

FernUniversität in Hagen

Fakultät für Mathematik und Informatik

**Projektarbeit**

im Studiengang Informatik

**Automatisches kartografieren einer unbekannten Umgebung mittels Multi Agent system**

von

|  |  |  |
| --- | --- | --- |
| Gerrit-Maximilian Söffker | 3244237 | B.Sc. Informatik |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Datum der Abgabe: XXX

Erstgutachter und Betreuer: Prof. Dr. habil. Zhong Li

Kurzfassung

XXX

Abstract

XXX

Inhaltsverzeichnis

[1 Einleitung 1](#_Toc198819021)

[2 Grundlagen 2](#_Toc198819022)

[2.1 Multi-Agent-System 2](#_Toc198819023)

[2.1.1 Definition 2](#_Toc198819024)

[2.2 Weg-Findungs-Algorithmen 3](#_Toc198819025)

[2.2.1 Random Drive 3](#_Toc198819026)

[2.2.2 A\*-Algorithmus 3](#_Toc198819027)

[2.2.3 Frontier-based Exploration Strategie 5](#_Toc198819028)

[2.3 Wahrnehmung 7](#_Toc198819029)

[2.3.1.1 Vollständig (360°) 7](#_Toc198819030)

[2.3.1.2 Eingeschränkt (< 360°) 7](#_Toc198819031)

[2.3.1.3 Reichweite 7](#_Toc198819032)

[2.3.1.4 Raycasting / LiDAR 7](#_Toc198819033)

[2.3.1.5 Bresenham Algorithmus 8](#_Toc198819034)

[2.4 Umgebungsmodell 9](#_Toc198819035)

[2.4.1 Kartographische Repräsentation 9](#_Toc198819036)

[2.4.1.1 Layout 9](#_Toc198819037)

[2.4.1.1.1 Grid (Raster Karten) 9](#_Toc198819038)

[2.4.1.1.2 HexGrid (Hexagonales Gitter) 10](#_Toc198819039)

[2.4.1.2 Hindernisse 10](#_Toc198819040)

[2.4.1.3 Randbedinungen 10](#_Toc198819041)

[2.4.1.3.1 Simulated Infinity 10](#_Toc198819042)

[2.4.1.3.2 Collision Boundary 10](#_Toc198819043)

[3 Hauptteil 10](#_Toc198819044)

[3.1 Technische Rahmenbedingungen 10](#_Toc198819045)

[3.2 Implementierung 11](#_Toc198819046)

[3.2.1 Simulationssoftware 11](#_Toc198819047)

[3.2.2 Simulationshardware 11](#_Toc198819048)

[3.2.3 UML-Diagramm 11](#_Toc198819049)

[3.2.4 Durchführung Simulation 11](#_Toc198819050)

[4 Auswertung 11](#_Toc198819051)

[4.1 Benennung 11](#_Toc198819052)

[4.2 Messgrößen 12](#_Toc198819053)

[4.3 Simulationen 12](#_Toc198819054)

[5 Zusammenfassung und Ausblick 12](#_Toc198819055)

[6 Anhang 13](#_Toc198819056)

[6.1 Rechnungen 13](#_Toc198819057)

[6.1.1 Stichprobengröße 13](#_Toc198819058)

[6.2 Daten der Simulationen 14](#_Toc198819059)

Erklärung

Ich erkläre, dass ich die schriftliche Ausarbeitung zum Seminar selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder grafische Darstellungen. Die Ausarbeitung wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Ausarbeitung nehme ich zur Kenntnis, dass diese mithilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

|  |  |  |
| --- | --- | --- |
| NAME, VORNAME  ORT, den XXX |  | NAME, VORNAME  ORT, den XXX |
|  |  |  |
|  |  |  |
| NAME, VORNAME  ORT, den XXX |  | NAME, VORNAME  ORT, den XXX |
|  |  |  |
|  |  |  |
| NAME, VORNAME  ORT, den XXX |  | NAME, VORNAME  ORT, den XXX |
|  |  |  |
|  |  |  |
|  |  |  |

Tabellenverzeichnis

[Tabelle 1: Verwendete Software und Versionen 11](#_Toc198820102)

[Tabelle 2: Messgrößen 12](#_Toc198820103)

Abbildungsverzeichnis

[Abb. 1: Pseudocode des A\* Algorithmus [8] 4](#_Toc198819951)

[Abb. 2: (L) LiDAR Scan eines Raumes, (R) Plot des Scans; Inwärts gerichtet Ecke bei ca. +30° [10] 8](#_Toc198819952)

