# SEMINAR MMI - VERFAHREN DER WEGEPLANUNG

Alexander Gran

alex@grans.eu

**Kurzfassung**

Wegeplanung ist ein vielseitig eingesetztes Werkzeug um Probleme aus den Bereichen Robotersteuerung, Computeranimation, Virtual Prototyping sowie Molekularbiologie zu lösen. Dieses Paper versucht die verschiedenen Verfahren zur automatisierten Berechnung von lokaler und globaler Wegeplanung der letzten drei Jahrzehnte vorzustellen. Es werden dabei sowohl Verfahren zur Überführung eines Wegeplanungsproblems in einen Graphen vorgestellt als auch direkte Lösungsansätze ohne explizite Berechnung des freien Konfigurationsraums. Sowohl klassische A*, Sichtgraph, Potential und Voronoi Ansätze, als auch neuere Probabilistic Roadmap Methoden und Rapidly-Exploring Random Trees werden kurz eingeführt und verglichen.

*Stichworte:* Konfigurationsraum, Voronoi, Potentialfelder, Zellenzerlegung, Sichtgraph, A*, Probabilistic Roadmap, Rapidly-exploring Random Tree

# 1 Einleitung

## 1.1 Motivation

Bei der einfachsten Form der Wegeplanung gilt es einen Weg von einem Startpunkt zu einem Zielpunkt zu finden, ohne dabei mit Hindernissen zu kollidieren. Dennoch hat die Wegeplanung verschiedene auf den ersten Blick nicht ersichtliche Anwendungen:

- Die Bewegung von Industrierobotern muss vor Arbeitsbeginn im Detail programmiert werden. Aufgrund der hohen Anzahl der Freiheitsgrade ist es aufwendig, diese alle manuell einzustellen. Die Benutzer fordern hier eine halbautomatische Programmierung anhand der gewünschte Lagen des Arbeitswerkzeugs. Die Zwischenlösungen für die einzelnen Achsen müssen daher automatisch berechnet werden können.
- Serviceroboter sind eine lange gehegte Vision der Menschen. Sie sollen autark in unbekanntem Gelände manövrieren und müssen dabei auch auf wechselnde und neue Umgebungen reagieren können. Es gilt Gefahren in Echtzeit zu erkennen und zu vermeiden.
- In der Computer Animation sollen - ähnlich den Industrierobotern - Softwaresysteme die Bewegungen von Objekten und Personen automatisch anhand von Ausgangs- und Zielwerten berechnen. Dabei ist ein natürliches Verhalten gefordert.
- Beim Virtual Prototyping ermöglicht die Wegeplanung eine automatische Vorhersage wie einfach (wenn überhaupt) der Zusammenbau von Komponenten ist; und das ohne sie vorher real zu modellieren.
- Selbst im Bereich der Biologie ergeben sich anspruchsvolle Anwendungen: Bei der Entwicklung von Medikamenten gilt es so genannte Ligandanlagerungen zu berechnen, niederenergetische Lösungen für die Anlagerungen Moleküle. Die Faltung von Proteinen wiederum ist ein so schneller Prozess, dass er kaum experimentell beobachtet werden kann, sondern nur mit Hilfe von Wegeplanungsmethoden berechnet werden kann

Diese Vielzahl von Anforderungen und die allgemeine NP-Schwere[1] haben dazu geführt, dass viele verschiedenartige Lösungsansätze entstanden sind, die alle ihre spezifischen Vor- und Nachteile haben. Nach einer kurzen formalen Einführung in die Terminologie der Wegeplanung werden diese im Folgenden vorgestellt und verglichen.

## 1.2 Problemstellung

In der einfachsten Formulierung des Problems heißt es oft schlicht: “Finde einen Weg von Start zu Ziel”. Man gelangt jedoch schnell zu weiteren Verfeinerungen, wie: “Finde einen möglichst schnellen Weg von Start zu Ziel”, oder auch “Finde schnell einen Weg von Start zu Ziel ohne die Umgebung komplett zu kennen”. Desweiteren ist die Definition von Weg, Start und Ziel im allgemeinen nicht direkt greifbar. Ein animierter Mensch hat zum Beispiel - je nach Modell - bis

[1] von einer nichtdeterministischen Turingmaschine in polynomieller Zeit lösbar

zu 57 Freiheitsgrade[PSL02], wodurch die Definition von Ziel und Weg nicht direkt ersichtlich ist. In diesem Kontext vermischt sich dann die eigentlich Pfadplanung unter anderem mit der sogennanten Schrittplanung. Es existieren inzwischen aber Algorithmen, die beide Probleme auf einmal lösen können.

# 2 Hauptteil

## 2.1 Formalismus

Die in dieser Arbeit verwendeten und im Folgenden eingeführten formalen Definitionen sind angelehnt an die Arbeit von Strandberg [Str04]. Ein Roboter $\mathcal{A}$ sei ein abstraktes Objekt, das eine bestimmte Konfiguration $q$ haben kann. Ob es sich dabei um einen starren Körper, ein Flugzeug oder eine kinematische Kette handelt ist für gewöhnlich zunächst nicht weiter von Bedeutung, sondern spiegelt sich in den gegebenen Einschränkungen wieder.

