Technische Hochschule Ingolstadt

Roboternavigation durch Potentialfelder: Attractive/Repulsive und Wavefront Potentiale

Intelligente Robotik WS2023/24
Praktische Arbeit

Carl Schünemann (Mat.Nr. 00107827)

20. Dezember 2023

Inhaltsverzeichnis

1 Ziel der Implementierung

Gegebene Aufgabenstellung:

Es soll ein Planungssystem zur Mikronavigation implementiert werden. Die Planung erfolgt auf Basis eines statisch gegebenen, einfachen Occupancy Gids mit Hilfe der Potenzialfeldmethode. Der Roboter besitzt eine rechteckige Form variabler Größer. Der gefundene Plan wird visualisiert. Planungsstatistiken werden geführt. Die Leistungsfähigkeit des Planers wird anhand verschiedener Planungsszenarien bewertet.

Umgesetzt durch: - Python Jupyter Notebook - Einzige externe Bibliothek zur Durchführung Mathematischer Operationen: numpy > Fokus der Implementierung - Robotermodell: - Rechteckiger Roboter mit Variabler Größe (width, length) - Rotation des Roboters um Änkerpunktïm und gegen den Uhrzeigersinn in 90° Schritten (0°, 90°, 180°, 270°) > Änkerpunkt- Ecke "links oben"bei 0° Rotation - Konfigurationsraum: - Roboter befindet sich in statischem Occupancy Grid: 2D Array variabler Größe: Wenn sich An Koordinate (X,Y) ein Hindernis befindet => False, sonst True - Um Roboterrotationen zu berücksichten: Transformation des 2D Occupancy Grid in einen 3D Konfigurationsraum: - Ebene in Z-Richtung entspr. Roboterrotation => 4 Ebenen - In jeder Ebene wird Occupancy Grid um die Dimensionen des Roboters in der jeweiligen Rotation erweitert - Potentialfelder: - Roboter wird Startpunkt/-rotation und Zielpunkt/-rotation im Occupancy-Grid gegeben - Berechnung von Potentialfeldern in Konfigurationsraum 1) Mit Attractive (anziehend zu Zielpunkt) / Repulsive Potentialen - Roboternavigation: - Berechnung der Potentialgradienten - Gradientenabstiegsverfahren

1

2 Technische Voraussetzungen

- Implementierung in Python mit einem Jupyter Notebook. - Quellcode auf GitHub: (QR-Code)

- Bedingungen zur Ausführung: Python >= 3.7 - Python Abhängigkeiten: > matplotlib zur Visualisierung > numpny für Mathematische Berechnungen => Installation aller Abhängigkeiten: pip install -r requirements.txt

- Empfehlung zur Ausführung: Anacoda Environment

2

3 Robotermodell

Robotermodell: - Anforderung: "Der Roboter besitzt eine rechteckige Form variabler Größer Implementierung: Robotergröße über Parameter ‘width‘ und ‘length‘ gesteuert

Roboterbewegung: - Translation in vier Richtungen aus Vogelperspektive: "links", "rechts", öbenünd ünten Rotation: Roboter kann sich in 90° Schritten um den Ankerpunkt drehen: ... 270° <= 0° => 90° => 180° => 270° => 0° ... - Der Ankerpunkt der Roboters (Definiert Position im Raum) = Ëcke links oben": Bei einer Rotation von 0°

** TODO: Abbildung Robotermodell + Translation/Rotation ***

4 Konfigurationsraum

Roboter befindet sich in Öccupancy-Grid": 2D Boolean Array öccupancy-grid"der Größe öccu-size-y"x öccu-size-x", das der Umgebung des Roboters entspricht, mit Ursprung "links oben"Für ein Hindernis an der Stelle (X,Y) steht im Occupancy-Grid "False", für eine freie Koordinate "True"

Translation: "x-1"(links), "x+1"(rechts), "y-1"(oben), "y+1"(unten)

