähigkeit zu geben als dem Räuber.

6.4.4 Beobachtungsraum

Wie im vorangegangenen Kapitel ist das Setting nur partiell beobachtbar und hat die gleichen Eigenschaften, wie in Abschnitt 5.4.3 beschrieben. Um die Skalierbarkeit auf viele autonome Agenten zu erleichtern, kann ein Agent nicht den vollständigen Zustand der Umgebung beobachten; stattdessen beschränkt sich seine Beobachtung auf ihn selbst, den Räuber und die n nächstgelegenen Nachbaragenten. Dies steht zum einen im Einklang mit verwandten Arbeiten, z.B. Boids, bei denen ebenfalls nur die lokalen Informationen über die nächsten Nachbarn benötigt werden. Es lässt sich aber auch biologisch erklären, wenn z.B. ein Fisch in einem Schwarm nicht den gesamten Schwarm, sondern nur seine lokalen Nachbarn beobachten kann. Zudem erleichtert es die Berechnung und hat die schöne Eigenschaft, dass der Beobachtungsvektor, der durch den Reinforcement Learning Algorithmus weitergeleitet wird, um eine Aktion zu erhalten, eine konstante Länge hat. Im Unterschied zu Abschnitt 5.4.3 befinden sich die Werte bezüglich des Räubers in der ersten Zeile der Beobachtungsmatrix. Zudem gibt es keinen festen Beobachtungsradius, sondern die Beobachtungsmatrix enthält immer die Beobachtungen für den Räuber, den Agenten selbst und die n nächstgelegenen Nachbaragenten, nach ihrer Entfernung geordnet.

6.4.5 Training

Wie bereits erwähnt, besteht ein valider Weg für das Training mehrerer homogener Agenten mittels Reinforcement Learning darin, nur einen Agenten zu trainieren und dann die erlernte Strategie auf alle Instanzen der homogenen Gruppe zu kopieren ([29]). Dies ähnelt auch der Natur, wo z.B. mehrere Schwarmfische der gleichen Verhaltensstrategie folgen.

Zum Zweck des Trainings wurden die DQN- und DDPG-Implementierungen von Keras-RL [116] verwendet. Keras-RL wurde ursprünglich für OpenAI Gym-Umgebungen ([14]) entwickelt, in denen nur einzelne Agenten mit diesen Umgebungen durch eine *step(action)*-Methode interagieren, der eine Aktion übergeben wird und welche eine Beobachtung, eine Belohnung und einen booleschen Statusindikator zurückgibt, der anzeigt, ob die aktuelle Episode beendet ist. Diese Schnittstelle wurde in der hier vorgeschlagenen Schwarmumgebung verwendet, um einen einzelnen Agenten darauf zu trainieren, dem anwesenden Räuber mit den zuvor erwähnten Belohnungen, Aktions- und Beobachtungs-

räumen auszuweichen. Während des Trainings eines Agenten sind auch die anderen Agenten anwesend, auf die die Verhaltensstrategie (d.h. das hinter den Reinforcement Learning Algorithmen liegende neuronale Netz) des lernenden Agenten nach jeder Episode kopiert wird. Eine Episode endet, wenn der lernende Agent vom Räuber gefangen wird oder 10.000 Zeitschritte ausgeführt wurden.

Während des Trainings betrug die Kantenlänge des Raumes 40×40 Pixel, wobei er an den Kanten jeweils mit der gegenüberliegenden Seite verbunden ist (Torus). Zu beachten ist, dass die Agenten und der Räuber bei jedem reellen Wert im Intervall $[0, 40]$ positioniert werden können. Die Werte in den 3-Tupeln der Beobachtung wurden jedoch auf [0,1] normalisiert. Die Agenten und der Räuber wurden durch Kreise mit einem Radius von 1 repräsentiert, wobei ein Agent als gefangen gilt, wenn die Entfernung seiner Position zur Position des Räubers unter 2 liegt. Während des Trainings waren 10 Agenten anwesend.

Mit den Hyperparametern des Reinforcement Learnings wird das Verhalten des Reinforcement Learning Algorithmus gesteuert (siehe [45]). Ein Beispiel hierfür wäre der Parameter γ, der beeinflusst, wie weitsichtig ein Agent agiert, d.h. ob er eher auf kurzfristig zu sammelnde Belohnungen setzt oder mehr Gewicht auf die potentielle Belohnung legt, die sich erst nach längeren Aktionsfolgen ergibt. Die Hyperparameter müssen vor dem eigentlich Trainingsprozess festgelegt werden und werden nicht von diesem bestimmt/optimiert wie es z.B. bei den Kantengewichten (auch Parameter genannt) der verwendeten künstlichen neuronalen Netze der Fall ist. Es gibt einen eigenen Forschungszweig, der sich mit der Optimierung von Hyperparametern beschäftigt (als weitere Ebene zum Optimierungsprozess, der durch das Reinforcement Learning betrieben wird). Eine relativ einfache Methode zur Hyperparameteroptimierung ist die Rastersuche (Grid Search). Dabei werden die relevanten Hyperparameter jeweils in eine Menge von diskreten Werten unterteilt und Modelle für alle Kombinationen evaluiert.

Dies geschah auch in diesem Fall, um eine gute Konfiguration für die Parameter der Reinforcement Learning Algorithmen zu finden. Die Qualität der Parameterkonfiguration eines Trainingslaufs wurde während der Testphase auf der Grundlage der kumulativen Belohnung bewertet, die der lernende Agent erzielen konnte und die im Wesentlichen der Anzahl der von ihm überlebten Zeitschritte entspricht. Dazu wurde das Verhalten nach dem Training über 1.000 Episoden getestet. Wie bereits beschrieben, endet eine Episode, wenn der lernende Agent vom Räuber gefangen wird oder er 10.000 Schritte (Aktionen) absolviert hat. Die Güte einer Parameterkonfiguration wurde rein an der Anzahl an Schritten (ausgeführten Aktionen) gemessen, die der lernende Agent insgesamt über die 1.000 Episoden durchführen konnte. Auch die Anzahl der beobachtbaren Nachbaragenten wurde als Parameter variiert, was als beste Konfiguration zu 4 beobachtbaren Nachbarn für DQN und einen Nachbarn für DDPG führte. Die Tabelle 6.1 zeigt die besten gefundenen Parameterkonfigurationen.

Hyperparameter	DQN	DDPG
Trainingsschritte	500.000	500.000
Verborgene Schichten	10	5
Neuronen pro Schicht	16	Actor: 16 Critic: 32
Aktivierungsfunktion verborgene Schichten	ReLU	ReLU
Aktivierungsfunktion Ausgabeschicht	linear	linear
γ	0,999999	0,999999
Optimierer	Adam	Adam
Lernrate	0,001	0,001
Replay Buffer Größe	50.000	100.000
Batch Größe	64	512
Exploration	ϵ-Greedy $\epsilon = 0{,}1$	Ornstein Uhlenbeck $\theta = 0{,}15$ $\mu = 0{,}0$ $\sigma = 0{,}3$
Beobachtbare Nachbarn	4	1

Tabelle 6.1: Hyperparameter für das Reinforcement Learning

Selbst bei der relativ geringen Anzahl von Agenten, die während des Trainings anwesend waren, konnte ein Schwarmverhalten beobachtet werden, wenn das erlernte Verhalten eines Agenten auf die anderen übertragen wurde. Da die Beobachtung eines Agenten partiell und somit auf das 3-Tupel $(dist_{neighbor_i}, direction_{neighbor_i}, orientation_{neighbor_i})$ für die n nächsten Nachbarn beschränkt ist, kann die Anzahl der Agenten sowie die Größe des Raumes erhöht werden, ohne die gelernte Verhaltensstrategie zu invalidieren. Dadurch lässt sich ein noch besseres Schwarmverhalten beobachten, welches im nächsten Abschnitt weiter evaluiert werden soll.

6.5 Simulationen und Ergebnisse

In diesem Abschnitt sollen die Ergebnisse des Reinforcement Learnings und das resultierende Schwarmverhaltens diskutiert werden. Da Boids der erste Algorithmus war, der für die algorithmische Simulation von Schwarmverhalten vorgeschlagen wurde, werden die Schwärme resultierend vom Multi-Agent Reinforcement Learning mit denen von Boids verglichen.

Zunächst soll ein Eindruck von den resultierenden Schwärmen vermittelt werden. Die Abbildung 6.2 zeigt die Bildung eines Schwarms in den ersten 40 Zeitschritten einer Testepisode von $SELFish_{DQN}$. Bei einem kontinuierlichen Aktionsraum zeigt $SELFish_{DDPG}$ ein ähnliches Verhalten, obwohl der Schwarm tendenziell dichter ist. Es kann angenommen werden, dass der Schwarm sich

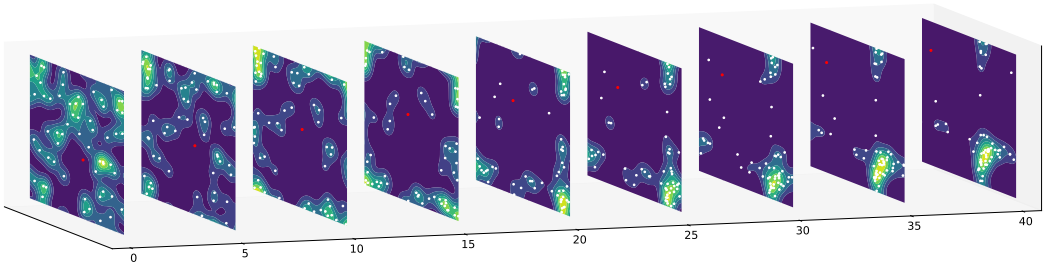


Abbildung 6.2: Schwarmformation in den ersten 40 Bildern einer Episode von $\text{SELFish}_{\text{DQN}}$. Agenten (weiß) und Räuber (rot) wurden zufällig initialisiert. Es kam eine Kerndichteschätzung [115] zum Einsatz, um die dichten Regionen des Multi-Agenten-Schwarms hervorzuheben. Bei der Visualisierung ist zu beachten, dass der Raum an den Rändern umschlägt bzw. die Kanten mit der gegenüberliegenden Seite verbunden sind.

bildet weil der lernende Agent entdeckt, dass das Raubtier von ihm abgelenkt werden kann, wenn er in der Nähe anderer Agenten bleibt, was sein Leben verlängert und damit seine akkumulierte Belohnung erhöht.

