Technische Hochschule
Brandenburg
University of
Applied Sciences
Fachbereich
Informatik und Medien

Künstliche Intelligenz – Entwicklung eines autonomen Sortierroboters

Camillo Fillinger, Majed Aldin Bakir, Vincent Berndt

Projekt im 5. Semester • Studiengang Informatik • Fachbereich Informatik und Medien • 10.01.2025

Einleitung

Der Sortierroboter (Abb. 2) ist ein autonomes System zur Navigation, Farberkennung und Sortierung von Bällen. Dieses Projekt kombiniert Robotik, Sensorik und Algorithmen zur Wegfindung, um eine automatisierte Lösung für Sortieraufgaben zu bieten.
Der Roboter wurde selbst aus LEGO gebaut und wird von einem AKSEN-Board gesteuert, das alle Berechnungen ausführt. Die Hardware umfasst selbstgebaute Antriebs- und Greifmechanismen, Optokoppler zur Navigation, Lichtsensoren zur Farberkennung und Taster zur Hinderniserkennung. Zusätzlich sind Motoren für die Fortbewegung und das Greifen der Bälle integriert.
Solche Technologien finden Anwendungen in Logistik, Recycling und Automatisierung.

Funktion

Der Sortierroboter arbeitet autonom und führt wiederholt eine Abfolge von Aufgaben aus, um Bälle entsprechend ihrer Farbe zu sortieren. Zunächst analysiert er die Karte und berechnet mit einer Kosten-Map per Breitensuche den kürzesten Weg zur nächsten Ballposition. Sobald die Route festgelegt ist, bewegt er sich dorthin. Am Ziel angekommen, nimmt er den Ball mit seinem Greifmechanismus auf und scannt dessen Farbe mit einem Lichtsensor. Abhängig von der erkannten Farbe entscheidet er, ob der Ball zur Zielposition für rote oder blaue Bälle transportiert werden muss. Anschließend berechnet der Roboter erneut den optimalen Weg, navigiert zur entsprechenden Zielposition und legt den Ball dort ab. Danach sucht der Roboter nach der nächsten Ballposition und wiederholt diesen Prozess, bis alle Bälle sortiert sind. Um bereits besuchte Felder bei der jeweils nächsten Routenplanung zu ignorieren, werden Positionen, an denen sich zuvor ein Ball oder ein Zielort für ein Ball befand, mit Hindernissen ersetzt.

Aufbau der Karte

Die virtuelle Karte des Roboters ist als 1D-Array implementiert und besteht aus 77 Feldern, beispielhaft als 2D-Karte in Abb. 1 dargestellt.
Jedes Feld hat eine bestimmte Bedeutung:

- X: Hindernis oder blockierter Bereich, der nicht befahren werden kann.
- .: Freie Fläche, auf der der Roboter sich bewegen kann.
- S: Startposition des Roboters.
- F: Positionen, an denen sich Bälle befinden.
- R: Zielposition für rote Bälle.
- B: Zielposition für blaue Bälle.

Ein Beispiel für eine solche Karte im Code:

```
char map[77] =
"xxxFxxxx.....xx..xx.Bxx.xx.xS.....xx.....Fx.x..xxx.....xx..xx.xx..xx.xxxRxxxx";
```

```
X X X F X X X
X . . . . . X
X . . X X . B
X X . X X . X
S . . . . . X
X . . . . . F
X . X . . X X
X . . . . . X
X . . X X . X
X . . X X . X
X X R X X X X
```