[Abb. 3: Darstellung der Plot Sequenzen des Algorithmus [12] 8](#_Toc198819953)

[Abb. 4: Pseudocode des Bresenham-Algoritmus [12] 9](#_Toc198819954)

Abkürzungsverzeichnis

|  |  |
| --- | --- |
| MAS | Multi-Agenten-Systeme |
| LiDAR | Light detection and ranging |

Einheitenverzeichnis

|  |  |  |
| --- | --- | --- |
| **Einheit** | **Name** | **Einheit** |
| Bewegung | Bewegung in Zellen | Zellen |
| Reichweite | Wieviel Zellen werden aufgedeckt | Zellen |
| Step | Zeiteinheit in der Bewegung durchgeführt werden können | - |

Symbolverzeichnis

|  |  |
| --- | --- |
| **Zeichen** | **Beschreibung** |
|  | Nicht normalisierter Werte für Distanz, Größe und Orientierung |
|  | Optimale Grenze |
|  | Erkannte Grenze i |
|  | Werte für Distanz, Größe und Orientierung |
|  | Gewichtungsfaktoren |
|  | Die geschätzten (Heuristik-)Kosten, um vom Knoten zum Zielknoten zu gelangen |
|  | Kosten der Grenze |
|  | Menge aller Grenzen |
|  | Die Gesamtkosten eines Knotens |
|  | Die tatsächlichen Kosten, um von Startknoten zum Knoten zu gelangen |
|  | Knoten |
|  | benötigte Anzahl an Simulationsläufen |
|  | Wert für das gewünschte Vertrauensniveau |
|  | empirische Standardabweichung der Simulationsausgaben |
|  | gewünschte Fehlermarge |
|  | benötigte Anzahl an Simulationsläufen für den Testdurchlauf |
|  | Ergebnis des i-ten Simulationsdurchlaufs |
|  | Mittelwert der Pilot-Simulationen |

1. Einleitung

Ob bei der Erkundung fremder Planeten, in Katastrophengebieten oder in sich ständig verändernden Umgebungen – das automatische Kartografieren unbekannter Gebiete stellt eine Herausforderung für die moderne Robotik dar. Ein vielversprechender Ansatz zur Bewältigung dieser Aufgabe bietet das Multi-Agenten-Systeme (MAS) bei dem eine Vielzahl von Agenten autonom arbeiten. Die einzelnen Agenten treffen ihre Entscheidungen selbstständig und kommunizieren untereinander, ohne dabei auf eine zentrale Kontrollinstanz zu vertrauen. Dieses ermöglicht eine Verminderung von Redundanzen in der Kommunikation, Zeitersparnis und höhere effizient als die zentrale Verwaltung und Steuerung. Einen wesentlichen Einfluss auf die Kartografierungseffizienz haben die eingesetzten „Weg-Findung“-Algorithmen, die Anzahl des eingesetzten Agenten sowie die eingesetzte Sensortechnik. Die dabei realisierten Ergebnisse in Hinblick auf eingesetzte Zeit und zurückgelegten Weg zeigen signifikante Unterschiede. Erste Forschungsergebnisse unter Laborbedingungen stützen diese Annahmen und geben einen Ausblick auf zukünftige Entwicklungen und Ergebnisse.

1. Grundlagen
   1. Multi-Agent-System
      1. Definition

Ein Multi-Agent-System ist ein System aus mehreren selbstständig handelnden Einheiten (Agenten) welche mittels Kommunikation ein gegebenes individual- oder kollektive Problem in einer geteilten Umgebung lösen [1]. Ein Agent ist ein autonom handelnder Akteur, der seine Umgebung aktiv wahrnimmt, auf Basis dieser Informationen eigenständig Entscheidungen ableitet und entsprechende Aktionen ausführt [2].

Auch wenn MAS in der Technik ein neues Forschungsfeld ist, wird diese Idee in der Natur oft beobachtet. So bewegen sich Tierschwärme wie Vögel oder Fische in einem großen dezentral gesteuerten Verbund und agieren in diesem als MAS in dem jedes Tier einen Agenten darstellt. Neuste Entwicklungen in der Technik probieren diese Idee zu übernehmen und so z.B. eine Verbesserung in der künstlichen Intelligenz und Schwarmintelligenz zu bewirken. Eine weitere Idee ist es ganze Produktion Bereich als einen individuellen Agenten anzusehen und so die Produktionsplanung und -steuerung zu verbessern [3, 4].