Boids fordert die Ausrichtung, Kohäsion und die Separation benachbarter Agenten. Dies kann durch Vektorberechnungen ausgedrückt werden, zusammen mit Gewichten, die diese drei Regeln in einen Kontext setzen. Um das Szenario dem Setting des Reinforcement Learnings ähnlicher zu machen, wurde eine weitere Kraft hinzugefügt, welche die Boids vom Räuber wegdrückt (zusammen mit einem Gewicht für dieses Verhalten, das es in einen Kontext zu den anderen Regeln setzt). Um eine gute Konfiguration für das Gewicht der Ausrichtung, Kohäsion, Separation und Raubtiervermeidung zu finden, wurden mehrere Läufe mit unterschiedlichen Parametereinstellungen durchgeführt. Auch hier wurde die Qualität einer Einstellung auf der Grundlage der Anzahl an Zeitschritten bewertet, die ein bestimmter Boid überleben konnte (genau wie der lernende Agent beim Reinforcement Learning).

Wenn es nur um das Überleben eines Agenten geht, könnte man sich vorstellen, dass sich ein Agent unabhängig von den ihn umgebenden anderen Agenten in die entgegengesetzte Richtung des Räubers dreht und sich von ihm weg bewegt. Diese Strategie wird im Folgenden als *TurnAway* bezeichnet und zum Vergleich angegeben.

6.5.1 Ausrichtung und Kohäsion

Weil der Boids Algorithmus die Ausrichtung und den Zusammenhalt der Agenten explizit forciert, sollen die Schwärme, die sich aus der Vermeidung des Räubers mittels Reinforcement Learning ergeben, mit Boids mittels dieser Metriken verglichen werden.

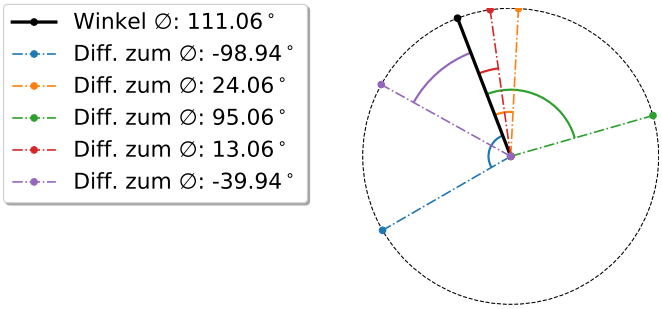


Abbildung 6.3: Betrachtet man die Orientierung von fünf Agenten im Raum, so kann ein mittlerer Winkel (schwarz) und die Abweichung davon in $(-180°, 180°]$ berechnet werden ([143]).

Da die globale Ausrichtung eines Agenten als Winkel in $[0°, 360°)$ gemessen wird (die Ausrichtung nach Osten entspricht 0°), kann die Ausrichtung der Agenten als Abweichung von einem mittleren Winkel innerhalb einer Gruppe gemessen werden (siehe Abbildung 6.3). Die absolute Abweichung der einzelnen Agenten von diesem mittleren Winkel wurde summiert und über die Anzahl der Agenten gemittelt.

Um die Kohäsion des Schwarms zu messen, wurde der durchschnittliche Abstand zwischen den Agenten berechnet. Hierfür wurde der Abstand zwischen allen Agenten i und j summiert (was der Summe über die obere Dreiecksmatrix der Distanzen entspricht) und durch die Anzahl der Agentenpaare gemittelt.

$$\begin{bmatrix} & dist_{1,2} & dist_{1,3} & \dots & dist_{1,n} \\ & & dist_{2,3} & \dots & dist_{2,n} \\ & & & \ddots & \vdots \\ & 0 & & & dist_{n-1,n} \\ & & & & \end{bmatrix}$$

Wenn man bedenkt, dass die Agenten vor dem Räuber fliehen und der Raum sich an den Rändern „umschlägt“, konnten sich, wie bereits aus den Abbildungen 6.1 und 6.2 ersichtlich, mehrere Gruppen mit unterschiedlicher Ausrichtung bilden, je nach ihrer Position in Bezug auf den Räuber. Aus diesem Grund erschien es nicht sinnvoll, Ausrichtung und Kohäsion über alle Agenten im Raum zu berechnen. Um dem entgegenzuwirken, wurde im Vorfeld das dichtebasierte Clustering-Verfahren DBSCAN [32] bzw. insbesondere dessen Scikit-Learn-Implementierung [112] verwendet und die durchschnittliche Abweichung vom mittleren Winkel und der mittlere Abstand zwischen den Agenten nur für Agenten in einem bestimmten Cluster berechnet (siehe Abbildung 6.4 für ein Beispiel-Clustering der Agentenpositionen). Die Messungen über alle Agenten werden zum Vergleich angegeben.

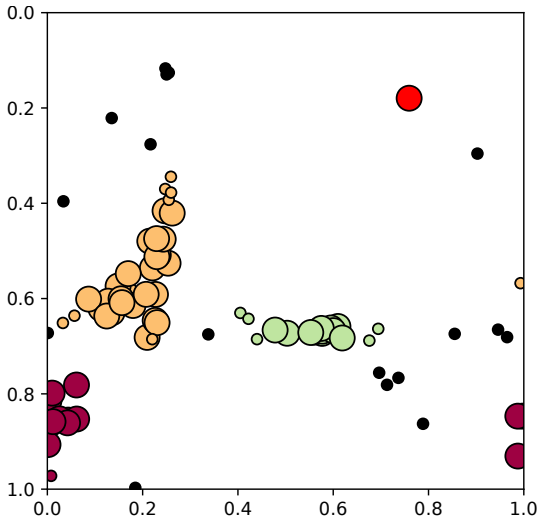


Abbildung 6.4: Beispiel-Clustering für $\text{SELFish}_{\text{DQN}}$ mit 40 Agenten (Räuber als roter Kreis) unter Verwendung des DBSCAN-Algorithmus [32]. Es wurden drei Cluster an Agenten gefunden (Orange, Grün, Bordeauxrot). Die als schwarze Punkte eingezeichneten Agenten wurden vom DBSCAN-Algorithmus als Rauschpunkte erkannt, da sie keinem der Cluster nahe genug sind, um diesen zugeordnet werden zu können.

Abbildung 6.5 zeigt die Anzahl der Agenten in einem bestimmten Cluster, wenn 40 Agenten in einem Raum von 40×40 Pixel anwesend waren. Es ist sichtbar, dass die TurnAway-Strategie im Durchschnitt viele Rauschpunkte erzeugt. Als Rauschpunkte werden im Zusammenhang mit dem DBSCAN-

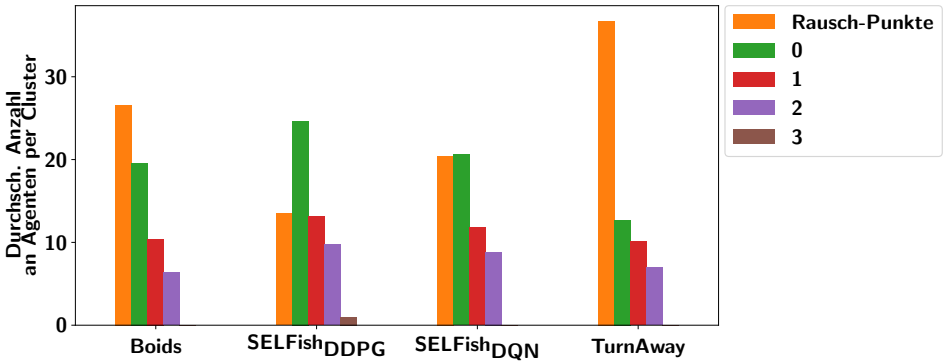


Abbildung 6.5: Durchschnittliche Anzahl von Agenten in einem jeweiligen Cluster (Cluster-ID angegeben), wobei Rauschpunkte Agenten sind, die keinem bestimmten Cluster zugeordnet werden konnten.

Verfahren Ausreißer bezeichnet, die nicht direkt einem Cluster zugeordnet werden können, weil sie an keinem dicht genug sind. Die Cluster, die für TurnAway gefunden werden, sind hauptsächlich darauf zurückzuführen, dass sich die Agenten in die gleiche Richtung bewegen, um dem Räuber auszuweichen. Boids und die beiden Reinforcement Learning Ansätze, die bei SELFish verwendet werden, DQN und DDPG, erzeugen im Durchschnitt ziemlich ähnliche Cluster-Anzahlen und -Größen, wobei DDPG die Tendenz hat, einen großen Cluster zu bilden.

Betrachtet man die durchschnittliche Abweichung vom mittleren Orientierungswinkel der Agenten innerhalb der Cluster (siehe Abbildung 6.6) kann man sehen, dass Boids die am stärksten ausgerichteten Gruppen von Agenten erzeugt, die sich im Allgemeinen in die gleiche Richtung bewegen. $\text{SELFish}_{\text{DQN}}$ und $\text{SELFish}_{\text{DDPG}}$ weichen stärker ab, vermutlich weil Agenten, die mit diesen Reinforcement Learning Algorithmen gelernte Verhaltensstrategien verfolgen, zu einer Art Zittern neigen. Auch zeigen diese Agenten das Verhalten, eine Linie an der „Grenze“ zu bilden, an der sie sich aufgrund der Torus-Eigenschaft der Umgebung wieder auf den Räuber zubewegen würden (vergleichbar mit den roten Linien in Abbildung 5.1b).

