Raum-Zeit-Scheduler

Bericht

Inhalt

1 Überblick

In einem klassischen Betriebssystem bestimmt der Scheduler, wann ein Prozess auf dem Prozessor ausgeführt wird. Im Falle eines Multiprozessorsystems ist zudem auch die Frage zu klären, auf welchem (wo) Prozessor der Prozess laufen soll. An einen Scheduler eines verteilten mobilen Betriebssystems werden weitere und abgewandelte Anforderungen gestellt.

Ein verteiltes Betriebssystem vereint mehrere physisch unabhängige Arbeitseinheiten unter sich, die in der realen Welt agieren. Zu erledigende Aufgaben bringen Spezifikationen zu Gebiet und Zeitraum mit sich, in dem bzw. zu der sie bearbeitet werden müssen. Diese könnte beispielsweise so aussehen,

dass an einem gewissen Ort ein Bild vom Sonnenuntergang aufgenommen werden soll. Um diese Aufgabe zu erfüllen, muss sich zu gegebener Zeit eine Arbeitseinheit (Roboter) an diesem Ort befinden, um das Foto schießen zu können.

Die Aufgabe des Schedulers ist zu einer Aufgabe einen Ort, eine Zeit und eine Arbeitseinheit zu bestimmen, welche die Spezifikation erfüllen. Dabei muss gewährleistet werden, dass es der Arbeitseinheit auch tatsächlich möglich ist, diesen Ort zur angegebenen Zeit zu erreichen, ohne dass andere Aufgaben in Verzug geraten.

2 Lösungsansatz

Bei der Zuteilung von Aufgaben muss der Scheduler drei Variablen belegen: Ort, Zeit und Arbeitseinheit. Diese müssen zum einen den gestellten Bedingungen der Spezifikation genügen und zum anderen realisierbar sein.

Der hier beschriebene Ansatz belegt nach und nach die Variablen mit konkreten Werten und überprüft sie auf Umsetzbarkeit. Erweist sich ein Wert einer zu Beginn belegten Variable als unergiebig, so wird ihr nach dem Backtracking-Prinzip ein neuer Wert zugewiesen bis entweder eine gültige Belegung gefunden wurde oder alle Kombinationsmöglichkeiten erschöpft wurden.

2.1 Welt

Die Arbeitseinheiten sind reale Objekte, die sich auf der Welt wiederfinden. Eine Einheit ist ein mobiles Gerät, welches sich in der Welt bewegt. Neben den Arbeitseinheiten existieren auch andere Objekte, die Hindernisse darstellen. Die Welt selbst wird auf eine zweidimensionale Ebene reduziert, sodass die Höhe der Arbeitseinheiten und Hindernissen nicht berücksichtigt wird.

Hindernisse werden in statische (bewegungslose) und dynamische (bewegte) Hindernisse eingeteilt. Ein statisches Hindernis wird durch ein Polygon dargestellt. Ein dynamisches Hindernis besitzt außerdem noch einen Bewegungspfad (Trajektorie). Die Arbeitseinheiten selbst stellen für andere Einheiten dynamische Hindernisse dar.

Eine weitere Eigenschaft der Welt ist die Unterscheidung zwischen Vergangenheit und Zukunft. Zukünftige Ereignisse können noch ein- oder umgeplant werden. Währenddessen sind vergangene Ereignisse unveränderlich.

2.2 Aufgaben und Spezifikationen

Eine Aufgabe wird von dem Scheduler einem Roboter zugewiesen, der diese an einem bestimmten Ort zu einer bestimmten Zeit ausführt. Dieser geht eine Spezifikation voraus, die Anforderungen an den Ort und den Zeitpunkt stellt. Außerdem gibt diese die Dauer zur Bearbeitung der Aufgabe an. Der Ort muss innerhalb eines definierten Areals liegen und der Zeitpunkt innerhalb eines Zeitintervalls. Der Scheduler nimmt solch eine Spezifikation entgegen und erzeugt daraus eine konkrete Aufgabe, die er einer Arbeitseinheit zuweist.

2.3 Ortsbestimmung

Der erste Schritt von der Spezifikation zur konkreten Aufgabe ist die Ortsbestimmung. Es wird innerhalb des spezifizierten Areals ein Punkt ermittelt, welcher sich nicht innerhalb eines statischen Hindernisses befindet. Dieser Punkt ist also ein Element aus der Vereinigung der freien Fläche und des Areals der Spezifikation (TODO: Abbildung).