Der betrachtete Raum $\mathcal{W}$ in dem sich der Roboter bewegt ist in den allermeisten Anwendungsfällen $\mathbb{R}^3$ oder $\mathbb{R}^2$, theoretisch sind aber beliebige Räume möglich. Die Art einer Konfiguration $q$, also der Zustand des Roboters, hängt natürlich vom Roboter $\mathcal{A}$ selbst ab:

- Kinematische Ketten haben Tupel aus Winkeln
- Ein Flugzeug hat Position und Orientierung
- Starre Körper haben allgemein Translationen und Rotationen

Die verschiedenen Darstellungsmöglichkeiten der Rotationen wie (Euler-)Winkel, Quaternionen oder Matrizen werden als allgemein bekannt angenommen. Welcher Typus konkret für eine Konfiguration gewählt wird, ist nicht direkt bedeutend, da sie sich alle ineinander überführen lassen.

Der Hindernisraum $\mathcal{O}$ ist eine Teilmenge von $\mathcal{W}$, die alle nicht passierbaren Punkte enthält. Sie ist nicht notwendigerweise endlich und auch nicht notwendigerweise vollständig gegeben.
Der Konfigurationsraum $\mathcal{C}$ enthält alle möglichen Konfigurationen $q$ und ist wiederum abhängig vom Roboter $\mathcal{A}$:

- Bei begrenzten kinematischen Ketten wird $\mathbb{R}^n$ verwendet, wobei $n$ die Anzahl der Verbindungsstellen ist
- Bei einem Flugzeug wird $R^{(3x2)}$ verwendet
- Rotationen erzeugen oft unbegrenzte viele Möglichkeiten, z.B. $S^1$, mit $S$ als Einheitskreis

$\mathcal{C}$ zerfällt in zwei disjunkte Teile, den blockierten Konfigurationsraum $\mathcal{C}_{obs}$ und den freien Konfigurationsraum ist $\mathcal{C}_{free}$ . In den allermeisten Fällen ist die Berechnung dieser beiden Teile nicht direkt möglich, da die Einschränkungen für $\mathcal{C}_{obs}$ beispielsweise nicht linear sind, oder durch Rotationen transzendente Gleichungssysteme entstehen, die nicht automatisiert lösbar sind. Falls eine numerische Lösung möglich ist, wird diese beinahe immer zu langsam sein.

Die Überführung von $\mathcal{O}$ nach $\mathcal{C}$ ist deshalb nur theoretisch möglich. Daher beschränkt man sich im Allgemeinen darauf, einzelne Lösungen zu Raten und dann zu Testen ob diese Werte in $\mathcal{C}_{free}$ liegen.

Das Wegefindungsproblem kann nun wie folgt formuliert werden: Finde einen freien Pfad, also eine Folge von Konfigurationen $q_i \in \mathcal{C}_{free}$, die die Startkonfiguration $q_s$ mit der Zielkonfiguration $q_z$ verbinden. Dieses "Piano Movers" Problem ist bereits NP-Vollständig [rei]. Es können weitere Eigenschaften gefordert sein, beispielsweise einen möglichst kurzen[2] Pfad zu finden.

Eine holonome Einschränkungen ist eine Bedingung der möglichen Konfigurationen die nicht nur von den Werten aus $\mathcal{W}$ abhängen. Zum Beispiel könnte man bei einem Flugzeug fordern, dass die erste Ableitung über der Zeit der X Koordinaten (die Vorwärtsgeschwindigkeit) immer eine bestimmte Größe haben muss. Bei einem Auto ist es auch nicht möglich seitwärts zu fahren, obwohl alle Koordinaten angefahren werden können, muss es dafür immer vor- bzw. rückwärts fahren. Man kann hier zwischen Freiheitsgraden im "Großen" und im "Kleinen" unterscheiden. Die (dimensionale) Größe des Konfigurationsraums bei einem Rad ist zum Beispiel zwei (Vorwärts/Rückwärts und Drehung), im Großen, also im umgebenden Raum, hat es hingegen drei Freiheitsgrade.

Das Wegefindungsproblem lässt sich grob in folgende Problemtypen gliedern:

- Holonome Systeme: Der Konfigurationsraum hat dabei so viele Dimensionen wie der umgebende Raum, $||\mathcal{C}|| = ||\mathcal{W}||$ Beispielsweise hat die Konfiguration eines Hubschrauber sechs Freiheitsgrade (je drei für Position und Orientierung), und der Hubschrauber unterliegt keiner holonomen Einschränkung.

- Holonome, redundante Systeme: Hier ist der Konfigurationsraum höherdimensional als der umgebende Raum, $||\mathcal{C}|| > ||\mathcal{W}||$ Die menschliche Hand hat, wie der Hubschrauber, sechs Freiheitsgrade, der Mensch verfügt aber über sieben Gelenke am Arm[3], um sie zu positionieren. Hier sind also unter Umständen mehrere Lösungen möglich, wovon eine ausgewählt werden muss.