An diesen Linien zirkulieren die Agenten, bis sich der Räuber in ihre Richtung bewegt. Bei TurnAway werden ohnehin nur Gruppen von Agenten als Cluster erfasst, die sich in die gleiche Richtung bewegen, wobei die durchschnittliche Orientierungsabweichung durch Agenten verzerrt wird, die von der anderen Seite des Raumes kommen und sich in die entgegengesetzte Richtung bewegen. Man könnte die Frage stellen, ob die für $\text{SELFish}_{\text{DQN}}$ oder $\text{SELFish}_{\text{DDPG}}$ gefundenen Schwärme (bzw. Cluster) auch allein daraus resultieren, dass die Agenten gelernt haben, sich vom Räuber abzuwenden und sich dadurch in die gleiche Richtung zu bewegen. Dem kann die Beobachtung entgegengesetzt werden, dass, wenn der Räuber an einer festen Position festgesetzt wird (er kann nicht vollständig entfernt werden, da es Teil der Beobachtung der

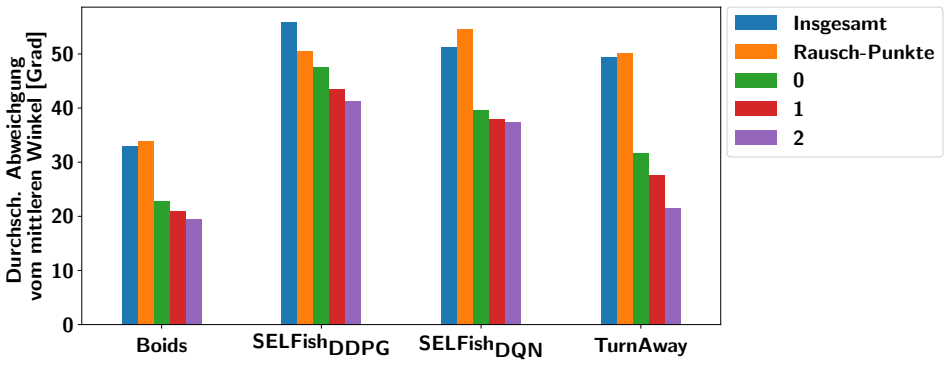


Abbildung 6.6: Durchschnittliche Abweichung vom mittleren Orientierungswinkel der Agenten über die Cluster hinweg.

Agenten ist), die trainierten Agenten immer noch einen Schwarm in größtmöglicher Entfernung vom Räuber bilden, in dem sie umeinander zirkulieren. Abbildung 6.7 zeigt den durchschnittlichen paarweisen Abstand zwischen Agenten entweder innerhalb von Clustern, zwischen Rauschpunkten oder zwischen allen Agenten in der Umgebung. Dieser Abstand ist über alle vier Agentenverhaltensstrategien hinweg relativ homogen, wobei nur $\text{SELFish}_{\text{DDPG}}$ dazu tendiert, etwas dichtere Agentengruppen zu erzeugen. Die Homogenität zwischen den Verhaltensstrategien in Bezug auf den durchschnittlichen paarweisen Abstand hängt allerdings auch mit der Abstands-/dichtebasierten Clusterbildung von DBSCAN zusammen.

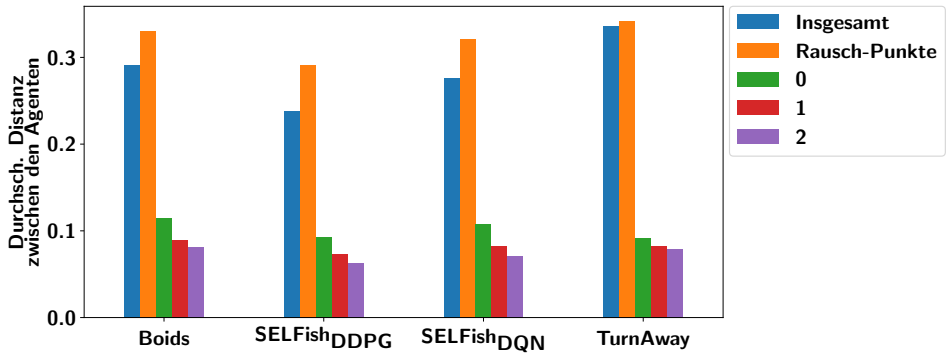


Abbildung 6.7: Durchschnittlicher paarweiser Abstand zwischen Agenten entweder innerhalb von Clustern, zwischen Rauschpunkten oder über alle Agenten hinweg. Kantenlängen des Raumes sind für die Abstandsberechnung auf 1 normiert.

6.5.2 Überleben der Agenten

Für die Reinforcement Learning-Algorithmen wurde die Belohnung so definiert, dass der einzelne Lernende als Belohnung für jeden überlebten Zeitschritt $+1$ und -1000 für das Gefangenwerden durch den Räuber erhielt. Die Maximierung der kumulativen Belohnung soll ihn dazu ermutigen, so lange wie möglich zu überleben. Nach dem Ende einer Episode, die damit endete, dass der lernende Agent gefangen wurde oder 10.000 Zeitschritte durchlaufen wurden, wurde die gelernte Verhaltensstrategie auf alle anderen Agenten kopiert. Aufgrund der teilweisen Beobachtbarkeit und der Normalisierung der Beobachtung kann die erlernte Strategie des Falles mit 10 Agenten auch in Szenarien mit anderen Parametern hinsichtlich der Anzahl der anwesenden Agenten oder der Größe der verfügbaren Fläche angewendet werden. Abbildung 6.8 zeigt die mittlere Episodenlänge für die verschiedenen Richtlinien, die im Wesentlichen der mittleren akkumulierten Belohnung der Lernenden entspricht. Für die statischen Verhaltensstrategien, Boids und TurnAway, entspricht sie der Zeit, die verging, bis ein bestimmter vordefinierter Agent gefangen wurde.

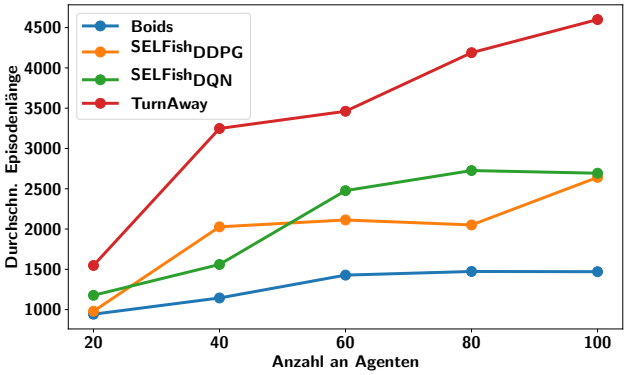


Abbildung 6.8: Durchschnittliche Episodenlänge für jede der Verhaltensstrategien mit unterschiedlicher Anzahl an Agenten in der Umgebung.

Abbildung 6.9 zeigt für jede der vier Verhaltensstrategien die absolute Anzahl der gefangenen Agenten in einer Episode geteilt durch die Länge der Episode (reduziert um eine transiente Phase von 100 Zeitschritten für die Schwarmbildung). Diese Messungen wurden dann wiederum über mehrere Episoden und Läufe (mit unterschiedlichen Seeds) gemittelt. Es ist zu beachten, dass, obwohl die Anzahl der Agenten in der Umgebung variiert wird, der Parameter für Boids oder die Verhaltensstrategien für SELFish$_{DQN/DDPG}$ immer noch diejenigen sind, die in kleineren Settings mit nur 10 Agenten ermittelt wurden.

Es ist in den Abbildungen 6.8 und 6.9 zu sehen, dass die Leistung der Verhaltensstrategie trotz der Tatsache, dass die Umgebungseinstellungen geändert werden, nicht zusammenbricht. Die Zunahme der durchschnittlichen Episodenlänge in Abbildung 6.8 bei einer höheren Anzahl von Agenten ist unter anderem

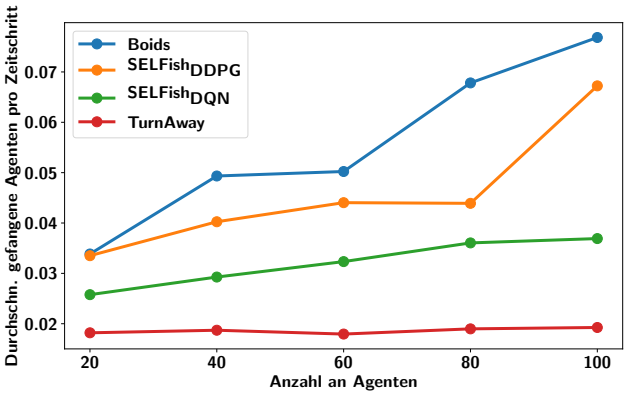


Abbildung 6.9: Durchschnittliche Anzahl an Agenten, die pro Zeitschritt gefangen wurden. Gemessen für die verschiedenen Verhaltensstrategien und mit variierender Agentenanzahl in der Umgebung.

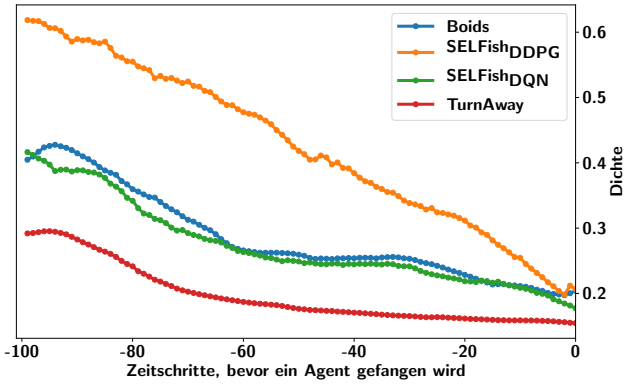


Abbildung 6.10: Die mittels Kerndichteschätzung [115] (vgl. Abbildung 6.2 und 6.11) ermittelte Dichte einer Region, in der sich ein Agent befindet in den letzten 100 Zeitschritten bevor er gefangen wird (Mittelwert über viele gefangene Agenten).

auf den Umstand zurückzuführen, dass mit mehr Agenten in der Umgebung die Wahrscheinlichkeit abnimmt, dass ein bestimmter Agent gefangen wird.