Dynamische Hindernisse werden an dieser Stelle noch nicht berücksichtigt, da ein genauer Zeitpunkt noch nicht bestimmt wurde. Möglicherweise führt der gewählte Ortspunkt zu keiner realisierbaren Belegung der übrigen Variablen, wenn beispielsweise dynamische Hindernisse den Weg versperren. In solch einem Fall muss ein neuer alternativer Ortspunkt gewählt werden. Die Anzahl der Alternativen wird dabei beschränkt, um eine endlose Suche zu vermeiden.

Der erste Punkt teilt das ursprüngliche Areal in vier Unterareale, indem es entlang der x- und y-Richtung geschnitten wird. Wenn aufgrund von Backtracking ein neuer Punkt gewählt werden soll, so wird ein innerer Punkt aus einem der vier Unterareale bestimmt. Es wird dabei immer das größte Unterareal gewählt. Dieser Punkt zerteilt nun wiederum das jeweilige Areal in vier Areale, welche der bisherigen Menge an Arealen hinzugefügt werden. Somit ergibt sich stets eine hinreichend gleichmäßige Verteilung an Punkten.

2.4 Arbeitseinheit- und Zeitraumbestimmung

Da nun ein Ortspunkt gewählt wurde, kann nun nach einer Arbeitseinheit gesucht werden, welche diesen Punkt auch innerhalb des erlaubten Zeitintervalls erreichen kann.

Eine Arbeitseinheit führt ihr aufgetragene Aufgaben in einer Liste nach und nach aus. Sie kann nur eine Aufgabe zur selben Zeit bearbeiten. Es kann also nur eine Einheit für eine neue Aufgabe ausgewählt werden, die auch die entsprechende Zeit dazu hat.

Im Normalfall muss eine Einheit vor und nach der einzuplanenden Aufgabe ebenfalls Aufgaben erledigen. Sie muss also genügend Zeit haben, um nach Abschluss der vorherigen Aufgabe zur neuen zu fahren, diese abzuarbeiten, um dann noch rechtzeitig die nachfolgende zu beginnen. Zudem muss die neue Aufgabe auch weiterhin innerhalb des spezifizierten Zeitraums begonnen werden.

Formal müssen also drei Bedingungen geprüft werden:

1. Die Zeit ist ausreichend, um von der vorherigen Aufgabe zur neuen zu fahren bevor sich das erlaubte Zeitfenster schließt:
$$t_{max} - t_1 \geq \frac{s_1}{v_{max}}$$
2. Die Zeit ist ausreichend, um von der neuen Aufgabe zur nachfolgenden zu fahren ohne das Zeitfenster zu verletzen:
$$t_2 - t_{min} \geq \frac{s_2}{v_{max}} + d$$
3. Die Zeit ist ausreichend, um die neue Aufgabe nach der vorherigen und vor der nachfolgenden abzüglich der Fahrzeit vollständig auszuführen:
$$t_2 - t_1 \geq \frac{s_1 + s_2}{v_{max}} + d$$

s_1 und s_2 sind die Strecken der Luftlinien zu und von der neuen Aufgabe. v_{max} ist die Höchstgeschwindigkeit der Arbeitseinheit. t_1 und t_2 sind die Zeitpunkte, an denen die vorherige Aufgabe beendet und die nachfolgende gestartet wird. d ist die Dauer der neuen Aufgabe. Das Zeitintervall, in der die neue Aufgabe gestartet werden muss, ist durch $[t_{min}, t_{max}]$ gegeben.

Die ersten beiden Bedingungen stellen sicher, dass die neue Aufgabe während des vorgegebenen Zeitintervalls begonnen wird. Die dritte Bedingung ist wahr, wenn zwischen Ende und Beginn der

bereits eingeplanten Aufgaben genügend Zeit für den Fahrweg und der Aufgabendauer vorhanden ist.

Jede Arbeitseinheit besitzt Pausen, in denen sie keine Aufgabe bearbeitet. In diesem Sinne sind Wege zu Aufgabenorten ebenfalls Bestandteile von Pausen. Nur in jenen Pausen, welche die drei Bedingungen erfüllen, kann die neue Aufgabe eingeplant werden. Wurde solch eine Pause gefunden, so muss nun noch geprüft werden, ob auch gültige Wege zu und von dem Aufgabenort existieren.