- Nichtholonome Systeme: Der Konfigurationsraum ist hier kleiner, $||\mathcal{C}|| < ||\mathcal{W}||$. Es ist nicht immer möglich, von einer möglichen Lösung zu einer anderen zu wechseln, auch wenn diese direkt daneben liegt. Ein auf der Ebene rollendes Rad ist ein einfaches Beispiel. Es kann alle Positionen erreichen, muss dafür aber einmal vor- und zurückfahren. Auch ist es möglich dass eine Position für sich zwar möglich ist, sie aber nicht erreicht werden kann. Beim Auto wäre dies eine Parklücke, die zwar groß genug ist, damit das Auto darin Platz findet, die aber nicht so groß ist, dass auch hineingefahren werden kann.

Desweiteren kann man die Probleme nach dem Wissen über den Hindernisraum $\mathcal{C}$ unterscheiden:

- Globale Probleme: In diesem einfachsten Fall ist $\mathcal{C}_{obs}$ und $\mathcal{C}_{free}$ von Beginn an komplett bekannt. Es existiert beispielsweise eine Karte des Raums.

[2] Hier braucht es natürlich eine passende Metrik

[3] Nimmt man den gesamten Körper dazu, wird das System noch redundanter

- Inkrementelle bzw. Lokale Probleme: Der Hindernisraum wird erst nach und nach bekannt. Beispielsweise erkennen Sensoren den Raum. Eventuell ist hier auch die Position des Roboters zunächst unbekannt, und muss anhand der Messwerte und bekannter Eigenschaften des Raumes zunächst bestimmt werden.
- Dynamische Probleme: $\mathcal{C}_{obs}$ ist nicht konstant, sondern kann sich während der Roboter arbeitet ändern. Das kann bedeuten, dass neue Hindernisse auftreten und alte verschwinden. Ein Algorithmus kann dynamische Probleme lösen, wenn er nicht bei jeder Änderung neu beginnen muss. Nicht dynamische Probleme werden statisch genannt.

Inkrementelle, dynamische Probleme sind typische Anwendungsfälle bei autonomen Servicerobotern, während beispielsweise bei Industrierobotern eher globale und statische Probleme gelöst werden müssen. Sind die Messwerte der Sensoren ungenau, muss man auch von einem dynamischen Problem ausgehen: Es ist dann möglich, dass sich ein ehemals in $\mathcal{C}_{free}$ gemessener Punkt plötzlich doch als blockiert herausstellt. Dieser "Fehler" muss dann einfach wie ein neues Hinderniss behandelt werden.

## 2.2 Voronoi

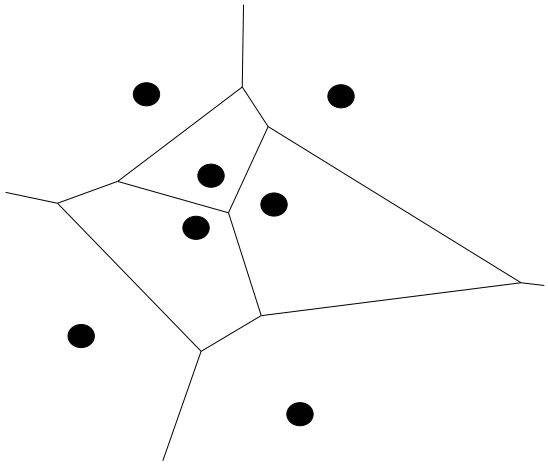

Abbildung 1: Voronoi Diagramm

Ein Voronoi Diagramm (VD) wird benutzt um die Ebene ($\mathbb{R}^2$) anhand von Messpunkten zu diskretisieren. Dieses ergibt dann einen Graphen, auf dem bekannte Graphenalgorithmen benutzt werden können. Dazu werden Messpunkte in der Ebene verteilt, und ihnen werden die Teilmengen der Ebene (=Flächen) zugeordnet, deren Punkte den geringeren Abstand [4] zu ihnen als zu allen anderen Messpunkten haben.

Die erste Verallgemeinerung hiervon ist das Generalized Voronoi Diagramm (GVD) welches nicht nur einzelne Punkte, sondern Punktemengen zulässt. Es ist immer noch beschränkt auf $\mathbb{R}^2$,

[4] Definiert über eine geeignete Metrik, für gewöhnlich der euklidische Abstand

wird aber für eine weitere Verallgemeinerung benötigt. Die Zuordnung der Ebenenpunkte zu den Messpunktemengen verläuft analog zum Voronoi Diagramm.

Der Generalized Voronoi Graph (GVG) ist die Verbindung von mehreren Generalized Voronoi Diagramms um höhere Dimensionen zu nutzen. Da er nicht notwendigerweise verbunden ist, existiert eine weitere Verallgemeinerung, der Hierarchical Generalized Voronoi Graph.

