Wintersemester 2017/2018
Motion Planning
Prof. Horsch, Rudi Scheitler

Praktikum 2 - Potential Field Method

David Welsch, Ken Hasenbank

17. November 2017

Beschreibung der Aufgabe

Das Ziel des Praktikums bestand darin, 3 Probleme mit Hilfe von Algorithmen der Potential-Field-Method zu lösen.

Der Potential-Field-Method-Algorithmus vereint ein Potenzial in Richtung des Ziels (Attractive) mit einem Potenzial, welches von den Hindernissen weg zeigt (Repulsive) (siehe Abbildung 1). Sinn dieses Algorithmus ist es, dass der Roboter sich in Richtung des Ziels bewegt und von vorhandenen Hindernissen abgestoßen wird und somit nicht mit diesen kollidiert.
Ein Problem dieses Algorithmus besteht darin, dass es möglich ist, dass lokale Minima entstehen, bei denen der Roboter in einem solchen zum Stehen kommt und das Ziel nicht erreicht.

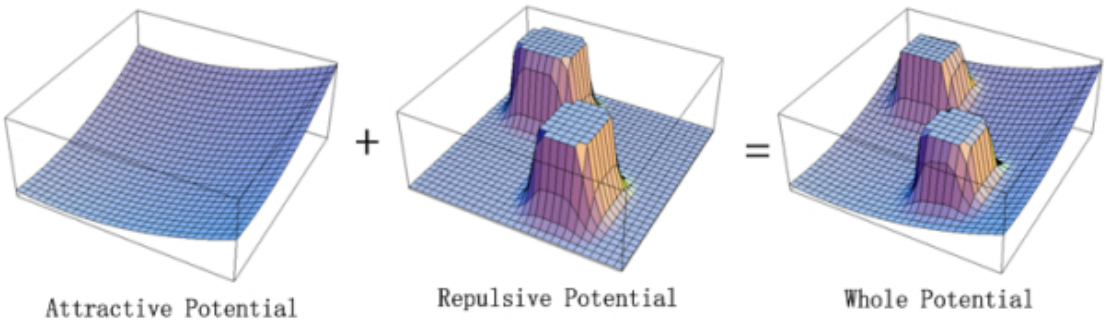

Abbildung 1: Potential Field Method

Eine Erweiterung der Potential-Field-Method stellt der Navigation-Function-Algorithmus dar. Dieser sorgt dafür, dass es im gesamten Raum nur ein Minimum gibt. Dieses Minimum ist bei korrekter Programmierung des Algorithmus und der richtigen Wahl des

