Mobile Schwarmintelligenz: Navigation eines Raspberry Pi Roboters mittels Schwarmintelligenz

im Studiengang Cybersecurity

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Jonas Grohe,
Steven Hartinger,
Moritz Reufsteck

Bearbeitungszeitraum: 18.10.2022 - 18.04.2023
Fachlicher Betreuer: Prof. Dr. Holger Hofmann

Erklärung zur Eigenleistung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: *Mobile Schwarmintelligenz: Navigation eines Raspberry Pi Roboters mittels Schwarmintelligenz* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, 17. April 2023
Ort, Datum

Steven Hartinger

Mannheim, 17. April 2023
Ort, Datum

Jonas Grohe

Mannheim, 17. April 2023
Ort, Datum

Moritz Reufsteck

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code Darstellungen

Zusammenfassung

Im Rahmen dieser Arbeit wird ein Roboter entwickelt, der in der Lage ist, eigenständig eine Strecke mithilfe eines Schwarmintelligenz-Algorithmus zu bewältigen. Für das Verständnis der Arbeit wird zunächst die Theorie der Schwarmintelligenz erläutert. Hierfür werden die Algorithmen Particle Swarm Optimization, Ant Colony Optimization, und Artificial Bee Colony erläutert sowie deren Stärken und Schwächen aufgezeigt. Anhand geeigneter Kriterien wird ein Artifical Bee Colony Algorithmus ausgewählt, der zur Lösung des Problems der Wegfindung auf einer Karte mit Hindernissen verwendet wird. Die Implementierung ist auf das Problem zur Wegfindung auf einer Karte mit Hindernissen angepasst. Dabei wird die stetige Gleichverteilungsfunktion verwendet, um zufällige Punkte innerhalb der Karte zu generieren, die dann auf ihre Gültigkeit überprüft werden. Die Bewertung eines Pfades wird durch die Summe der euklidischen Abstände zwischen den Punkten berechnet. Die lokale Suche des Bienenalgorithmus wurde dabei genutzt, um die Punkte in einer bestimmten Nachbarschaft gegebenenfalls zu optimieren. Das Ergebnis des Algorithmus liefert den besten Pfad der gefunden werden konnte. Für die Umsetzung des Vorhabens für den Roboter werden drei Konzepte erläutert, miteinander verglichen und anschließend das passendste Konzept anhand von Kriterien ausgewählt. Dabei werden Faktoren wie die verfügbare Hardware und vorhandene Kenntnisse berücksichtigt. Besonderes Augenmerk wird auf die Kommunikation zwischen den einzelnen Komponenten gelegt, wobei Aspekte wie Sicherheit, Authentifizierung und Validierung von Daten einbezogen wird. Hierbei wird eine Ansteuerung mittels einer REST API gewählt. Damit hierdurch übertragene Daten korrekt verarbeitet werden können, wird eine Funktion zur Übersetzung von Instruktionen für die Anwendung auf dem Einplatinencomputer implementiert und integriert. Schließlich wird die Ergebnisse der Planung mit den Hardware-Komponenten vereint, eine geeignete Plattform für den Roboter konstruiert und alles in einem Prototyp zusammengeführt.

Kapitel 1

Einleitung

1.1 Problemstellung

In der modernen Robotik gibt es immer mehr Anwendungen, in denen mehrere mobile Roboter zusammenarbeiten müssen, um eine bestimmte Aufgabe auszuführen. Ein Beispiel hierfür ist die Exploration von unbekannten Umgebungen, bei der mehrere Roboter zusammenarbeiten müssen, um eine möglichst große Fläche abzudecken. Ein wichtiger Aspekt in diesen Anwendungen ist die Fähigkeit der Roboter, effizient und koordiniert zu navigieren, um Kollisionen und unnötige Rückwege zu vermeiden. Für einen Austausch der Informationen unter den einzelnen Robotern muss eine zuverlässige Kommunikation stattfinden. Ein Ansatz, um dieses Problem anzugehen, ist die Anwendung von Pathfinding-Algorithmen, die es den Robotern ermöglichen, einen Weg durch die Umgebung zu finden, der ihre Ziele erreicht und gleichzeitig die Ressourcen wie zum Beispiel Energie und Zeit der Roboter optimiert. In dieser Arbeit soll untersucht werden, wie Pathfinding-Algorithmen in einer Umgebung mit mehreren mobilen Robotern angewendet werden können, um eine koordinierte und effiziente Navigation zu ermöglichen.

1.2 Ziel der Arbeit

Schwarmintelligenz beschreibt das Phänomen, dass ein Schwarm von Lebewesen oder künstlichen Agenten in der Lage ist, gemeinsam komplexe Aufgaben zu lösen, die für einzelne Individuen unlösbar wären. Schwarmintelligenz findet sich in der Natur bei vielen Arten, wie beispielsweise Ameisen, Bienen oder Schwärmen von Vögeln oder Fischen. Der Schlüssel zur Schwarmintelligenz liegt in der Fähigkeit der Individuen, miteinander zu kommunizieren und Informationen auszutauschen. In der Technologie wird Schwarmintelligenz zunehmend genutzt, um komplexe Aufgaben zu lösen. Mobile Schwarmintelligenz beschreibt dabei den Einsatz von Schwarmintelligenz in mobilen Systemen wie beispielsweise Robotern oder Drohnen. Diese können sich eigenständig in einer Umgebung bewegen und gemeinsam Aufgaben erledigen, die für einzelne Roboter oder Drohnen zu schwierig wären. Um Schwarmintelligenz in mobilen Systemen zu nutzen, gibt es eine Vielzahl von Algorithmen. Beispiele hierfür sind der Schwarm-Algorithmus, der Ameisenalgorithmus oder der Partikel-Schwarm-Algorithmus. Diese Algorithmen nutzen unterschiedliche Strategien, um Schwarmintelligenz in mobilen Systemen zu erzeugen. Im Rahmen dieser Arbeit soll ein geeigneter Algorithmus für die Navigation eines Einplatinencomputers durch einen Hindernisparcours mittels Schwarmintelligenz ermittelt werden. Hierfür sollen verschiedene Algorithmen vorgestellt und miteinander verglichen werden. Der ausgewählte Algorithmus soll es dem Einplatinencomputer ermöglichen, Hindernisse eines Parcours zu verarbeiten und eine optimale Route durch den Parcours zu finden. Damit die Schwarmintelligenz-Algorithmen auf dem Einplatinencomputer ausgeführt werden können, ist eine zuverlässige Kommunikation mit einem Server notwendig. Hierfür können beispielsweise WLAN- oder Bluetooth-Verbindungen genutzt werden. Die Kommunikation muss dabei praktisch und zuverlässig umgesetzt werden, um eine fehlerfreie Navigation durch den Hindernisparcours zu gewährleisten. Um den Einplatinencomputer für die Navigation durch den Parcours fit zu machen, muss dieser zu einem Roboter umgebaut werden. Hierfür werden geeignete Bauteile benötigt, die dem Roboter die Fähigkeit zum Fahren, Lenken und Orientieren geben. Die

Bewegung des Roboters kann mithilfe von Servomotoren umgesetzt werden, die eine präzise Steuerung der Bewegungen ermöglichen. Insgesamt zeigt sich, dass Schwarmintelligenz ein vielversprechendes Konzept für die Navigation von mobilen Systemen in komplexen Umgebungen ist. Durch die Nutzung von Schwarmintelligenz-Algorithmen können Roboter oder Drohnen eigenständig und effizient Aufgaben erledigen, die für einzelne Individuen zu schwierig wären.

1.3 Vorgehensweise

Zu Beginn der Arbeit wird der Begriff Schwarmintelligenz erläutert und in Bezug auf die Nutzung mobiler beziehungsweise beweglicher Geräte gebracht. Im folgenden Schritt werden verschiedene Pathfinding-Algorithmen erläutert und ihre Anwendbarkeit auf die vorliegende Problematik evaluiert. Daraufhin erfolgt eine genaue Beschreibung, inklusive Zielsetzung, des praktischen Anwendungsbeispiels. In der Planungsphase sollen Entscheidungen darüber getroffen werden, welche technischen Lösungen ausgewählt werden. Mögliche Entscheidungsfindungen sind hier beispielsweise verwendete Hardware, Bibliotheken oder Frameworks. Anschließend sollen die zuvor erlangten Kenntnisse und Entwürfe dazu genutzt werden, eine Umsetzung in Form eines Prototyps zu entwickeln. Zuletzt wird ein Fazit gezogen und die Ergebnisse kritisch reflektiert.

Kapitel 2

Schwarmintelligenz

2.1 Ansätze aus der Natur

In dieser Arbeit werden grundlegend drei verschiedene Algorithmen der Schwarmintelligenz für unsere Problematik erläutert und evaluiert. Jede dieser Algorithmen sind Analogien zu dem Verhalten von Lebewesen aus der Natur. Schwarmoptimierungsalgorithmen (SOAs) imitieren die kollektive Explorationsstrategie von Schwärmen in der Natur bei Optimierungsproblemen. Diese Algorithmen verwenden einen populationsbasierten Ansatz für die Probleme. Diese Gruppe von Algorithmen wird als populationsbasierte stochastische Algorithmen bezeichnet [**Yuce2013**].

Der erste Algorithmus ist der Bees Algorithm (BA). Dieser basiert auf der Nahrungssuche von Honigbienen. Honigbienen sammeln ihr Essen über mehrere Kilometer, weshalb ein organisierter Ablauf und Kommunikation die Schlüssel des Erfolges sind. Die Bienen gehen dabei so vor, dass sie sogenannte Späher-Bienen zur Nahrungsfindung aussenden. Sind diese Späher erfolgreich, kehren sie zurück zum Bienenstock und informieren die anderen Bienen per Tanz über den Standort des Essens. Anhand dieser Information können die anderen Honigbienen das Essen sammeln, während die Späher versuchen, weitere Blumenbeete ausfindig zu machen [**Brownlee2011**].

Ein weiterer relevanter Algorithmus zur Optimierung eines Schwarms ist die Particle Swarm Optimization (PSO). Dieser nutzt das Verhalten von

Populationsgruppen wie beispielsweise Herdentieren oder Vogel- bzw. Fischschwärmen. Dabei steht der Schwarm für die Population und jedes Mitglied der Population wird Partikel genannt. Die Partikel sind bei der Optimierung entscheidend, da sie das globale Optimum finden sollen. Dies erreichen, indem sie iterativ versucht, Partikel im Hinblick auf ein bestimmtes Qualitätsmaß zu verbessern. Dabei wird ein Partikel abhängig von seiner Position und Geschwindigkeit zu der besten bekannten Position geführt und informiert die anderen Partikel über seine Position. Falls nun ein anderer Partikel eine bessere Position als die vorherige gefunden hat, werden die anderen Partikel benachrichtigt und der Schwarm in die Richtung gelenkt. Dies geschieht so lange, bis der Schwarm innerhalb des Suchraumes die optimale Position gefunden hat [**Brownlee2011**].

Der letzte Algorithmus, der in dieser Arbeit vorgestellt wird, ist der Ant System Algorithmus. Inspiriert ist dieser Algorithmus von der Essenssuche von Ameisen. Ameisen sind so gut wie blind weshalb sie bei der Nahrungsfindung über Pheromone kommunizieren. Schüttet eine Ameise Pheromone aus, bedeutet es, dass sie Nahrung gefunden hat. Dies geschieht jedes Mal, sobald eine Ameise Essen findet. Durch das positive Feedback wissen die Ameisen also, welcher Route sie folgen müssen. Über die Zeit verfallen die Pheromone, damit alte Pfade nicht mehr gefolgt werden [**Brownlee2011**]

2.2 Mobile Schwarm Intelligenz

Durch komplexer werdende Probleme in allen Bereichen des Lebens ist es von Relevanz, dass schnelle und effiziente Ansätze zur Lösung von Problemen gefunden werden. Besonders wenn ein Informationsaustausch oder lokale Kommunikation einer Mehrzahl von Systemen stattfindet, kann Schwarm Intelligenz (oder auch Kollektive Intelligenz genannt) bei der Lösung von Problemstellungen helfen. Als Inspiration vieler Grundlagen Schwarm-basierter Systeme dient das biologische Verhalten von Vorkommnissen in der Natur, wie das von Insekten wie beispielsweise Ameisen, oder das Verhalten von in Schwärmen agierenden Lebewesen wie Fischen oder Vögeln [**Blum2008**]. Ein Schwarm, der in der Natur als kollektives Verhalten von organisierten, jedoch

dezentralisierten Populationen bekannt ist [**Kiranyaz2013**], wird im Jahr 1999 mit der Informationstechnik von [**Bonabeau1999**] in Zusammenhang gebracht und als Eigenschaft bezeichnet, die es individuellen Komponenten durch Interaktion ermöglicht, Aktionen durchzuführen, die nicht durch das alleinige Handeln einer einzeln agierenden Komponente umsetzbar sind. Interaktionen, wie beispielsweise das Abgeben von Pheromonspuren auf gefundenen Wegen zwischen dem Nest einer Kolonie von Ameisen, oder dem sogenannten "waggle dance"von Bienen, deren Schwärmer diesen praktizieren, wenn sie auf neue Futterquellen gestoßen sind [**Blum2008**, **Panigrahi2011**], können in Form anderer Kommunikations- oder Interaktionsverfahren im Netzwerk- oder Datenverkehr angewendet werden. Ersteres basiert darauf, dass einzelne Ameisen das abgegebene Pheromon anderer Ameisen als indirektes Kommunikationsmedium nutzen, um damit den kürzesten Weg zu einer Futterquelle zu finden [**Blum2008**]. Die Ameisen nutzen dabei die Dichte des Pheromons, um die Richtung zu bestimmen, in die sie sich bewegen sollen. Um dieses Verhalten auf Anwendung in der Informationstechnik übertragen zu können, werden "künstliche PheromonSSpuren geschaffen, deren Informationsgehalt dadurch genutzt werden kann, um probabilistische Entscheidungen zu treffen. Abgegeben werden diese künstlichen Spuren dann von Agenten, die die künstlichen Ameisen darstellen [**Gendreau2010**].

Ein bekannter Algorithmus, der in vielen wissenschaftlichen Arbeiten wie [**Blum2008**, **Parpinelli2011**, **Brownlee2011**] thematisiert und erläutert wird, ist die Ant-Colony-Optimization (ACO). Hierbei handelt es sich um eine metaheuristische Technik im Verhalten von Ameisen. Das Ziel der Spezies ist es, den kürzesten Weg einer Kolonie bis zu einer Futterquelle zu ermitteln [**Parpinelli2011**]. Der aus der Biologie bekannte und definierte Schwarm wird im Jahr 1999 von [**Bonabeau1999**] als Eigenschaft bezeichnet, die es individuellen Komponenten durch Interaktion ermöglicht, Aktionen durchzuführen, die nicht durch das alleinige Handeln einer einzeln agierenden Komponente umsetzbar sind. Die in der Natur vorkommenden Phänomene von Schwarmintelligenz haben zudem Forscher dazu motiviert, ïntelligent mutliagentSSysteme zu entwerfen, deren Verhalten sich stark an den Vorkommnissen und Vorbildern in der Natur orientiert. Besonders Optimierungsproble-

me, beispielsweise in den Bereichen des Routings von Telekommunikationsdaten, profitieren durch die Implementierung von Schwarmintelligenz Algorithmen [**Blum2008**]. Zudem ist das Themengebiet der Robotik ebenfalls ein Anwendungsfall, für den das ursprünglich in der Natur vorkommende Phänomen als Vorbild dient. [s. 167 Blum2008] Im Folgenden werden drei bekannte Anwendungsfelder, die als verschiedene Anwendungsfälle für Schwarmintelligenz in einer Vielzahl von Literaturwerken [**Blum2008**, **Parpinelli2011**, **Eberhart2001**, **Spears2005**] erläutert werden. Das Phänomen der Ameisen, aus dem die metaheuristische Technik ACO (Ant Colony Optimization) hervorgeht [**Blum2008**, **Panigrahi2011**, **Gendreau2010**], stellt eine effektive Methode dar, um Routing Probleme in Netzwerken zu lösen. Eine weit verbreitete und umfangreich erforschte Technik hierfür ist AntNet. AntNet basiert darauf, dass Agenten zu gleicher Zeit die Gegebenheiten, das heißt, Knoten und Kanten in einem Netzwerk erforschen und währenddessen die gesammelten Informationen austauschen. Wie im biologischen Vorbild findet dieser Informationsaustausch ebenfalls indirekt statt [**DiCaro2011**]. Eine weitere relevante Optimierungsmethode von Problemlösungen ist die PSO (Particle Swarm Optimization), die zum ersten Mal 1995 in [**Kennedy1995**] thematisiert wurde und ihren Ursprung wie der ACO ebenfalls an einem der Biologie zugehörigen Phänomen hatte. Hierbei handelt es sich um einen probabilistischen Algorithmus, bei dem individuelle Partikel als Teil eines Schwarms in einem Suchraum vorhanden sind. Jedes einzelne Partikel stellt eine potenzielle Lösung für das zu behandelnde Problem dar. Diese Partikel bewegen sich durchgehend durch den Suchraum. Das Ziel hierbei ist, dass eine optimale oder ausreichend gute Lösung gefunden wird. Während des gesamten Vorgangs und weiterführenden Iterationen beobachten Partikel jeweils die Positionen der anderen Partikel. Nach jeder Iteration speichert ein Partikel den besten Fitnesswert, der während einer Suche gefunden wurde [**Blum2008**]. Ein weiterhin relevanter Algorithmus, der unter anderem in den Bereichen Robotik und Netzwerk Routing Anwendung findet, ist der Bees Algorithm (BA) Einer der Algorithmen basiert auf der dezentralisierten Entscheidungsfindung von Bienenschwärmen bei der Nahrungssuche. Bienen praktizieren hierbei einen Tanz, passen ihn je nach Profitabili-