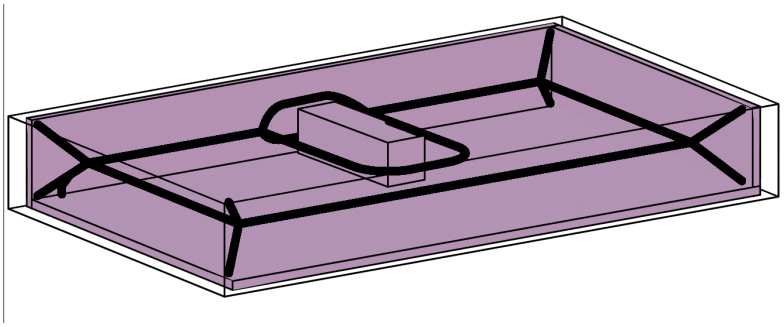

Abbildung 2: Hierarchical Generalized Voronoi Graph, aus [Cho96]

Er ist in der Lage anhand von nach und nach gemessenen Punkten über einen $m$-dimensionalen Raum einen Graphen (Roadmap) zu legen, um dann mit üblichen Graphalgorithmen darauf einen Weg zu finden. Dieser Aufbau erfolgt immer inkrementell, so dass auch eine stückweise Konstruktion möglich ist, wenn Sensoren den Raum erst noch erkunden müssen. Sollte ein Weg existieren, ist der Graph immer verbunden.

Nachteilig in der praktischen Anwendung ist, dass die verwendete Mathematik relativ komplex ist und dass noch wenig detaillierte Ergebnisse vorliegen, die das Verfahren mit anderen vergleichen. Details liefert [Cho96], sowie [JM00]. Choset liefert keinen Ansätze, um mit Voronoi Graphen dynamische Probleme zu lösen. Generalized Voronoi Diagrams können auch eingesetzt werden, um enge Passagen zu erkennen. Hybridverfahren können diese so erkannten Engstellen dann speziel lösen. Ein HGVG hingegen behandelt enge Passagen zunächst nicht speziell.

## 2.3 Sichtgraph

Der Sichtgraph Algorithmus sucht in der Ebene nach einem Weg anhand der Eckpunkte der Hindernisse. Er blickt von Ecke zu Ecke und baut so die Roadmap für den Weg auf. Es werden also die Ecken der Hindernisse mit Knoten identifiziert, die mit einer Kante verbunden sind, wenn sie sich gegenseitig sehen. Sehen bedeutet, dass eine Strecke zwischen den beiden Knoten existiert, die kein anderes Hindernis schneidet. Für Start und Ziel wird ebenfalls jeweils ein Knoten eingefügt, der dann mit allen von ihm sichtbaren Ecken verbunden wird.

Der Sichtgraph Algorithmus funktioniert nur im $\mathbb{R}^2$, auch ist das Finden der Ecken nicht trivial. Wenn die Hindernisse komplex werden, kann die Eckenanzahl dramatisch ansteigen. Dennoch ist der Algorithmus im Allgemeinen sehr schnell und kann aufgrund seiner Konstruktion sowohl lokal also auch global arbeiten. Die Sensoren müssen dann lediglich genau die nächsten

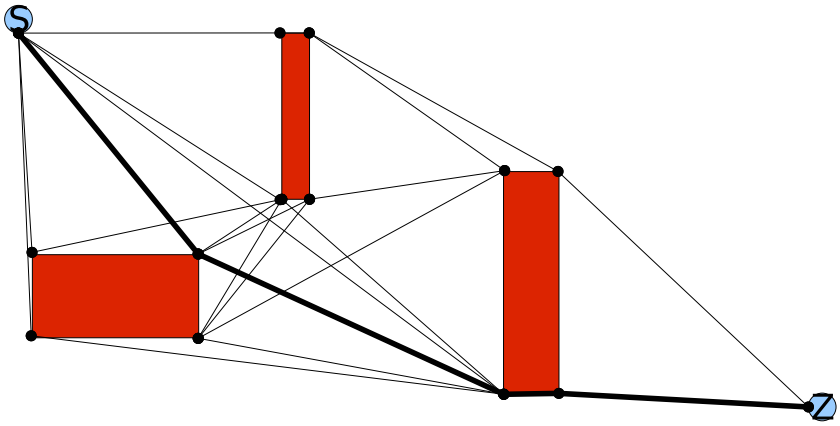


Abbildung 3: Ein Sichtgraph mit Weg von Start zu Ziel

Ecken erkennen, welche sofort eingefügt werden können. Natürlich ist der so gefundene Weg unter Umständen nicht optimal. Auch dynamische Probleme lassen sich mit dem Sichtgraph Algorithmus lösen: In der Ebene kann die Berechnung der neuen Verbindungen bzw. der nicht mehr mögliche Verbindungen lokal durchgeführt werden, so dass das Einfügen von Objekten keine komplette Neuberechnung erfordert. Enge Passagen stellen für den Sichtgraph Algorithmus keine besondere Schwierigkeit dar, wichtig ist hier lediglich die Form der Hindernisse, die die Passage bilden.