*** TODO: Abbildung 2D Occupancy Grid ***

Umsetzung der Kombination aus variabler Größe + Roboterrotation: Ansatz aus Literatur ... (TODO): - Transformation der Bewegung des Roboters der Größe "width", "lengthïn die Punkt-Bewegung des Roboters mit Größe width=1, length=1 - Deshalb für jede der möglichen 4 Rotationen des Roboters (0°, 90°, 180°, 270°): neues Occupancy-Grid in dem das ursprüngliche Occupancy-Grid um die Roboterdimensionen der jeweiligen Rotation erweitert wurde => Ergebnis: 4x 2D-Occupancy-Grids mit erweiterten Hindernis = 4x "Rotationsebene"

*** TODO: Abbildung der erweiterten Occupancy Grids"

=> Zusammenfassen zu einem 3D-Array "configuration-space[rotation][y][x] der Dimension 4 x öccu-size-y"x öccu-size-y-> Rotation: in Rotationsbene "rotation": - "(rotation+1) mod 4"(Rotation um 90° im Uhrzeigersinn) - "(rotation-1) mod 4"(Rotation um 90° gegen Uhrzeigersinn)

*** TODO: Abbildung 3D Plot ***

3D Konfigurationsraum ermöglicht somit 1. kollisionsfreie Translation 2. kollisionsfreie Rotation

5 Berechnung der Potentialfelder

- Anforderung: Die Planung erfolgt [...] mit Hilfe der Potenzialfeldmethode.

Def. Potenzialfeldmethode = jeder Koordinate des diskretisierten Konfigurationsraums wird ein physikalisches Potential zugewiesen:

Vereinfacht: je Höher die "potenzielle Energieëiner Koordinate, desto weiter ist der Punkt auf dem aktuellen Pfad vom Ziel entfernt.

Berechnung dieses Potenzialfelds für den Konfigurationsraum ist Vorbedingung zur Roboternavigation im nächsten Kapitel.

Zur Berechung des Potenzialfelds wurden in dieser Implementierung zwei unterschiedliche Ansätze der Literator verfolgt.

5.1 Attractive/Repulsive Potentiale

Idee: Gesamtpotenzial an einer Koordinate = Kombination aus - Anziehendem Potenzial: Berücksichtigt ausschließlich die Entfernung zum Ziel (TODO: Formel)

*** TODO: Abbildung mit Plot ***

- Abstoßendes Potenzial: Hier wird ausschließlich der Einfluss umliegender Hindernisse berücksichtig. (TODO: Unterschiedliche Formeln in Literatur, hier:)

*** TODO: Abbildung mit Plot ***

Kombination der Potenziale unterschiedlich in Literatur. Hier gemäß ... (TODO) Addition beider Potenziale in einem Punkt

*** TODO: Abbildung mit Plot ***

Problem: Lokale Minima

5

5 Berechnung der Potentialfelder

5.2 Wavefront Potentiale

XYZ stellt sogenannte "Wavefront Potenzialeäls Alternative zu anziehenden und abstoßenden Potenzialen vor.

Berechnung über sog. "WavefrontÄlgorithmus - Initialisierung: > jeder Punkt mit Potenzial 0 > Hindernisse haben nicht definiertes Potenzial (hier np.nan) > Warteschlange = [(Zielpunkt, 2)]

- Wiederhole bis Warteschlange leer: > (Aktueller Punkt, aktuelles Potenzial) = Vorderstes Element in Warteschlange > Setze aktuelles Potenzial für aktuellen Punkt > finde aktuell erreichbare Nachbarn: (x-1, x+1, ... (z-1) mod 4) + kein Hindernis + nicht außerhalb der grenzen des Occupancy-Grids > Nachbarn als Tupel in Warteschlange setzen (Nachbar, Potenzial+1)

- Dieser Breitensuchen-ähnliche Algorithmus breitet immer größer werdende Potenziale "wellenartigïm erreichbaren Raum aus - Punkte, die hinter Hindernissen liegen erhalten somit höhere Potenziale als direkt vom Ziel erreichbare Punkte

*** TODO: Abbildung mit ein paar Iterationen ***

6

6 Roboternavigation

Basierend auf berechneten Potenzialen für jede freie Koordinate im Konfigurationsraum: Roboternavigation durch Gradientenabstiegsverfahren = In jedem Punkt in die Richtung der größten Verringerung des Potenzials gehen.