Um genauer zu verstehen wie das Fangen der Agenten durch den Räuber abläuft, wird diese in Abbildung 6.11 dargestellt. Wenn sich der Räuber in Richtung des Schwarms bewegt, bewegt dieser sich kollaborativ weg, wobei einige wenige Agenten zurückbleiben. Die Gemeinschaft der Agenten wird immer kleiner und kleiner, bis einer von ihnen abgetrennt wird. Wenn ein Agent separiert ist, besteht kaum eine Möglichkeit, dass der Räuber von anderen Agenten abgelenkt wird und Agent wird zur Beute. Dies wird deutlich, wenn man die Dichte der Agenten in den Zeitschritten bevor sie gefangen werden betrachtet. Abbildung 6.10 zeigt die Dichte der Region eines Agenten, ermittelt mit der Kerndichteschätzung [115] (vgl. Abbildung 6.2 und 6.11) in den letzten 100-Zeitschritten seines Lebens.

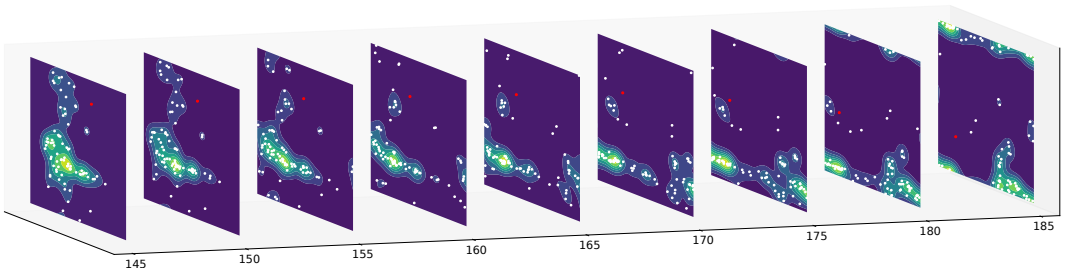


Abbildung 6.11: Separation der Agenten (weiße Punkte) vom Schwarm, bevor sie vom Räuber (roter Punkt) gefangen werden.

6.5.3 Nash-Gleichgewichte in Schwärmen

Wie bereits erwähnt, befanden sich während des Trainings 10 homogene Agenten auf einer Fläche von 40×40 Pixeln, wobei nur ein Agent tatsächlich mittels Reinforcement Learning trainiert wurde und seine Verhaltensstrategie nach jeder Episode auf alle anderen Agenten übertragen wurde. Eine Episode endete, wenn der lernende Agent vom Räuber gefangen wurde (d.h. mit ihm kollidierte) oder 10.000-Schritte ausgeführt wurden. Die Belohnung war so strukturiert, dass der Lernende ermutigt wurde, möglichst lange am Leben zu bleiben. Der Räuber hat die Eigenschaft, dass er von mehreren Beuteagenten in seiner Nähe abgelenkt werden kann, indem er sich zufällig für einen von ihnen entscheidet und diesem folgt. Deshalb lernten die Agenten Schwärme zu bilden, um ihre Überlebenschancen zu erhöhen. Eine weitere fest kodierte Strategie, bei der sich ein Agent immer in die entgegengesetzte Richtung des Räubers drehte und floh, ohne auf das Verhalten anderer Agenten zu achten, wurde unter anderem zum Vergleich eingesetzt (*TurnAway*). Abbildung 6.8 zeigte, dass die mit 10 Agenten auf einer 40×40 Pixel Fläche erlernte Strategie in Settings mit mehr Agenten eingesetzt werden kann, ohne zu versagen bzw. in der Performance einzubrechen. Darüber hinaus zeigte sich aber auch, dass die statische Strategie des Abwendens in Bezug auf die Überlebenszeit besser abschneidet als die erlernte Verhaltensstrategie.

Um dieses Phänomen genauer zu untersuchen, wurde ein weiteres Experiment durchgeführt, bei welche nur Agenten anwesend waren, die entweder der erlernten $\text{SELFish}_{\text{DQN}}$-Verhaltensstrategie (trainiert mit 10 homogenen Agenten auf 40×40 Pixeln) oder der statischen TurnAway-Strategie folgten. Die Ergebnisse sind in Abbildung 6.12 dargestellt.

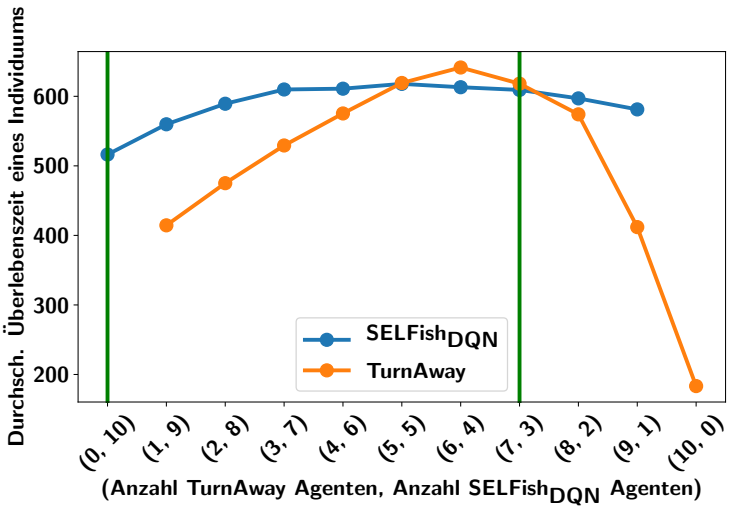


Abbildung 6.12: Überlebenszeit von $\text{SELFish}_{\text{DQN}}$ (trainiert mit 10 Agenten auf 40×40 Pixel) gegenüber TurnAway auf einer Fläche von 40×40 Pixeln. Nash-Gleichgewichte mit einer grünen Linie markiert.

Sie zeigt einen Aufbau, bei dem 10 Agenten in einem Bereich von 40×40 Pixeln anwesend sind. Die Agenten, die der SELFish$_{\text{DQN}}$-Strategie folgen neigen dazu, Schwärme zu bilden, da sie gelernt haben, dass dies ihre Überlebenschancen erhöhen könnte (angesichts der Eigenschaft eines ablenkbaren Räubers), während TurnAway-Agenten sich nicht um andere kümmern (außer um den Räuber). Die Abszisse veranschaulicht das Mengenverhältnis der Agenten des jeweiligen Typs. Zum Beispiel bedeutet der Eintrag (6,4) auf der Abszisse, dass es in der Umgebung 6 Agenten gibt, die TurnAway ausführen, und 4 Agenten, die SELFish$_{\text{DQN}}$ ausführen. Auf der Ordinate ist die Überlebenszeit (in Zeitschritten) der Agenten angegeben, die entweder SELFish$_{\text{DQN}}$ oder TurnAway ausführen (gemittelt über Individuen innerhalb jeder Verhaltensstrategie). Das Experiment wurde über 10-Läufe mit unterschiedlichen Seeds durchgeführt, jeweils mit 10-Episoden, die 100.000-Schritte dauerten (ohne weitere Abbruchkriterien). Das Ergebnis waren Tausende von gefangenen Agenten in beiden Gruppen, deren Überlebenszeit dann jeweils in ihrer Gruppe gemittelt wurde. Um die Mengenverhältnisse der Agenten gleich zu halten, werden gefangene und damit gestorbene Agenten direkt neu erstellt. Dabei kehrten sie an der dichtesten Stelle des Schwarms zurück in die Umgebung (ermittelt mit der Kerndichteschätzung [115]). Das Vorgehen wurde gewählt, weil es für dieses Experiment von besonderem Interesse ist, sich vom Schwarm wegzubewegen bzw. die anderen Agenten zu ignorieren, und das Schwarmverhalten nicht durch neu (potenziell auf der Freifläche) in die Umgebung eingesetzte Agenten gestört werden sollte.

Das Experiment offenbart zahlreiche interessante Erkenntnisse:

1. Die Leistung von SELFish$_{\text{DQN}}$ im Hinblick auf die gemessene Überlebenszeit einzelner Individuen in der Umgebung, in der die Agenten tatsächlich auf diese Strategie trainiert wurden, ist besser als TurnAway. Das bedeutet, dass die Verwendung einer in einem teilweise beobachtbaren Modell erhaltenen Verhaltensstrategie in einem Setting mit anderen Parametern möglich ist, aber möglicherweise nicht zu einer konsistenten Leistung in dem trainierten Setting führt (im Vergleich zu Abbildung 6.8).