## 2.4 Potentialfelder

Potentialfeld Algorithmen [HA92, WC00] versuchen das natürliche Verhalten von beispielsweise Elektronen nachzuahmen [MM02]. Man modelliert den Raum derart, dass Wegpunkte die Nähe von Hindernisse meiden, während Freiräume und das Ziel die Wegpunkte “anziehen”. Dazu wird zunächst jedem Hindernis ein hohes Potential zugewiesen, das Ziel bekommt ein niedriges Potential. Die Potentialverteilung (bzw. Funktion) ergibt nun eine “Landkarte” für den Weg, wobei Gebiete hohen Potentials gemieden werden sollten und solche mit niedrigem Potential bevorzugt werden. Die Wahl der Potentialfunktion ist natürlich von enormer Bedeutung für den Erfolg eines Algorithmus und unterliegt ständiger Diskussion und Forschung. Ein einfacher Algorithmus [5] läuft nun immer in Richtung von niedrigem Potential, also senkrecht zu den Gradienten. Dabei tritt u.U. ein Problem mit lokalen Minima auf, diese verhalten hier wie Sackgassen. Lokale Minima müssen also vermieden werden, oder es muss eine Strategie gefunden werden, um ihnen zu entrinnen. Andernfalls findet der Algorithmus möglicherweise keine Lösung, obwohl eine existiert. Es gibt hier verschiedene Ansätze: Die Potentialfunktion so zu wählen, dass das einzige Minimum im Ziel liegt, oder aber randomisiert mehrere Möglichkeiten zu probieren. Beispielsweise versucht der Algorithmus von Wang und Chirikjian [WC00] die minimalen Täler der Landkarte kollisionsfrei zu verbinden. Schlägt dies fehl, wählt er einen

[5] ähnlich zu dem Verfahren von Iwamura, Yamamoto und Mohri[IYM00]

neuen Weg.

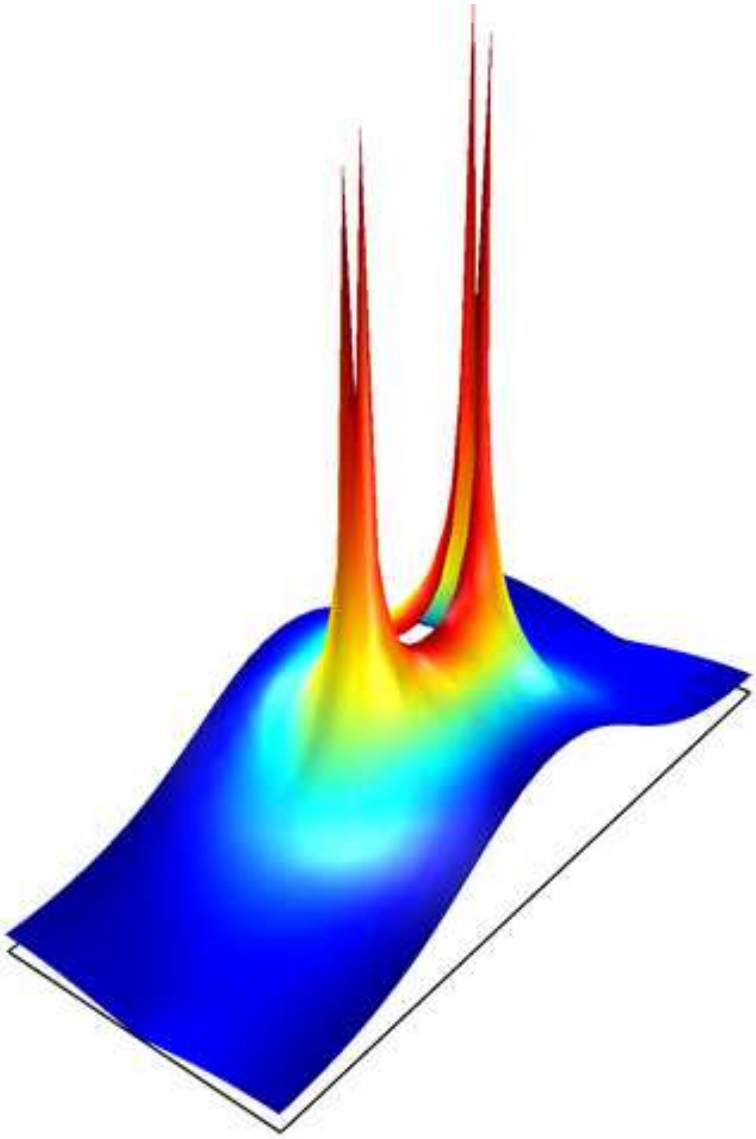

Abbildung 4: Ein Potentialfeld, von www.comsol.se

Potentialfeld Algorithmen haben jedoch mehrere Probleme: Das Verfahren welches über Täler läuft kann zwar mit lokalen Minima umgehen, scheitert aber bei höher dimensionalen Räumen an der Laufzeit. Auch gibt es Probleme, bei denen eventuell sehr viele Möglichkeiten probiert werden müssen. Der einfachere Gradientenansatz muss lokale Minima vermeiden, es gibt also Probleminstanzen, die er nicht lösen kann. Dies macht die Wahl der Potentialfunktion sehr schwierig und zu einer zentralen Aufgabe. Beiden ist gemein, dass sie keine Möglichkeiten haben, Engpässe zu erkennen und gesondert zu bearbeiten. Dies kann Laufzeit und Erfolgswahr-

scheinlichkeit stark beeinträchtigen. Vorteile sind dagegen die sonst recht schnelle Laufzeit und die simple Transformation von $\mathcal{C}_{free}$ in einen Graphen. Dynamische Probleme lassen sich mit den allermeisten Potentialfeldern nicht lösen, da das Einfügen bzw. Löschen von Hindernissen sich im Allgemeinen auf die gesamte Potentialverteilung auswirkt.