2.5 Wegplanung

Bis hierhin wurden ein Ortspunkt und eine Pause einer bestimmten Arbeitseinheit als Kandidaten für die Einplanung einer Aufgabe gewählt. Ob dies jedoch auch möglich ist, ergibt sich erst bei der Wegplanung. Wie bereits in 2.4 erwähnt, besitzt die einzuplanende Aufgabe im Regelfall einen Vorgänger und einen Nachfolger. Daher muss die Arbeitseinheit vom Vorgänger zur neuen Aufgabe fahren, diese ausführen, und anschließend zum Nachfolger fahren. Es sind also jeweils Trajektorien für beide Fahrwege zu ermitteln, welche sämtliche Kollisionen mit Hindernissen vermeiden.

Da die Aufgabe selbst lediglich ein Zeitintervall vorgibt, in welchen diese zu beginnen ist, wurde bisher noch kein konkreter Zeitpunkt dafür bestimmt. Dieses Zeitintervall wird durch die gewählte Pause aus dem vorigen Schritt eingeschränkt. Bei der Wegplanung kann also der schnellstmögliche Weg ermittelt werden. Allerdings muss hierbei beachtet werden, dass die Arbeitseinheit auch genügend Zeit zum Ausführen der Aufgabe hat, ohne dass es einem dynamischen Hindernis in die Quere kommt.

Für die zweite Trajektorie muss die Wegplanung eine Trajektorie liefern, die als Zielzeitpunkt den Start des Nachfolgers verwendet. Es wird hier also nicht nach dem schnellstmöglichen Weg gesucht.

Für die Wegfindung gilt an dieser Stelle zu beachten, dass Arbeitseinheiten für andere Einheiten ebenfalls dynamische Hindernisse darstellen. Ihre Trajektorien liefern dazu das Bewegungsmuster. Neben den Fahrwegen müssen auch Trajektorien für die Aufenthalte während der Ausführung von Aufgaben berücksichtigt werden.

Zur Bestimmung des Weges selbst wird der Ansatz der Pfad-Geschwindigkeits-Dekomposition verwendet (Kant & Zucker, 1986), welcher im Bericht „Path-Velocity-Decomposition“ erläutert wurde.

3 Implementierung

Die Implementierung ist in Java realisiert. Zur Repräsentation der äußeren Welt sind geometrische Objekte und Operationen nötig, für die auf die *Java Topology Suite* zurückgegriffen wird.

3.1 Welt

Die Klasse `World` implementiert die Repräsentation der äußeren Umgebung in der sich die Arbeitseinheiten bewegen. Sie wird zu Beginn konfiguriert und ist anschließend unveränderlich. Daher bildet die Klasse nur die statische bzw. unveränderliche Umgebung ab.

3.1.1 Arbeitseinheit

Arbeitseinheiten gehören zwar im eigentlichen Sinne zur Welt, sind können allerdings ihr Bewegungsmuster ändern und werden daher von der Klasse `World` nicht berücksichtigt.

Die Perspektive auf die Welt ist von Einheit zu Einheit verschieden. Dies liegt daran, dass jede Einheit auf Punktgröße reduziert wurde, um die Wegplanung zu vereinfachen. Die Folge ist, dass alle übrigen Hindernisse (und Arbeitseinheiten) um den Durchmesser der betrachteten Einheit erweitert wird.

Eine Arbeitseinheit speichert ihre Fahrwege segmentweise ab. Ein Segment stellt dabei den Weg von einer Aufgabe zur nächsten dar. Der Aufenthalt während der Bearbeitung einer Aufgabe wird ebenfalls durch ein Segment dargestellt sowie der Stillstand der Einheit, wenn sie die letzte Aufgabe erledigt hat und noch keine neue Aufgabe erhalten hat. Durch die Segmentierung können Teile der Fahrwege unabhängig von den übrigen Teilen neuberechnet werden. Implementiert werden drei Arten von Segmenten wie bereits geschildet: `MovingWorkerUnitObstacle`, `OccupiedWorkerUnitObstacle` und `IdlingWorkerUnitObstacle`. Diese stellen zugleich dynamische Hindernisse der Klasse `DynamicObstacle` dar.

3.1.2 Statische Hindernisse

Statische Hindernisse sind unbewegte Objekte innerhalb der Welt. Sie werden durch Objekte der Klasse `StaticObstacle`, welche ein `Polygon` enthält.

3.1.3 Dynamische Hindernisse

Dynamische Hindernisse werden durch die Klasse `DynamicObstacle` implementiert und bestehen aus einem `Polygon` und einer `Trajectory`. Sie können ebenso wie statische Hindernisse der Welt hinzugefügt werden.