2. Das Schwarmverhalten, welches sich aus dem Reinforcement Learning in dieser Umgebung ergibt (mehrere homogene Agenten, die einem Raubtier ausweichen/ablenken), ist ein Nash-Gleichgewicht (grüne Linie bei $(0, 10)$ in Abbildung 6.12). Das bedeutet, bei 10 Agenten, die SELFish$_{\text{DQN}}$ mit einer Neigung zum Schwärmen/Gruppieren ausführen (und null TurnAway-Agenten), hat kein Agent einen Anreiz von dieser Strategie des „Schwärmens“ abzuweichen, während alle anderen ihre Strategie beibehalten. In der Simulation ist beobachtbar, dass das Verlassen des Schwarms und das Ignorieren der Anderen (wie TurnAway) den Agenten auf die freie Fläche führen, wo er eine leichte Beute für den Räuber ist. Die Intuition wurde bestätigt, indem die Nash-Gleichgewichte in diesem

Setting mit dem Gambit-Software-Tools für Spieltheorie [97] berechnet wurden. Dies geschah auf die Weise, dass die durchschnittliche Überlebenszeit eines Agenten, der eine bestimmte Strategie verfolgt, als Auszahlung in Sinne eines Normalformspiels betrachtet wurde. Das bedeutet zum Beispiel, dass, wenn 9 Agenten $\text{SELFish}_{\text{DQN}}$ ausführen, beträgt die Auszahlung für jeden Agenten, der diese Strategie verfolgt, 561, während der eine Agent, der die TurnAway Strategie verfolgt, eine Auszahlung von 415 erhält. Gambit zeigte auch ein weniger offensichtliches Nash-Gleichgewicht bei $(7, 3)$. Bei diesem Mengenverhältnis hat ein Agent, der TurnAway (Auszahlung 619) ausführt, keinen Anreiz seine Strategie auf $\text{SELFish}_{\text{DQN}}$ umzustellen (Auszahlung 614 von $\text{SELFish}_{\text{DQN}}$ am Punkt $(6, 4)$). Zusätzlich, hat ein $\text{SELFish}_{\text{DQN}}$-Agent (Auszahlung 610) keinen Anreiz zu TurnAway (Auszahlung 575 bei $(8, 2)$) zu wechseln, während die anderen Agenten ihre Strategie beibehalten (siehe Tabelle 6.2 für die genauen Auszahlungswerte bzw. Überlebenszeiten).

Tabelle 6.2: Überlebenszeit/Auszahlung von 10-Agenten mit unterschiedlichen Verhaltensstrategien auf einer Fläche von 40×40 Pixeln.

Anzahl an Agenten		Überlebenszeit	
Turn Away	$\text{SELFish}_{\text{DQN}}$	Turn Away	$\text{SELFish}_{\text{DQN}}$
0	10	nan	517
1	9	415	561
2	8	476	590
3	7	530	611
4	6	576	612
5	5	620	619
6	4	642	614
7	3	619	610
8	2	575	598
9	1	412	582
10	0	183	nan

3. Verhaltensstrategien in Multi-Agenten-Szenarien, die mit DQN und der von in [29] vorgeschlagenen Methode ermittelt wurden, können nur die äußeren Punkte $(0, 10)$ und $(10, 0)$ einnehmen, da alle Agenten die gleiche Strategie verfolgen. Im Beispiel einer Flächengröße von 40×40 Pixeln ist $(0, 10)$ sicherlich besser, während ein Verhältnis von $(6, 4)$ die beste Mischung aus beiden Strategien ist, wobei in dieser Umgebung unter Umständen noch eine bessere Leistung im Sinne der Überlebenszeit der Agenten erzielt werden kann.

Um die Ergebnisse weiter zu untermauern, wurde das Experiment in Flächengrößen von 80×80 und 20×20 Pixeln wiederholt. Die Ergebnisse sind in den Abbildungen 6.13 und 6.14 zu sehen. Sie zeigen, dass die Ergebnisse für verschiedene Flächengrößen variieren, wie z.B. reines TurnAway, das im Fall von 80×80 Pixeln reines $\text{SELFish}_{\text{DQN}}$ deutlich übertrifft. Zudem wurden weniger offensichtliche Nash-Gleichgewichte wie $(4, 6)$ im Fall von 20×20 Pixeln gefunden.

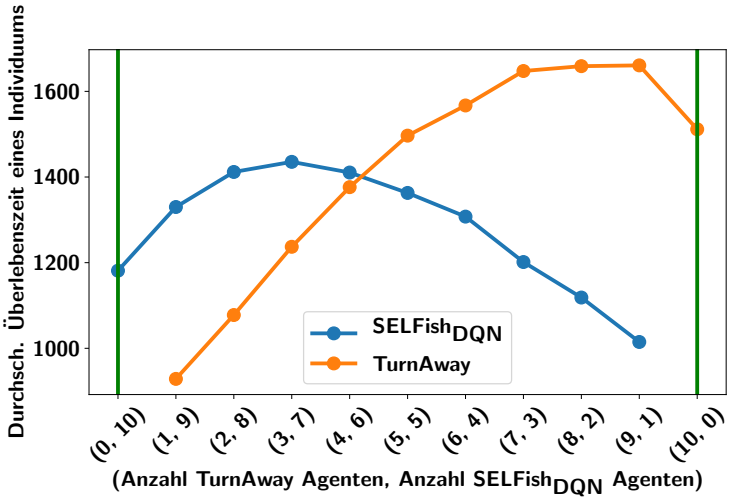


Abbildung 6.13: Überlebenszeit $\text{SELFish}_{\text{DQN}}$ (trainiert mit 10 Agenten auf 40×40 Pixeln) gegenüber TurnAway auf einer Fläche von 80×80 Pixeln. Nash-Gleichgewichte mit einer grünen Linie markiert.

6.6 Diskussion und Zusammenfassung

Mit SELFish wurde gezeigt, dass Schwarmverhalten allein durch eigennützige Agenten entstehen kann. Sie wurden durch Multi-Agent Reinforcement Learning darauf trainiert, zu vermeiden, von einem Räuber gefangen zu werden, der sich von mehreren Agenten in seinem Sichtradius ablenken lässt. Es wurde jeweils nur ein Agent mit einer Belohnungsstruktur trainiert, die dazu ermutigt, das Gefangenwerden so lange wie möglich zu vermeiden. Nach jeder Episode wurde die erlernte Verhaltensweise auf alle anderen Agenten übertragen. Die Ergebnisse für $\text{SELFish}_{\text{DQN}}$ und $\text{SELFish}_{\text{DDPG}}$ in Bezug auf die Ausrichtung und den Zusammenhalt, aber auch in Bezug auf die Überlebenschancen der Agenten wurden mit Boids verglichen, einem gängigen Ansatz für algorithmische Schwarmsimulationen. Die Ergebnisse zeigen, dass die Messungen für den durch Multi-Agent Reinforcement Learning erzeugten Schwarm denen von Boids ähnlich sind. Im Gegensatz zu Boids, wurde den Agenten jedoch nie explizit ein Modell zur Verfügung gestellt, das es erlaubt, einen Schwarm zu erkennen

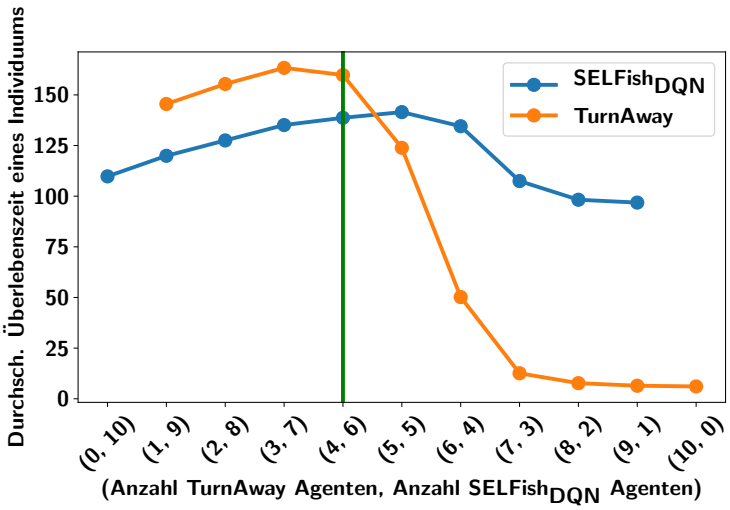


Abbildung 6.14: Überlebenszeit von SELFish$_{DQN}$ (ausgebildet mit 10 Agenten auf 40×40 Pixeln) gegenüber TurnAway auf einer Fläche von 20×20 Pixeln. Nash-Gleichgewichte mit einer grünen Linie markiert.

oder zu bilden. Die Experimente zeigen also, dass es möglich ist, Schwärme durch rein emergentes Verhalten zu erzeugen.

Durch Veränderung/Erweiterung des Setups konnte gezeigt werden, dass die durch Reinforcement Learning in teilweise beobachtbaren Szenarien gewonnenen Verhaltensstrategien in anderen Umgebungen ohne Leistungseinbruch eingesetzt werden können, obwohl eine konsistente Leistung (im Vergleich zu der Umgebung, in der die Strategie tatsächlich trainiert wurde) nicht garantiert werden kann. Darüber hinaus wurde gezeigt, dass der aus Reinforcement Learning mit mehreren Agenten resultierende Schwarm in einem Räuber/Beute-Szenario ein Nash-Gleichgewicht hat.

Dies wirft ein wenig Licht auf die Gründe für das Entstehen von Schwärmen. Die Motivation des Kapitels begann mit einer Auflistung der Vorteile, die bei biologischen Schwärmen beobachtet wurden. Die vorgestellte Forschung könnte einen weiteren Grund für die Entstehung von Schwärmen auflisten, und zwar den sozialen Druck. Die Existenz eines Schwarmes von beträchtlicher Größe kann das Überleben von nicht schwärmenden Individuen aktiv behindern und sie somit dazu drängen, sich dem Schwarm anzuschließen, selbst wenn dies für das Überleben aller Individuen suboptimal ist. Dabei ist zu beachten, dass dies rationale Agenten betrifft, d.h. Schwarmteilnehmer, die bei jeder einzelnen ihrer Entscheidungen lokal optimal handeln.

Interessanterweise ist das Phänomen des Drucks, der durch fehlende Kommunikations- und Kontrollstrukturen entsteht, auch in der natürlichen Evolution beobachtet worden [22]. Daher können Schwärme unter bestimmten Bedingungen auch als sich selbsterhaltend interpretiert werden, was bedeutet, dass sie bei ihrem Einsatz in der Praxis mit zusätzlicher Vorsicht behandelt

werden sollten. Sich selbsterhaltende Schwärme können zusätzliche Ziele für emergentes Verhalten einführen, die den beabsichtigten Zweck des Systementwicklers beeinflussen. Es liegt an der zukünftigen Forschung, das Zusammenspiel zwischen der Nutzung eines solchen emergenten Verhaltens und seiner Kontrolle zu untersuchen, um Schwarmanwendungen nützlich einzusetzen.