## 2.5 A*-Algorithmus

Der A*-Algorithmus[Bec01]war einer der ersten (1988) Ansätze, um heuristische[HNR68, HNR72] und formale[Dij59] Methoden miteinander zu verknüpfen. Er hat viele Anwendungen und kann auch zur Pfadplanung eingesetzt werden. A* sucht kürzeste Wege auf Graphen und versucht dabei unter Zuhilfenahme einer geeigneten Heuristik, naheliegende Knoten zuerst zu untersuchen. Die Heuristik wird genutzt, um den Weg über einen Knoten bis zum Ziel zu schätzen. Dadurch ist es möglich schneller zu arbeiten als beispielsweise Dijkstra[Dij59]. Wenn die Heuristik monoton[6] ist, sind die Wege sogar immer minimal. Ist die Wahl einer monotonen Heuristik nicht möglich, findet der Algorithmus nicht sicher den kürzesten Weg. In einer alternativen Version des Algorithmus reicht auch eine zulässige[7] Heuristik aus, was in der Praxis aber keinen großen Unterschied macht. Die meisten bekannten Heuristiken sind entweder monoton oder nicht zulässig. Da der A*-Algorithmus auf Graphen arbeitet, ist es zunächst notwendig, den Raum $\mathcal{W} = \mathbb{R}^3$ bzw $\mathcal{W} = \mathbb{R}^2$ zu diskretisieren.

### Exkurs: Transformation von $\mathbb{R}^2$ in einen Graphen mit Quadtrees

Zur effizienten Transformation von $\mathbb{R}^2$ in einen Graphen kann man sogenannte Quadtrees verwenden; Bei $\mathbb{R}^3$ können mit einem analogen Vorgehen Octtrees verwendet werden. Diese teilen den Raum rekursiv in Rechtecke auf, wobei große, zusammenhängende Teile zusammengefasst werden, um damit Platz (und später natürlich Laufzeit) zu sparen. Die Konstruktion von Quadtrees ist auch inkrementell möglich. In Abbildung 5 ist eine beispielhafte Probleminstanz abgebildet.

Die Diskretisierung läuft nun rekursiv ab wie folgt:

1. Wähle ein Rechteck $R$
2. Enthält $R$ kein Hindernis (d.h. $R \subset \mathcal{C}_{free}$ ) markiere $R$ als frei, fertig
3. Enthält $R$ ausschließlich Hindernisse (d.h. $R \subset \mathcal{C}_{obs}$ ) markiere $R$ als belegt, fertig
4. Andernfalls zerteile $R$ in zwei gleich große Rechtecke $R_1$ und $R_2$

Die gesamte Ebene wird dadurch - bis zu einer unteren Schranke - rekursiv durch Halbierungen in Rechtecke zerlegt. Große freie bzw. behinderte Bereiche bilden dabei ein Rechteck. Das