tät der Futterquelle an und locken damit andere Bienen an. Müssen Bienen zwischen zwei Kandidaten entscheiden, wählen sie diejenige, deren waggle-dance stärker ausgeprägt ist [**Panigrahi2011**, **Yuce2013**, **Parpinelli2011**]. Ist die Futterquelle einer anderen Biene vielversprechender als die, zu der sich eine Biene zu diesem Zeitpunkt bewegt, verwirft sie die eigene und macht sich auf den Weg zu der Quelle, die die andere Biene preisgegeben hat [**Bonabeau1999**]. Die Analyse des resultierenden Algorithmus zeigt die Existenz von zwei Typen von Bienen: Scout-Bienen und Arbeiter-Bienen. Die Arbeiter-Bienen nutzen die bereits von den Scout-Bienen gesammelten Informationen, um eine erfolgreiche Lokalisierung von Nahrungsquellen zu erreichen. Die Scout-Bienen hingegen erkunden die Umgebung stochastisch und teilen ihre gewonnenen Lösungen mit der Kolonie [**Parpinelli2011**]. Ant-Colony-Optimization (ACO), Bee-Algorithm (BA) und Particle-Swarm-Optimization (PSO) sind drei wichtige Algorithmen für Schwarmintelligenz. Sie nutzen Konzepte aus dem Verhalten von Tieren in Schwärmen oder Kolonien, um Lösungen für komplexe Probleme zu finden. ACO nutzt das Verhalten von Ameisen, um komplexe Pfade zu finden. BA nutzt das Verhalten von Bienen, um optimale Lösungen zu identifizieren. PSO nutzt Partikel-basierte Bewegungen, um optimale Lösungen zu erreichen. Sie stellen Ansätze bereit, um Probleme mithilfe eines kollektiven Verhaltens von Agenten, in diesen Fällen Ameisen, Bienen und Partikel, zu lösen und ermöglichen die Modellierung von komplexen Systemen. Zusammenfassend lässt sich sagen, dass die Erläuterung der relevanten Algorithmen für Schwarmintelligenz ein wichtiger Schritt ist, um die Konzepte mobiler Schwarmintelligenz zu verstehen. Mobiler Schwarmintelligenz beschreibt die Anwendung von Prinzipien der Schwarmintelligenz in einem mobilen Kontext, bei dem eine Gruppe von selbständigen Geräten oder Agenten zusammenarbeitet, um gemeinsame Ziele zu erreichen. Gängigerweise sind die mobilen Systeme in der Lage, sich innerhalb einer physischen Umgebung fortzubewegen und diese zu erkunden. Die Algorithmen ACO, BA und PSO haben somit hohe Relevanz für die Anwendung in mobilem Schwarmverhalten aufgrund ihrer Ursprünge in der Natur zur Problemlösung mithilfe von kollektiven Verhaltens- und Zusammenarbeitsstrategien für Modellierung und Optimierung. Es ist jedoch zu

berücksichtigen, dass die Wahl des geeigneten Algorithmus abhängig von den spezifischen Anforderungen und Einschränkungen des zu lösenden Problems ist. Es kann vorkommen, dass andere Algorithmen in bestimmten Kontexten besser geeignet sind.

Kapitel 3

Schwarmintelligenz Algorithmen

3.1 Partikelschwarmoptimierung

Die Partikelschwarmoptimierung/particle swarm optimization (PSO) wurde zuerst von Dr. Kennedy und Dr. Eberhart in 1995 vorgestellt[**kennedy1942particle**]. Sie ist inspiriert vom Verhalten von Vogelschwärmen und Fischschulen. Jeder Teil wird als Partikel bezeichnet, die Gesamtheit als Schwarm.
Die Partikel werden gleichverteilt über dem Suchbereich verteilt. Sie erhalten eine zufällige Startgeschwindigkeit. Der Suchraum hat D Dimensionen und N ist die Menge an Partikeln. Nun hat das i-te Partikel die Position $X_i = (x_{i1}, x_{i2}, x_{i3}, ... \ ...x_{iD})$ und die Geschwindigkeit $V_i = (v_{i1}, v_{i2}, v_{i3}, ... \ ...v_{iD})$. Außerdem speichert jedes Partikel seine beste Position $\mathrm{P_{ibest}}$ und erhält die insgesamt beste Position $\mathrm{P_{gbest}}$.
Jedes Partikel updated seine Position nach den folgenden Gleichungen:

$$V_i^{k+1} = wV_i^k + c_1 r_1 (P_{ibest} - X_i^k) + c_2 r_2 (P_{gbest} - X_i^k)$$

$$X_i^{K+1} = X_i^K + V_i^{K+1}$$

- k: Nummer der Iteration
- i: Nummer des Partikels

- w: Startgewicht, gibt an wie stark/schwach sich die Geschwinidgkeit pro Iteration verändert, um Divergenz zu vermeiden sollte es kleiner als 1 gewählt werden
- c1,c2: kognitives und soziales Gewicht, positive Konstanten
- r_1, r_2: uniform verteilte Zufallswerte im Bereich [0,1]

Schritte der gründsätzlichen Durchführung der Partikelschwarmoptimierung:

1. Initialisierung: Die Partikel werden gleichverteilt initialisiert und erhalten eine Startgeschwindigkeit
2. Evaluierung: Die Partikel werden nach einer Fitnessevaluierung ausgewertet
3. Update P: Der so gewonnene Fitnesswert wird dem bisher besten Fitnesswert des Partikels verglichen, ist er besser wird $P_{ibest} = P_i$. Ist dieser Wert auch besser als P_{gbest} so ersetzt $P_i P_{gbest}$
4. Update Partikel: Die Position und Geschwindigkeit werden nach den obigen Formeln verändert.
5. Wiederholung: Schritt 2 bis 4 werden nun bis Erreichen des Abbruchkriteriums wiederholt

Die Partikelschwarmoptimierung kann auf unterschiedliche Weisen umgesetzt werden. Das Abbruchkriterium kann unterschiedlich gewählt werden. Standardmäßig werden hier entweder die Anzahl der Iterationen im Vorhinein festgelegt oder der Algorithmus endet nach einer zu geringen Änderung nach mehreren Iterationen. Um die Konvergenz zu beschleunigen, kann der P_{ibest}-Wert anstatt des persönlich besten Wertes den besten Wert eines bestimmten Umfelds speichern. Der Wahl der Fitnessevaluierung kommt bei PSO ein großer Wert zukommen. Eine gut gewählte Fitnessfunktion kann die Konvergenz stark beschleunigen und somit den benötigten Rechenaufwand minimieren.
PSO ist einfach zu implementieren und hat nur wenige Parameter die gesetzt werden müssen. Es braucht keine skalierten Daten und ist einfach zu

parallelisieren [**poli2007particle**]. Eine Umsetzung mit einem großen kognitiven Gewicht c_1 führt zu einer ausgiebigen Exploration, während ein großes soziales Gewicht c_2 zu einer schnellen Konvergenz führt. Daher können diese Parameter dynamisch gewählt werden, um anfangs den gesamten Suchraum zu erforschen und später trotzdem schnell zu konvergieren [**suganthan1999particle**]. Umsetzungen von PSO lassen sich in zwei Kategorien einteilen: synchrone (S-PSO) und asynchone (A-PSO) Partikelschwarmoptimierung. Im ursprünglichen, asynchronen PSO wurden die Geschwindigkeiten und Positionen aller Partikel gelichzeitig am Ende jeder Iteration verändert. Im asynchronen PSO wird die Geschwindigkeit und Position der Partikel dauerhaft basieren auf den verfügbaren Informationen aktualisiert. S-PSO erreicht eine höhere Ausnutzung, während A-PSO eine besser Exploration, sowie eine schneller Konvergenz liefert. [**carlisle2001g**], [**ab2014synchronous**] PSO teilt viele Merkmale mit genetischen Algorithmen. Beide basieren auf einer zufälligen Initialisierung ihrer Agenten und entwickeln diese Anhand einer Fitnessevaluierung weiter. Allerdings erfolgt der Informationsaustausch grundlegend unterschiedlich. Während bei einem genetischen Algorithmus alle Agenten untereinander Informationen austauschen, erfolgt dies bei einer PSO nur in eine Richtung, vom besten Partikel zu allen anderen.

3.2 Ameisen Algorithmen

Ameisenalgorithmen sind von der Futtersuche der Ameisen abgeleitet. Ameisen kommunizieren über Stigmergie. Das heißt,sie kommunizieren nur indirekt indem sie ihre lokale Umwelt verändern. Jede Ameise scheidet auf ihrem Weg Pheromone aus, welche mit der Zeit verdunsten. Folgende Ameisen wählen wahrscheinlicher einen Weg mit größerer Pheromonkonzentration. Existieren nun zwei unterschiedlich lange Wege mit gleicher Pheromonkonzentration, entscheiden sich etwa gleich viele Ameisen für beide Wege. Da die Ameisen auf dem kürzeren Weg in gleicher Zeit allerdings öfter laufen, steigt hier die Konzentration schneller als auf dem längeren Weg. Infolgedes-

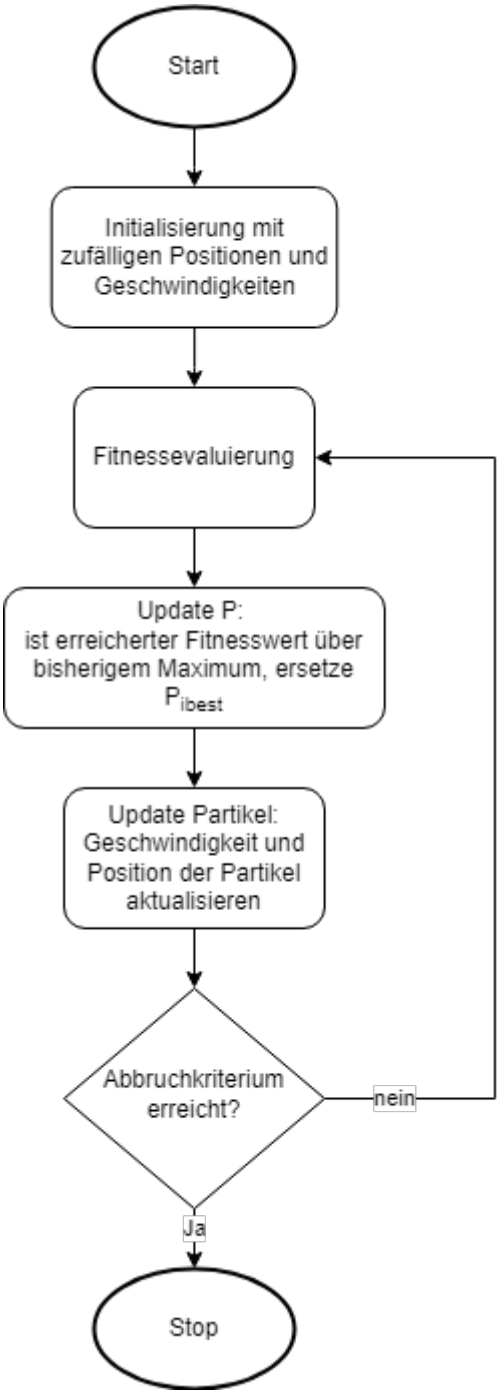


Abbildung 3.1: Flussdiagramm von PSO

sen laufen immer mehr Ameisen den kürzeren Weg und es bildet sich eine Ameisenstraße.
Es gibt viele verschiedene Umsetzungen der Ameisenalgorithmen. Der erste Ansatz wurde von Marco Dorigo 1991 vorgestellt[**Dorigo1991AntSA**] und 1996 nochmal verbessert[**484436**]. Sein Ansatz namens *Ant System* lieferte Grundlagen welche er 1997 in einem neuen System namens *Ant Colony System* weiter verbesserte[**585892**]. In 2000 veröffentlicht Stützle Hoos einen weiteren ACO Algorithmus namens *MAX-MIN Ant System (MMAS)* [**STUTZLE2000889**]. Es exisitieren noch einige weiter ACO Algortihmen, wie das *best-worst Ant System (BWAS)*[**cordon2000new**] oder das *Hyper-Cube Ant System*[**blum2004hyper**]. Diese sind allerdings nicht so verbreitet und werden daher im Folgenden nicht weiter behandelt.
Im Folgenden werden wir erst auf AS eingehen und dann die Unterschiede zu MMAS und ACS erklären.

Ant System

Jede Ameise bewegt sich von Punkt x zu Punkt y mit folgender Wahrscheinlichkeit:

$$p_{xy}^k = \frac{(T_{xy}^a)(n_{xy}^b)}{\Sigma(T_{xy}^a)(n_{xy}^b)}$$

- p_{xy}: Wahrscheinlichkeit des Übergangs von x auf y
- T_{xy}: Menge an Pheromonen auf dem Übergang von x nach y
- a: Gewicht des Einflusses von T_{xy}
- n_{xy}: Maß wie wünschenswert dieser Übergang ist
- b: Gewicht des Einflusses von n_{xy}

n_{xy} wird berechnet durch:

$$n_{xy} = \frac{1}{d_{xy}}$$

- d_{xy}: Abstand zwischen x und y.

Nun bewegt sich jeder Agent nach der von ihm berechneten Wahrscheinlichkeit, bis er den Zielzustand erreicht. Außerdem erhält jeder Agent eine Liste der bereits besuchten Zustände, die er nichtmehr besuchen darf, um einen Kreislauf zu verhindern. Dann werden die Pheromone folgendermaßen aktualisiert:

$$T_{xy}^k = (1-p)T_{xy}^k + \Delta T_{xy}^k$$

- p: Pheromonverdunstung
- k: Nummer der Ameise
- ΔT_{xy}^k: in diesem Schritt ausgeschüttete Pheromone auf dem Schritt von x nach y

ΔT_{xy}^k berechnet sich nun folgendermaßen:

$$\Delta T_{xy}^k = \begin{cases} \frac{Q}{L_k} & \text{wenn Zustand besucht} \\ 0 & \text{sonst} \end{cases}$$

- Q: Konstanter Wert, wieviel Pheromone ein Agent ausschüttete
- L_k: Länge des Pfades der Ameise

Die Wahl der Pheromonverdunstung p hat einen großen Einfluss auf den Erfolg des Algorithmus. Er bestimmt Erkundung und Ausnutzung der Agenten. Ein zu hoher Wert führt zu zuviel Erkundung und das Verlorengehen von Ameisen. Ein zu niedriger Wert schränkt die Erkundung zusehr ein und der optimale Pfad wird unter Umständen nicht gefunden.

MAX-MIN Ant System

Das MAX-MIN Ant System(MMAS) wurde eingeführt um die Performance des Systems auf großen Daten zu verbessern. Gegenüber AS gibt es zwei große Unterschiede. Es gibt es einen maximalen sowie einen minimalen Wert für die Pheromonkonzentration auf einem Übergang, T_{xy}. Die minimal und maximal Werte T_{min} und T_{max} werden für ein Problem spezifisch empirisch

gewählt[**socha2002max**]. Allerdings gibt es dafür Richtlinine die bei einer guten Wahl helfen kann[**STUTZLE2000889**].
Bei MMAS kann außerdem nur der beste Agent die Pheromone verändern.
ΔT_{xy} wird berechnet mit:

$$\Delta T_{xy} = \begin{cases} \frac{1}{L_{best}} & \text{wenn Übergang benutzt} \\ 0 & \text{sonst} \end{cases}$$

L_{best} kann jetzt entweder der beste bisherige Pfad,der beste Pfad aus der letzten Iteration sein oder eine Kombination der beiden.