Die MAS bietet viele Vorteile verglichen zur klassischen zentralen Steuerung:

**Dezentrale Struktur und Autonomie**

Dieses System hat den Vorteil das der Ausfall eines Individuums nur einen marginalen, bis keinen Einfluss auf das Gesamtsystem und deren Problemlösung hat. Andere Agenten können den Ausfall kompensieren und das System kann weiterhin bestehen. In einer dezentral gesteuerten Umgebung hätte der Ausfall der Zentralstelle eine fatale Auswirkung auf das Gesamtsystem und würde somit handlungsunfähig werden.

**Skalierbarkeit und Flexibilität**

Durch den Einsatz autonomer, unabhängiger Agenten wird eine hohe Flexibilität erreicht bei dem die Anzahl der Agenten bei Bedarf erhöht oder verringert werden kann. Dabei braucht es keine Änderung oder Anpassung der Zentrallen Steuerung.

**Kosteneffizienz**

Im Fall einer zentralen Steuerung ist eine leistungsstarke und komplexe Steuerung erforderlich, da alle Entscheidungen und Koordinationen im System für alle Agenten zentral verwaltet werden müssen. Die Form der Steuerung gibt es bei dem Multiagent System nicht, womit weniger leistungsstarke Steuerungen in jedem Agenten eingesetzt werden können.

**Echtzeitreaktionen**

Wird ein Agent mit einem Problem konfrontiert, muss er dieses nicht an eine zentrale Stelle zur Auswertung weiterleiten und auf eine Antwort warten. Stattdessen kann sofort eigenständig mit der Lösung des Problems begonnen werden.

* 1. Weg-Findungs-Algorithmen
     1. Random Drive

Im Random Drive wird im Vorfeld keine definierte Route berechnet. Der Agent fährt zufällige Wege und ohne Berücksichtigung, ob der Bereich:

* Un-/bekannt ist
* schon befahren wurde
* geografisch gut gelegen ist
* ein anderer Agent sich hier aufhält
* mit Hindernissen blockiert ist

Dabei wird eine gradlinige Bewegung ausgeführt, bis es zum Zeitablauf oder der Kollision mit einem Hindernis kommt. Dieses Ereignis führt zu einer zufälligen Änderung der Richtung. Der Vorteil ist die eine einfache Implementierung. Nachteilig sind die Ineffizienz und der Mangel an einer Zielorientierung [5].

* + 1. A\*-Algorithmus

Der A\*-Algorithmus gehört zu den informierten Such-Algorithmen und hat zum Ziel in einem bekannten Graphen den kürzesten Weg zwischen zwei Knoten zu ermitteln. Der Algorithmus ist eine Erweiterung des Dijkstra-Algorithmus. Anders als bei uniformierten Algorithmen wird zusätzlich zur Kostenfunktion eine Schätzfunktion (Heuristik) eingesetzt. Der Algorithmus ist vollständig und optimal. [6]

Der Algorithmus benutzt zu Ermittlung des optimalen Weges zwei Funktionen, der die Kosten vom Startknoten zu einem bestimmten Knoten angibt, sowie der die geschätzten Kosten von Knoten zum Ziel repräsentiert. Die Funktion ordnet jedem bekannten Knoten die Gesamtkosten als Summe von und zu. Diese Werte dienen als Grundlage für die Auswahl des nächsten zu untersuchenden Knotens. [6]

|  |  |
| --- | --- |
|  | ( 1 ) |

Zur Ermittlung der heuristischen Größe gibt es verschiedene Ansätze.

**Euklidische Distanz**  
Die Euklidische Distanz misst den Abstand zwischen zwei Konten auf einer zweidimensionalen Fläche oder kann bei Bedarf auf den Raum angepasst werden. Diese Idee wird vor allem bei kontinuierlichen Problemen eingesetzt [7].

|  |  |
| --- | --- |
|  | ( 2 ) |

**Manhattan-Distanz**  
Die Manhatten-Distanz beschreibt die Summe der absoluten Differenzen der Koordinaten. Diese Idee wird primär verwendet, wenn es sich um eine Gitter-basierten Umgebungen handelt [7].

|  |  |
| --- | --- |
|  | ( 3 ) |

**Chebyshev-Distanz**  
Die Chebyshev-Distanz ermittelt das maximum der absoluten Differenzen in der Ebene. Diese Idee findet Anwendung, wenn die Weg-Findung in alle Richtungen (inklusive Diagonal) stattfinden kann [7].