[6] Eine monotone Heuristik überschätzt die Weglänge zwischen zwei Knoten nie

[7] Eine zulässige Heuristik überschätzt die Weglänge zum Ziel nie

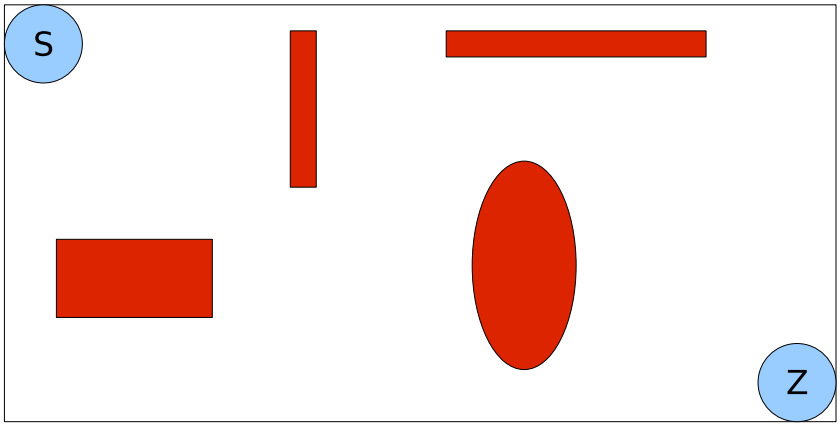


Abbildung 5: Eine Probleminstanz

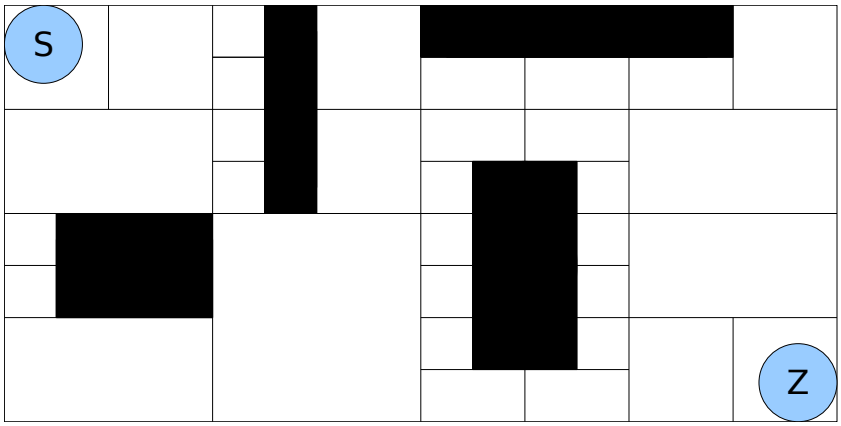


Abbildung 6: Die dazugehörige Aufteilung in einen Quadtree

Ergebnis ist in Abbildung 6 zu sehen. Die rekursive Aufteilung induziert direkt einen Baum, auf dem dann beispielsweise der A*-Algorithmus angewendet werden kann. Die Schranke erzeugt natürlich einen Fehler, so dass versucht werden muss, sie klein genug zu wählen, um jede Probleminstanz lösen zu können. Gerade bei gekrümmten Hindernissen ist dies nicht immer effizient möglich.

Bei einer Quadtree Zerlegung des Raumes lassen sich dynamische Probleme effizient lösen, da nur die betroffenen Knoten des Baumes neu berechnet werden müssen. Da A* selber keine Lernphase hat, sondern einen bereits bestehenden Graphen braucht, hat dort die Änderung im Graphen keine Auswirkung auf die Laufzeit. Auch eine inkrementelle Konstruktion ist mit einem ähnlich Ansatz denkbar, hier werden einfach zusätzliche Baumknoten eingefügt.

Der A*-Algorithmus überführt also zuerst den Raum $\mathcal{W}$ in einen Graphen. Dann werden die Kosten $f(v)$ jedes Knoten auf $\infty$ gesetzt, der Startknoten bekommt Kosten von $0$. Mittels einer

Prioritätswarteschlange wird immer der Knoten $v$ mit dem kleinsten Wert $f(v) := g(v) + h(v)$ betrachtet. Dabei sind $g(v)$ die Kosten vom Start bis zu $v$ und $h(v)$ die Heuristikkosten über $v$ zum Ziel. $f(v)$ sind also die geschätzten Kosten, um über $v$ von Start zu Ziel zu laufen. An den derartig ausgewählten Knoten $v$ werden nun bei allen Nachbarn die Werte für $f$ neu berechnet. Der Algorithmus endet wenn die Kosten des Zielknotens das erste Mal berechnet werden erfolgreich, oder wenn ein Knoten zum zweiten Mal ausgewählt wurde mit einem Fehler. Im Erfolgsfalle wird der Weg rückwärts berechnet und ausgegeben. Die Laufzeit ergibt sich, mit einem Fibonacci Heap als Prioritätswarteschlange, zu $O(|V| \log |V| + |E|)$. Hierbei ist $|V|$ die Anzahl der Knoten und damit der Rechtecke im Baum und $|E|$ die Anzahl der Baumkanten (logarithmisch abschätzbar gegen $|V|$). Grundsätzlich sind für den A*-Algorithmus auch andere Diskretisierungen denkbar, die ihn auch bei höhreren Räumen nutzbar machen. Die Wahl der Heuristik wird dadurch natürlich erschwert.

## 2.6 PRM Methoden

Auch bei Probabilistic Roadmap Methods wird das Pfadplanungsproblem in zwei Phasen zerlegt. Zunächst wird eine Transformation in einen Graphen, hier eine sogenannte Roadmap, durchgeführt, auf der dann die Wege gesucht werden. In dieser ersten Lernphase, auch Preprocessing oder Konstruktionsphase genannt, werden zufällig gewählte Konfigurationen $q$ draufhin untersucht, ob sie in $\mathcal{C}_{free}$ oder $\mathcal{C}_{obs}$ liegen. Motiviert ist dieses Vorgehen durch die oft sehr komplexe Beschreibung von $\mathcal{C}_{obs}$ , die in höherdimensionalen Räumen vorliegt. Beispielsweise können nicht-holonome Einschränkungen vorliegen, oder es sind durch Rotationen transzendente Gleichungen entstanden. Wurde nun eine zufällig gewählte Konfiguration in $\mathcal{C}_{free}$ gefunden, wird versucht mit Hilfe eines lokalen Planers diese mit anderen Konfigurationen zu verbinden. Dieses Problem ist oft einfacher, da nur ein Ausschnitt des gesamten Problems betrachtet werden muss. Außerdem ist es nicht notwendig eine existierende Verbindung zwingend sofort zu finden. Es kann einfach in einem späteren Durchlauf mit mehr Konfigurationen wieder probiert werden. Eine derartige Roadmap (im simplen zweidimensionalen Fall, in Anlehnung an[LF00]) ist in Abbildung 7 zu sehen.

