Suchen 2 - Labyrinth Navigation mittels A Stern

Storyboard

Damit sich ein Roboter autonom durch ein Produktionsumfeld bewegen kann, muss dieser einen Pfad von seiner aktuellen Position zu einer Zielposition finden. Bei einem Linienfolger ist dieser Pfad statisch und bereits vorgegeben und erlaubt nur das wiederholte Abfahren desselben Pfades. Ein anderer, flexibler Ansatz ist es, den Pfad anhand einer Karte zu Planen. Dieses Vorgehen erlaubt es dem Roboter, verschiedene Stationen in unterschiedlicher Reihenfolge anzufahren. Da die abgefahrenen Pfade bei diesem Ansatz aber unterschiedlich sind, müssen diese erst vom Roboter geplant werden. Das passiert in der Regel anhand einer Karte. Die Karte kann dabei vorab bekannt sein oder dynamisch anhand von Sensordaten aufgebaut werden.

Wie können Bewegungen durch auf einer Karte geplant werden? Das Pfadplanungsproblem wird so dargestellt, dass es durch einen Suchalgorithmus gelöst werden kann (siehe AIAV Video Suche). Dazu erfasst der Roboter seine Umgebung, erstellt sich aus den Sensordaten eine Karte und bestimmt seine derzeitige Position. Der Suchalgorithmus sucht dann nach einem Pfad von der derzeitigen Position des Roboters zum Ziel. Befahrbare Wegpunkte werden dabei als Knotenpunkte, sogenannte "Nodes", dargestellt. Die Nodes sind durch einen oder mehrere Pfade miteinander verbunden. Das aus den Nodes und Pfaden resultierende Netzwerk nennt man Graph. Dieser bildet alle bekannten Verzweigungen und Pfade ab. Abbildung 1 zeigt ein Beispiel für so einen Graphen.

Abbildung 1 zeigt ein Beispiel für so einen Graphen. Die Nodes (z.B. Stationen oder Kreuzungen) sind miteinander durch Linien verbunden. Diese Linien stellen mögliche Verbindungen zwischen den Nodes dar. Jede der Linien ist mit einer Zahl, den Kosten, bemessen. Diese Kosten beschreiben wie "schwierig" das Abfahren einer Verbindung ist. Solche Kosten können zum Beispiel vom Abstand zwischen den Nodes oder von der benötigten Zeit zum Zurücklegen der Verbindung abhängig sein. Mit einem Beispiel ist das ganze gleich anschaulicher: Wir haben eine Station für unseren Roboter und eine Kreuzung. Zwischen Station und Kreuzung gibt es eine Verbindung, einen drei Meter langen Gang. Also stellen wir sowohl die Station, als auch die Kreuzung als Nodes dar und verbinden diese mit einem Pfad mit Kosten von drei.