Fokus der Implementierung: Beheben lokaler Maxima (durch Kräftegleichgewicht)

6.1 Berechnung der Gradienten

Berechnung der Gradienten über np.gradients => Gradienten = Kraftvektoren in x, y und Rotations-Richtung Zeigen in Richtung der größten Verringerung des Potenzials der benachbarten Koordinaten entlang einer Achse

*** Abbildung mit Kraftvektoren in Konfigurationsraum ***

Problem 1: np.gradients berechnet Gradienten an Grenzen von Array als Differenz der letzten beiden Werte (edge-order = 1) Lösung: Manuelles clipping der Gradienten: Wenn Gradient in Richtung der Grenze => Wird auf 0 geclippt Wichtig: durch Clipping können an Grenzen können lokale Minima oder Plateaus geschaffen werden: *** TODO: Abbildung ***

6.1.1 Gradienten an Hindernissen

Problem 2: np.gradients Gradienten an Hindernissen (np.nan) np.nan als Gradienten. Lösung: Behandlung von Hindernissen als Array-Grenzen: (Funktion "compute-obstacle-gradients") - "Grenze rechts"(x+1 == np.nan): Gradient = ... Beispiel: *Abb* - "Grenze links"(x-1 == np.nan): Gradient = ... Beispiel: *Abb* - "Grenze vorne"(y-1 == np.nan): Gradient = ... Beispiel: *Abb* - "Grenze hinten"(y+1 == np.nan): Gradient = ... Beispiel: *Abb* - "Grenze oben"(z-1 mod 4 == np.nan): Gradient = ... Beispiel: *Abb* - "Grenze unten"(z+1 mod 4 == np.nan): Gradient = ... Beispiel: *Abb* Wichtig: durch Clipping können an Hindernissen lokale Minima und Plateaus geschaffen werden: Wenn beide anderen Kraftvektoren ebenfalls 0 sind

6.1.2 Behandlung lokaler Maxima

Problem: lokale Maxima > In jeder Achse haben die beiden entgegengesetzten Nachbarn gleich großes Potenzial > Unterschied lokales Minimum und Plateau: Potenzial der Nachbarn ist dabei niedriger als das der aktuellen Position

7

 6 Roboternavigation

Lösung: > Durch Implementierung von "compute-obstacle-gradients"kann lokales Maximum nicht an Grenze oder Hindernis existieren > Gedachte Grenze zu einem Nachbarn setzen: Berechne Differnz zu einem der Nachbarpotenziale Gradienten

*** Abbildung mit Beispiel ***

6.2 Gradientenabstieg

Lokales Minimum oder Plateau: 0-Kraftvektor in allen Achsen

Startpunkt in 3D Konfigurationsraum

Wiederhole bis Kraftvektor x = Kraftvektor y = Kraftvektor rotation = 0: - Diskretisierung der Gradienten in Translationen des 3D Konfigurationsraum: Für betragsmäßig größten (oder gleich große) Gradienten in x, y oder Rotations-Richtung wird neue Position berechnet - Aus allen möglichen neuen Positionen wird diejenige gewählt, die noch nicht besucht wurde und deren Gesamtkraft maximal ist-

*** Abbildung mit paar Iterationen bis Ziel gefunden ***

6.3 Vergleich der Potenzialfeldmethoden

Anziehendes/Abstoßendes Potenzial: - Nachteil : lokale Minima ... - Nicht-Lineare Gradienten

- Vorteil Wavefront Algorithmus: - Garantierte Konvergenz zum Ziel => Keine Lokalen Minima ... Lineare Gradienten

 8

7 Grenzen der Implementierung

9