Abb. 1: Beispielkarte in 2D

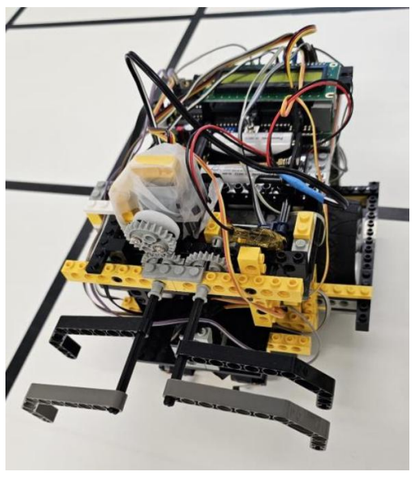

Abb. 2: Soro der Sortierroboter

Planung und Navigation

Die Navigation des Roboters basiert auf drei zentralen Schritten: Berechnung der Kosten-Map (findAllCosts()), Ermittlung des optimalen Pfads (findBestPath()) und Umwandlung des Pfads in Navigationsanweisungen (getNavigation()).

Berechnung der Kosten-Map – findAllCosts()

Die Funktion findAllCosts() nutzt die Breitensuche, um die Anzahl der Schritte von der aktuellen Position zu jedem erreichbaren Feld zu berechnen. Zunächst wird costMap[77] mit -1 gefüllt, um unbesuchte Felder zu markieren und die Startposition (starPosition) sowie Zielposition (zielPosition) werden bestimmt. Die Startposition wird in die Warteschlange (queue[77]) eingefügt und in der costMap auf 0 gesetzt. Anschließend wird solange die Queue nicht leer ist, das erste Element (current) entfernt und die vier möglichen Bewegungsrichtungen (rechts, links, oben, unten) überprüft. Jede davon neue gültige Position (kein Hindernis, innerhalb der Karte, unbesucht), wird in die Queue eingefügt und ihr Kostenwert als costMap[current] + 1 in der costMap gespeichert. Am Ende enthält costMap für jedes Feld den kürzesten Abstand in Schritten von der Startposition, was als Grundlage für die Pfadberechnung mit findBestPath() dient.

Ermittlung des optimalen Pfads – findBestPath()

Nachdem costMap die minimalen Schritte gespeichert hat, wird mit findBestPath() der kürzeste Pfad von der Zielposition zurück zur Startposition rekonstruiert. Indem der Algorithmus bei der Zielposition beginnt und rückwärts den kürzesten Weg zur Startposition sucht. Er schaut dabei, welcher seiner Nachbarn den niedrigsten Wert hat, bis er bei 0 also der Startposition landet. Die Positionen, über die der Algorithmus geht, speichert er in path[]. Zuletzt wird path[] umgedreht, sodass es den Pfad der Positionen vom Start zum Ziel beinhaltet.

Umwandlung zu Steuerbefehlen – getNavigation()

Die Funktion getNavigation() übersetzt den Pfad-Index in eine Reihenfolge von Bewegungsanweisungen (navigation[]) für den Roboter. Dabei werden jeweils zwei aufeinanderfolgende Positionen verglichen, gestartet mit den ersten Beiden. Anhand der Differenz der Positionen wird bestimmt, in welcher Richtung sich die Positionen zueinander befinden. Diese Richtung wird dann mit der vorberechneten Roboterrichtung (currentDirection) verglichen und daraus der Steuerbefehl abgeleitet (g = gerade, l = links, r = rechts). Die (vorberechnete) Richtung des Roboters wird dann aktualisiert und die Steuerbefehle so lange berechnet, bis alle Positionen von path[] durchgearbeitet wurden. Am Ende enthält navigation[] eine Sequenz von Steuerbefehlen für den Roboter.

Verarbeitung der Steuerbefehle

Der Roboter verarbeitet die Steuerbefehle indem er bei jeder Kreuzung, diese erkennt er mithilfe seiner Optokoppler, den nächsten Befehl ausliest und ausführt. Dies so lange bis die Sequenz durch gearbeitet ist, daraufhin führt er das Greifen oder Loslassen des Balles durch.

Betreuung Dipl.-Inform. Ingo Boersch • Technische Hochschule Brandenburg • Prof.Dr. Emanuel Kitzelmann • Technische Hochschule Brandenburg