Interface gebundene Konstruktion eines Control Sets nach Pivtoraiko and Kelly

Vincent Dahmen (6689845), Tina Van (6707886)
Fakultät Informatik der Universität Hamburg

...

1 Zusammenfassung

In diesem Report wird unsere Implementation eines Algorithmus zur Erstellung einer Sammlung von atomare Bewegungen für Agenten, die nur über eine eingeschränkte Bewegungsfreiheit verfügen, erläutert. Hierfür fassen wir zunächst Pfade mit ähnlichem Verlauf zu Gruppen zusammen. Anschließend sortieren wir alle nicht atomaren Gruppen aus. Im letzen Schritt übernehmen wir die kürzeste Bewegung jeder Gruppe als deren Repräsentant. Die Menge der Repräsentanten kann anschließend für verschiedene Pfadfindungsalgorithmen benutzt werden und garantiert dabei dass alle Pfade begehbar sind. Der gesamte Algorithmus orientiert sich dabei stark an dem theoretischen Entwurf[1] von Pivtoraiko und Kelly

Einleitung

Die meisten Pfadfindungsalgoritmen wie A* und ähnliche arbeiten mit einer Gitter-Repräsentation ihrer Umgebung. Die meisten Agenten wie ferngesteuerte Autos oder Quadcopter sind aber nicht in der Lage ihre Umgebung als "Punkteänzusteuern. Um also Pfadfindungsalgoritmen (im folgenden auch als suchbasierte Algorithmen) schlagen Pvitoraiko und Kelly ein Konzept namens "Control-Sets"vor. Hierbei versucht man sogenannte "Motionprimitivesßu finden. Gemeint sind die Grundbewegungen, also jene aus denen sich alle anderen Bewegungen zusammensetzen. Da eine "Motionprimitveälso eine kleinstmögliche unteilbare Bewegung ist, werden wir den Begriff ätomare"Bewegung im folgenden synonym verwenden.

Mithilfe dieser elementaren Bewegungen lässt sich von einem Startpunkt ein Koordinaten-Netz herleiten. Hierfür wird erst jede Bewegung einmal ausgeführt und deren Endpunkte markiert. Anschließend wird von jedem Endpunkt wieder jede Bewegung ausgeführt und so weiter. Mit diesem Verfahren erhält man ein beliebig großen Gitter-Netz der Umgebung, auf den nun mit beliebigen suchbasierten Algorithmen weitere Pfadplannung betrieben werden kann.

Wir beschäftigen uns mit der konkreten Implementation eines Control-Set Generators. Aufgrund der Vielfalt an Agenten versuchen wir ein möglichst einfaches Konzept vorzustellen, dass sich für möglichst viele Agenten umsetzten lässt. Um das zu gewährleisten haben wir einige Stellen lediglich als Interface definiert. Dahinter steckt die Idee das ein Entwickler unsere Implementation einfach benutzen kann. Deswegen verweisen wir im weiteren auch auf einen möglichen Nutzer. Gemeint ist jemand der den Code für sein Projekt benutzt.

Die größte Herausforderung ergab sich für uns aus genau dieser Kompatibilität. Um sicherzustellen dass alle Agenten den gleichen Code benutzen können mussten wir zum Beispiel ein möglichst einfaches Konzept für die

Modelierung n-dimensionaler Position unterbringen.

Im wesentlichen behandeln wir im Kapitel 2 alle Teilprobleme und ihre Bewältigung und im Kapitel 3 den genauen Aufbau des Generators.

Im Anhang folgt der Code unserer Java-Implementation welcher auch in einem öffentlichen GitHub-Repostory zur Verfügung steht.

Inhaltsverzeichnis

2 Definitionen der Datenstrukturen

2.1 Control-Set

Ein Control-Set ist eine Menge atomaren Bewegungen, die ein Agent ausführen kann. Das Control-Set wird von suchbasierten Algorithmen(Erläuterung!) genutzt.

Ein Grund für die Nutzung des Control-Sets ist, dass an Zeit und Rechenaufwand für die lokale Plannung gespart wird, weil das Erstellen eines Control-Sets nur einmalig notwendig ist und dieses wiederverwendbar ist. Weitere Nutzen des Control-Sets werden im Paper erwähnt.

Im Code wird das Control-Set als eine Map von Positionen und Bewegungen implementiert.

[strange...] Bei der Erzeugung eines Control-Sets wird auf den Bewegungserzeuger und einem Intervall, den man selbst bestimmt, geachtet.

Bei der Nutzung des Control-Sets ist es Vorteilhaft, dass die Bewegungen in dieser Art Abgespeichert werden, weil man sich schnell und gezielt Bewegungen in Abhängigkeit der Position aus dem Control-Set abrufen kann.

Auch ist das Control-Set abhängig von den Bewegungserzeugern, was dem Nutzer die Freiheit gibt, den Code anzupassen.

Zum Beispiel kann der Benutzer zwei Control-Set generieren lassen mit unterschiedlichen Bewegungserzeugern und den selben Intervall und diese miteinander vergleichen.