|  |  |
| --- | --- |
|  | ( 4 ) |

Die genaue Funktionsweise des Algorithmus ist in folgender Abbildung (vgl. Abb. 1) erläutert.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | * **Input:** A graph *G(V,E)* with source node and goal node end. |  |
|  | * **Output:** Least cost path from *start* to *end*. |  |
|  | * **Steps:** |  |
|  | * Initialize |  |
|  | * + *open\_list = {start}* | /\*List of nodes to be traversed\*/ |
|  | * + *closed\_list = {}* | /\*List of already traversed nodes\*/ |
|  | * + *g(start) = 0* | /\*Cost from source node to a node\*/ |
|  | * + *h(start) = heuristic\_function (start, end)* | /\*Estimated cost from node to goal node\*/ |
|  | * + *f(start) = g(start) + h(start)* | /\*Total cost from source to goal node\*/ |
|  |  |  |
|  | * while *open\_list* is not empty |  |
|  | * + m = Node on top of *open\_list*, with least *f* |  |
|  | * + if *m == end* |  |
|  | * + - return |  |
|  | * + remove *m* from *open\_list* |  |
|  | * + add *m* to *closed\_list* |  |
|  | * + for each *n* in *child(m)* |  |
|  | * + - if *n* in *closed\_list* |  |
|  | * + - * continue |  |
|  | * + *cost=g(m)+ distance(m,n)* |  |
|  | * + if *n* in *open\_list* and *cost < g(n)* |  |
|  | * + - remove *n* from *open\_list* as new path is better |  |
|  | * + if *n* in *closed\_list* and *cost <g(n)* |  |
|  | * + - remove *n* from closed\_list |  |
|  | * + if *n* not in *open\_list* and *n* not in *closed\_list* |  |
|  | * + - add n to open\_list |  |
|  | * + - g(n) = cost |  |
|  | * + - h(n) = heuristic\_function(n,end) |  |
|  | * + - f(n) = g(n)+h(n) |  |
|  |  |  |
|  | return failure |  |

Abb. 1: Pseudocode des A\* Algorithmus [8]

* + 1. Frontier-based Exploration Strategie

Die Frontier-based Exploration Strategie ist ein Vorgehen zur Aufdeckung unbekannter Bereiche innerhalb einer gegebenen Umgebung. Diese basiert auf der Idee der Unterteilung der Umgebung in bekannte und unbekannte Bereiche, welche aufgedeckt werden sollen. Die Grenzen (engl. Frontier) werden durch verschiede Kriterien unterteilt und bewertet. Eine Grenze mit hoher Bewertung wird einer grenze mit geringer Bewertung vorgezogen. Die daraus entstehende Hierarchie soll eine effiziente Aufdeckung der unbekannten Bereiche garantieren.

Die Strategie verfolgt dabei immer dem gleichen Zyklus.

**Karten-Update**  
Der Agent erstellt eine Karte für seinen Sichtbereich und ergänzt die schon bestehende Karte mit diesen Informationen.

**Grenzen-Update**  
In der Karte werden die neuen Grenzen (Bereiche an dem Unbekannte Bereiche and bekannte Bereiche grenzen) ermittelt.

**Grenzen-Auswahl**  
Die Grenzen werden mittels Distanz zum Agenten, Orientierung zum Agenten und Länge der Grenze bewertet und hierarchisch eingeordnet. Die optimale Grenze wird ausgewählt.

**Routenauswahl**  
Der Agent bestimmt die optimale Route zur ausgewählten Grenze, wobei Hindernisse umfahren werden müssen.

**Agenten Bewegung**Der Agent bewegt sich zum ausgewählten Ziel.

Nach Erreichen des fünften Schritts wird der Zyklus von vorne begonnen.

Die Grenzauswahl spielt bei der Effektivität der Strategie eine entscheidende Rolle [9].

|  |  |
| --- | --- |
|  | ( 5 ) |

Unter der Bedingung der optimalen Grenze :

|  |  |
| --- | --- |
|  | ( 6 ) |

mit:

XXX FRAGE: SOLLTE DIE SUMMER DER λ NICHT AUCH MAXIMAL 1 SEIN?

Mit der Normalisierung [9]:

|  |  |
| --- | --- |
|  | ( 7 ) |

Mit:

Die vorgehende Formel ( 8 ) und ( 9 ) beschreibt die Entscheidung [9].