Ant Colony System

Ant Colony System (ACS) bringt mehrere kleinere Verbesserungen mit sich. Die wichtigste Veränderung ist die Erweiterung des Pheromonupdates. Diese wird nun in zwei Schritten vorgenommen. Der erste Schritt ist das lokale Pheromonupdate. Jeder Agent aktualisiert nach jedem Schritt die Pheromone auf seinem letzten Übergang mit folgender Gleichung:

$$T_{xy} = (1 - \varphi) \cdot T_{xy} + \varphi \cdot T_0$$

- φ: Pheromon-Verdunstungskoeefizient, Wert zwischen 0 und 1
- T_0: Initialer Wert des Pheromons

Das Ziel des lokalen Updates ist die Erweiterung der Suche in einer Iteration. Durch das Vermindern der Konzentration an bereits besuchten Übergängen werden folgende Agenten begünstigt, andere Übergänge zu wählen und somit andere Lösung zu erzeugen. Dies vermindert die Überlagerung von gleichen Lösungen, welche aufgrund der Updates nur mit den besten Lösungen unerwünscht wird[**dorigo1997ant**]. Die zweite Updateregel, das Offline Pheromon Update wird wie bei MMAS mit dem besten Pfad insgesamt oder dem besten Pfad der Iteration durchgeführt. Die Formel ist nur leicht abgeändert:

$$T_{xy} = \begin{cases} (1-p) \cdot T_{xy} + p \cdot \Delta T_{xy} & \text{wenn Übergang auf dem besten Pfad} \\ T_{xy} & \text{sonst} \end{cases}$$

Ein weiterer Unterschied ist die Nutzung einer pseudorandomisierten Auswahl. Ist die Wahrscheinlichkeit einen Pfad zu wählen über einem gewählten Schwellwert, so wird der Agent definitiv diesen Pfad wählen. Durch die Abnutzung der besuchten Pfade kann dies nun genutzt werden, um die Suche zielgerichteter zu gestalten, ohne die anderen Pfade komplett auszuschließen[**gambardella1996so**

Schritte der grundsätzlichen Durchführung von ACO:

1. Initialisierung: Die Übergänge zwischen Start und Ziel bekommen einen Wert für n zugewiesen
2. Jede Einheit bewegt sich einen Schritt übereinstimmend mit der berechneten Wahrscheinlichkeit. Wiederholen, bis das Ziel erreicht ist
3. Die Länge der Pfade messen und Pheromonkonzentration aktualisieren
4. Die Länge aller Pfade mit dem bisherigen besten Pfad vergleichen, den besten Pfad speichern
5. Wiederholung: Schritt 2-4 wiederholen bis Abbruchkriterium erreicht ist

3.3 Bienen Algorithmen

Der Bienen Algorithmus ist von der Futtersuche der Bienen abgeleitet.ABC wurde in 2005 von Karaboga vorgestellt[**karaboga2005idea**]. Ein kleiner Teil des Bienenschwarms fungiert als Kundschafter für die Kolonie. Sie sind konstant zufällig auf der Futtersuche. Finden sie eine Futterquelle analysieren sie deren Effektivität und kehren zum Bienenstock zurück. Die Effektivität hängt von Faktoren wie der Menge, Entfernung und Zuckergehalt

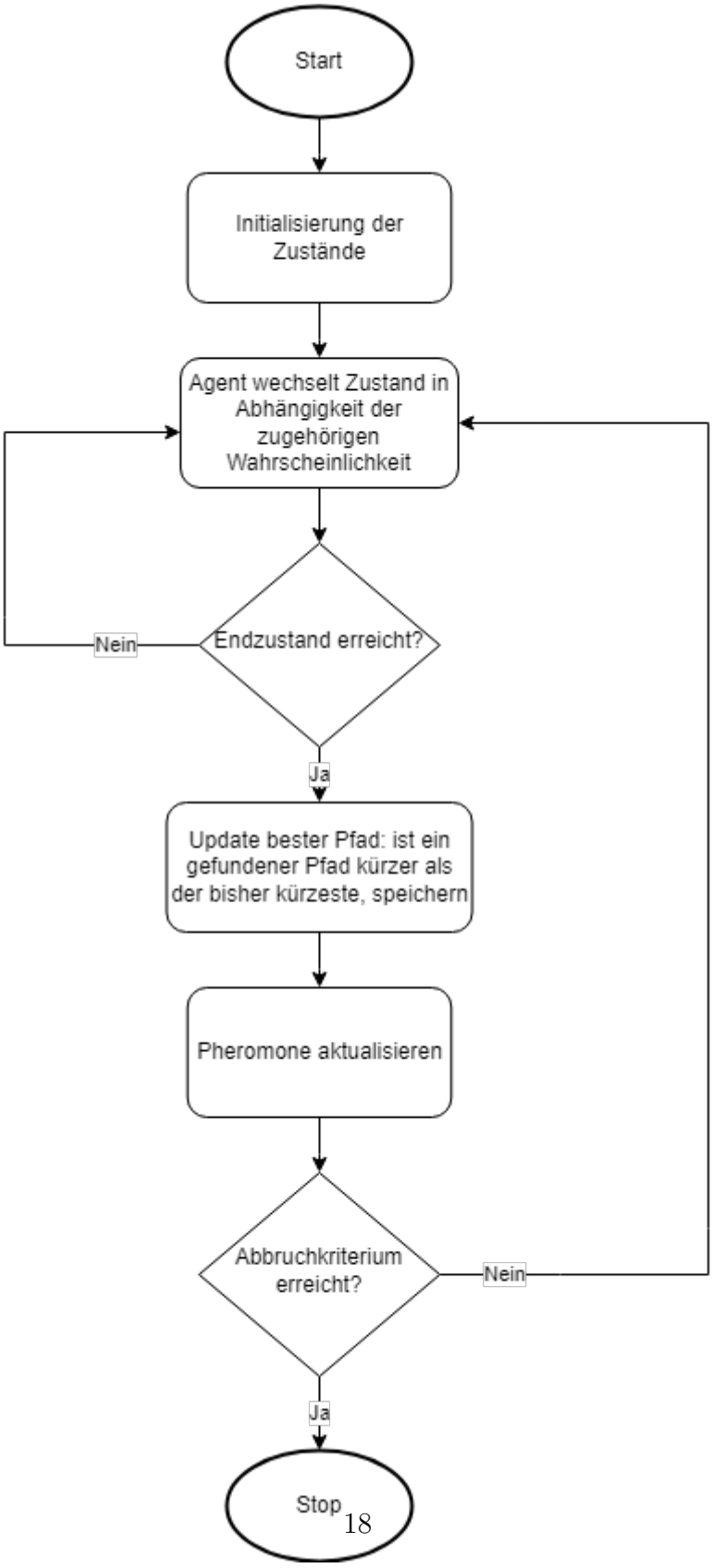


Abbildung 3.2: Flussdiagramm von ACO

ab.[**PHAM2006454**] Die Bienen, die eine effektive Futterquelle gefunden haben, führen nun einen sogenannten *waggle dance* auf dem *dance floor* aus [**Seeley+1995**]. Damit gibt sie die Wegbeschreibung zur Futterquelle weiter. Unbeschäftigte Arbeiterbienen schauen auf dem dance floor nach Wegbeschreibungen. Umso effektiver die Futterquelle, umso länger der Tanz, daher werden auch mehr Arbeiterbienen diese ausnutzen.[**KARABOGA2009108**]
ABC ereicht gute Resultate für relativ niedrige Rechenaufwände [**karaboga2008performance**]. Außerdem hat es nur sehr wenige Kontrollparameter die gewählt werden müssen.

Im Artificial Bee Colony (ABC) Algorithmus gibt es drei Gruppen an Bienen:

1. Kundschafter (Scout) Bienen: Sie fliegen zufällig auf der Futtersuche umher

2. Betrachtende Bienen: Sie stehen am dance floor und beobachten die Tänze. Sie suchen sich den besten aus und werden nun zu beschäftigten Bienen

3. Beschäftigte Bienen: Sie fliegen zu der von ihr gewählten Futterquelle. Nachdem sie zurückgekehrt sind kommunizieren sie ihre Informationen auf dem dance floor weiter

Die Position einer Futterquelle repräsentiert eine mögliche Lösung des Optimierungsproblems, die Effektivität der Futterquelle die damit verbundene Fitness. Jede beschäftige Biene ist mit einer Lösung verknüpft. Das heißt, es gibt gleich viele beschäftigte Bienen wie Futterquellen.[**pham2005bees**]
Ist eine Futterquelle erschöpft, so werden die damit verknüpften beschäftigten Bienen zu betrachtenden Bienen. Dies wird im Algorithmus durch einen Parameter *limit* umgesetzt. Für jede Lösung wird eine Versuchsanzahl pro Iteration hochgezählt, erreicht diese den *limit*-Wert wird die Lösung nichtmehr weiter besucht und die Agenten suchen sich neue Lösungen.
Der Algorithmus beginnt mit der Initialisierungphase. Es werden zufällige Futterquellen nach folgender Formel initialisiert:

$$x_{m,i} = l_i + r \cdot (u_i - l_i)$$

- $x_{m,i}$: Der Wert von m in der Dimension i
- l_i: unterer, beziehungsweise oberer Rand von $x_{m,i}$
- r: zufälliger Wert im Bereich [0,1]

Die Fitnesswerte der so erhaltenen Lösungen werden evaluiert und gespeichert.
Dann kommt die Phase der beschäftigten Bienen. Die beschäftigten Bienen suchen eine bessere Futterquelle/Lösung in der Nähe ihrer Quelle (x_m) nach folgender Gleichung:

$$v_{m,i} = x_{m,i} + \Phi_{m,i}(x_{m,i} - x_{k,i})$$

- x_k: zufällig gewählte Futterquelle
- i: zufällige Dimension
- $\Phi_{m,i}$: zufällige Zahl im Bereich [-1,1]

Nun wird die Fitness von v_m evaluiert, ist sie besser als x_m ersetzt v_m x_m als Futterquelle.
Nun folgt die Phase der Beschäftigten Bienen. Sie bekommen die Futterquellen der Beschäftigten Bienen sowie deren Fitnessevaluierung. Nun wählen sie eine Futterquelle abhängig von folgender Wahrscheinlichkeit:

$$p_i = \frac{f_i}{\Sigma_{n=0}^{N} f_n}$$

- f_i: Fitness des Lösung
- N: Anzahl aller Lösungen

Auch hier wird die Fitness des erhaltenen Wertes evaluiert und im Falle einer Verbesserung der alte Wert ersetzt.
In der Kundschafter Bienen Phase werden die Anzahl der Versuche einer

Quelle überprüft. Wurde die Futterquelle schon öfter erfolglos versucht zu optimieren, wie der *limit* Parameter vorgibt, so wird die damit verknüpfte beschäftigte Biene eine Kundschafter Biene. Kundschafter Bienen benutzen die Formel aus der Initialisierung, um zufällige Lösungen zu suchen und zu evaluieren. Treffen sie so zufällig auf eine Lösung mit einem guten Fitnesswert, so werden sie die beschäftigte Biene dieser Lösung.
Eine verbesserte Umsetzung für ABC wurde im Jahr 2014 von Karaboga veröffentlicht. Die quick Artificial Bee Colony (qABC) verändert das Verhalten der betrachtenden Bienen, um diese der realen Welt besser anzupassen und die Optimierung zu verschnellern. In der echten Welt, bekommen die betrachtenden Bienen nur eine Region durch den waggle dance mitgeteilt, und suchen sich dort die für sie beste Option zum Ausnutzen. Bei der normalen Umsetzung von ABC wird allerdings die gleiche Formel zugrunde gelegt. Daher führt der qABC Algorithmus folgende Formel für die betrachtenden Bienen ein:

$$v_{m,i^{best}} = x_{m,i^{best}} + \Phi_{m,i}(x_{m,i^{best}} - x_{k,i})$$

- $x_{m,i^{best}}$: beste gefundene Lösung in der Nachbarschaft

Das heißt, die betrachtenden Bienen bekommen von den beschäftigten Bienen einen Punkt, welchen sie als Mittelpunkt für eine Nachbarschaft wählen. Dann suchen sie in dieser Nachbarschaft den für sie besten Punkt und errechnen den dazugehörige Fitnesswert. Um die Größe der Nachbarschaft zu bestimmen, wird der Parameter r eingeführt, welcher der Radius der Nachbarschaft angibt. Bei einem Radius r=0 wird der qABC wieder zu einem standartmäßigen ABC.
Schritte der gründsätzlichen Durchführung des Bienen Algorithmus:

1. Initialisierung: Initiale Lösungen werden gewählt
2. Fitness der initialen Lösungen berechnen
3. Jeder Lösung wird eine beschäftigte Biene zugewiesen,sie sucht in der Umgebung nach einer besseren Lösung

4. Ihre Lösung wird evaluiert und, wenn besser als die bisherige, gespeichert
5. Jede betrachtende Biene wählt eine Lösung, sie sucht in der Umgebung nach einer besseren Lösung
6. Ihre Lösung wird evaluiert und, wenn besser als die bisherige, gespeichert
7. Kundschafter Bienen suchen zufällig nach neuen Lösungen
8. Ihre Lösung wird evaluiert und, wenn ähnlich gut wie bisherige, gespeichert
9. Wiederholung: Schritt 2-8 wiederholen bis Abbruchkriterium erreicht ist

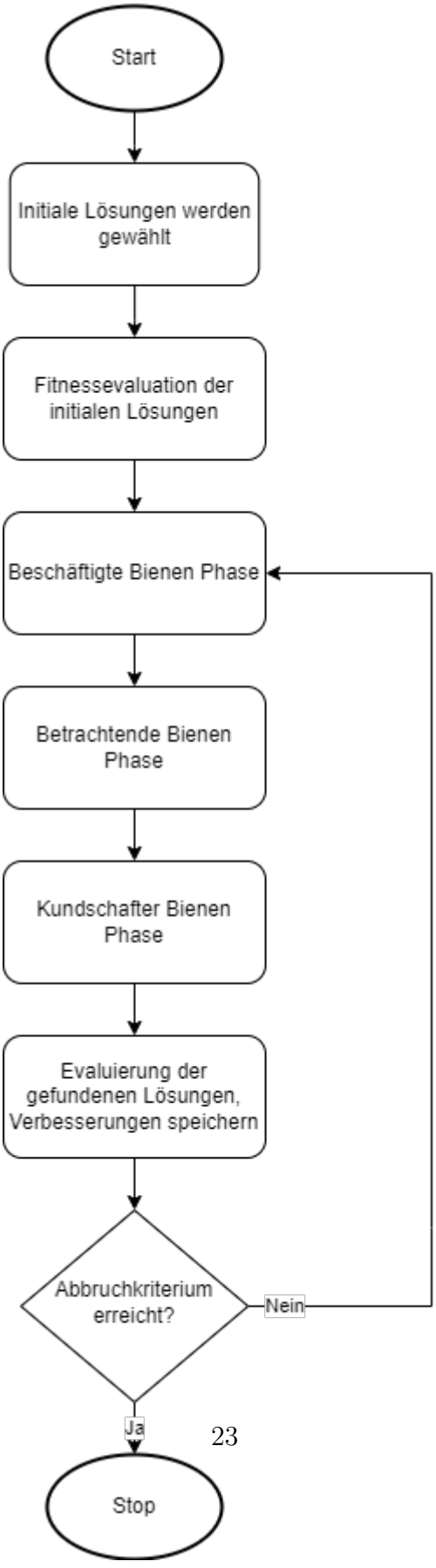


Abbildung 3.3: Flussdiagramm von ABC

Kapitel 4

Vergleich der Algorithmen

4.1 Vor- und Nachteile von Schwarmalgorithmen

Schwarmalgorithmen sind aufgrund ihrer dezentralen Struktur sehr robust. Probleme einzelner Agenten beeinflussen den Algorithmus kaum. Schwarmalgorithmen sind zudem meist einfach strukturiert und simpel zu implementieren, gerade gegenüber anderen Algorithmen in ähnlichen Anwendungsfeldern. Die Agenten können gut auf veränderungen der Umwelt reagieren, daher funktionieren die Algorithmen auch in dynamischen Umgebungen. Durch die Simplizität der einzelnen Agenten lassen sich die Algorithmen auch einfach skalieren und parallelisieren.Sie haben einen signifikanten Vorteil bei mehreren Zielen und komplexen dynamischen Veränderungen gegenüber normalen Algorithmen.