— Definition Problem (was soll es lösen) Vorteile von Controll-Sets gegenüber reiner "listeÏmplementation

2.2 Bewegung

Die Bewegung stellt eine abstrahierte "Bewegung"wie sie auch natursprachlich verwendet wird dar. In unserem Modell wird jede Bewegung von etwas erzeugt (siehe Bewegungserzeuger) und kann beliebig oft reproduziert werden.

Außerdem hat sie eine variable Startposition und einen davon abhängigen Verlauf.

Für ein Control-Set ist eine Bewegung die elementare Datenstruktur in unserer Implementation hat eine Bewegung einen relativen Verlauf, einen Erzeuger samt Erzeuger-Argumente und eine Start Position.

Mithilfe des Erzeugers kann die Bewegung tatsächlich von einem Agenten äusgeführt"werden.

Der relative Verlauf ist eine Liste, in der jeder Eintrag einem SZwischenstoppëntspricht.

Jeder Eintrag besteht aus einer Berechnung für jede Dimension in der sich etwas ändert.

(Das Konzept der Dimensionen in underem Modell wird 2.4 näher erläutert)

Mit dieser Abstraktion kann der Verlauf von einer Bewegung mit einem beliebigen Startpunkt berechnet werden.

— definition vereinfachen "funktion"3 und 5 zusammen führen

2.3 Bewegungserzeuger

Unter dem Konzept der Bewegungserzeuger versuchen wir in unserem Modell alle Möglichkeiten eines Agenten eine Bewegung zu erzeugen zu modellieren.

Ein Bewegungserzeuger zeichnet sich dadurch aus, dass er eine endliche Menge von Bewegungen erzeugen kann.

Hierbei haben wir die Implementation bewusst frei gelassen, um eine maximale Anpassung zu erlauben.

In unserem Konzept definiert man für einen Agenten einen sinnvollen Bewegungserzeuger (z.B. einen Motor) und kann danach nur noch die allgemeinere Klassen "Bewegung"benutzen.

Insbesondere bei Austausch des Antriebssystem kann die komplette Plannung (solange sie auf Control-Sets beruht) beibehalten werden. Es muss lediglich ein anderer Bewegungserzeuger benutzt werden.

In unserem Modell ist ein Agent nichts weiter als eine Menge von Bewegungserzeugern, mit denen er sich fortbewegt.

Diese Erzeuger stellen die Gesamtmenge an Bewegungen dar.

In dem Algorithmus entsteht aus genau dieser großen Menge nachher das Control-Set.

In unserer Implementation existiert lediglich ein Interface (siehe oben).

Das Zusammenspiel von Bewegungserzeuger entspricht der Fabrikmethode nach GoF[2]. — Problem nach vorne (sehr viele mögliche antriebe) Beispiel besser integrieren (als eigener Absatz)

2.4 Position

Eine Position ist ein benutzerdefinierte Koordinate. Sie enthält, die vom Nutzer gewählten Werte und Vektoren und ist der kleinste Bestandteil einer Bewegung.

Die Position wird für die Definition einer Bewegung verwendet. Sie beinhaltet Informationen wo der Agent sich befindet und in welchem Zustand.

In unserem Modell lässt sich der Gesamtzustand eines Agenten als Reihe von Werten für Dimensionen beschreiben. Eine solche "Dimension"kann dabei sowohl eine tatsächliche physikalische Dimension wie eine X-Achse, als auch eine beliebige Variable samt Belegung, wie die Posiition eines Greifarmes sein.

Wir untersscheiden lediglich zwischen Dimensionen die als Strecke dargestellt werden können (wie z.B. Zeit, x-Achse, ...) und sogennaten Üser defined Dimensions"(UdFs) welche alles andere abdeckt.

Möchte man z.B. die Entfernung zwischen zwei Positionen wissen, so kann man einfach die Differenz aller Dimensionen die keine UdFs sind entweder als Tupel oder als, mit der geometrischen Summenformel verrechneten, Gesamtwert angeben.

Technisch ist die Belegung aller Dimensionen ein double, da dies genug Genauigkeit bietet.

Zusätzlich hat jede Dimension eine Toleranz der ebenfalls in Form eines double hinterlegt wird. Dieser Wert beschreibt wie groß die Differenz in dieser Dimension maximal sein darf, so dass der Wert noch als gleich angenommen wird.

In unserem Modell werden alle Bewegungen über ihren Bewegungserzeuger erstellt, so dass jeder Bewegungserzeuger für jede Bewegung individuell Dimensionen und Toleranzen verändern kann.

Somit kann jede denkbare Art einer Bewegung modelliert werden.

— problem vorweg (zu viele eigenschaften, bewegung)

2.5 Bewegungstunnel

Wenn man eine Bewegung als einfachen Vektor interpretiert, dann ist ein Bewegungstunnel eine Menge dieser Vektoren.

Diese Tunnel kann man sogar graphisch darstellen, wenn man die Vektorgraphen verbindet und ähnelt dem path equivalence class in Pivtoraiko und Kelly[1].

Als spezielle Eigenschaft haben alle Vektoren ein gemeinsamen Startpunkt und einen gemeinsamen Endpunkt.