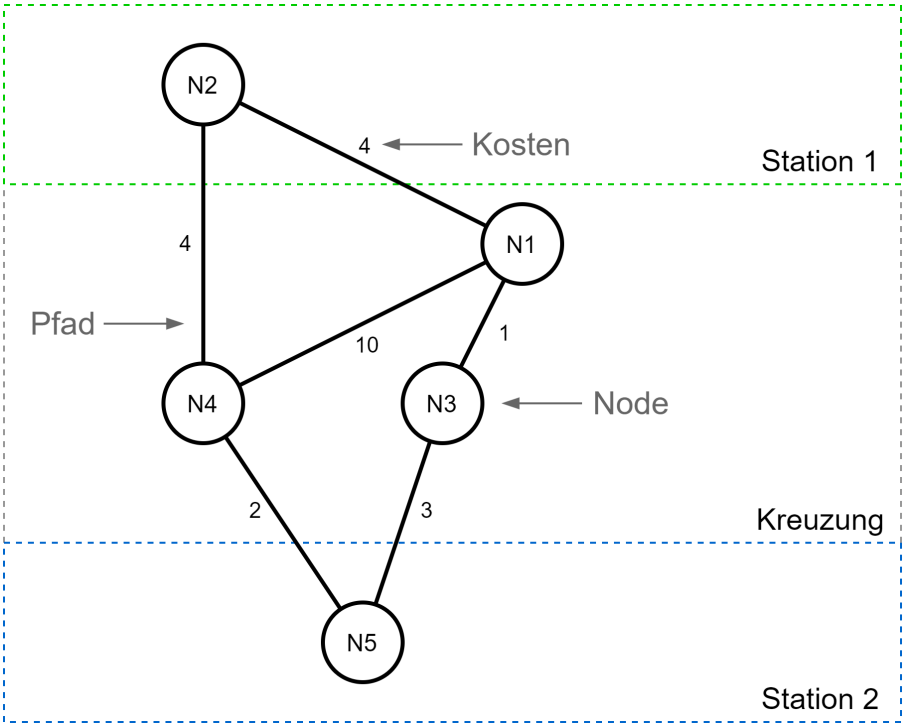

Abbildung 1: Die Nodes (N1-N5) stellen Punkte in der echten Welt, wie z.B. eine Station oder Kreuzung, dar. Sie sind mittels möglichen Pfaden miteinander verbunden. Jeder dieser Pfade ist mit einer Zahl, den Kosten, bemessen. Diese Kosten beschreiben wie "schwierig" das Abfahren einer Verbindung ist. Die Kosten können so z.B. die Länge der Verbindung beschreiben. Wir speichern die Kosten im Graphen, da sie für einige Suchalgorithmen benötigt werden.

Suchalgorithmen suchen einen Weg zwischen zwei Knotenpunkten auf dem Graphen. Beispiele für Suchalgorithmen sind Breiten- und Tiefensuche, sowie Dijkstra's und A* (A Stern) [1]. Abhängig von der Art des Suchalgorithmus wird dabei unterschiedlich vorgegangen und Knoten in einer unterschiedlichen Reihenfolge durchsucht.

In diesem Usecase wurde der A* Suchalgorithmus implementiert. Die Idee hinter diesem Algorithmus ist es, systematisch den kürzesten Weg zu bekannten Nodes zu finden. Dafür ist jedoch eine Karte der Umgebung notwendig. Ist keine Karte vorhanden, muss sie zunächst erstellt werden. Auf der Karte werden dann Nodes vom Startpunkt aus abgesucht bis die Zielnode erreicht wird. Die Reihenfolge in welcher abgesucht wird, hängt dabei von der sogenannten Heuristik ab. Diese wird für jeden Knotenpunkt als die Summe aus den Kosten um den Punkt zu erreichen und der Entfernung zum Zielknoten berechnet. Da durch die Heuristik sowohl der zurückgelegte Weg als auch die Entfernung zum Ziel berücksichtigt wird, müssen potentiell weniger Knoten durchsucht werden als bei anderen Suchalgorithmen. Ein weiterer Vorteil dieses Vorgehens ist, dass der optimale Pfad zum Ziel gefunden wird. Voraussetzung dabei ist dass der Pfad exestiert und eine gültige Heuristik eingesetzt wird.

Die Suche wird mittels zwei Listen durchgeführt: einer offenen und einer geschlossenen Liste. Der Ablauf des Algorithmus ist wie folgt:

Zunächst wird der Startknoten auf die offene Liste gelegt und aufgeklappt. Beim Aufklappen einer Node werden alle mit ihr verbundenen Knoten auf die offene Liste gelegt, sofern der jeweilige Knoten noch nicht auf der offenen oder geschlossenen Liste ist. Die Nodes auf der offenen Liste werden anhand der Heuristik sortiert. Die Node mit dem niedrigsten Wert für die Heuristik wird von der offenen

Liste in die geschlossene Liste abgelegt und wiederum aufgeklappt. Nun wird so lange neu sortiert und aufgeklappt, bis der Zielknoten erreicht wird oder die offene Liste leer wird. Sobald die offene Liste leer ist, bricht der Algorithmus ab, da keine Lösung existiert. Wurde der Zielknoten jedoch erreicht, wird abschließend der optimale Pfad vom Zielknoten aus basierend auf den bekannten Nodes in der geschlossenen Liste konstruiert. Abbildung 2 zeigt Ablauf des Aufklappens und Ablegens anhand eines beispielhaften Graphens.