Bei großen Problemen neigen Schwarmintelligenz Algorithmen häufig zu verfrühten Konvergenzen auf lokalen Extremwerten. Um dies zu verhindern, werden häufig weiter Parameter eingeführt, was allerdings zu erhöhtem (Rechen-)Aufwand im Vorfeld führt, und die Anpassungsfähigkeit einschränkt.[**wu2022review**] Eine korrekte Wahl der Parameter kann einen großen Einfluss haben. Sind Parameter schlecht gewählt, kann der Algorithmus nur sehr langsam oder gar nicht auf eine Lösung kommen.

4.2 Vergleich der Schwarmalgorithmen

Nachdem wir oben die drei Algorithmen Particel Swarm Optimization(PSO), Ant Colony Optimization(ACO) und Artificial Bee Colony (ABC) vorgestellt haben, werden wir sie im folgenden zuerst generell und dann spezifisch für unseren Anwendungsfall vergleichen

4.2.1 Particel Swarm Optimization

PSO ist einfach zu implementieren, mit wenigen Parametern. Die Einflüsse der Parameter sind klar erkennbar, was ihre Wahl vereinfacht. Die Partikel kommunizieren nur in eine Richtung, vom allgemein oder lokalen besten zu den andern. Die Ausgabe von PSO liegt in den besten gefundenen Positionen der Partikel. PSO lässt sich gut auf Funktionsoptimierung anwenden. PSO bietet eine schnelle Konvergenz und eine hohe Effiizienz, besonders zu Beginn. Es bietet allerdings nur eine geringe Suchgenauigkeit und fällt schnell in lokale Optimas. [**yu2015swarm**] PSO ist der ausgereifteste SI-Algorithmus. Er wurde bereits häufig und für viele unterschiedliche Probleme eingesetzt.

4.2.2 Ant Colony Optimization

ACO biete eine hohe Robustheit sowie eine besonders gute Parallelisierung. ACO kann mit verschiedenen anderen heuristischen Algorithmen kombiniert werden, um es auf ein Problem abzustimmen. Die Kommunikation bei ACO funktioniert über das Verändern der Umwelt, die Ausgabe von ACO liegt in den zurückgelassenen Pheromonen. Allerdings kommt diese Flexibilität häufig mit leichten Performance-Nachteilen. Außerdem hat ACO wie PSO eine schlechte lokale Suchgenauigkeit und fällt schnell in lokale Optimas. ACO ist besonders für Routingprobleme, wie das Traveling Salesman Problem [**stutzle1997max**] oder das Vehicle Routing Problem [**gambardella1999macs**] optimal.

4.2.3 Artificial Bee Colony

ABC bietet eine sehr schnelle Konvergenz in frühen Phasen. Allerdings werden diese gegen Ende langsamer. ABC hat nur sehr wenige Parameter und ist daher sehr einfach aufzusetzen. Es bietet eine sehr gute Performance für viele Probleme. Da ABC sehr allgemein gehalten ist, bietet es auch eine große Flexibilität und Robustheit. Durch die Kombination aus lokaler und globaler Suche ist die Wahrscheinlichkeit des Findens des globalen Optimums erhöht. Allerdings braucht eine ABC-Umsetzung überdurchschnittlich viele Funktionsevaluierungen,was die Performance beeinträchtigen kann. Die Ausgabe des Algorithmus kann indirekt im Tanz der Bienen gefunden werden. Die Biene mit dem attraktivsten Tanz, beziehungsweise der Agent mit der besten Fitnessevaluation, hat die beste Lösung des Problems.

4.3 Wahl eines Algorithmus

Wir haben uns aus verschiedenen Gründen für die umsetzung einer ABC entschieden. Diese werden im Folgenden dargelegt:
Artificial bee colony ist optimal für Optimierungen geeignet, besonders für Optimierungen für nur ein Ziel (in unserem Fall der kürzeste Weg). Es hat keine großen Kontrollparameter, die erst auf das Problem eingestellt werden müssen und kann daher effizient auf Veränderungen reagieren. Laut einer Studie von Karaboga [**karaboga2008performance**] schlägt es PSO und weiter Algorithmen klar in einem klassischem Benchmark-Test von Krink [**krink2004noisy**]. ACO ist hier nicht sehr relevant, da es nicht ursprünglich für Optimierungsprobleme konzipiert wurde und daher bei den meisten Problemen nicht mit PSO und ABC mithalten kann. In einem weiter Test mit verschiedenen SI-Algorithmen (unter anderem auch PSO und ACO) schneidet ABC im Durchschnitt als der Schnellste Algorithmus ab [**ab2015comprehensive**]. Auch bei einem großen Traveling Salesmen Problem schlägt es PSO und ACO in Performance [**sabet2016comparison**]. Aufgrund er Kombination von Geschwindigkeit und einer hohen Anpassung an unser Problem haben wir uns für die Nutzung von ABC entschieden.

Kapitel 5

Anwendung von Schwarmintelligenz

5.1 Implementierung in Python

Nachdem in Abschnitt 4.1 evaluiert wurde, welcher Algorithmus für unsere Problematik am besten angewendet werden kann wurde die Implementierung des „Bees Algorithms“ in der Programmiersprache Python programmiert.

Zu Beginn des Programmes musste die Karte mit den dazugehörigen Hindernissen initialisiert werden. Die Karte wurde in Form eines 2D-Koordinatensystems umgesetzt (siehe Abbildung 5.1).

Die Menge, in der sich der Roboter bewegen darf, ergibt sich aus der Gesamtzahl der Karte K und den Hindernissen innerhalb der Karte H:

$$K\_frei = K - H$$

Als nächstes wird, wie bereits in Kapitel 3 beschrieben, die Population initialisiert. Für unseren Fall wird der Bienen Algorithmus nicht verwendet um einen optimalen Punkt zu finden, sogar um den optimalen Pfad zu finden. Dabei bildet jede Biene einen Pfad ab. Die Pfade bestehen aus k vielen Punkten in Form eines Tuples *(x,y)*. Jeder Pfad wird mit den Koordinaten *(0,0)* initialisiert und muss am Ende am Ziel *(16,16)* ankommen. Die Funktion

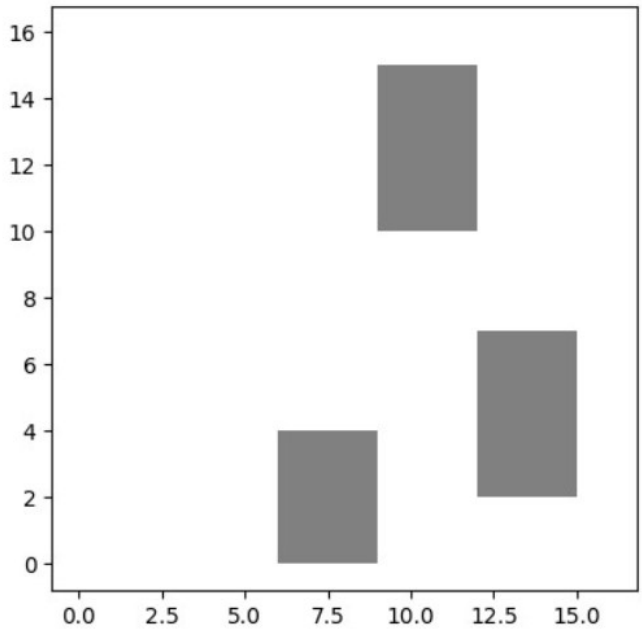


Abbildung 5.1: Initialisierte Karte mit Hindernissen

iteriert von $i = 1$ bis k, da der erste Punkt des Pfades bereits gesetzt ist:

```
def init_population():
    k=7
    i = 1
    path = [(0,0)]
    goal = (16,16)
 while i <= k:
    x = int(random.uniform(0, map_width))
    y = int(random.uniform(0, map_height))
    position = (x,y)
    if is_feasible(position, path[i-1],obstacle_list=obstacle_list):
        path.append((x,y))
        i+=1
    else:
        x = int(random.uniform(0, map_width))
        y = int(random.uniform(0, map_height))
        position = (x,y)

```

```
18  # Wirf Pfad weg falls das Ziel nach 10 Schritten nicht erreicht ist.
19  count = 0
20  while not is_feasible(goal,path[i-1],obstacle_list=obstacle_list):
21     count += 1
22     x = int(random.uniform(0, map_width))
23     y = int(random.uniform(0, map_height))
24     if is_feasible((x,y), path[i-2], obstacle_list=obstacle_list):
25         path[i-1] = (x,y)
26     if count > 10:
27         return initPopulation()
28
29  path.append(goal)
30  return  path
```

Code Darstellung 1: init_population Funktion

Im Quellcode der Code Darstellung 1 benutzt die stetige Gleichverteilung oder auch uniform distribution function genannt. Die Gleichverteilungsfunktion ist eine Wahrscheinlichkeitsverteilungsfunktion, die angibt, wie wahrscheinlich es ist, dass eine Zufallsvariable einen bestimmten Wert innerhalb eines bestimmten Intervalls annimmt. Die Gleichverteilungsfunktion ordnet jedem Wert innerhalb des Intervalls die gleiche Wahrscheinlichkeit zu. Das bedeutet, dass jede Zahl innerhalb des Intervalls mit gleicher Wahrscheinlichkeit gezogen wird.
Die Formel für die Stetige Gleichverteilung lautet:

$$f(x) = 1/(b-a),\ a \leq x \leq b$$

$$f(x) = 0,\ x < a\ oder\ x > b$$

Dabei ist a der untere und b der obere Endpunkt des Intervalls [**casella2021statistical**]. Im Falle dieser Bienen Algorithmus Implementierung wird die Stetige Gleichverteilungsfunktion verwendet, um zufällige Pfadpunkte in den Grenzen der Karte zu erstellen.
Nach der Erstellung der Punkte muss überprüft werden, ob der Punkt valide ist. Ein Punkt innerhalb eines Pfades P wird als valide angesehen, wenn die folgenden Bedingungen erfüllt sind [**Darwish2018**]:

1. $P \cap H = \emptyset$

2. $\forall i, \forall j \quad \overline{p_i p_j} \cap H = \emptyset$

Aus diesen zwei Formeln kann man ableiten, dass in einem validem Pfad weder ein Punkt innerhalb eines Hindernisses liegen darf, noch darf eine Verbindung von zwei Punkten der Menge P ein Hindernis aus der Menge H schneiden darf. Die zwei dazugehörigen Funktion sehen wie folgt aus:

```
def pointInsideObstacle(point, obstacle):
    # Überprüfen, ob der Punkt innerhalb des Hindernisses liegt
    for obs in obstacle:
        if point[0] == obs[0] and point[1] == obs[1]:
            return True

    return False
```

Code Darstellung 2: Überprüfen, ob der Punkt in einem Hindernis liegt

```
def intersects(point1, point2, obstacle):
    # Überprüfen, ob die Strecke von point1 zu point2 das Hindernis
    ↪ schneidet

    x_min = min(point1[0], point2[0])
    x_max = max(point1[0], point2[0])
    y_min = min(point1[1], point2[1])
    y_max = max(point1[1], point2[1])

    for obs in obstacle:
        if obs[0] >= x_min and obs[0] <= x_max and obs[1] >= y_min and
        ↪ obs[1] <= y_max:
            return True
    return False
```

Code Darstellung 3: Überprüfen, ob Schnittpunkt existiert

Aus diesen zwei Funktion setzt sich die Funktion *is_feasible()* zusammen, die in der Methode *init_population()* verwendet wird. Ein neuer Punkt wird also nur hinzugefügt, falls *is_feasible* wahr ist. Eine Ausnahme die existiert entsteht bei dem Anfügen des letzten Punktes *(16,16)*. Es kann passieren, dass der letzte generierte Punkt des Pfades keine valide Verbindung zu dem Zielpunkt herstellen kann, da immer ein Hindernis im Weg ist. In diesem Falle wird der letzte Punkt des Pfades wieder ersetzt durch einen neuen Punkt. Im Anschluss wird überprüft, ob dieser Punkt nun auf gültigem Wege zum Ziel kommt. Falls dies nach 10 Versuchen nicht funktioniert wird der Pfad komplett wegeworfen und neu initialisiert. Diese Variante stellte sich nach vermehrten Überprüfungen als effizientesten Weg heraus.

Für die weiteren Schritte des Bienen Algorithmus benötigen wir die Bewertungsfunktion. Die Bewertung eines Pfades wird mithilfe des Euklidischen Abstands berechnet:

$$f = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

Der Euklidische Abstand berechnet den Abstand zwischen zwei Punkten innerhalb einer Ebene [**Friedrich2019**]. Dies wird für jeden Punkt durchgeführt und aufsummiert. Die passende Funktion dazu sieht wie folgt aus:

```
1     def evaluate_path(path):
2         x_points = []
3         y_points= []
4         for i in range(len(path)):
5             x, y = path[i]
6             x_points.append(x)
7             y_points.append(y)
8         i=1
9         total_distance= 0
10        while i < len(path)-1:
11            total_distance += np.sqrt((x_points[i+1] - x_points[i]) **2 +
              ↪  (y_points[i+1] - y_points[i])**2)
```

```
12              i+=1
13              return total_distance
```

Code Darstellung 4: Evaluierungsfunktion

In Kombination mit der Evaluierungsfunktion wird die Lokale Suche des Bienen Algorithmus angewandt, um in einer bestimmten Umgebung, auch Nachbarschaft genannt, zu suchen. Dafür kann die Größe der Nachbarschaft *ngh* beliebig initialisiert werden. Nachfolgend wird für jeden Punkt des Pfades der übergeben wird, außer für Start und Ziel, ein neuer Punkt mittels Stetiger Gleichverteilungsfunktion in einer bestimmten Begrenzung erstellt. Dabei muss allerdings sichergestellt werden, dass ein valider Punkt dem Pfad hinzugefügt wird. Falls der Punkt ungültig ist wird ein neuer Punkt erstellt. Zum Schluss der Funktion wird überprüft ob die Fitness des neuen Pfades höher ist als die des alten Pfades. Wenn das der Fall ist wird der Pfad akualisiert. Auf diese Weise versucht man die bestehenden Pfade zu optimieren.

```
1     def local_search(solution, fitness):
2         ngh = 2
3         i = 1
4         while i <= len(solution) - 2:
5          count= 0
6          new_solution = solution.copy()
7          new_point =  (int(random.uniform(solution[i][0]-ngh,
           ↪  solution[i][0]+ngh)),int(random.uniform(solution[i][1]-ngh,
           ↪  solution[i][1]+ngh)))
8          new_solution[i] = new_point
9          while not path_is_feasible(new_solution):
10            new_point =  (int(random.uniform(solution[i][0]-ngh, solution
              ↪  [i][0]+ngh)),int(random.uniform(solution[i][1]-ngh,
              ↪  solution[i][1]+ngh)))
11             new_solution[i] = new_point
12             count +=1
13             if count > 20:
14                return solution
15            new_fitness = evaluate_path(new_solution)
16            if new_fitness < fitness:
```

```
17                 solution = new_solution
18             i+= 1
19         return solution
```

Code Darstellung 5: Lokale Suche des Bienen Algorithmus

Neben der lokalen Suche findet eine globale Suche statt. Hier werden erneut randomisiert eine Menge an Pfaden erstellt und per Evaluationsfunktion bewertet. Dieses Mal wird allerdings nicht die stetige Gleichverteilungsfunktion verwendet, sondern wählt man eine zufällige Stelle im Pfad aus und generiert dort eine zufällige Änderung bewirkt. Diese neu-erstellten Pfade wird im Anschluss zurückgegeben und zur Gesamtmenge der gefunden Pfade hinzugefügt.

```
1     def generate_new_solutions(solutions, best_solution):
2         new_solutions = []
3         for solution in solutions:
4             # Wenn die Lösung mit der besten Lösung übereinstimmt, wird
              ↪  sie übersprungen
5             if solution == best_solution:
6                 continue
7 
8             # Zufällige Stelle im Pfad auswählen
9             index = random.randint(1, len(solution)-2)
10 
11            # Neue Lösung generieren durch zufällige Änderung an der
              ↪  ausgewählten Stelle
12            new_solution = solution.copy()
13            new_x = new_solution[index][0]
14            new_y = new_solution[index][1]
15            new_solution[index] = (new_x, new_y)
16 
17            # Neue Lösung zur Lösungsmenge hinzufügen
18            new_solutions.append(new_solution)
19 
20        return new_solutions
```

Code Darstellung 6: Globale Suche des Bienen Algorithmus

Zuletzt müssen diese Methoden auf die Weise, wie in Kapitel 3 erläutert wurde, in einer Funktion zusammengefügt werden. Diese Funktion nimmt als Parameter die Anzahl an Kundschafter Bienen, die auch als Anzahl der Iterationen gesehen werden kann, und sieht wie folgt aus:

```
1     def bees_algorithm(num_scouts):
2      i=0
3      solutions = []
4      while i <= num_scouts:
5         path = initPopulation()
6         solutions.append(path)
7             i+=1
8
9      for i in range(num_scouts):
10     # Führe Local Search für jede Lösung durch
11        for j in range(len(solutions)):
12            solutions[j] = local_search(solutions[j],
              ↪  evaluate_path(solutions[j]))
13
14        # Berechne Fitness-Werte der Lösungen
15        fitnesses = [evaluate_path(solution) for solution in solutions]
16
17        # Aktualisiere die beste Lösung (findet niedrigsten Wert im Array)
18        best_solution_index = np.argmin(fitnesses)
19        best_solution = solutions[best_solution_index]
20
21        # Generiere neue Lösungen durch globale Suche
22        new_solutions = generate_new_solutions(solutions, best_solution)
23
24        # Sortiere Array aufsteigend
25        fitnesses = sorted(fitnesses)
26
27        # Aktualisiere die Lösungsmenge mit den neuen Lösungen
28        solutions[num_scouts:] = new_solutions
29
30     return solutions, best_solution
```

Code Darstellung 7: Gesamtheit des Bienen Algorithmus

Zu Beginn des Algorithmus wird die Population initialisiert auf die Anzahl der Kundschafter Bienen. Anschließend wird für jeden Pfad die lokale Suche durchgeführt, die Fitness berechnet und die beste Lösung aktualisiert. Zudem wird die Globale Suche angewandt und das Array aufsteigend anhand der Fitness-Werte sortiert. Die neu-generierten Fitness-Werte werden nun an den Index der Anzahl von Kundschafter Bienen des Arrays angehängt, damit die Menge der Pfade nicht zu groß wird und somit die Funktion effizienter ist. In dieser Implementierung wurde als Stoppkriterium die Menge an Iterationen gewählt, jedoch lässt sich dieses beliebig wählen. Ein weiteres sinnvolles Stoppkriterium wäre beispielsweise ein bestimmter Fitness-Wert.