Für das geschilderte Szenario selbst bleibt zu untersuchen, ob andere vorteilhafte Eigenschaften eines Schwarms, wie eine erhöhte Bewegungseffizienz in der Gruppe, die hier noch nicht modelliert wurden, ebenfalls zur Schwarmbildung in einem Reinforcement Learning Szenario führen. Die Co-Evolution des Verhaltens des Räubers und seiner Beute durch Reinforcement Learning erwies sich in ersten Experimenten als möglich aber instabil. Dies wäre ein interessantes Thema für weiterführende Forschung. Die Agenten konnten sich in einer torusartigen Umgebung ohne Hindernisse oder Kollisionen frei bewegen. Natürlich gibt es dazu Verbesserungen wie das Hinzufügen von Wänden, Hindernissen und Kollisionen zwischen den Agenten.

7 Zusammenfassung und Ausblick

Die Anwendungsfälle für sich autonom bewegende und organisierende Entitäten, in der Literatur auch Agenten genannt, sind grenzenlos. Dies beginnt bei bereits weit verbreiteten Staubsaug- oder Rasenmährobotern, ist aber auch für Nicht-Spieler-Charaktere in Computerspielen interessant und besonders für Szenarien der Industrie 4.0 von Bedeutung. Dazu gehören autonom fahrende Kraftfahrzeuge, welche von verschiedenen Interessengruppen stark vorangetrieben werden. In der Logistik sind neben den autonom fahrenden Kraftfahrzeugen auch Paketroboter oder insbesondere Paketdrohnen bedeutsam.

In der vorliegenden Arbeit wurde in Kapitel 3 zunächst ein Ansatz zur intuitiven Vorhersage von dynamischen Hindernisbewegungen vorgeschlagen. Danach werden in diesem Kapitel zwei Ansätze entwickelt, um die erlernten Vorhersage in die Wegplanungsentscheidungen eines Agenten zu integrieren. Die Bewegungsvorhersage basiert auf künstlichen neuronalen Netzen und hat den entscheidenden Vorteil, dass sie sich dynamisch an geänderte Umgebungsbedingungen anpassen kann. Zudem hat das künstliche neuronale Netz den Vorteil, dass es sich im Hinblick auf den Berechnungsaufwand schneller auswerten lässt, als dies bei einem exakten physikalischen Modell der Fall wäre, was aber auch Einbußen im Hinblick auf die Genauigkeit nach sich zieht. Im ersten vorgeschlagenen Ansatz wird das erlernte Modell vom Agenten verwendet, um ausgehend vom aktuellen Zustand der Umgebung einen Baum möglicher Folgezustände aufzuspannen, die sich aus seinen Aktionen und den abgeschätzten, zukünftigen Positionen der Objekte in seiner Umgebung ergeben. Aus diesen simulierten Folgezuständen wählt er dann den führ ihn am lohnenswertesten Zustand aus, um die zu ihm führende Aktionsfolge tatsächlich auszuführen. Die Evaluation des Ansatzes zeigt, dass ein Agent, der auf dem erlernten Modell beruht, im Hinblick auf die erhobenen Metriken ähnliche Performance-Werte erreicht wie ein Agent, der auf einem exakten physikalischen Modell seiner Umgebung beruht. Das erlernte Modell lässt sich aber im Hinblick auf die Rechenzeit deutlich schneller auswerten, was den Planungsvorgang signifikant beschleunigt. Da bei dieser Baumsuche über Folgezustände der Umgebung Berechnungen mehrfach ausgeführt werden, wird im Kapitel 3 ein weiterer Ansatz zur Verwendung der erlernten Vorhersagemodelle entwickelt. Mit diesem wird untersucht wie sich die Bewegungsvorhersage in einen Graphen integrieren lassen. Es wird vorgeschlagen die Integration über zeitabhängige Kantenkosten vorzunehmen, so dass eine Kollisionsteuerung auf globaler Ebene erfolgen kann,

indem Kollisionsszenarien schon bei der Wegplanung mittels eines angepassten A* Algorithmus vermieden werden und der Agent gar nicht erst in die Verlegenheit kommt, Kollisionen auf lokaler Ebene z.B. durch einen Potential Field Ansatz oder einem einfachen Notstopp vermeiden zu müssen.

Betrachtet man Szenarien der Industrie 4.0, bei denen sich Fertigungsroboter autonom in Industriehallen bewegen, muss noch vor der Verhinderung von Kollisionen mit statischen und dynamischen Hindernissen zunächst mit der Bewegung auf einer Freifläche umgegangen werden. Diese wird in der Regel zunächst diskretisiert, um sie algorithmisch greifbar zu machen. Deshalb wird in Kapitel 4 dieser Arbeit ein ganzheitliches Verfahren vorgeschlagen, mit welchem zunächst mit einem Ansatz namens Stable Growing Neural Gas Freiflächen diskretisiert werden, um auf dem entstehenden Graphen multiple Agenten parallel routen zu können. Stable Growing Neural Gas ist von der gleichmäßigen Ausbreitung von Gas-Molekülen im Raum inspiriert und diskretisiert auf diese Weise einen Raum, indem heuristisch Knoten erzeugt, gleichmäßig im Raum verteilt und miteinander verbunden werden. Kapitel 4 enthält mehrere Vorschläge zur Verbesserung von Stable Growing Neural Gas, wenn es, wie hier, zur Erstellung von sich dynamisch anpassenden Routing-Graphen verwendet wird. Der Ansatz umfasst zudem einen Vorschlag, wie sich auf der Grundlage einer Potential Field Methode Kollisionen zwischen verschiedenen Agenten oder dynamischen Hindernissen auf lokaler Ebene verhindern lassen. Zur Pfadplanung auf dem entstehenden Graphen wird die Verwendung des A* Algorithmus vorgeschlagen und es wird dargestellt, welche positiven Effekte durch das Zusammenwirken der gewählten Methoden zur Diskretisierung, Kollisionsvermeidung und Pfadplanung ergeben. In der Evaluation zeigt der entworfene Ansatz im Hinblick auf häufige Änderungen der Umgebung oder die Unterstützung multipler Agenten überlegene Leistung im Vergleich zu verwandten Arbeiten.

Für den Fall, dass eine ganze Gruppe von Agenten zu einem Punkt gesteuert werden bzw. einem Zielobjekt folgen soll, wurde in Kapitel 5 ein auf Reinforcement Learning basierender Ansatz vorgeschlagen. Das Setting ist inspiriert von einem Futtersuche-Szenario, bei welchem z.B. ein Schwarm von Fischen einer Futterquelle folgt. Der Beitrag des Ansatzes besteht vor allem in der speziell geformten Beobachtung der Agenten und in der eigens gestalteten Belohnungsfunktion, mit der die Agenten darauf trainiert werden, den Abstand zu einem virtuellen Zielpunkt zu minimieren. Dafür hat sich die Methode, einen Agenten zu trainieren und dessen Wissen allen anderen Agenten zur Verfügung zu stellen, als wirksam erwiesen. Mit der entwickelten Methode konnten die Agenten erfolgreich darauf trainiert werden einem Zielpunkt zu folgen. Darüber hinaus ist ihnen dies auch möglich, wenn sie den direkten Sichtkontakt zum Zielpunkt verlieren, da sie gelernt haben, sich aneinander zu orientieren. Ein Beitrag dieses Kapitels ist des Weiteren, dass die Agenten selbständig lernen voneinander einen gewissen Abstand zu halten, ohne explizit darauf trainiert zu werden.

Die gleiche Trainingstechnik wurde in Kapitel 6 auf das umgekehrte Szena-

rio angewendet, bei welchen eine Gruppe von Agenten einem auf sie zukommenden Objekt nicht folgen, sondern ausweichen soll. Dies ist wiederum an einen Fisch- oder Vogelschwarm angelehnt ist, bei welchem die Individuen einem Raubtier ausweichen. Der Räuber könnte allerdings auch als intelligentes Hindernis angesehen werden, welchem ausgewichen werden soll. Dem Szenario liegt zusätzlich die Annahme zugrunde, dass der Angreifer/Räuber von multiplen möglichen Beutetieren in seiner Umgebung abgelenkt ist und sich nur schwer für eines entscheiden kann. Die Ergebnisse zeigen, dass die eigennützig handelnden Agenten emergent einen Schwarm bilden, ohne dass sie in irgendeiner Form explizit darauf trainiert wurden, abgesehen davon, ihre Überlebenszeit zu maximieren. Zudem zeigen die erlernten Verhaltensweisen große Ähnlichkeit zu einem etablierten Ansatz zur algorithmischen Erzeugung von Schwärmen, welcher auf starren Regeln basiert. Das entstehende Verhalten ist damit zu erklären, dass der Aufenthalt in einer Gruppe die Überlebenschance eines einzelnen Individuums in Anwesenheit des Räubers erhöht. Dies erlaubt aufschlussreiche Einblicke in die Dynamiken, welche sich aus der Interaktion multipler autonome Agenten ergeben können und kann für die Untersuchung von Evakuierungsszenarien von Bedeutung sein. Geht man davon aus, das in Zukunft unter Umständen ganze Drohnenschwärme zum Einsatz kommen, um Pakete auszuliefern oder gegebenenfalls zu militärischen Einsatzzwecken, so liefert das gelernte Verhalten einen Ansatzpunkt, um mit möglichen Angreifern umzugehen.