Die Gewichtungsfaktoren werden für drei Bereiche gebraucht:

**Distanz**Ist eine Grenze näher, kann in diese Grenze mit weniger Zeit und Ressourcen erkundet werden

**Größe**  
Eine größere Grenze weist tendenziell auf ein größeres unbekanntes Areal dahinter hin. Dabei ist zu beachten, dass das System minimiert wird, eine große Grenze also zu einem kleinen führen muss.

**Orientierung**   
Ist ein Agent schon in eine Richtung einer Grenze gedreht, werden keine Ressourcen und Zeit zur Reorientierung aufgewendet.

Zur Bestimmung des Gewichtungsfaktoren werden in der Literatur mehrere verfahren beschrieben.

**Empirische Kalibrierung**Im rahmen einer Studie erden experimental verschieden Werteparte ausprobiert und verglichen. Die besten Ergebnisse werden ausgewählt. Dieses Vorgehen erfordert eine hohe Anzahl an Tests. XXX Quelle

**Optimierungstechniken**  
In dieser Variante werden spezielle Optimierungstechniken wie Grid Search oder Random Search eingesetzt, um Wertepaare mit optimalen Bedingungen zu ermitteln. Dieses beinhaltet die systematische automatische Durchsuchung des Parameterraums. XXX Quelle

**Maschinelles Lernen**  
Im Maschinellen Lernen werden dynamisch die Wertepaarte angepasst, um so zu ermitteln in welcher Situation welches Wertepaarte die besten Ergebnisse liefert. XXX Quelle

* 1. Wahrnehmung
     + 1. Vollständig (360°)

In einem Vollständigen Sichtfeld ist der Agent in der Lage alle Zellen, um sich herum in einer definierten Reichweite zu kartografieren. Die Ausrichtung des Agenten hat keinen Einfluss auf sein Sichtfeld.

* + - 1. Eingeschränkt (< 360°)

Der Agent kann nur einen begrenzten Bereich in Fahrtrichtung (x-Achse) wahrnehmen und kartografieren. Bereiche außerhalb des Sichtfelds des Agenten sind nicht bekannt und müssen durch eine Änderung der Fahrtrichtung ermittelt werden.

* + - 1. Reichweite

Die Reichweite ist eine Größe, die den maximalen Abstand vom Ausgangspunkt des Agenten, bis zu dem eine Kartografierung erfolgen kann, angibt. Ein Hindernis blockiert die Sicht, sodass keine Informationen über die Zellen hinter den Hindernissen vorliegen, es sei denn, diese wurden bereits zuvor erfasst.

* + - 1. Raycasting / LiDAR

Das Raycasting ist eine Simulation Methode zur Ermittlung von Objekt in einem Raum zur Bilderstellung. Die Idee basiert auf dem Aussenden einer Welle (Licht, Laser etc.) und dem zurückwerfen dieser Welle von der Oberfläche eines Objektes. Von einer Quelle wird eine Welle mit bestimmter Winkelabstand und Länge in den Raum emittierend. Diese Welle wird nun entweder durch eine Oberfläche eines Objektes zurück zur Quelle geworden, oder im Raum absorbiert. Kommt es durch eine Oberfläche zu einer Reflexion kann mittels Zeitdifferenz der Abstand dieses Punktes / Objekts im Raum ermittelt werden. Quelle XXX

Ein Sensor, z.B. ein “light detection and ranging” (LiDAR), ist eine Hardware, der diese Technik nutzt, die durch die Idee des Raycasting simuliert wird. Die nachstehende Abb. 2 veranschaulicht die funktionsweiße des Raycasting / LiDAR [10].

|  |  |
| --- | --- |
|  |  |

Abb. 2: (L) LiDAR Scan eines Raumes, (R) Plot des Scans; Inwärts gerichtet Ecke bei ca. +30° [10]

Die Änderung der Wellenlänge durch eine Bewegung des Sensors (Doppler-Effekt [11]) wird in dieser Arbeit vernachlässigt.

* + - 1. Bresenham Algorithmus

Um auf einem Raster eine Linie in Rasterform darzustellen, wird ein spezieller Algorithmus benötigt. Ein Algorithmus, der dieses Problem löst, ist der „Bresenham-Algorithmus“ [12].

Um diesen Algorithmus anwenden zu können werden zwei Felder auf einem Raster benötigt. Durch partielle Verschiebung der Start Koordinaten wird so eine vollständige Linie im Raster aufgebaut. So können bestimmte Linie mit einem Winkel zum Koordinatenkreuz dargestellt und ermittelt werden [12].