Nach 30 Iterationen hat der Bienen Algorithmus den folgenden Pfad errechnet:

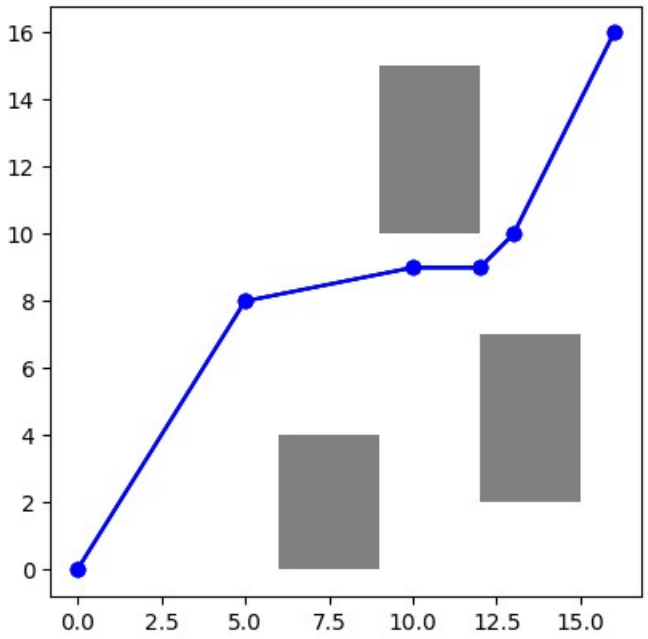


Abbildung 5.2: Schnellster Pfad mit 30 Iterationen

Zur Veranschaulichung, welche Wege der Bienen Algorithmus verwendet hat, wurde die Gesamtheit aller Pfade graphisch dargestellt:

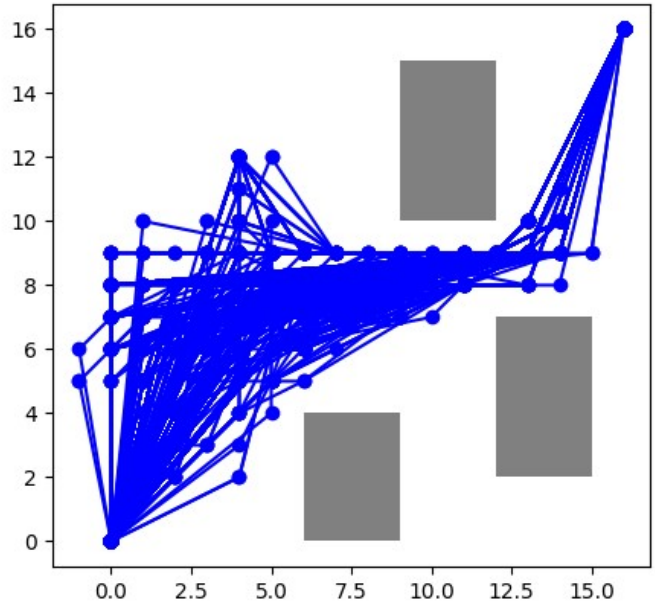


Abbildung 5.3: Initialisierte Karte mit Hindernissen

In Abbildung 5.3 ist die Funktionsweise des Bienen Algorithmus zu erkennen. Verschiedene Punkte mittels der Lokalen Suche wurden ausgetestet, um den bestmöglichen Weg zu finden.

Kapitel 6

Einplatinencomputer als Roboter

6.1 Konzepte

Die theoretischen Grundlagen des Konzepts der Schwarmintelligenz in der Robotik werden in diesem Kapitel erläutert und in Bezug auf eine Anwendung mit Raspberry Pi Einplatinencomputer gebracht. Diese sollen durch einen zufällig generierten Hindernis-Parcours fahren die vorhandenen Hindernisse hierbei umfahren. Zudem soll durch mehrere Iterationen an Durchläufen die Effizienz der Lösung verbessert werden. Für die Umsetzung und die Lösung des Problems wurden mehrere Ideen und Ansätze entwickelt und in diesem Kapitel erläutert und miteinander verglichen. Außerdem werden die einzelnen Komponenten des Konzepts erläutert und die einzelnen Schritte der Implementierung beschrieben. Die Problemstellung bei allen Konzepten soll sein, dass eine durch X- und Y-Koordinaten definierte Strecke mit einem Hindernis-Parcours so verarbeitet wird, dass durch diese ein Pfad gefunden wird, der Hindernisse vermeidet. Außerdem soll aus diesen gefundenen Pfaden der effizienteste Pfad ermittelt werden.

6.1.1 Umsetzung des Konzepts ausschließlich in Code

Die Umsetzung des Konzepts in Code ist die einfachste und schnellste Möglichkeit, um das Konzept in die Realität umzusetzen. Allerdings ist diese Variante verglichen mit einer Umsetzung mit Hardware weniger realitätsnah und weist einen geringeren Bezug zum eigentlichen Problem, der mobilen Schwarmintelligenz, auf. Für diesen Ansatz werden keine Hardware-Komponenten benötigt. Das zu lösende Problem soll mithilfe eines Ärtifiziellen Schwarmeseinen Pfad zu finden, der Hindernisse vermeidet umgesetzt werden. Als Programmiersprache soll Python verwendet werden, da Python durch diverse Module und Bibliotheken, wie zum Beispiel Numpy, Matplotlib oder Scipy, einen Vorteil gegenüber anderen Programmiersprachen bietet, mit denen sich die Umsetzung des Konzepts umständlicher umsetzen ließe. In Listing XY wird eine Methode für die Erstellung einer 20x20 Matrix exemplarisch dargestellt. Eine solche Matrix kann als eine Art ”Karte”verstanden werden, auf der Hindernisse und Wege dargestellt werden können. Die Matrix wird mit Nullen initialisiert und an den Stellen, an denen Hindernisse sein sollen, mit Einsen belegt. Mit den exemplarischen Parametern werden 18 Hindernisse zufällig gesetzt. Die Matrix wird anschließend ausgegeben.

```
1  import numpy as np
2
3  def obstacles():
4      """
5      Creates a 20x20 grid with obstacles randomly placed.
6      Grid containing 1 is an obstacle, containing 0 is empty.
7      Returns a numpy array of the grid.
8      """
9      obs = np.zeros((20,20))
10     for i in range(18):
11         obs[np.random.randint(20)][np.random.randint(20)] = 1
12
13     return obs
```

Code Darstellung 8: Matrizen Generierung

Die Abbildung, die mit dem Python Package ‘matplotlib‘ erstellt wurde, visualisiert eine Matrix, die durch die in Listing XY dargestellte Funktion erstellt wurde. Der Startpunkt ist bei den Koordinaten (0/0) als grüner Punkt angezeigt. Das Ziel am Punkt (20/20) als roter Punkt. Die weiße Fläche stellt den freien und somit durchquerbaren Bereich und die schwarzen Flächen den durch Hindernisse blockierten dar. Diese Daten sind die Ausgangslage für einen darauf angewandten Schwarmintelligenz Algorithmus dar, dessen Output den bestmöglichen Pfad durch diese Fläche ermitteln soll siehe Abbildung 6.1.

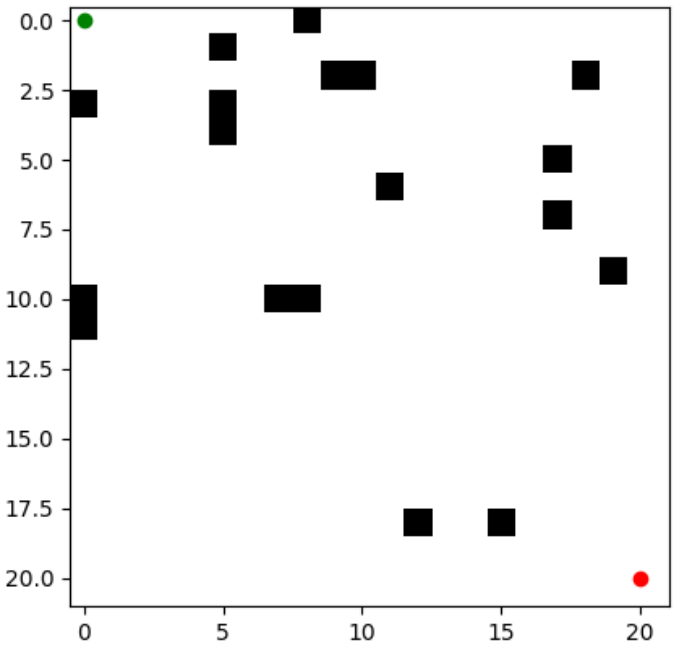


Abbildung 6.1: Visualisierung der Matrix

In Folge der Ermittlung kann der optimale Pfad verwendet werden, um den effizientesten Weg durch die Matrix, beziehungsweise die Hindernisse grafisch darzustellen.

6.1.2 Umsetzung des Konzepts in der physischen Variante

In dieser Implementierung wird ein Schwarmintelligenz-Algorithmus verwendet, um einen optimalen Pfad basierend auf Daten aus der realen Umgebung zu finden. Im Gegensatz zur reinen Code-basierten Variante werden die Daten nicht zufällig generiert, sondern von Sensoren des Roboters in der physischen Umgebung erfasst. Denkbar wären hier beispielsweise die Nutzung von Ultraschallsensoren, die an Einplatinencomputer wie dem Raspberry Pi an den vorhandenen GPIO-Pins angeschlossen werden können. Die Sensoren sammeln regelmäßig Daten, die in einer Datenbank gespeichert und periodisch abgefragt werden könnten, um sie dem Schwarmintelligenz-Algorithmus zur Berechnung des optimalen Pfades zur Verfügung zu stellen. Der Algorithmus liefert dann den optimalen Pfad, den der Roboter durch die Umgebung nutzen kann. Dieser soll dann in der Lage sein, den Pfad so zu durchfahren, dass er nicht mehr auf die Sensoren zur Erkennung der Hindernisse angewiesen ist. Sowohl beim Erlangen der Daten, wobei der Roboter an einem Startpunkt beginnt, die Umgebung zu durchfahren, als auch beim Durchfahren ohne die Sensoren, müssen die Koordinaten des Roboters zum jeweils aktuellen Zeitpunkt bestimmt werden können. Dies ist nur dann möglich, wenn anhand der Umdrehungen der Servomotoren, die zur Fortbewegung genutzt werden, ermittelt werden kann, wie weit und in welche Richtung sich ein Roboter jeweils fortbewegt hat. Für die Ermittlung wird hierfür weiterhin die Größe der verwendeten Räder benötigt. Sind diese beiden Messgrößen bekannt, können anhand dieser die Position und auch Anweisungen zum Fahren übermittelt werden.
Die Qualität der Daten hängt hierbei stark von der verwendeten Hardware ab. Zum einen, weil anhand von Umdrehungen der Servomotoren bestimmt wird, wo sich ein Roboter zum jeweiligen Zeitpunkt befindet und zum anderen von der Genauigkeit der verwendeten Ultraschallsensoren, wenn diese Hindernisse in ihrer Umgebung detektieren. Als zusätzlicher Schritt, welcher bei der Code-basierten Variante nicht anfällt, ist in dieser Umsetzung ein zusätzlicher Schritt notwendig, um die erfassten Daten in eine Form zu bringen,

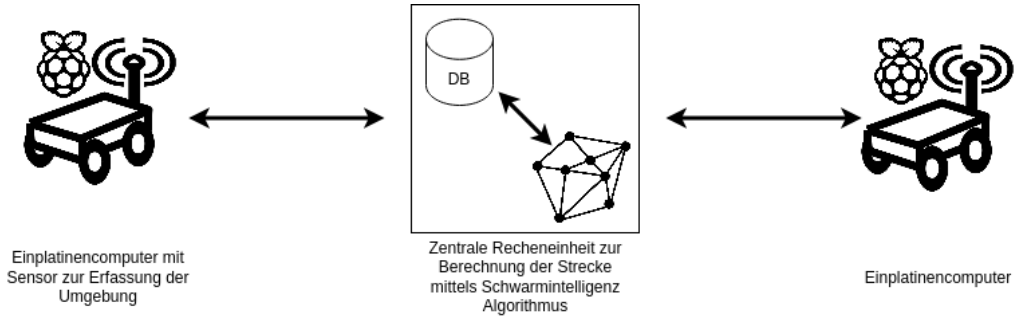


Abbildung 6.2: Übersicht Code-basierte Variante

die der Schwarmintelligenz-Algorithmus verarbeiten kann. Denkbar wäre hier ebenfalls die Nutzung von Bibliotheken wie beispielsweise ‘numpy‘, um eine Matrix in der Größe des Kartenbereichs zu erstellen. Anhand der aktuellen Position des erfassenden Raspberry Pi Roboters könnten in Abhängigkeit des Abstands zu einem Hindernis die entsprechenden X- und Y-Koordinaten in der Matrix auf 1 gesetzt werden. Eine solche Matrix könnten dann wie in der Code-basierten Variante an den Schwarmintelligenz-Algorithmus zur Verarbeitung übergeben werden.

6.1.3 Umsetzung des Konzepts hybrider Variante

Bei der Umsetzung in hybrider Variante handelt es sich um ein gemischtes Vorgehen aus der reinen Code-basierten Variante und der physischen Variante. Die Daten der zu verarbeitenden Umgebung werden wie bei der Code-basierten Variante durch einen Algorithmus generiert. Nach Verarbeitung der Daten erfolgt jedoch ein Schritt in die physische Umsetzung. Der durch den verwendeten Schwarmintelligenz Algorithmus ermittelte Pfad wird nach der Verarbeitung in Instruktionen übersetzt, die der Roboter in der physischen Umgebung ausführen kann. Hierfür ist eine Verbindung zwischen dem Algorithmus und dem Roboter notwendig. Diese kann beispielsweise durch eine WLAN-Verbindung hergestellt werden. Weiterhin muss eine Schnittstelle geschaffen werden, über welche der Client, der den Algorithmus und die Berechnungen ausführt, mit dem Roboter kommunizieren kann. Wenn auf das Kriterium der Echtzeit verzichtet werden kann, eignet sich hierfür eine

reguläre API Schnittstelle. Der Raspberry würde in diesem Fall einen Endpunkt darstellen, an den Instruktionen in Form des gewählten Datenformats wie beispielsweise JSON oder XML gesendet werden. Diese Schnittstelle würde sich zudem auch für die Übermittlung von Signalen wie Start, Stopp oder Reset eignen.