Um zu prüfen ob ein Bewegung sich in andere zerlegen lässt, nach der Simulations Studie von Pivtoraiko und Kellys path decomposition, sind Bewegungstunnel hilfreich.

Man muss lediglich überprüfen ob der Tunnel mit anderen Bewegungen "durchquerbarïst. (siehe Abschnitt 3)

Es ist auch leicht möglich die Bewegung zu finden, die möglichst dicht in der Mitte liegt. Hiefür wählt man einfach den Vektor mit der kleinsten geometrischen Länge aus. — allgemeine umformulierungen (...)

3 Implementation

Ein Control-Set besteht aus Bewegungen, die in einer Map abgespeichert sind. Um eine bestimmte Bewegung aufzurufen, wird eine Position angegeben und die Bewegung wird ausgegeben.

Zu erst werden alle Bewegungen erzeugt und als eine Menge von Bewegungen abgespeichert. Dies ist wie der erwähnte reachability graph in Pivtoraiko und Kellys Paper[1]. Für diese werden, wenn sie eine bestimmter Länge, (die entweder kleiner oder gleich dem bestimmten Intervall) haben,

Endpunkte berechnet, die wieder als eine Menge von Bewegungen abgespeichert werden. Damit bestimmt man die größe der Pfade im Control-Set und werden in Pivtoraiko und Kellys Paper[1] als Control-Set Radius beschrieben. Daraus lassen sich Bewegungstunnel finden. Um abzusichern, dass ein Bewegungstunnel b_1 nicht weiter zerteilt werden kann, wird b_1 mit allen anderen Bewegungstunneln verglichen, ob man b_1 mit einer der anderen Bewegungstunneln den Endpunkt von b_1 erreichen kann. Wenn dies der Fall ist, wird b_1 entfernt und der Algorithmus wählt einen weiteren Bewegungstunnel als b_1 aus der Bewegungstunnel Menge und wiederholt den Vorgang. Hiermit implentieren wir das in 2.6 beschriebene Verhalten und erreichen damit dass alle "ähnlichen"Bewegungen in Gruppen (bei uns Bewegungstunnel) unterteilt werden. Nun werden noch die übrigen Tunneln normalisiert, wobei die kürzesten Bewegunge jeder der Tunneln im Control-Set abgespeichert werden.

4 Fazit

fazit:

a) Ziel? Unser Ziel war, dass wir ein Control-Set erstellen, nach dem Verfahren von Pivtoraiko und Kelly, mit unerem Code, wobei wir nicht die Information benötigen wollen, welche Parameter oder Variablen eine gültige Bewegung ist.

b) wie erreicht? Eine Abstrakte Klasse Bewegungserzeuger musste angelegt werden, da wir nicht wissen, was eine Sinnvolle Bewegung ist. Dies überlassen wir dem Nutzer. Dieser Bewegungserzeuger übergibt uns, alle Möglichkeiten, die der Agent fähig ist, sich fort zu bewegen vom Ausgangspunkt. Somit haben wir alle nötigen Informationen um ein spezifisches Control- Set für diesen Agenten zu erstellen. Nach Pivtoraiko und Kelly haben müssen wir die Bewegungen diskretisieren. Deshalb fordern wir vom Nutzer ein Wert für die Toleranz und das Intervall. Um im nächsten Schritt die Bewegungen zu zerlegen, haben wir das Prinzip von Bewegungstunneln eingeführt. Diese hat die equivalence class von Pivtoraiko und Kelly als Grundidee. Die dadurch gewonnen Repräsentanten der einzelnen Bewegungstunneln sind das Control-Set.

Dadurch haben wir einerseits einen flexiblen Code erhalten, der n- beliebige Dimensionen von Bewegungen haben kann. Das führt dazu, dass der Nutzer selbst definieren muss, was er eine gültige Bewegung ist und welche Variablen von Nutzen sind. Damit, kann ein Nutzer nach seinen Bedürfnissen, den Code anpassen.

Andererseits, geben wir dem Nutzer wenige Vorgaben, an denen er sich Richten kann. Das kann dazu führen, dass unser Code uneffizient verwendet wird, da keine "default"Bewegung definiert ist.

Ein nächster Schritt wäre, die noch nicht im Code berücktsichtigte Be-

rechnugszeit zu optimieren.

struckturierung von b) Probleme Pos -entferungn -beliebig viele "Bewegungabstrakt -relativen Verlauf Lösung Pos als Liste, unterscheidung udd "messbaren"pythagoras über messbare als Länge

wechsel zu relativen verlauf und fabrikmethode

etc probleme nicht vohersebar waren, rückbezug zu paper

5 Quellen

M. Pivtoraiko, A. Kelly, Generating near minimal spanning control sets for constrained motion planning in discrete state spaces, In Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), 2005, pp. 3231–3237.

Erich Gamma, Richard Helm, Ralph Johnson, Jon Vlissides, published by Kevin Zhang, Design Patterns, Elements of Reusable Object Oriented Software, letzter Zugriff: 30.08.2015, http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

6 Appendices

ControlSet.java

Bewegung.java

BewegungsErzeuger.java

Position.java

NeutralePosition.java

Bewegungstunnel.java