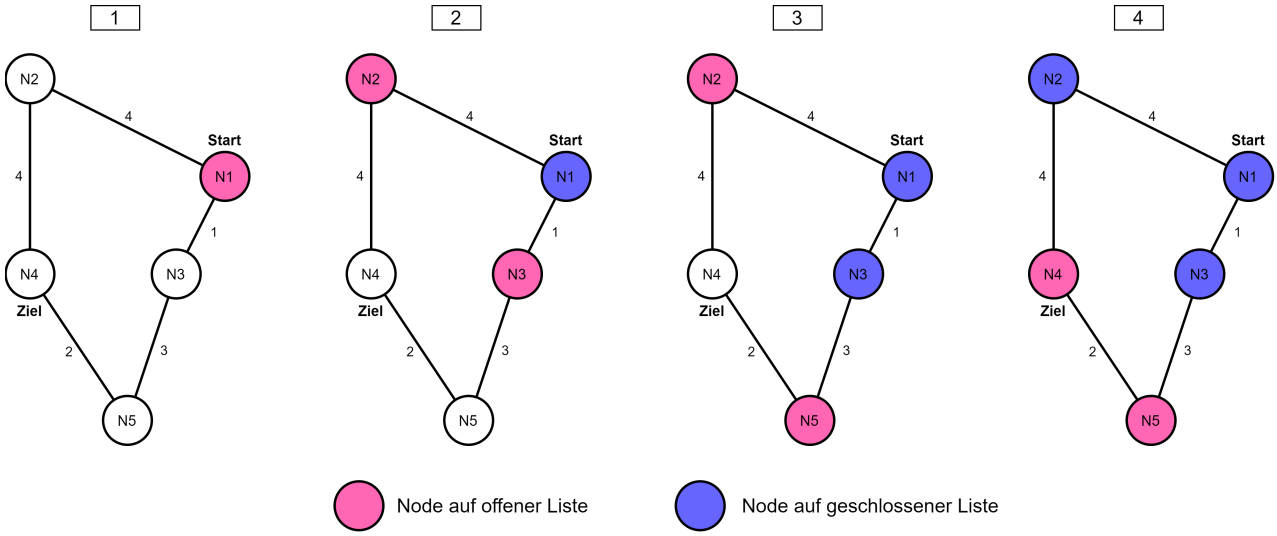

Abbildung 2: Beispielhafter Ablauf des A Algorithmus auf einem Graphen. Die Nodes werden nacheinander aufgeklappt, damit systematisch der kürzeste Weg zu jeder Node gefunden wird. Die Reihenfolge in der aufgeklappt wird, kommt von der sogenannten Heuristik. Diese berechnet sich aus den Kosten des bereits zurückgelegten Pfads und der Entfernung zum Ziel. Es werden so lange bekannte Nodes aufgeklappt, bis das Ziel erreicht wurde oder keine Nodes zum Aufklappen übrig sind.*

In der praktischen Implementierung des Usecases wird ein Mobiler Roboter in einem zufällig generierten Labyrinth plaziert. Das Ziel des Roboters ist dabei, einen Pfad vom Eingang links unten zum Ausgang rechts oben zu finden. Die Struktur des Labyrinths ist dem Roboter dabei unbekannt, das heißt, dass er sich den Graphen erst aufbauen muss.

Jetzt wenden wir die Idee des Graphen auf ein "echtes" Umfeld, also unser Labyrinth, an. Der Graph besteht aus einem Gitter aus Nodes (siehe Abbildung 3). Alle Pfade zwischen den Nodes sind 1,8 Meter lang, damit jede Zelle des Labyrinths einer Node entspricht. Die Pfade zwischen den Nodes signalisieren offene Durchgänge und werden anhand der Laserscan Sensoren am Roboter ermittelt. Der Laserscan liefert den Abstand bis zur nächsten Wand rund um den Roboter. Mithilfe der Position des Roboters am Graphen und den Laserscan Daten wird der Graph so lange aktualisiert, bis das komplette Labyrinth bekannt ist, oder das Ziel erreicht wurde.