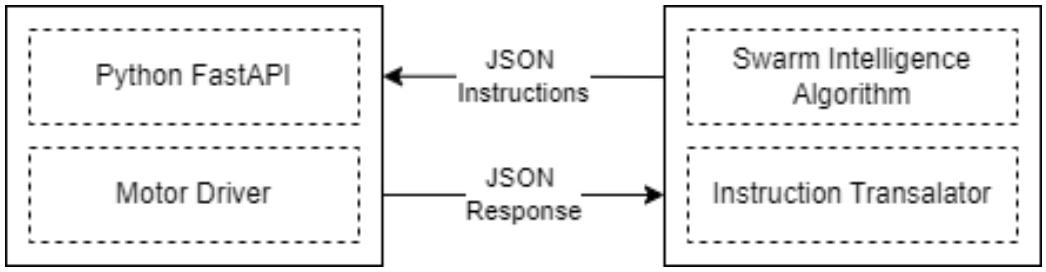


Abbildung 6.3: Übersicht hybride Variante

6.2 Auswahlkriterien für das verwendete Vorgehen

Bei der Auswahl von Kriterien für den Vergleich zwischen der reinen Codebasierten Implementierung, der physischen Implementierung und der hybriden Implementierung des Konzepts der mobilen Schwarmintelligenz werden die folgenden Aspekte berücksichtigt:

- Realitätsnähe: Realitätsnähe im Kontext der Implementierung eines Schwarmverhaltens bezieht sich auf die Fähigkeit der Implementierung, das tatsächliche Verhalten des Schwarmes in der realen Umgebung genau vorherzusagen. Es geht darum, wie nahe die Simulation der realen Umgebung kommt und wie gut die Implementierung die Gegebenheiten und Bedingungen der realen Welt berücksichtigt.

- Komplexität: Komplexität im Kontext der Implementierung sowie einer möglichen physischen Umsetzung bezieht sich auf den Schwierigkeitsgrad der Implementierung des Schwarmintelligenz-Algorithmus sowie der physischen Umsetzung. Hierbei werden die Kriterien Program-

mieraufwand, zusätzlich benötigte Kenntnisse und die Komplexität der benötigten Hardware berücksichtigt.

- Kosten: die Kategorie Kosten bezieht sich in erster Linie auf eine Implementierung der physischen Variante. Hierbei werden die Kosten für die benötigte Hardware berücksichtigt. Es soll erwähnt werden, dass eine Implementierung mit bereits vorhandenen und durch die Lehrstätte zur Verfügung gestellten Ressourcen durchgeführt werden soll.

- Skalierbarkeit: Skalierbarkeit bezieht sich auf die Fähigkeit der Implementierung, auf größere oder kleinere Umgebungen angepasst zu werden. Das bedeutet, dass die Implementierung in der Lage sein sollte, auf unterschiedliche Schwarmgrößen, räumliche Gegebenheiten, unterschiedliche Sensordaten oder andere Anforderungen anpassbar zu sein.

- Geschwindigkeit: Geschwindigkeit bezieht sich in diesem Zusammenhang auf die Fähigkeit des Algorithmus, Eingabedaten schnell zu verarbeiten und Entscheidungen zu treffen. Eine Implementierung mit hoher Geschwindigkeit kann schnell auf Änderungen in der Umgebung oder auf neue Anforderungen reagieren und somit effektiver arbeiten.

- Visualisierung: Visualisierung bezieht sich darauf, wie die Ergebnisse der Implementierung dargestellt und realitätsnah dargestellt werden können. Es soll betrachtet werden, wie gut das Verhalten des Schwarmes in der realen Umgebung simuliert werden kann und wie gut die Ergebnisse der Implementierung dargestellt werden können.

Tabelle 6.1: Entscheidungsmatrix: verwendetes Konzept

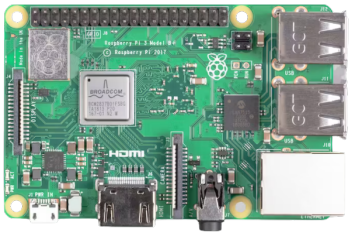
	Code-basiert	**Physisch**	**Hybrid**
Kriterium	Punkte	Punkte	Punkte
Realitätsnähe	1	5	3
Komplexität	4	1	3
Kosten	5	3	4
Skalierbarkeit	4	2	3
Geschwindigkeit	4	2	3
Visualisierbarkeit	2	5	4
Summe	20	18	20

Die Punkte der einzelnen Kriterien wurden in einer Skala von 1 bis 5 vergeben. Dabei steht 1 für die schlechteste Bewertung und 5 für die beste Bewertung. Die Summe der Punkte der einzelnen Kriterien ergibt die Gesamtpunktzahl für das jeweilige Konzept. Die ermittelten Punkte liegen wie in Tabelle 6.1 alle sehr nah beieinander. Während die Umsetzung des Vorhabens in der physischen Variante einen erhöhten Aufwand und gegenüber den anderen Vorgehensweisen mit einer deutlich höheren Komplexität einhergeht, bietet sie gleichzeitig die beste Realitätsnähe. Die Umsetzung in der Code-basierten Variante ist hingegen mit einem geringeren Aufwand verbunden, bietet jedoch die geringste Realitätsnähe. Die Umsetzung in der hybriden Variante bietet eine gute Realitätsnähe und ist mit einem vergleichbaren Aufwand verbunden. Eine Umsetzung in der hybriden Variante wird daher als Basis für die weitere Implementierung des Vorhabens verwendet.

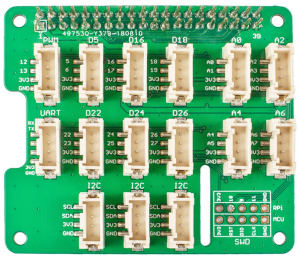
6.3 Hardware

In diesem Abschnitt werden die verwendeten Hardware-Komponenten für die Umsetzung der hybriden Variante beschrieben. Es ist zu erwähnen, dass bei der gewählten und verwendeten Hardware vorzugsweise auf Materialien zurückgegriffen wurde, die bereits durch die Lehrstätte zur Verfügung gestellt wurden. Möglicherweise gibt es bessere Alternativen, die zum Zeitpunkt der

Umsetzung jedoch nicht zur Verfügung standen. Die verwendeten Hauptkomponenten sind ein Raspberry Pi 3, ein Grove Base HAT und ein Servo Motor und ein eigens dafür konstruiertes und anschließend mit einem 3D-Drucker angefertigtes Gehäuse, das alle Komponente vereint. Die Komponenten sind in Abbildung 6.4 dargestellt.

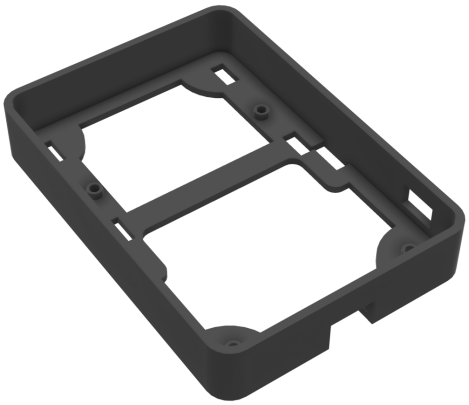


(a) Raspberry Pi 3 [**raspberry-pi**]



(b) Grove Base HAT [**seeedstudio**]

(c) Servomotor [**addicore_fs90r_micro_servo**]



(d) Roboter Gehäuse

Abbildung 6.4: Verwendete Bauteile

Der Raspberry Pi dient hierbei als Einplatinencomputer, der die Steuerung des Roboters übernimmt. Der Grove Base HAT, in Abbildung 6.4b dargestellt, bietet die Möglichkeit verschiedene Sensoren und Aktoren anzuschließen. Servomotoren, wie in Abbildung 6.4c dargestellt, dienen hierbei als An-

trieb des Roboters. Das Gehäuse, in Abbildung 6.4d dargestellt, vereint alle Komponenten und bietet eine einfache Montage und Demontage der einzelnen Komponenten. Die Wahl des Servomotors ist aufgrund der geringen Größe und der niedrigen benötigten Versorgungsspannung auf ein Modell dieser Bauart gefallen. Außerdem sind diese Art von Servomotoren günstig erhältlich sowie energieeffizient [**Sustek2017**]. Die Wahl des Grove Base HAT ist aufgrund der einfachen Verwendung und der Möglichkeit, verschiedene Sensoren und Aktoren anzuschließen, auf dieses Modell gefallen. Bei diesem Bauteil handelt es sich um eine Addon Board, das es erlaubt, Bauteile wie Sensoren und Aktoren über einen einfachen Steckverbinder anzuschließen [**seeedstudio**].

6.4 Steuerung und Kommunikation

Für die Kommunikation des Roboters stehen verschiedene Übertragungs- und Kommunikationsmöglichkeiten zur Verfügung. Aufgrund der Anforderung, dass das Gerät mobil betrieben werden soll, eignet sich eine kabellose Variante.
Vergleichen werden hierfür die Möglichkeiten MQTT, REST API und Web Socket. Für den Anwendungsfall relevant sind die folgenden Eigenschaften:

- Einfache Implementierung
- Keine zusätzliche Hardware notwendig
- Flexibilität
- Skalierbarkeit
- Zuverlässigkeit

Während sich Kriterien für den Einsatz von Robotik hauptsächlich auf eine hohe Zuverlässigkeit, geringe Latenzen sowie eine hohe Flexibilität beziehen [**AMARAN2015400**], werden im Rahmen dieser studentischen Arbeit andere Kriterien als relevant angesehen, mit unter, weil diese für die Umsetzung

des Vorhabens eine Erleichterung mit sich bringen. So ist die einfache Implementierung ein wichtiges Kriterium, da die Umsetzung in einer begrenzten Zeit erfolgen soll. Die zu verwendende Hardware soll sich, wenn möglich auf bereits vorhandene Komponenten beschränken. Flexibilität ist insofern relevant, als bei der Umsetzung des Vorhabens eine hohe Anpassungsfähigkeit an die Umgebung gewünscht ist. Wenn Änderungen notwendig werden, soll diese möglichst einfach umsetzbar sein, ohne dass die gesamte Umsetzung neu durchgeführt werden muss. Skalierbarkeit soll möglich sein, sodass das System bei einer Erweiterung der Anzahl Geräten, auf das es verteilt wird, in gleicher Weise funktionieren soll. Zuverlässigkeit ist ein wichtiges Kriterium, da die Kommunikation zwischen den einzelnen Geräten nicht unterbrochen werden darf.

Tabelle 6.2: Entscheidungsmatrix: Verwendete Kommunikationsprotokolle

	MQTT	**REST API**	**Web Socket**
Kriterium	Punkte	Punkte	Punkte
Einfache Implementierung	2	4	3
Keine zusätzliche Hardware	1	5	5
Flexibilität	5	3	4
Skalierbarkeit	4	4	4
Zuverlässigkeit	5	4	4
Summe	17	20	20

Die Punkte der einzelnen Kriterien wurden in einer Skala von 1 bis 5 vergeben. Dabei steht 1 für die schlechteste Bewertung und 5 für die beste Bewertung. Die Summe der Punkte der einzelnen Kriterien ergibt die Gesamtpunktzahl für die jeweilige Technologie. Obwohl in den Bereichen Robotik und IoT häufig MQTT verwendet wird [**AMARAN2015400**], wird es für diese studentische Arbeit nicht in Betracht gezogen. Nicht zuletzt, weil keine Vorkenntnisse der bearbeitenden vorhanden sind.

6.4.1 REST API

Unter Einbeziehung der in Tabelle 6.2 aufgezeigten Ergebnisse der Entscheidungsfindung für eine geeignete Möglichkeit der Kommunikation mit dem verwendeten Raspberry Pi Einplatinencomputer, fällt die Wahl für die Implementierung einer REST API Schnittstelle auf die in Python zu implementierende Lösung *FastAPI* [**tiangolo_fastapi_features**]. In Abbildung 6.5 werden die für das Vorhaben benötigten API-Endpunkte in Form einer durch die Umsetzung automatisch generierten API-Dokumentation aufgezeigt.

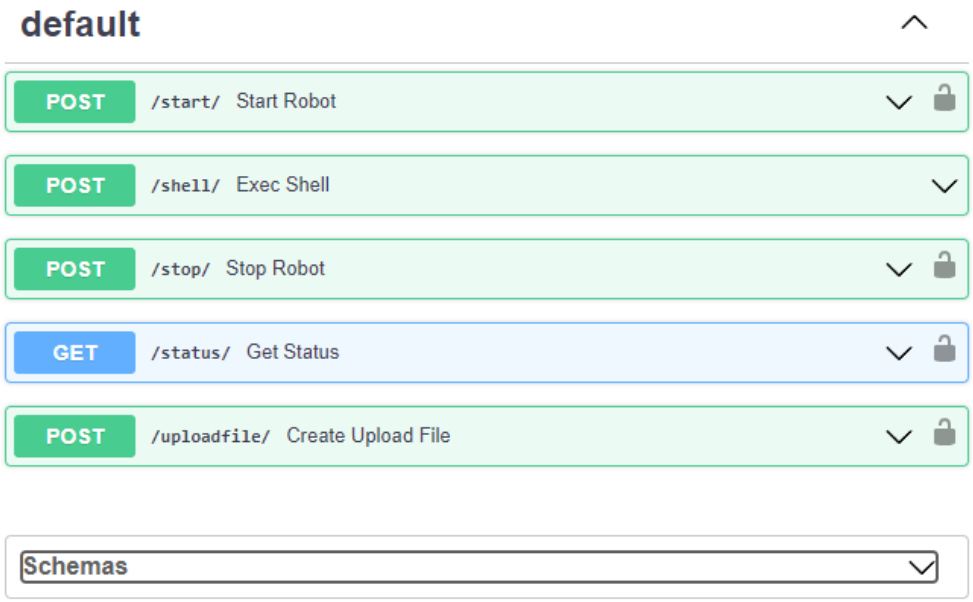


Abbildung 6.5: API Docs

Bei den implementierten Endpunkten handelt es sich um die in Abbildung 6.5 dargestellten. Folgende Funktionalitäten werden durch die Endpunkte bereitgestellt:

- **/start/**: dieser Endpunkt stellt die Funktionalität bereit, den Roboter unter Verwendung einer vorhandenen Instruktions-Datei zu starten. Dieser Vorgang wird nur dann ausgeführt, wenn noch kein aktiver Fahrvorgang existiert. Ist keine valide Instruktions-Datei vorhanden wird der Vorgang ebenfalls nicht gestartet.

- **/stop/**: dieser Endpunkt stellt die Funktionalität bereit, einen fahrenden Roboter zu stoppen und somit die Ausführung einer Instruktions-Datei zu unterbrechen. Diese Funktion könnte dann von Relevanz sein, wenn das Fahrverhalten Fehler aufweist.

- **/shell/**: dieser Endpunkt liefert die Möglichkeit, reguläre Linux Shell Befehle über die API-Schnittstelle an den verwendeten Raspberry Pi zu übermitteln. Dies ist sowohl mit einzelnen, als auch Aneinanderreihung von Befehlen und Parametern möglich. Hiermit kann der Raspberry beispielsweise neu gestartet oder heruntergefahren werden. Einen Vorteil bietet der Endpunkt insofern, als keine gesonderte SSH-Verbindung für das Ausführen kleinerer Befehle aufgebaut werden muss.

- **/status/**: dieser Endpunkt stellt eine Funktionalität bereit, mit der der aktuelle Status eines Roboters angefragt werden kann. Zum Zeitpunkt der Implementierung beschränkt sich dieser lediglich auf die beiden Zustände *active* und *inactive*. Weiterhin denkbar wären jedoch die Bereitstellung zusätzlicher Informationen wie dem Fortschritt einer aktuell abzuarbeitenden Instruktions-Datei.

- **/uploadfile/**: dieser Endpunkt stellt die Funktionalität bereit, mit der ein Anwender die Instruktionen, der durch den Schwarmintelligenz-Algorithmus errechneten Pfadinformationen, übertragen kann. Es ist zu beachten, dass ein Roboter jeweils einen Datensatz an Instruktionen verarbeiten kann und zum aktuellen Zeitpunkt keine Warteschlange für die Aneinanderreihung mehrerer Instruktions-Dateien vorgesehen ist. Wählt der Anwender eine valide Datei für die Übertragung aus, wird diese nach vorheriger Überprüfung in ein */tmp/* Verzeichnis übertragen. Möglicherweise bereits existierende Instruktionen werden hierbei überschrieben. Befindet sich der Roboter zum Zeitpunkt der Anfrage im Zustand *active*, ist ein Übermitteln einer neuen Instruktions-Datei nicht möglich.