Obwohl bereits in der Zusammenfassung am Ende eines jeden Kapitels skizziert, soll im Folgenden noch einmal auf Ansatzpunkte eingegangen werden, welche Raum für Folgearbeiten bieten. Betrachtet man die Modelle der intuitiven Physik aus Kapitel 3, welche später verwendet werden um die Vorhersage von Objektbewegungen in den Planungsprozess des Agenten zu integrieren, so lassen diese ohne Zweifel noch Verbesserungen zu. Dies betrifft z.B. die Art oder Anzahl der unterstützten Bewegungsmuster (oder deren Unterscheidung) aber auch die Methodik des Trainings oder die Ausgabe im Hinblick auf Format und Informationsgehalt. Hierzu gibt es rege eigene Forschungsgebiete, die sich z.B. mit der Vorhersage von Bewegungsmustern menschlicher Bewegung beschäftigen. Die in dieser Arbeit verwendeten, auf künstlichen neuronalen Netzen basierenden Bewegungsmodelle dienen eher der Demonstration der Durchführbarkeit des Ansatzes. So wurde zu Beginn des zweiten Kapitels gezeigt, dass sich zusammen mit den Vorhersagen zukünftiger Positionen auch Aussagen darüber extrahieren lassen, wie sicher das Modell in der Vorhersage ist. Aus Komplexitätsgründen wurden die Informationen bzw. der Korridor an vorhergesagten möglichen Bewegungsverläufen, der sich daraus ergibt, im späteren Pfadplanungsverfahren des Agenten aber nicht weiterverwendet. Dies wäre z.B. beim PCMP im Vergleich zur klassifikationsbasierten Vorhersage des Belegungsgitters interessant.

Generell kommen bei allen in dieser Arbeit vorgestellten Ansätzen zweidimensionale Räume zum Einsatz, in der Vorstellung der Draufsicht auf einen

Gebäudeplan. Es ist davon auszugehen, dass sich sämtliche in dieser Arbeit angesprochenen Algorithmen und Ansätze auf drei Dimensionen erweitern lassen. Dies gilt für die Bewegungsvorhersage mit intuitiver Physik, das Stabel Growing Neural Gas mit dem Potential Field Ansatz aber auch das Reinforcement Learning zur Untersuchung von Schwarmverhalten. Die Erweiterung des Schwarmansatzes zur Anwendung in 3D ist interessant, weil in diesem Hinblick vor allem fliegende Drohnen als Anwendungszweck sinnvoll erscheinen.

Ein weiterer Ansatzpunkt wäre die physikalische Korrektheit der Simulationen. Zunächst wurde aus Ermangelung echter Roboter auf Simulationen zurückgegriffen. Zur Komplexitätsreduktion wurde in diesen dann aber auf die physikalisch korrekte Modellierung sämtlicher Gegebenheiten verzichtet. So haben die Agenten in der hier vorliegenden Arbeit in der Regel keine Masse oder Trägheit und unterliegen in ihrer Bewegung z.B. keinen vorgegebenen Kurvenradien. Auch in diesem Hinblick ging es vor allem um die Demonstration der Machbarkeit und weniger um die abschließende detailgetreue Umsetzung bis hin zur Produktreife. Erste getätigte Erweiterungen der hier vorgestellten Ansätze deuten aber an, dass die Integration weiterer physikalische Bedingungen durchaus möglich ist, ohne den Ansatz selbst unbrauchbar zu machen.

Im Hinblick auf die Schwarmbildung durch Reinforcement Learning sei erwähnt, dass in diesem Bereich viele weitere Szenarien denkbar sind. Eines davon ist, dass alle Agenten parallel lernen, siehe [57]. Dies macht das Lernverfahren zwar instabiler, förderte aber eine Strategie zutage, welche etwas performanter ist als die in Kapitel 5 ermittelte. Auch die Koevolution, das heißt, das parallele Lernen von Angreifer und Beute, sowie das Training auf reinen Pixeldaten anstelle von vorgefertigten Features wurde erfolgreich erprobt, fand aber keinen Eingang in diese Arbeit.

Eine weitere Dimension, um welche die Ansätze dieser Arbeit, die multiple Agenten umfassen, erweitert werden können, ist die explizite Kommunikation zwischen den Agenten. Dabei kann es sich um vorgegebene Protokolle handeln, die für das Routing auf einem gemeinsamen Graphen, wie in Kapitel 4, von Vorteil sein könnten. Für die Ansätze in Kapitel 5 und 6 wäre insbesondere eine erlernte Kommunikation zwischen den Agenten interessant, was ein aktives Forschungsfeld im Multi-Agent Reinforcement Learning darstellt, um so gemeinschaftlich, effizientere Verfolgungs- oder Ausweichstrategien zu ermitteln. Dies würde die Ansätze auch noch näher an ihre biologischen Vorbilder heranführen und damit Vergleiche zwischen künstlich erlerntem Verhalten und in der Biologie/Soziologie beobachtetem Verhalten begünstigen.

Abkürzungsverzeichnis

DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network/Learning
GNG	Growing Neural Gas
KNN	Künstliches Neuronales Netz
LSTM	Long Short-Term Memory
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
NGDPP	Neural Gas Dynamic Path Planning
NG	Neural Gas
PCMP	Predictive Collision Management Path Planning
POMDP	Partially Observable Markov Decision Process
PRM	Probabilistic Roadmap
ReLU	Rectified Linear Unit
RRT	Rapidly Exploring Random Tree
SGNG	Stable Growing Neural Gas
SOM	Self-Organizing (Feature) Map
VFF	Virtual Force Field

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[2] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] A. Albadvi, S. K. Chaharsooghi, and A. Esfahanipour. Decision making in stock trading: An application of promethee. *European Journal of Operational Research*, 177(2):673–683, 2007.

[4] H. Alt, E. Welz, and B. Wolfers. Piecewise linear approximation of bézier-curves. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 433–435. ACM, 1997.

[5] S. Ammoun and F. Nashashibi. Real time trajectory prediction for collision risk estimation between vehicles. In *ICCP 2009*, Aug. 2009.

[6] C. Barrios and Y. Motai. Improving estimation of vehicle's trajectory using the latest global positioning system with kalman filtering. *IEEE T. Instrumentation and Measurement*, 60:3747–3755, 12 2011.

[7] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

[8] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2372–2377. IEEE, 2006.

[9] R. Bellman. Dynamic programming. *Princeton Press*, 1957.

[10] L. Belzner. Time-adaptive cross entropy planning. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 254–259, New York, NY, USA, 2016. ACM.

[11] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[12] K. Berns and T. Kolb. *Neuronale Netze für technische Anwendungen*. FZI-Berichte Informatik. Springer Berlin Heidelberg, 2013.

[13] M. Bowling and M. Veloso. An analysis of stochastic game theory for multiagent reinforcement learning. Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2000.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[15] E. Catmull and R. Rom. A class of local interpolating splines. In *Computer aided geometric design*, pages 317–326. Elsevier, 1974.

[16] G. Chalkiadakis, E. Elkind, and M. Wooldridge. Computational aspects of cooperative game theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(6):1–168, 2011.

[17] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *CoRR*, abs/1612.00341, 2016.

[18] O. Chatain. Cooperative and non-cooperative game theory. *University of Pennsylvania*, 2014.

[19] F. Chollet et al. Keras. `https://keras.io`, 2015.

[20] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[22] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 1976.

[23] E. M. de Cote, A. Lazaric, and M. Restelli. Learning to cooperate in multi-agent social dilemmas. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 783–785, New York, NY, USA, 2006. ACM.

[24] B. Dellinger, R. Jenkins, and J. Walton. Automated waypoint generation with the growing neural gas algorithm. 2017.

[25] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. A case for time-dependent shortest path computation in spatial networks. pages 474–477, 01 2010.

[26] L. Deng and D. Yu. Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387, 2014.

[27] M. Dietzfelbinger, K. Mehlhorn, and P. Sanders. *Algorithmen und Datenstrukturen: Die Grundwerkzeuge*. Springer-Verlag, 2014.

[28] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[29] M. Egorov. Multi-agent deep reinforcement learning. *CS231n: Convolutional Neural Networks for Visual Recognition*, 2016.

[30] A. Eidehall and L. Petersson. Statistical threat assessment for general road scenes using monte carlo sampling. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):137–147, March 2008.

[31] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[32] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, 1996.

[33] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[34] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[35] L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 327–341. SIAM, 2011.

[36] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *CoRR*, 2015.

[37] M. Friedrich, A. Ebert, C. Hahn, G. Schneider, L. Obermeier, A. Erk, and I. Jennes. A distributed metadata platform for hybrid radio services. In *19th International Conference on Innovations for Community Services (I4CS 2019)*, 2019.

[38] M. Friedrich, C. Roch, S. Feld, C. Hahn, and P.-A. Fayolle. A flexible pipeline for the optimization of construction trees. In *Proceedings of the 28th International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2020.

[39] B. Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pages 625–632, 1995.

[40] C. Fulgenzi, A. Spalanzani, C. Laugier, and C. Tay. Risk based motion planning and navigation in uncertain dynamic environment. Research report, Oct. 2010.

[41] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[42] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[43] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

[44] T. Gindele, S. Brechtel, and R. Dillmann. Learning driver behavior models from traffic observations for decision making and planning. *IEEE Intelligent Transportation Systems Magazine*, 7(1):69–79, Spring 2015.

[45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[47] H.-M. Gross, H. Boehme, C. Schroeter, S. Müller, A. König, E. Einhorn, C. Martin, M. Merten, and A. Bley. Toomas: interactive shopping guide robots in everyday use-final implementation and experiences from long-term field trials. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2005–2012. IEEE, 2009.

[48] H. Guo and Y. Meng. Distributed reinforcement learning for coordinate multi-robot foraging. *J. Intell. Robotics Syst.*, 60(3-4):531–551, Dec. 2010.

[49] J. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *AAMAS Workshops*, pages 66–83, 11 2017.

[50] D. Ha and J. Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[51] C. Hahn. Framework for hybrid positioning. In *13. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste (LBAS 2016)*, 2016.

[52] C. Hahn. Intuitive pathfinding of autonomous agents. In *14. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste (LBAS 2017)*, 2017.

[53] C. Hahn and S. Feld. Collision avoidance using intuitive physics. In *2018 International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2018.