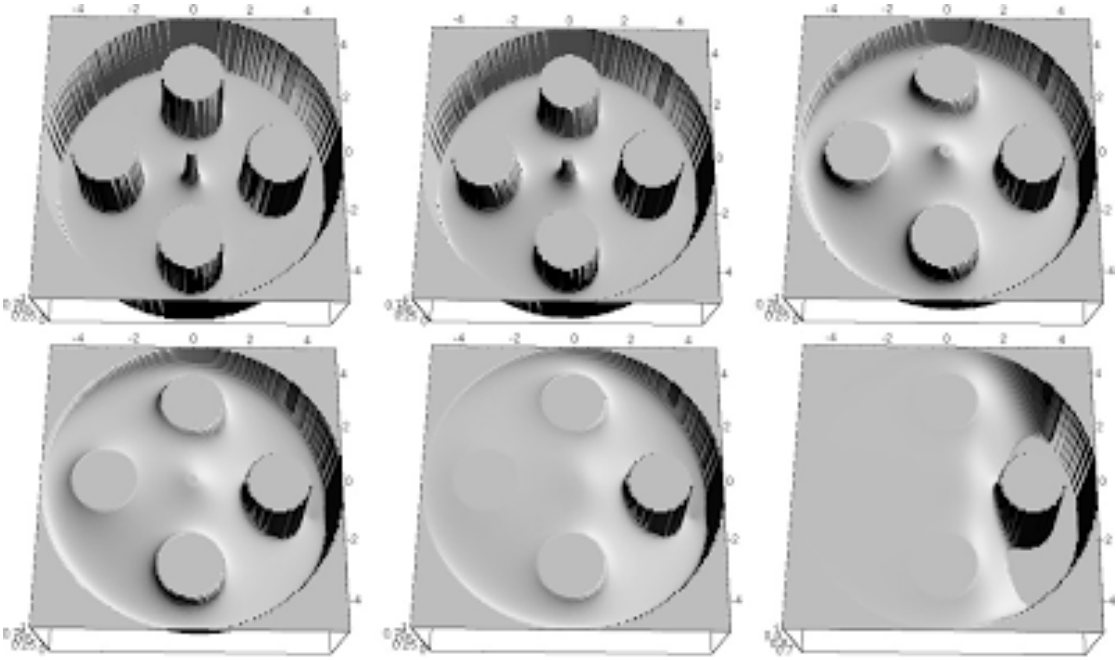

Abbildung 2: Navigation Function

Parameters K das Ziel. In Abbildung 2 ist dieser Algorithmus mit verschiedenen Parametern K dargestellt.

Das Praktikum besteht aus drei Aufgaben. Diese unterscheiden sich folgendermaßen:

- Aufgabe 1
  In der ersten Aufgabe ging es darum, den Potential-Field-Algorithmus zu implementieren.
  a) In dieser Aufgabe waren die Objekte rechteckig.
  b) In dieser Aufgabe waren die Objekte zylindrisch.
- Aufgabe 2 (Zusatzaufgabe)
  In dieser Aufgabe sollte der Navigation Function Algorithmus implementiert werden. Die Objekte und Hindernisse waren dieselben wie in Aufgabe 1.b).

Beschreibung der Lösung

Zur Lösung haben wir den vorgegeben Code-Rahmen verwendet. Wir haben die jeweiligen Funktionen in der Datei *Potential.cpp* verwendet, um die jeweiligen Aufgaben zu lösen.

- Für Aufgabe 1.a) wurde die Funktion *update_box* genutzt. In dieser wird zuerst geschaut, ob das Ziel erreicht wurde (Zeile 118). Dies wird mit der Funktion *goalReached* geprüft. Diese vergleicht die aktuelle Position mit der Zielposition und gibt

beim unterschreiten eines Wertes, welcher durch die Konstante *GOAL_ERROR* beschrieben wird, *true* zurück. Ist das Ziel erreicht, bricht die Funktion ab und gibt *true* zurück.

Ansonsten wird zuerst der Vektor vom Roboter zum Ziel errechnet, dann die Distanz zum Ziel. Ist diese Distanz größer als eine festgelegte Mindestdistanz (*DIST_MIN_GOAL*), wird der Vektor mit einem Skalierfaktor und der Mindestdistanz multipliziert, sowie durch die Länge geteilt. Dieser Vektor beschreibt dann das attraktive Potenzial, welches in Richtung des Ziels zeigt.

Anschließend wird das abweisende (repulsive) Potenzial berechnet. Hierzu wird zu jedem Objekt der Abstand zum Roboter berechnet, um zu sehen, ob das Objekt nah genug am Roboter ist, um den Roboter abzulenken. Dieser Abstand wird beschrieben durch die Konstante *DIST_MIN_OBST*. Wenn ja, wird anhand der in der Aufgabenstellung gegebenen Formel $U_{rep,i}(q)$ berechnet.

Anschließend werden die beiden Potenziale addiert. Der daraus enstehende Vektor wird dann normalisiert und der Roboter in die durch den Vektor gegebene Richtung bewegt.

Damit ist die Aufgabe 1.a) abgeschlossen.

- Die Aufgabe 1.b) wurde in der Funktion *update_cylinder* gelöst. Der Code entspricht genau dem der Aufgabe 1.a), lediglich die Variablen *robot* sowie *obstacle* sind nun Arrays der Klasse *Cylinder* statt *Box*.
- Die Aufgabe 2 (Zusatzaufgabe) wurde mit der Funktion *update_cylinder_navigation* gelöst. Hier wird ebenfalls zuerst geprüft, ob das Ziel bereits erreicht wurde. Anschließend werden alle für die Berechnung der Ausrichtung benötigten Variablen berechnet. Diese sind in der Aufgabenstellung durch die Variable $\Delta\ \gamma(q)$ beschrieben. Hierfür haben wir mehrere Unterfunktionen erstellt, welche im folgenden beschrieben werden:
  - *calc_beta_i*
    In dieser Funktion wird die Variable $\beta_i(q)$ aus der Aufgabenstellung berechnet. Hierfür wird zuerst der Mittelpunkt des Roboters in der Variable *pt* gespeichert. Ist i größer als 0, wird anschließend von diesem Punkt der Mittelpunkt des zu betrachtenden Hindernisses abgezogen und die Länge berechnet. Zu dieser wird der Radius der beiden Objekte (Roboter und Hindernis) quadriert und von dem eben berechneten Wert abgezogen. Dies gilt nur, wenn $i > 0$ ist. Ist $i == 0$, so wird der zuerst berechnete Wert abgezogen und vom quadrierten Radius der beiden Objekte abgezogen.
  - *calc_beta*
    In dieser Funktion wird die Variable $\beta(q)$ berechnet. Hierfür wird die eben