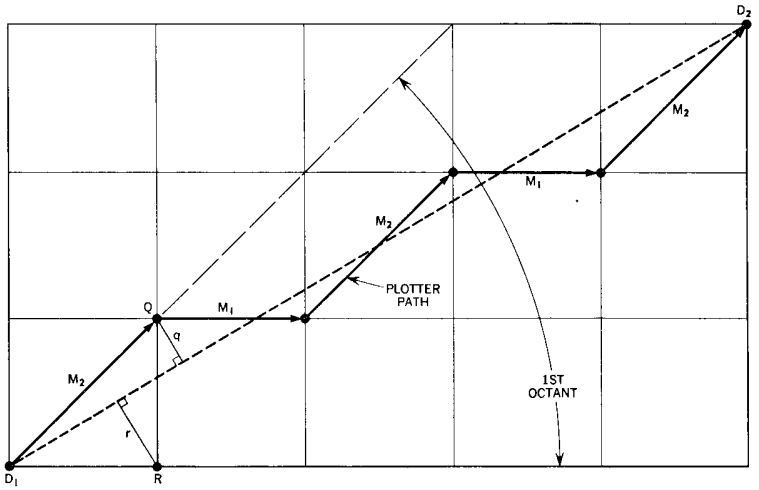


Abb. 3: Darstellung der Plot Sequenzen des Algorithmus [12]

Der Abb. 3 ist der schrittweise Ablauf des Algorithmus zu entnehmen. Der detaillierte Ablauf ist der nachstehenden Abb. 4 zu entnehmen.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | * **Input:** Two points with (x0, y0) and (x1, y1) |  |
|  | * **Output:** Coordinates for the grid patterned line |  |
|  | * **Steps:** |  |
|  | * **Initialize:** |  |
|  |  |  |
|  | *dx* = *xe* - *x* |  |
|  | *dy* = *ye* - *y* |  |
|  | *xstep* = *1* |  |
|  | *ystep* = *1* |  |
|  |  |  |
|  | if *dx* < *0*{ |  |
|  | * + *dx* = *-dx* |  |
|  | * + *xstep* = *-1* |  |
|  | } |  |
|  | * if *dx < 0{* |  |
|  | * + *dx* = *-dx* |  |
|  | * + *xstep* = *-1* |  |
|  | * } |  |
|  | * if *dy* < *0{* |  |
|  | * + *dy* = *-dy* |  |
|  | * + *ystep* = *-1* |  |
|  | * } |  |
|  |  |  |
|  | * *a* = *2*\**dx* |  |
|  | * *b* = *2*\**dy* |  |
|  | * if *dy* <= *dx{* |  |
|  | * + *f* = *-dx* |  |
|  | * + while(*x0* <> *x1*){ |  |
|  | * + - print(*x,y*) |  |
|  | * + - *f* = *f* + *b* |  |
|  | * + - if *f* > *0* { |  |
|  | * + - * *y0* = *y0* + *ystep* |  |
|  | * + - * *f* = *f* – *a* |  |
|  | * + - } |  |
|  | * + - *x0* = *x0* + *xstep* |  |
|  | * + } |  |
|  | * } |  |
|  | else{ |  |
|  | * + *f* = *-dy* |  |
|  | * + while(*y0* <> *y1*){ |  |
|  | * + - print(*x0,y0*) |  |
|  | * + - *f* = *f* + *a* |  |
|  | wenn *f* > *0*{ |  |
|  | * + - * *x0* = *x0* + *xstep* |  |
|  | * + - * *f* = *f* – *b* |  |
|  | * + - } |  |
|  | * + - *y0* = *y0* + *ystep* |  |
|  | * + } |  |
|  | * + print(*x0,y0*) |  |
|  | * } |  |
|  |  |  |

Abb. 4: Pseudocode des Bresenham-Algoritmus [12]

* 1. Umgebungsmodell
     1. Kartographische Repräsentation
        1. Layout
           1. Grid (Raster Karten)

Ein Grid ist ein grundlegender 2D kartentyp mit quadratischem Aufbau und einer Vielzahl an Zellen mit vier Seiten. Jede Zelle hat vier direkte Nachbarn sowie vier weitere benachbarte Zellen in vertikaler Richtung. Ein Agent kann sich je nach eingesetztem System zu allen acht Nachbar Zellen (Euklidische Distanz) oder vier Nachbarn (Manhattan-Distanz) bewegen. Die Position jeder Zelle kann durch zwei Koordinaten genau bestimmt werden – eine für Horizontal, eine Vertikal.