[54] C. Hahn, S. Feld, and H. Schroter. Predictive collision management for time and risk dependent path planning. In *28th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2020)*. ACM, 2020.

[55] C. Hahn, S. Feld, and H. Schroter. Predictive collision management for time and risk dependent path planning. *arXiv e-prints*, page arXiv:2011.13305, Nov. 2020. (Extended Version).

[56] C. Hahn, S. Feld, M. Zierl, and C. Linnhoff-Popien. Dynamic path planning with stable growing neural gas. In *11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, 2019.

[57] C. Hahn and M. Friedrich. Using existing reinforcement learning libraries in multi-agent scenarios. In *1st International Symposium on Applied Artificial Intelligence (ISAAI'19)*, 2019.

[58] C. Hahn, S. Holzner, L. Belzner, and M. T. Beck. Empirical evaluation of a distributed deployment strategy for virtual networks. In *International Conference on Mobile, Secure, and Programmable Networking*, pages 88–98. Springer, 2017.

[59] C. Hahn, T. Phan, S. Feld, C. Roch, F. Ritz, A. Sedlmeier, T. Gabor, and C. Linnhoff-Popien. Nash equilibria in multi-agent swarms. In *12th International Conference on Agents and Artificial Intelligence (ICAART 2020)*, 2020.

[60] C. Hahn, T. Phan, T. Gabor, L. Belzner, and C. Linnhoff-Popien. Emergent escape-based flocking behavior using multi-agent reinforcement learning. *The 2019 Conference on Artificial Life*, (31):598–605, 2019.

[61] C. Hahn, F. Ritz, P. Wikidal, T. Phan, T. Gabor, and C. Linnhoff-Popien. Foraging swarms using multi-agent reinforcement learning. *Artificial Life Conference Proceedings*, (32):333–340, 2020.

[62] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[63] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[64] M. Hüttenrauch, A. Sosic, and G. Neumann. Deep reinforcement learning for swarm systems. *CoRR*, abs/1807.06613, 2018.

[65] N. Kaempchen, K. Weiss, M. Schaefer, and K. Dietmayer. Imm object tracking for high dynamic driving maneuvers. pages 825 – 830, 07 2004.

[66] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug 1996.

[67] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE Intl Conference on Robotics and Automation*, volume 2, March 1985.

[68] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[69] A. Khosravi, E. Mazloumi, S. Nahavandi, D. Creighton, and J. W. C. van Lint. Prediction intervals to account for uncertainties in travel time prediction. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):537–547, June 2011.

[70] B. Kim, C. Kang, J. Kim, S. Lee, C. Chung, and J. Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th Intl Conference on Intelligent Transportation Systems (ITSC)*, Oct 2017.

[71] Y.-H. Kim, J.-I. Jang, and S. Yun. End-to-end deep learning for autonomous navigation of mobile robot. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6. IEEE, 2018.

[72] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[73] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation, 1991. Proceedings.*, pages 1398–1404. IEEE, 1991.

[74] Y. Koren, J. Borenstein, et al. Potential field methods and their inherent limitations for mobile robot navigation. In *ICRA*, volume 2, pages 1398–1404, 1991.

[75] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[76] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[77] A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017.

[78] A. Kushleyev and M. Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *2009 IEEE Intl Conf. on Robotics and Automation*, May 2009.

[79] J. E. Laird. Using a computer game to develop advanced ai. *Computer*, 34(7):70–75, 2001.

[80] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, page 1–101, 2016.

[81] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017)*, volume 31. Neural Information Processing Systems (NIPS), 2017.

[82] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[83] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[84] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[85] S. Lefevre, D. Vasquez, and C. Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1, 07 2014.

[86] P. Legendre, F.-J. Lapointe, and P. Casgrain. Modeling brain evolution from behavior: a permutational regression approach. *Evolution*, 48(5):1487–1499, 1994.

[87] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*, pages 464–473. IFAAMAS, 2017.

[88] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[89] W. Li, A. Leonardis, and M. Fritz. Visual stability prediction and its application to manipulation. *CoRR*, abs/1609.04861, 2016.

[90] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[91] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[92] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[93] H. Liu and L. Wang. Human motion prediction for human-robot collaboration. *Journal of Manufacturing Systems*, 44:287–294, 2017.

[94] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[95] T. Martinetz, K. Schulten, et al. A "neural-gas"network learns topologies. 1991.

[96] L. Matignon, G. j. Laurent, and N. Le fort piat. Review: Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *Knowl. Eng. Rev.*, 27(1):1–31, Feb. 2012.

[97] R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory. 2016.

[98] I. Millington and J. Funge. *Artificial intelligence for games*. CRC Press, 2009.

[99] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[101] O. Montiel, U. Orozco-Rosas, and R. Sepúlveda. Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles. *Expert Systems with Applications*, 42(12):5177–5191, 2015.

[102] K. Morihiro, H. Nishimura, T. Isokawa, and N. Matsui. Learning grouping and anti-predator behaviors for multi-agent systems. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 426–433. Springer, 2008.

[103] L. Noriega. Multilayer perceptron tutorial. 2005.

[104] A. Nowé, P. Vrancx, and Y.-M. De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.

[105] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv e-prints*, page arXiv:1811.03378, Nov. 2018.

[106] R. O'Grady, C. Pinciroli, R. Groß, A. L. Christensen, F. Mondada, M. Bonani, and M. Dorigo. Swarm-bots to the rescue. In *European Conference on Artificial Life*, pages 165–172. Springer, 2009.

[107] J. Orkin. Agent architecture considerations for real-time planning in games. In *AIIDE*, pages 105–110, 2005.

[108] A. B. Özgüler and A. Yıldız. Foraging swarms as nash equilibria of dynamic games. *IEEE transactions on cybernetics*, 44(6):979–987, 2013.

[109] S. Park, B. Kim, C. Kang, C. Chung, and J. Choi. Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture. pages 1672–1678, 06 2018.

[110] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[111] J. Patel, S. Shah, P. Thakkar, and K. Kotecha. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, 42(1):259 – 268, 2015.

[112] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[113] J. Perolat, J. Z. Leibo, V. Zambaldi, C. Beattie, K. Tuyls, and T. Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017)*, 30, 2017.

[114] R. Philippsen, B. Jensen, and R. Siegwart. Toward online probabilistic path replanning in dynamic environments. pages 2876–2881, 10 2006.

[115] S. J. Phillips, R. P. Anderson, and R. E. Schapire. Maximum entropy modeling of species geographic distributions. *Ecological modelling*, 190(3-4), 2006.

[116] M. Plappert. keras-rl. https://github.com/keras-rl/keras-rl, 2016.

[117] A. Pokle, S. A. Baby, and L. Li. Analysis of emergent behavior in multi-agent environments. *CS234: Reinforcement Learning Winter 2018*, 2018. `https://ashwinipokle.github.io/assets/docs/234_final_report.pdf`.

[118] X. Qiu, E. Meyerson, and R. Miikkulainen. Quantifying point-prediction uncertainty in neural networks via residual estimation with an i/o kernel. *arXiv preprint arXiv:1906.00588*, 2019.

[119] Intuitive Physics Workshop at NIPS 2016. Online, 2016. `http://phys.csail.mit.edu/`, letzter Abruf: 12.11.2020.

[120] T. Rashid, M. Samvelyan, C. S. Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4292–4301, 2018.

[121] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

[122] C. W. Reynolds. Steering behaviors for autonomous characters. `https://www.red3d.com/cwr/steer/gdc99/`, march 1999.

[123] F. Ritz, F. Hohnstein, R. Müller, T. Phan, T. Gabor, C. Hahn, and C. Linnhoff-Popien. Towards ecosystem management from greedy reinforcement learning in a predator-prey setting. In *Artificial Life Conference Proceedings*, volume 32, pages 518–525, 2020.

[124] R. Rojas. *Theorie der neuronalen Netze: eine systematische Einführung*. Springer-Verlag, 2013.

[125] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[126] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[127] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques. Pythonrobotics: a python code collection of robotics algorithms, 2018.

[128] K. Schmid, L. Belzner, T. Gabor, and T. Phan. Action markets in deep multi-agent reinforcement learning. In *International Conference on Artificial Neural Networks*, pages 240–249. Springer, 2018.

[129] E. Semsar-Kazerooni and K. Khorasani. A game theory approach to multi-agent team cooperation. In *2009 American Control Conference*, pages 4512–4518. IEEE, 2009.

[130] D. Silver. Cooperative pathfinding. 01 2005.

[131] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[132] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.

[133] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[134] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[135] M. Tan. Multi-agent reinforcement learning: independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann Publishers Inc., 1993.

[136] F. Tencé, L. Gaubert, J. Soler, P. De Loor, and C. Buche. Stable growing neural gas: A topology learning algorithm based on player tracking in video games. *Applied Soft Computing*, 13(10):4174–4184, 2013.

[137] Q. Tran and J. Firl. Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 918–923, June 2014.

[138] J. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2366–2371, May 2006.

[139] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

[140] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101:158–168, 2016.

[141] C. J. Watkins and P. Dayan. Q-Learning. *Machine learning*, 1992.

[142] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[143] G. Watson. *Statistics on spheres*. University of Arkansas lecture notes in the mathematical sciences. Wiley, 1983.

[144] A. Weinstein and M. L. Littman. Open-loop planning in large-scale stochastic domains. In *AAAI*, 2013.

[145] M. Werner, C. Hahn, and L. Schauer. Deepmovips: Visual indoor positioning using transfer learning. In *7th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016)*, 2016.

[146] Y. Xu, Z. Piao, and S. Gao. Encoding crowd interaction with deep neural network for pedestrian trajectory prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5275–5284, 2018.

[147] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng. Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction, 2019.

[148] L. Zhao, T. Ohshima, and H. Nagamochi. A* algorithm for the time-dependent shortest path problem. 01 2008.

[149] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *CoRR*, abs/1712.00600, 2017.