6.4.2 API Sicherheitsaspekte

Da der durch den Raspberry bereitgestellte Endpunkt eine Möglichkeit bietet, den Raspberry und die daran angeschlossene Hardware zu steuern, gilt es einige Grundsätze zu beachten, dass durch die Implementierung dieser Schnittstelle keine Schwachstellen offengelegt werden. Besonders, da eine Shell-Kommunikation, zur erweiterten Steuerung, sowie dem Sammeln von Debugging-Informationen genutzt werden soll.

API Key Authentifizierung

Eine gängige Maßnahme um API-Endpunkte gegen unberechtigte Nutzung und gegebenenfalls schadhafte Anfragen zu schützen, ist die Implementierung der Abfrage eines API-Keys zur Authentifizierung des Anwenders [**De2017**]. Aus diesem Grund werden die Routen durch die Authentifizierung mittels eines API-Keys geschützt, sodass eine Ausführung nur dann erlaubt ist, wenn ein valider API Key im Header der jeweiligen Anfrage mitgeliefert wird. Eine valide Anfrage, wie sie vom Endpunkt des Raspberry Pis akzeptiert wird, ist in 9 dargestellt. Die verwendete Länge des API Keys wird anhand der Empfehlungen aus [**NISTSP800-57pt3r1**] gewählt und in der Implementierung umgesetzt. Weiterhin wird der API Key nicht in den geschriebenen Code, sondern in die Systemumgebungsvariablen des ausführenden Systems integriert. Innerhalb des Codes wird diese Variable zu dem Zeitpunkt aufgerufen, wenn eine Überprüfung des API Keys notwendig ist.

```
curl --location --request POST 'http://192.168.178.108:8080/shell/' \
--header 'X-API-KEY:
↪  8k60dJhd8U6sZiM7L1nG1Q2ySiHv6G32NcP0Z48vm3S2sJ19BmUdK05Gd8hZoP1y' \
--header 'Content-Type: application/json' \
--data-raw '{"command" : "pwd"}'
```

Code Darstellung 9: Curl API Request

Eine Methode eines API-Endpunkts, der durch die Abfrage des API Keys abgesichert ist, ist in 10 dargestellt.

```python
@app.post("/shell/")
async def exec_shell(request: Request):
    """
    Triggers a shell command.
    Body needs to have a "command" key with
    the command to be executed as value.
    """
    body = await request.json()
    if 'command' in body:
        command_list = body['command'].split()
        message = run_command(command_list)
        return {"message": message}
    else:
        return {"message": "No command sent."}
```

Code Darstellung 10: Geschützter Endpunkt

Validierung der JSON Instruktionen

Neben API-Anfragen, die eine direkte Auswirkungen auf angeschlossene Hardware-Elemente haben können, handelt es sich bei dem Endpunkt *upload-file* um eine Schnittstelle, bei dem der Anwender einen Request ausführen kann, bei dem der Anwender eine JSON-Datei auf den Raspberry Pi übertragen kann. Die Möglichkeit dieses Vorgangs bietet potenziell die Möglichkeit, Dateien und Dateiformate zu übertragen, welche nicht dafür vorgesehen sind oder sogar schadhafte Auswirkungen auf das Zielsystem haben können. Aus diesem Grund wird eine Validierung der JSON-Datei als notwendig betrachtet. Zudem bietet sich hierdurch der Vorteil, dass die Struktur und der Aufbau der JSON-Datei im gleichen Zug überprüft werden kann und so von vornherein keine inkorrekten Dateien, bei deren Ausführung es zu Fehlern kommen könnte, auf den Raspberry Pi übertragen werden können.

```python
if not file:
    return {"message": "No upload file sent"}

if not file.filename.endswith('.json'):
```

```
5       raise HTTPException(status_code=400, detail="Invalid file type. Only
        ↪  JSON files are allowed.")
6
7   contents = await file.read()
8   try:
9       json.loads(contents.decode('utf-8'))
10  except (json.JSONDecodeError, UnicodeDecodeError):
11      raise HTTPException(status_code=400, detail="Invalid file encoding.
        ↪  Only UTF-8 encoded JSON files are allowed.")
```

Code Darstellung 11: API JSON Validierung

6.5 Verarbeitung der Instruktionen

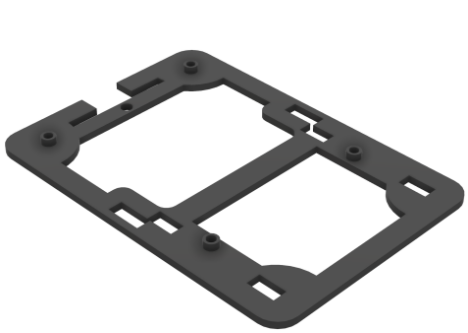
Damit der Roboter übermittelte Instruktionen umsetzen kann und sich der Roboter fortbewegen kann, wird eine Möglichkeit benötigt, um mit den verwendeten Motoren zu kommunizieren. Hierfür wird durch die Lehrstätte eine Hardware-Komponente bereitgestellt, die diese Anbindung vereinfacht. Das dafür verwendete Grove Base Hat, in Abbildung 6.4b dargestellt, bietet die Möglichkeit mittels einem einzigen Stecker die Anschlüsse *GND, VCC, Signal* zu verbinden. Bei den Instruktionen, die für den Raspberry Pi Roboter benötigt werden, handelt es sich um die Funktionen vorwärts Fahren, rückwärts Fahren, rechts, sowie links Drehen. Diese Manöver werden folgendermaßen implementiert: Die verwendeten Motoren können sich sowohl vorwärts als auch rückwärts drehen. Für den Vorgang des vorwärts Fahrens werden beide Motoren gleichzeitig in Richtung vorwärts angesteuert. Das gleiche Vorgehen wird für den Vorgang des rückwärts Fahrens mit rückwärts Drehungen umgesetzt. Für die Vorgänge der Rechts- sowie Linksdrehung wird jeweils nur ein Motor zur Zeit angesteuert. Eine Drehung in einem bestimmten Winkel wird implementiert, indem zuvor durch Experimente bestimmt wird, wie viele MS ein Motor angesteuert werden muss, um den Roboter beispielsweise um 90 Grad zu drehen. Die Möglichkeit anderer Winkel wird mit diesem ermittelten Faktor errechnet. Für die vorwärts und rückwärts Bewegung muss durch eine Instruktion eine Zeit in Sekunden angegeben werden, für die der Roboter sich

bewegen soll. Nach Ablauf dieser Zeit werden die Motoren gestoppt und es können weitere Instruktionen ausgeführt werden.

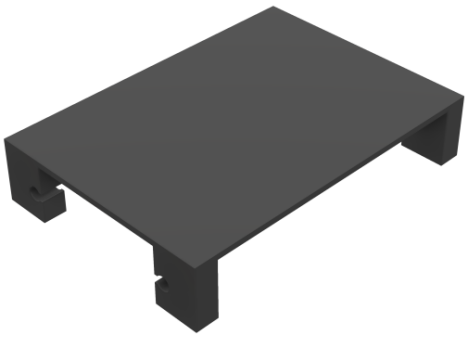
6.6 Umbau des Einplatinencomputers

Um alle Komponenten des Roboters miteinander zu vereinen wurde eine Plattform mithilfe der Konstruktionssoftware *Fusion 360* erstellt. Die Anforderungen an dieses Modell sind wie folgt:

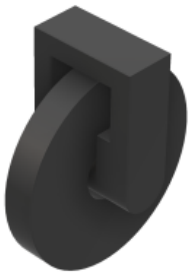
- Die Plattform muss die Möglichkeit bieten, den Raspberry Pi aufzunehmen und zu fixieren.
- Es muss eine Möglichkeit geben, die Motoren zu fixieren.
- Die Plattform muss die Möglichkeit bieten, die Anschlusskabel der Motoren und des Raspberry Pi zu verbinden.
- Der SD-Kartenslot sowie die USB-Anschlüsse des Raspberry Pi müssen zugänglich sein.



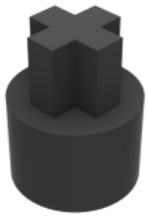
(a) Plattform



(b) Motor Halterung



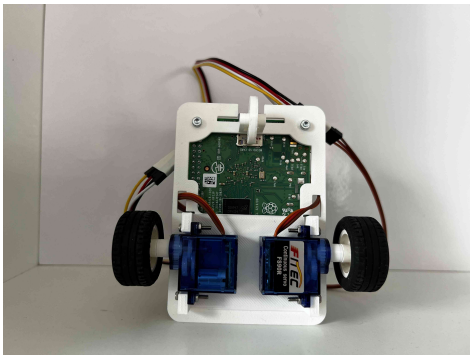
(c) Vorderrad



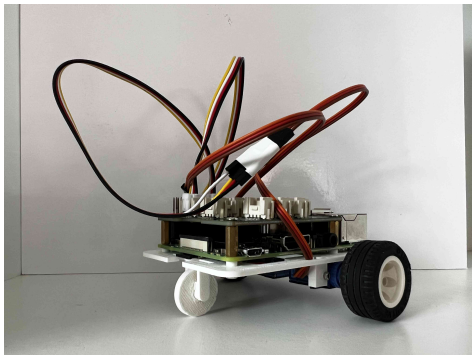
(d) Rad Adapter

Abbildung 6.6: Konstruierte Bauteile

Weiterhin müssen an die beiden Motoren die verwendeten Räder angebracht werden. Hierfür wurde ebenfalls ein Adapter konstruiert. Neben den genannten Bauteilen wurde weiterhin ein einzelnes Vorderrad für den Roboter konstruiert. Die Konstruktion der Bauteile ist in Abbildung 6.6 dargestellt. Nach Fertigstellung sowie mehreren Anpassungen der Konstruktionen wurden die Bauteile mit einem 3D-Drucker hergestellt. Die einzelnen Komponenten wurden miteinander verklebt und anschließend die übrigen Hardware-Komponenten mit Feingewinde-Schrauben befestigt. Der fertige Roboter ist in Abbildung 6.7 dargestellt. Eine größere Darstellung des Roboters ist zudem dem Anhang in Anhang B zu entnehmen.



(a) Ansicht von unten



(b) Seitenansicht

Abbildung 6.7: Zusammengebauter Roboter

6.7 Funktionsweise im Überblick

Im Folgenden soll eine gesamtheitliche Übersicht geschaffen werden, anhand der die Funktionsweise und Abläufe des Roboters deutlich werden. Die Grundvoraussetzung für die Kommunikation mit dem Roboter ist, dass sich das Gerät, auf dem der Schwarmintelligenz-Algorithmus einen Pfad berechnet und daraus die Instruktionen für den Roboter erstellt, im gleichen Netzwerk angesiedelt sind. Die Kommunikation zwischen den Geräten erfolgt über das Protokoll *HTTP*. Zu Beginn muss an den in Abbildung 6.5 dargestellten Endpunkt */uploadfile/* eine *JSON*-Datei mit den Instruktionen für den Roboter gesendet werden. Der Aufbau muss wie in Abschnitt 6.7 dargestellt vorliegen. Wenn diese Datei erfolgreich an den Endpunkt des Raspberry Pi übertragen wurde, kann der Roboter gestartet werden. Dies erfolgt über den Endpunkt */start/*, der im Anschluss der Übertragung aufgerufen werden muss. Nach Aufruf dieses Endpunkts wird auf dem Roboter die zuvor übertragene *JSON*-Datei geladen und anschließend die vorhandenen Instruktionen der Reihenfolge nach durch eine Übersetzungsfunktion ausgeführt. Der vollständige Code zur Ausführung der Instruktionen ist in 15 dargestellt. Jeder der Endpunkte gibt eine Rückmeldung, ob die jweilige Aktion erfolgreich war.

```
{
    "instructions": [
      {
        "action": "drive_straight",
        "seconds": 2
      },
      {
        "action": "turn_right",
        "degree": 45
      }
    ]
  }
```

Code Darstellung 12: JSON Instructions

Soll ein neuer Fahrvorgang gestartet werden, muss der Anwender eine neue Datei mit Instruktionen an den Endpunkt senden. Dieser Vorgang ersetzt die zuvor eventuell bereits vorhandene Datei und ermöglicht einen neuen Durchlauf. Der Roboter kann jederzeit über den Endpunkt */stop/* gestoppt werden.

6.8 Probleme bei der Umsetzung

Im Folgenden sollen Probleme aufgezeigt werden, die bei der Umsetzung, sowohl bei der Konzeption als auch bei der Implementierung, aufgetreten sind. Diese Probleme wurden in der Praxis gelöst und sollen nun in diesem Kapitel erläutert werden. Damit eine mobile Anwendung des Roboters möglich ist, ist eine Spannungsversorgung mittels eines Akkus notwendig. Der Raspberry Pi 3B+ benötigt eine Versorgungsspannung von 5V, die über den Micro-USB anschluss bereitgestellt werden kann [**raspberrypi-docs**]. Optimalerweise wird für den Betrieb der Motoren weiterhin eine weitere externe Spanungsquelle verwendet. Diese steht im Rahmen der studentischen Arbeit jedoch nicht zur verfügung und konnte nicht durch das Inventar der Lehrstätte zur Verfügung gestellt werden. Daher wurde der Rasperry Pi ausschließlich über den Micro-USB-Anschluss mit Strom versorgt, was die Mobilität aufgrund

des angeschlossenen Kabels stark einschränkt. Bei den vorerst verwendeten Motoren handelt es sich um Motoren, die für den Einsatz in Modellbauanwendungen konzipiert wurden. Hierbei stellte sich zu einem späteren Zeitpunkt der Nachteil heraus, dass diese Motoren nur einen Drehradius von 180° besitzen. Dieser ist für die Bewegung des Roboters nicht ausreichend. Daher wurde ein neuer Motor mit einem Drehradius von 360° verwendet, mit dem die Bewegung des Roboters nun möglich ist. Weiterhin stellten sich bei der Ansteuerung der Motoren zusätzliche Probleme heraus. Die Ansteuerung der an die GPIO-Pins des Rapsberry angeschlossenen Motoren erfolgt über die Anpassung Der *PWM*-Frequenz. Die PWM-Frequenz wird hierbei über die Bibliothek *pigpio* eingestellt. Jedoch stellte sich heraus, dass die Anpassung der PWM-Frequenz nicht zuverlässig funktioniert. Durch die Anpassung sollte es normalerweise möglich sein, die Geschwindigkeit der Motoren zu steuern. Mit der Anpassung war es jedoch lediglich möglich, die Richtung der Drehung der Motoren anzupassen, weswegen eine Anpassung der Fahrgeschwindigkeit des Roboters nicht möglich war. Dies hatte weiterhin zur Folge, dass eine Rechts- sowie Linksdrehung des Roboters nicht während dem Fahren erfolgen kann. Deshalb werden Instruktionen zur Navigation des Rechts- und Linksfahrens nur während des Stillstands des Roboters ausgeführt und Anweisungen hierfür werden inkrementell verarbeitet.

Kapitel 7

Anwendung des Algorithmus auf den Roboter

In diesem Kapitel werden die Inhalte der vorherigen Abschnitte zusammengefügt und somit der Roboter zum Fahren des Pfades gebracht. Der Roboter bekommt dafür von dem Algorithmus aus Kapitel 5 den optimalen Pfad des Durchlaufs geliefert und soll diesem folgen, um das Ziel zu erreichen.

Für das Fahren des Roboters von einem Punkt zu dem nächsten muss zunächst die Steigung zwischen zwei Punkten mittels der Steigungsformel berechnet werden:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Mit der Steigung ist definiert wie weit sich der Roboter drehen muss, um auf dem richtigen Pfad zu fahren. Für die Ermittlung der Distanz zwischen zwei Punkte wird die folgende Formel verwendet:

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Allerdings muss zuerst verglichen werden, die aktuelle Position der nächsten Position gleicht. Falls das der Fall ist, wird diese Koordinate übersprungen und direkt mit dem nächsten Punkt fortgefahren.