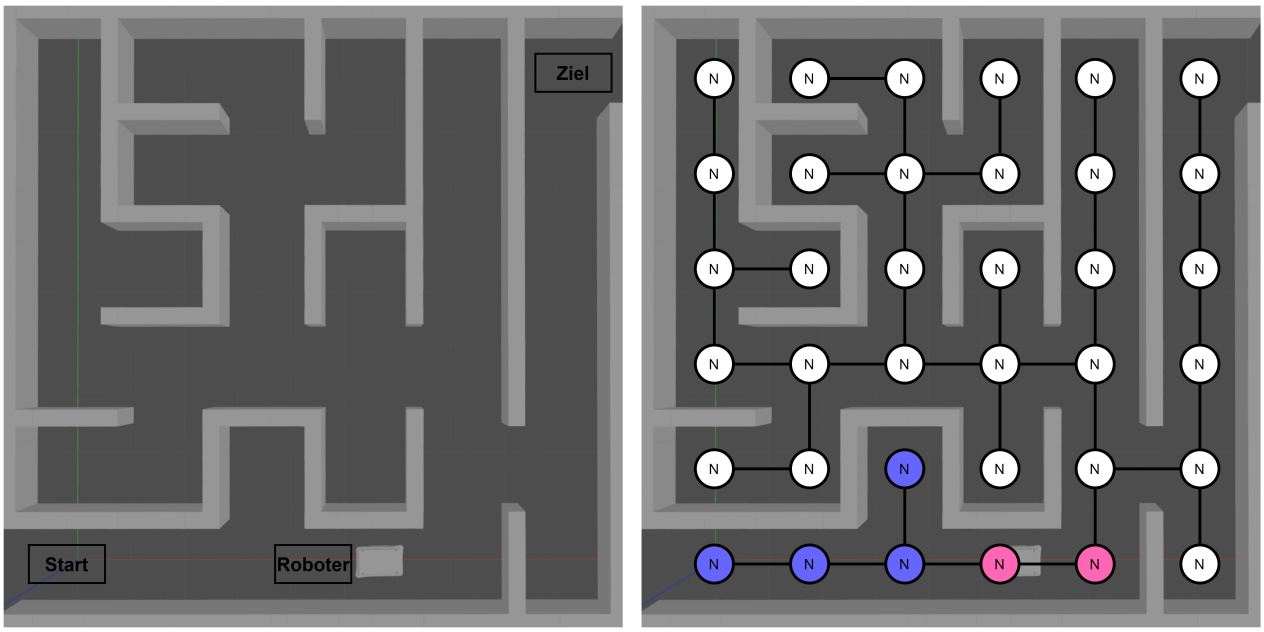

Abbildung 3: Das Labyrinth wird als Graph dargestellt, damit der Roboter mittels A navigieren kann. Dabei baut sich der Roboter zunächst den Graphen anhand der Sensordaten auf. Dann werden laufend Nodes anhand der Heuristik aufgeklappt, bis die Zielnode erreicht wird. Damit wurde ein Graph ermittelt, auf welchem ein Weg zwischen Start und Ziel besteht. Dieser Pfad wird anschließend direkt aus der vom A Algorithmus ermittelten geschlossenen Liste geladen und abgefahren.

Ein Video, welches den Aufbau der Karte im Labyrinth zeigt, finden Sie hier. Dabei fährt der Roboter alle aufgeklappten Nodes ab und erweitert anhand der Sensordaten so lange die Liste der bekannten Nodes, bis das Ziel gefunden wurde.

Ein Video, welches das Abfahren des ermittelten Pfades im Labyrinth zeigt, finden Sie hier. Hier hat sich der Roboter erfolgreich eine Karte aufgebaut. Auf dieser wird mittels A* Algorithmus ein Pfad zwischen Start und Ziel ermittelt. Dieser ist im rechten Fenster in grün eingezeichnet. Sobald der Pfad erfolgreich ermittelt wurde, fährt in der Roboter ab.

Da der Roboter das Labyrinth anfangs noch nicht kennt, fährt er zunächst alle Nodes an, welche der A* Algorithmus aufklappt. Dabei wird so lange der Graph aus den Sensordaten aufgebaut, bis die Zielnode aufgeklappt und erreicht wurde. Dannach wird der optimale Pfad zwischen Start und Ziel mithilfe der kompletten geschlossenen Liste ermittelt. Abschließend fährt der Roboter auf diesem Pfad zurück zum Start.

Quellen

[1] Russell, S. and Norvig, P., 2002. Artificial intelligence: a modern approach.