beschriebene Funktion *calc_ beta_ i* verwendet. Sie wird für jedes Objekt im Array der Hindernisse aufgerufen und zusammen multipliziert. Anschließend wird der Wert zurück gegeben.

- *calc_ p_ beta_ i*
  In dieser Funktion wird der Vektor $\beta_i(q)$ aus der Aufgabenstellung berechnet. Dieser wird berechnet, indem die Mittelpunkte von Roboter und zu betrachtendem Objekt voneinander abgezogen und mit 2 multipliziert werden. Ist $i == 0$ wird der Vektor vor der Rückgabe noch negiert, um der Definition gerecht zu werden.
- *calc_ p_ beta*
  In dieser Funktion wird der Vektor $\beta(q)$ aus der Aufgabenstellung berechnet. Hierfür werden die Funktionen *calc_ p_ beta_ i* sowie *calc_ beta_ i* verwendet. Diese werden entsprechend der Formel auf dem Aufgabenblatt aufsummiert bzw. multipliziert.

Außerdem wurden noch folgende Variablen berechnet:

- *qminusgoal*
  Hier wird der Vektor zwischen der aktuellen Konfiguration (dem Mittelpunkt des Roboters) sowie der Zielposition bestimmt. Der Vektor zeigt vom Roboter zum Ziel.
- *distgoal*
  In dieser Variable wird die Länge des eben berechneten Vektors gespeichert.
- *beta*
  In dieser Variable wird der Funktionswert von *calc_ beta* gespeichert, damit die Funktion nur ein mal aufgerufen werden muss, obwohl der Wert in der endgültigen Berechnung mehrfach benötigt wird.
- *p_ beta*
  Hier wird das Ergebnis von *calc_ p_ beta* gespeichert, aus dem gleichen Grund wie oben.
- *distGoalBeta*
  Hier wird die Variable *disgoal* mit $2 \times k$ potenziert und die Variable *beta* hinzu addiert. Da dieser Wert bei unseren Berechnungen immer negativ wurde und er in der späteren Berechnung noch mit einer negativen Zahl potenziert werden muss, was zu einem undefinierten Ergebnis geführt hat, nehmen wir von dieser Zahl den Betrag.

Aus diesen Variablen kann dann der Vektor berechnet werden, der die Richtung der Bewegung des Roboters beschrieben soll. Unsere gesamte Formel dafür lautet:
$2 * qminusgoal$
$* pow(distGoalBeta, 1/k)$
$- pow(distgoal, 2) * (1/k) * pow(distGoalBeta, (1/k) - 1)$
$* (2 * k * pow(distgoal, (2 * k) - 2) * qminusgoal + p_beta)$

)/(
$pow(distGoalBeta, 2/k)$

In den verbleibenden Zeilen wird der Roboter dann in die eben berechnete Richtung fortbewegt und *false* zurück gegeben, um der aufrufenden Funktion zu signalisieren, dass das Ziel noch nicht erreicht wurde.

Damit ist auch die Aufgabe 2 abgeschlossen.

Was wir gelernt haben

Nachdem wir im ersten Praktikum bereits anschaulich gesehen haben, wie Wegfindungsalgorithmen aussehen können, war es nun sehr interessant für uns einen weiteren solchen Algorithmus zu implementieren und auch die Unterschiede zu sehen. Besonders die Zusatzaufgabe stellte dabei eine große Herausforderung dar, da die Formeln zum Teil schwer nachzuvollziehen waren.