Nachdem diese Operationen für die zwei aktuell betrachteten Punkte durchgeführt sind können neue Instruktionen der JSON Datei hinzugefügt werden, die später per „upload“ Methode aus Kapitel 6 auf den Raspberry Pi hochgeladen wird. Diese Vorgehensweise wird iterativ für jeden der Punkte durchgeführt und im Anschluss kann der Roboter per post Methode „/start“ gestartet werden (siehe Abbildung 13).

```
def drive_path(path, upload_url, start_url, headers):
    for i in range(len(path)-1):
        current_pos = path[i]
        next_pos = path[i+1]
        if current_pos != next_pos:
            # berechne Steigung
            m = (next_pos[1] - current_pos[1])/(next_pos[0] -
            ↪  current_pos[0])

            #drehe roboter um die Gradzahl der Steigung * 10
            instructions["instructions"].append({"action": "turn_right",
            ↪  "degree": m*10})

            #Länge der Stecke berechnen
            distance = math.sqrt((next_pos[0] - current_pos[0])**2 +
            ↪  (next_pos[1] - current_pos[1])**2)
            instructions["instructions"].append({"action":
            ↪  "drive_straight", "seconds": distance})

    with open('instruc.json', 'w') as f:
        json.dump(instructions, f)

    # Anfrage bauen und senden
    files = [('file', open('instruc.json', 'rb'))]
    upload_response = requests.post(upload_url, headers=headers,
    ↪  files=files)
    response = requests.post(start_url, headers=headers)
```

Code Darstellung 13: drive_path Funktion

Kapitel 8

Fazit

8.1 Ausblick

Das Feld der Metaheuristik ist ein Feld, welches sich schnell verändert und wächst. Es werden immer wieder neue Verbesserungen zu bestehenden oder kommplett neue Algorithmen veröffentlicht. Außerdem wurden hier nur drei Algorithmen näher betrachtet. Um eine bestmögliche Performance sicherzustellen, müssten mehr Algorithmen sowie dazugehörige Verbesserungen betrachtet und bestenfalls getestet werden. Zusätzlich müsste dies immer wieder aktualisiert werden, sobald eine mögliche Verbesserung vorliegt. Der beschriebene Bienenalgorithmus zur Wegfindung auf einer Karte mit Hindernissen hat vielversprechende Ergebnisse gezeigt und könnte in der Zukunft für die Navigation von autonomen Robotern oder Drohnen eingesetzt werden. Weiterführende Forschung könnte sich darauf konzentrieren, den Algorithmus für dynamische Umgebungen zu optimieren und seine Anwendbarkeit auf größere Karten zu erweitern. Zudem könnten verschiedene Bewertungsfunktionen für Pfade getestet werden, um die Effizienz des Algorithmus zu verbessern. Insgesamt hat die Arbeit gezeigt, dass der Bienenalgorithmus eine vielversprechende Methode für die Wegfindung in schwierigen Umgebungen darstellt.
Für die Umsetzung des Roboters wurden drei mögliche Konzepte miteinander verglichen. Die Wahl für die Implementierung und Umsetzung des Roboters

fiel auf das Konzept mit einer hybriden Umsetzung, wobei ein Client einen Schwarmintelligenz-Algorithmus ausführt, daraus Instruktionen für den Roboter generiert und diese an den Roboter sendet. Bei dieser Form der Implementierung werden jedoch keine Umgebungsdaten aktiv durch den Roboter gesammelt. Um dieses Vorhaben zu realisieren, hätte der Roboter weiterhin mit Sensoren ausgestattet werden müssen, welche die Umgebung erfassen. Diese Daten müssten dann in eine Form gebracht werden, mit der der verwendete Schwarmintelligenz-Algorithmus umgehen kann. Dies hätte die Implementierung im Rahmen dieser studentischen Arbeit deutlich erschwert, nicht zuletzt, weil mit der Ansteuerung der verwendeten Motoren bereits ein hoher Aufwand verbunden war und besonders die Kontrolle der Geschwindigkeiten der Motoren nicht zu lösen war. Die Implementierung des Roboters mit der hybriden Umsetzung hat gezeigt, dass die Umsetzung eines Roboters mit Schwarmintelligenz-Algorithmus möglich ist. Zukünftig wäre eine umfangreichere Umsetzung unter der Verwendung von Sensoren denkbar, um die Umgebung des Roboters zu erfassen und eine Umsetzung dieser Variante zu ermöglichen. Hierbei würden die durch die Sensoren erfassten Umgebungsdaten durch den Schwarmintelligenz-Algorithmus verarbeitet werden, wodurch eine gesteigerte Realitätstauglichkeit des Roboters erreicht werden könnte.

Anhang A

Code Darstellungen

```python
from fastapi import FastAPI, HTTPException, Depends, UploadFile, Request
from fastapi.security.api_key import APIKeyHeader, APIKey
from starlette.status import HTTP_403_FORBIDDEN
import os
import json
from typing import Union
from utils.proc_actions import start_proc, kill_proc, is_process_running
from utils.call_shell import run_command

app = FastAPI()

API_KEY = os.environ.get("API_KEY")
api_key_header = APIKeyHeader(name="X-API-Key", auto_error=False)

async def api_key_verification(api_key_header: str =
↪  Depends(api_key_header)):
    if api_key_header is None or api_key_header != API_KEY:
        raise HTTPException(status_code=HTTP_403_FORBIDDEN,
        ↪  detail="Invalid API key")
    return api_key_header

@app.post("/start/")
async def start_robot(api_key: APIKey = Depends(api_key_verification)):
    """
    Start the robot if it is not already running.
```

```
    This endpoint will start the robot if it is not already running.
    """
    if not is_process_running('call_robot.py'):
        await start_proc('call_robot.py')
        return {"message": "Robot started."}
    else:
        return {"message": "Robot started."}


@app.post("/shell/")
async def exec_shell(request: Request):
    """
    Triggers a shell command.
    Body needs to have a "command" key with
    the command to be executed as value.
    """
    body = await request.json()
    if 'command' in body:
        command_list = body['command'].split()
        message = run_command(command_list)
        return {"message": message}
    else:
        return {"message": "No command sent."}


@app.post("/stop/")
async def stop_robot(api_key: APIKey = Depends(api_key_verification)):
    """
    Stop the robot if it is running.

    This endpoint will stop the robot if it is running.
    If it is not running, it will return a message saying so.
    If it is running, it will stop the robot and return a message saying
    ↪  so.
    """
    if is_process_running('call_robot.py'):
        await kill_proc('call_robot.py')
        return {"message": "Robot stopped."}
    else:
```

```
62          return {"message": "Robot not running."}
63
64  @app.get("/status/")
65  async def get_status(api_key: APIKey = Depends(api_key_verification)):
66      """
67      Get the status of the robot.
68
69      This endpoint will return a message saying if the robot is running or
        ↪  not.
70      """
71      if is_process_running('call_robot.py'):
72          return {"message": "Robot running."}
73      else:
74          return {"message": "Robot not running."}
75
76
77
78  @app.post("/uploadfile/")
79  async def create_upload_file(file: Union[UploadFile, None] = None,
    ↪  api_key: APIKey = Depends(api_key_verification)):
80      """
81      Upload a JSON driving instructions file.
82
83      This endpoint allows to upload a JSON file containing driving
        ↪  instructions.
84      It checks if the file is a JSON file and if it is encoded in UTF-8.
85      If the file is valid, it will be saved on the endpoint device.
86      """
87      if not file:
88          return {"message": "No upload file sent"}
89
90      if not file.filename.endswith('.json'):
91          raise HTTPException(status_code=400, detail="Invalid file type.
            ↪  Only JSON files are allowed.")
92
93      contents = await file.read()
94      try:
95          json.loads(contents.decode('utf-8'))
96      except (json.JSONDecodeError, UnicodeDecodeError):
```

```
97          raise HTTPException(status_code=400, detail="Invalid file
            ↪  encoding. Only UTF-8 encoded JSON files are allowed.")
98
99      file_path = "tmp/driving_instructions/instructions.json"
100     if os.path.exists(file_path):
101         os.remove(file_path)
102
103     with open(file_path, "wb") as buffer:
104         buffer.write(contents)
105
106     return {"message": "Uploaded file!"}
```

Code Darstellung 14: API

```
1   import RPi.GPIO as GPIO
2   import time
3   import json
4   import os
5
6   SERVO_R = os.environ.get("SERVO_R")
7   SERVO_L = os.environ.get("SERVO_L")
8
9
10  class Robot:
11      def __init__(self, servo_pin1=SERVO_R, servo_pin2=SERVO_L,
        ↪  pwm_freq=50, duty_cycle=0):
12          self.servo_pin1 = servo_pin1
13          self.servo_pin2 = servo_pin2
14          self.pwm_freq = pwm_freq
15          self.duty_cycle = duty_cycle
16          self.start_time = None
17
18          GPIO.setmode(GPIO.BCM)
19          GPIO.setwarnings(False)
20
21          GPIO.setup(self.servo_pin1, GPIO.OUT)
22          self.pwm1 = GPIO.PWM(self.servo_pin1, self.pwm_freq)
23          self.pwm1.start(self.duty_cycle)
24
```

```
25          GPIO.setup(self.servo_pin2, GPIO.OUT)
26          self.pwm2 = GPIO.PWM(self.servo_pin2, self.pwm_freq)
27          self.pwm2.start(self.duty_cycle)
28
29      def drive_straight(self, seconds=2):
30          self.pwm1.ChangeDutyCycle(5)
31          self.pwm2.ChangeDutyCycle(10)
32          self.how_long(seconds)
33
34      def drive_backwards(self, seconds=2):
35          self.pwm1.ChangeDutyCycle(10)
36          self.pwm2.ChangeDutyCycle(5)
37          self.how_long(seconds)
38
39      def turn_left(self, degree=90):
40          self.pwm1.ChangeDutyCycle(5)
41          self.how_long(1.2*degree/90.0)
42
43      def turn_right(self, degree=90):
44          self.pwm2.ChangeDutyCycle(10)
45          self.how_long(1.2*degree/90.0)
46
47      def stop(self):
48          self.pwm1.ChangeDutyCycle(0)
49          self.pwm2.ChangeDutyCycle(0)
50          end_time = time.time()
51          print("Time elapsed: ", end_time - self.start_time)
52
53      def cleanup(self):
54          self.pwm1.stop()
55          self.pwm2.stop()
56          self.__init__(self.servo_pin1, self.servo_pin2, self.pwm_freq,
            ↪  self.duty_cycle)
57          GPIO.cleanup()
58
59
60      def how_long(self, seconds):
61          self.start_time = time.time()
62          while True:
```

```
63              current_time = time.time()
64              elapsed_time = current_time - self.start_time
65              if elapsed_time >= seconds:
66                  self.stop()
67                  break
68
69      def execute_instructions(self, instructions_file):
70          with open(instructions_file, 'r') as f:
71              instructions = json.load(f)
72          for instruction in instructions['instructions']:
73              action = instruction['action']
74              print("Executing action: ", action)
75              if action == 'drive_straight':
76                  self.drive_straight(instruction['seconds'])
77              elif action == 'drive_backwards':
78                  self.drive_backwards(instruction['seconds'])
79              elif action == 'turn_left':
80                  self.turn_left(instruction['degree'])
81              elif action == 'turn_right':
82                  self.turn_right(instruction['degree'])
83              time.sleep(1)
```

Code Darstellung 15: Driving Logic

Anhang B

Darstellungen

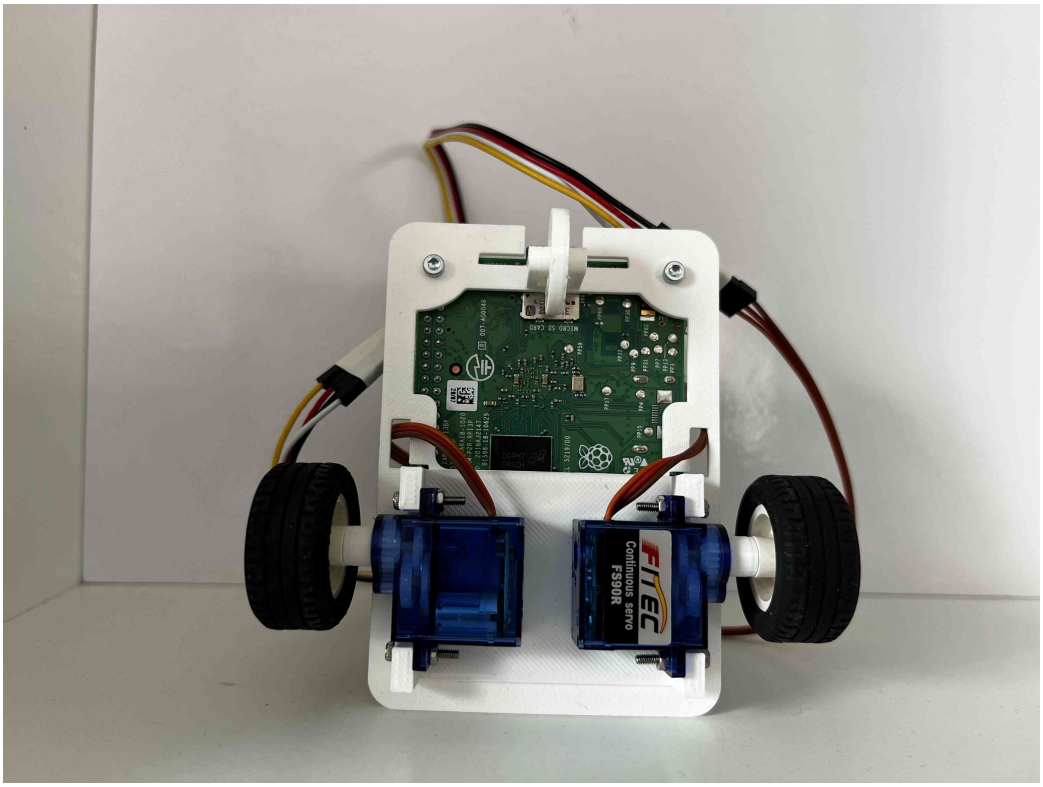


Abbildung B.1: Ansicht von unten (groß)

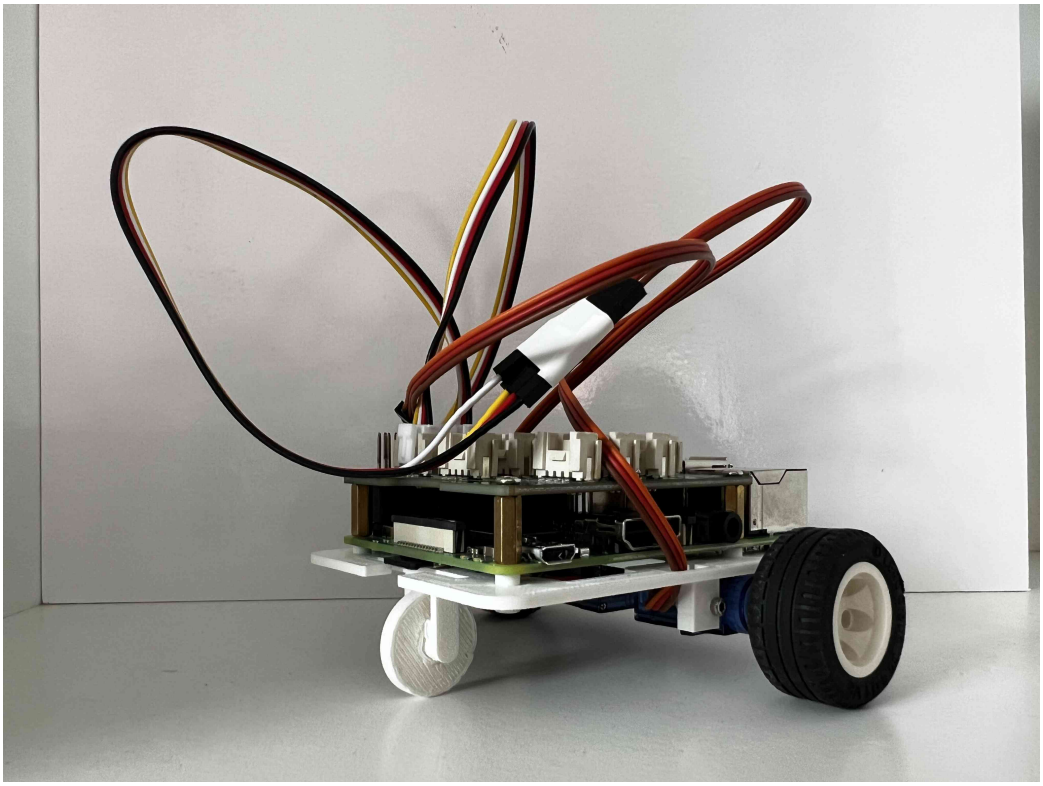


Abbildung B.2: Seitenansicht (groß)