Obwohl auch Arbeitseinheiten dynamische Hindernisse sind, werden diese der Welt nicht explizit hinzugefügt, weil sie mit jeder neuen Aufgabe ihre Bewegungspfade ändern können. Außerdem darf eine Einheit sich nicht selbst „sehen“, da sie ansonsten mit sich selbst in der Wegplanung fälschlicherweise kollidiert. Arbeitseinheiten werden bei der Wegfindung implizit betrachtet.

3.1.4 Trajektorien

Trajektorien der Klasse `Trajectory` beschreiben einen Pfad durch Zeit und Raum. Es handelt sich dabei um eine abstrakte Klasse, die durch `SimpleTrajectory` und `DecomposedTrajectory` implementiert wird.

`SimpleTrajectory` ist ein einfacher Container für die Orts- und Zeitpunkte einer Trajektorie. `DecomposedTrajectory` speichert dagegen die Orts- und Zeitkomponente getrennt voneinander ab. Entgegen der `SimpleTrajectory`, welche einen $x \times y \times t$-Pfad speichert, liegt hier die Bahn als $x \times y$- und $s \times t$-Pfad vor. Dies ermöglicht eine getrennte Berechnung beider Pfade und ist bei der Aktualisierung des Geschwindigkeitsprofils von Nutze, da dann nur der `VelocityPathfinder` erneut ausgeführt werden muss, während eine Berechnung des Ortspfades entfällt.

3.1.5 Zeit

Die Unterscheidung zwischen Vergangenheit und Zukunft wurde noch nicht umgesetzt. Bisher liegen alle Ereignisse stets in der Zukunft. Daher können Aufgaben stets eingeplant werden.

3.2 Aufgaben

Eine Aufgabe ist eine Tätigkeit, die eine bestimmte Arbeitseinheit an einem bestimmten Ort (x, y) zu einem bestimmten Zeitpunkt t ausführt und dazu eine gewisse Zeit d benötigt. Die Klasse `Task` implementiert die Aufgabe und kann als ein vierstelliger Tupel dargestellt werden:

$$(x, y, t, d)$$

Aufgaben werden in Listen der Arbeitseinheiten gespeichert, denen sie zugeordnet wurden.

3.3 Aufgabenspezifikationen

Eine Spezifikation stellt Anforderungen, die das Tupel einer Aufgabe erfüllen muss. Sie gibt ein Areal vom Typ `Geometry` an, in dem sich der Ort (x, y) der Aufgabe befinden muss. Zwei Zeitpunkte geben das Zeitfenster $[t_{min}, t_{max}]$, in der die Aufgabe gestartet werden muss. d gibt die Dauer der Aufgabe an. Ein Objekt der Klasse `TaskSpecification` speichert diese Werte, das an den Scheduler zur Einplanung einer neuen Aufgabe übergeben werden kann.

3.4 Arbeitseinheit

Die Klasse `WorkerUnit` bildet die Eigenschaften einer Arbeitseinheit ab. Dazu gehören ihre Form, Höchstgeschwindigkeit sowie der Ort und Zeitpunkt, wo die Einheit startet. Außerdem führt die Einheit eine chronologisch sortierte Liste ihrer Aufgaben sowie ihrer Fahrsegmente, die den Weg zu den jeweiligen Aufgaben als dynamische Hindernisse beschreiben.

Wenn der Scheduler einer Einheit eine neue Aufgabe zuweist, so gibt dieser auch immer ein neues Fahrsegment zu dieser Aufgabe an. Sollte es eine Nachfolgeaufgabe geben, so wird dieses Segment zu dieser Aufgabe aktualisiert.

Eine Einheit kann eine Liste ihrer Pausen innerhalb eines gegebenen Zeitintervalls erstellen. Dies ist nützlich, wenn eine freie Einheit für eine neue Aufgabe gesucht wird.

3.5 Ortsbestimmung

Die Ortsbestimmung erfolgt durch die Klasse `LocationIterator`, welche eine beliebige Menge an Punkten aus einer gegebenen Fläche extrahiert. Die Anzahl der Punkte muss begrenzt werden, da eine Fläche unendlich viele Punkte enthält.

Aus dem Areal einer Aufgabenspezifikation und der Weltkarte wird eine Fläche erstellt, die alle Freiflächen des Areals repräsentiert. Somit wird gewährleistet, dass kein Punkt innerhalb eines statischen Hindernisses gewählt wird.