* + - * 1. HexGrid (Hexagonales Gitter)

Ein HexGrid ist ein Kartentyp, bei dem das Gitter aus einer Vielzahl von Hexagonen besteht. Jede Einzelne Zelle hat genau sechs Nachbarn. Ein Agent kann sich in alle angrenzenden Zellen bewegen.

* + - 1. Hindernisse

Das Hindernis in einer Karte stellt einen Bereich da, der von einem Agenten nicht überfahren werden kann. Ein Ausweichen des Agenten ist nötig. Die Hindernisse beeinflussen die Weg-Findung und müssen mit eingebunden werden. In einem Multigrid system kann dieselbe Zelle von zwei oder mehr Agenten simultan genutzt werden. Dem gegenüber gestellt ist das Singlegrid System, hier kann dieselbe Zelle nur von einem Agenten genutzt werde. Diese genutzte Zelle stellt zu diesem Zeitpunkt ein Hindernis für einen andern Agenten da und muss umfahren werden.

* + - 1. Randbedinungen
         1. Simulated Infinity

In der Literatur beschreibt diese Konzepte (auch „toroidal Karten“ und „Hypermaps“ bekannt) die Idee der simulierten Unendlichkeit einer Karte. Kommt ein Agent zum Rand der gegebenen Karte, kann dieser Rand passiert werden und er jeweilige Agent wird auf der jeweiligen anderen gegenüberliegenden Seite wieder eingesetzt. Ein Agent kann nie das Ende eine Karte erreichen [13].

* + - * 1. Collision Boundary

Bei der „Collision Boundary“, auch „Impassable Boundary“ oder „Rigid Boundary“ genannt, stellt der Rand einer jeder Karte ein nicht zu Überwindens Hindernis da. Wird dieses Hindernis / Rand erreicht, kann sich der Jeweilige Agent nicht mehr weiter in diese Richtung bewegen. Ein Umkehren ist nötig [13].

1. Hauptteil
   1. Technische Rahmenbedingungen

Im Rahmen dieser Arbeit werden folgende Rahmenbedingungen angenommen und gelten solange nicht expliziert anders erwähnt: Ein 2D „Grid“ / „HexGrid“ XXX (vgl. Abschnitt 2.4.1) mit „Simulated Infinity“ / „Collision Boundary“ XXX (vgl. Abschnitt 2.4.1.3) in einem „Grid System“ / „Multigrid system“ XXX (vgl. Abschnitt 2.4.1.2). Der Agent kann sich in der „Euklidische Distanz“ / „Manhattan-Distanz“ XXX (vgl. Abschnitt 2.4.1.1.1) bewegen.

Die Darstellung der Zeit wird mittels schritten (steps) realisiert. Jeder Step beschreibt dabei die Zeiteinheit, in der jeder Agent eine definierte Anzahl an Bewegungen auf der Karte durchführen kann. Bewegungen und Distanzen werden in Zellen gemessen.

Das System terminiert ordentlich, wenn XXX % der Karte kartografiert wurden. Gelingt dieses in XXX steps nicht, wird der lauf abgebrochen.

XXX vllt auch ein Verhältnis von SUMME schritten zur Kartengröße XXX

Folgende Variablen werden im Laufe der Arbeit geändert und gegenübergestellt: Größe Gird, Wegfinde algo, Anzahl Agents, Wahrnehmung (Distanz und Winkel), Kommunikationsradius (XXX), Anzahl an Bewegungen XXX

* 1. Implementierung
     1. Simulationssoftware

Zum Erstellen der Simulation wird die Programmiersprache „Python“ in der Version 3.13.3 verwendet. Zur Unterstützung des MAS wird das Framework „Mesa: Agent-based modeling in Python“ in der Version 3.1.5 angewendet. Die Darstellung wird mit dem Framework „Solara“ in der Version XXX umgesetzt. Dabei handelt es sich um eine Web-Framework zur Darstellung von Daten. XXX

Zusätzlich und inkludiert wurden folgende Pakete und Softwares genutzt:

Tabelle 1: Verwendete Software und Versionen

|  |  |  |
| --- | --- | --- |
| Name | Version | Datum |
| Python | 3.13.3 |  |
| Mesa: Agent-based modeling in Python | 3.1.5 |  |
| Solara |  |  |
| XXX |  |  |

* + 1. Simulationshardware

XXX Sollten wir gucken ob wir das brauchen

* + 1. UML-Diagramm

Zur Darstellung des UML-Diagramms wird die Software XXX verwendet.