Objekte der Klasse `Geometry` bieten die Funktion an, einen inneren Punkt zu extrahieren. Um weitere Punkte zu extrahieren, werden sukzessive immer kleinere Teilflächen der Freifläche betrachtet. Wie bereits in 2.3 beschrieben teilt ein neuer Punkt die Teilfläche, aus der er entsprungen ist, in vier neue Teilflächen ein. Die neuen Teilflächen werden in eine nach Fläche sortierten Warteschlange hinzugefügt. Bereits verwendete Flächen werden entfernt.

3.6 Arbeitseinheit- und Zeitraumbestimmung

Der `WorkerUnitSlotIterator` sucht nach den Pausen der Arbeitseinheiten, welche die Bedingungen aus 2.4 genügen. Dazu betrachtet er jede Arbeitseinheit und lässt sich eine Liste ihrer Pausen erzeugen, die sich im Zeitfenster der Spezifikation befinden.

3.7 Wegplanung

Die Planung der Wege zur neuen Aufgabe und der darauffolgenden wird vom `TaskPlanner` übernommen. Er leitet die Berechnung von räumlichen Pfaden und Geschwindigkeitsprofilen ein. Die räumliche Planung übernimmt ein `SpatialPathfinder` während die zeitliche von einem `VelocityPathfinder` erledigt wird.

Zunächst erfolgt mithilfe des `StraightEdgePathfinder`s die räumliche Planung der Trajektorien, die zur neuen Aufgabe verlaufen bzw. von ihr wegverlaufen. Da hierdurch alte Trajektorien überschrieben werden, ändern sich somit auch die dynamischen Hindernisse. Dadurch kann es vorkommen, dass die Trajektorien anderer Arbeitseinheiten dieser Einheit ausweichen, die unterdessen einen neuen Kurs eingeschlagen ist. Durch eine Neuberechnung der Geschwindigkeitskomponente solcher Trajektorien kann dieser Umstand behoben werden. Der `TaskPlanner` stellt daher vor der Geschwindigkeitsplanung fest, welche Trajektorien neuberechnet werden müssen.

Daraus entsteht eine Liste an Fahrsegmenten inklusive der beiden neu erstellten, die ein neues Geschwindigkeitsprofil erhalten. Diese Liste wird anhand der „Lässigkeit“ der jeweiligen Segmente sortiert. Diese ist so definiert, dass Trajektorien mit geringerem Spielraum hinsichtlich der Geschwindigkeit einen geringeren Lässigkeitswert besitzen. Konkret handelt es sich dabei um die Zeit pro räumliche Länge des Fahrsegments, die die Arbeitseinheit in Stillstand verbringen kann und dennoch in der Lage ist ihr Ziel pünktlich zu erreichen. In dieser Reihenfolge werden dann auch die Geschwindigkeitsprofile errechnet, wodurch die dynamischen Hindernisse durch jene Segmente nach und nach angereichert werden. Der Effekt ist, dass Arbeitseinheiten jenen Arbeitseinheiten ausweichen, die es eiliger haben.

Für das erste Stück der neuen Fahrsegmente wird ein `MinimumTimeVelocityPathfinder` verwendet. Somit wird erreicht, dass die Arbeitseinheit so schnell wie möglich zu ihrer Aufgabe gelangt. Das zweite Stück zur nächsten Aufgabe wird durch ein `FixTimeVelocityPathfinder` berechnet, sodass diese Aufgabe pünktlich zum eingeplanten Zeitpunkt begonnen werden kann. Das Aktualisieren bestehender Fahrsegmente wird ebenfalls durch einen solchen Pathfinder durchgeführt.

Die beiden `VelocityPathfinder` haben gemein, dass sie sogenannte Verbotene Regionen mithilfe des `ForbiddenRegionBuilder`s erstellen. Dieser berechnet anhand dynamischer Hindernisse und eines Raumpfades jene Positionen des Pfades, welche zu einer bestimmten Zeit nicht passierbar sind. Siehe dazu auch den Bericht *„Path-Velocity Decomposition“*. Anhand dieser Regionen kann ein Graph erstellt werden, der gültige Wege in der $s \times t$-Ebene angibt. Dazu wird der `ArcTimeMeshBuilder` verwendet. Dieser ist in zwei Implementierungen vorhanden: `FixTimeMeshBuilder` und `MinimumTimeMeshBuilder`. Zur Bestimmung der kürzesten Pfade innerhalb der Graphen wird die *JGraphT*-Bibliothek verwendet.

4 Literaturverzeichnis

Kant, K., & Zucker, S. W. (Fall 1986). Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research , 5* (3), S. 72-89.