XXX

* + 1. Durchführung Simulation

Es werden zwei Fälle der Simulation unterschieden. Wird eine Einzelsimulation grafisch dargestellt, wird das Web-Framework Solara verwendet. Dabei können folgende werte dynamisch zur Laufzeit geändert werden (vgl. XXX).

Werder eine Große Anzahl an Simulationen durchgeführt bei dem eine detaillierte grafische Betrachtung vernachlässigt werden kann, wird das in Mesa verwendete System des „Batchrunners“ verwendet. Bei dieser wird die Simulation ohne grafische Komponente ausgeführt.

1. Auswertung
   1. Benennung

XXX Akronyme beschreiben XXX

* 1. Messgrößen

Zur Gegenüberstellung der Ergebnisse werden folgende Größen definiert.

Tabelle 2: Messgrößen

|  |  |  |
| --- | --- | --- |
| **Name** | **Beschreibung** | **Einheit** |
| Zeit | Zeit Schritte, die das System benötigt, um zu terminieren | Steps |
| Weg | Summe aller Bewegungen aller Agenten bis zur Terminierung | Zellen |

* 1. Simulationen

Im Rahmen dieser Arbeit wurden folgende Parameter miteinander vergleichen:

* Konstant XXX
* Konstant XXX

1. Zusammenfassung und Ausblick

XXX

1. Anhang
   1. Rechnungen
      1. Stichprobengröße

Die Stichproben Größe *n* in einer Simulation wird wie folgt ermittelt:

|  |  |
| --- | --- |
|  | ( 10 ) |

Mit:

Für das Vertrauensniveau wird ein Wert von 95% angenommen, somit einen Wert von 1,96, sowie einer akzeptierten Fehlermarge von ±0,01.

Dabei wird *s* empirisch mit *ntest* = 50 Testläufen ermittelt. Folgende Formel wird dabei eingesetzt.

|  |  |
| --- | --- |
|  | ( 11 ) |

Daraus ergeben sich folgende Daten:

Somit wird eine Stichprobenmenge pro veränderlichen Wert von XXX Durchläufen angenommen.

* 1. Daten der Simulationen

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | A. M. Uhrmacher und D. Weyns, in *Multi-Agent Systems*, Boca Raton, CRC Press, 2018, p. VII. |
| [2] | V. Marik, O. Stepankova, H. Krautwurmova und M. Luck, in *Multi-Agtent Systems and Applications II*, Berlin, Heidelberg, Springer-Verlag, 2003, p. 5 ff.. |
| [3] | l. Marik und V. Vyatkin, „Holonic and Multi-Agent Systems for Manufacturing,“ *Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems,* Bd. 1, Nr. 1, p. 195 ff., 2007. |
| [4] | M. Wooldridge, in *An Introduction to Multiagent Systems*, Chichester, John Wiley & Sons, LTD, 2002, p. 7 ff.. |
| [5] | T. Bräunl, in *Mobile Robot Programming*, Perth, Springer Cham, 2023, p. 39 ff.. |
| [6] | P. E. Hart, N. J. Nilsson und B. Raphael, „A Formal Basis for the Heuristic Determination of Minimum Cost Paths,“ *IEEE Transactions on Systems Science and Cybernetics,* Bd. 4, Nr. 2, p. 100–107, 1968. |
| [7] | V. Kotu und B. Deshpande, in *Data Science*, Cambridge, Elsevier, 2019, pp. 65-163. |
| [8] | H. Sharma, P. Leskovsky, A. Alekseychuk, O. Hellwich, R. S. Anand, N. Zerbe und P. Hufnagl, „Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics,“ *Diagnostic Pathology,* Bd. 7, Nr. 1, p. 134, 2012. |
| [9] | W. Gao, M. Booker, A. Adiwahono, M. Yuan, J. Wang und Y. W. Yun, „An improved Frontier-Based Approach for,“ *IEEE,* Bd. 1, Nr. 1, p. 1 ff., 2018. |
| [10] | T. Bräunl, in *Robot Adventures in Python and C*, Bern, Springer Cham, 2020, p. 49. |
| [11] | D. Nolte, „The fall and rise of the Doppler effect,“ *Physics Today,* Bd. 1, Nr. 2, pp. 30-35, 2020. |
| [12] | J. E. Bresenham, „Algorithm for computer control of a digital plotter,“ *IBM Systems Journal,* Bd. 4, Nr. 1, pp. 25-30, 1965. |
| [13] | R. Miikkulainen und J. C. Príncipe, „Advances in Self-Organizing Maps,“ *7th International Wrokshop,* Bd. 1, Nr. 1, p. 247 ff., 2009. |