Auf dieser Roadmap werden dann Start- und Zielkonfiguration eingefügt und diese dann mit einem lokalen Planer mit dem jeweils nächsten Punkt verbunden. Dannach kann auf dem Graphen einfach eine Suche nach dem kürzesten Weg durchgeführt werden, was im Beispiel das Ergebnis aus Abbildung 8 liefert. Dazu ist es wieder notwendig mit einer geeigneten Metrik den Kanten des Graphen Kosten zuzuweisen. Dieses Verfahren ist besonders bei statischen Problemen effizient, da die Kosten der Lernphase sich während der Abfragephasen amortisieren. Ist das Problem dynamischer Natur, das heißt $\mathcal{C}$ verändert sich, ist ein Lazy-PRM Ansatz notwendig. Hierbei werden die teuren Schritte erst während der Wegesuche durchgeführt. Dennoch haben auch PRM Methoden Schwierigkeiten mit engen Passagen. Die uniform zufällige Auswahl der Konfigurationen macht es dem lokalen Planer schwer, diese zu überwinden. Allerdings sind komplexere Lernmethoden möglich, die Anhand des Graphen versuchen, beispielsweise die Punkteauswahl optimierend zu beeinflussen [CR00]. Ladd und Kavraki[LK04] betrachten die Wahl der richtigen Kantengewichte genauer.

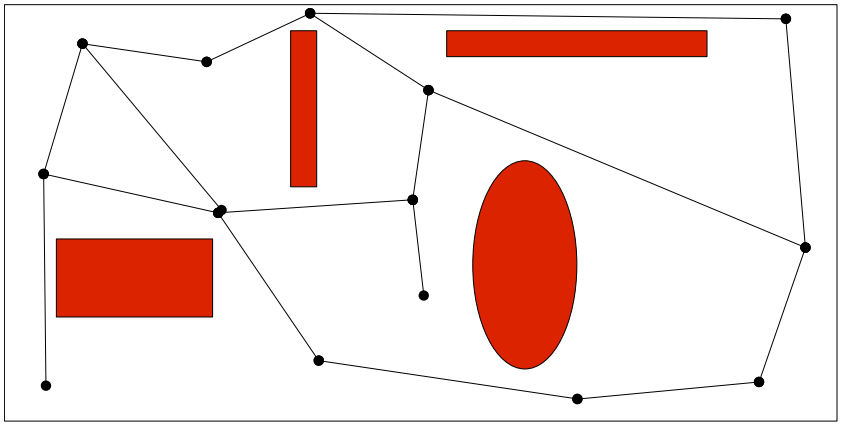

Abbildung 7: Eine einfache Roadmap

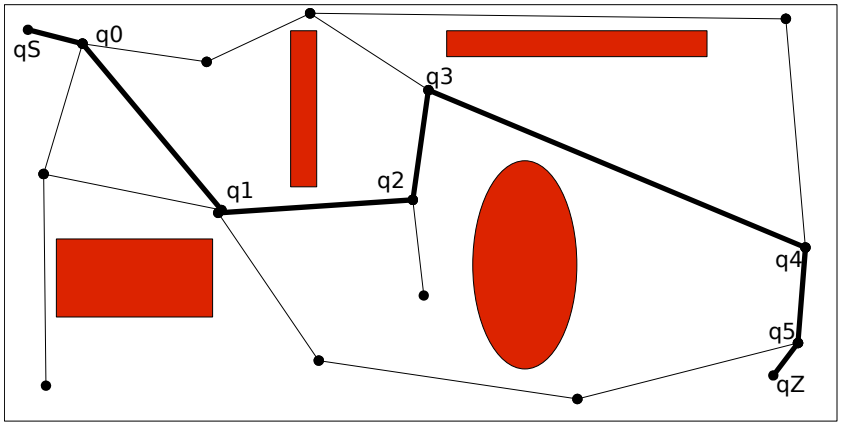


Abbildung 8: Eine einfache Roadmap mit einem möglichen Pfad

## 2.7 Rapidly-Exploring Random Tree

Rapidly-Exploring Random Tree versuchen die Unzulänglichkeiten der PRM Methoden, vorallem in Hinblick auf enge Passagen, zu umgehen, da sie dazu streben den gesamten Konfigurationsraum effizient zu entdecken (daher der Name)[LaV98]. Mit RRTs können auch nichtholonome Systeme hohen Gerades (z.B. $\mathbb{R}^{12}$ und $\mathbb{R}^{57}$(!) in [Str04] oder [KO03]) effizient gelöst werden. Sie streben in unentdeckte Gebiete des Konfigurationsraums und benötigen keine Suche nach Verbindungen zwischen Konfigurationen. Statt dessen wird Konvergenz verwendet, um immer nur in sehr kleinen Schritten die Roadmap entstehen zu lassen.

Das Vorgehen ähnelt Probabilistic Roadmap Methods. Es wird allerdings kein generischer Graph, sondern ein Baum konstruiert, was die Suche auf dem Graphen nachher auch noch deutlich vereinfacht. Die Knoten des Baumes sind die bereits gefundenen Konfigurationen.

Während der Konstruktion des Baumes wird zunächst eine zufällige Konfiguration $q_{rand}$

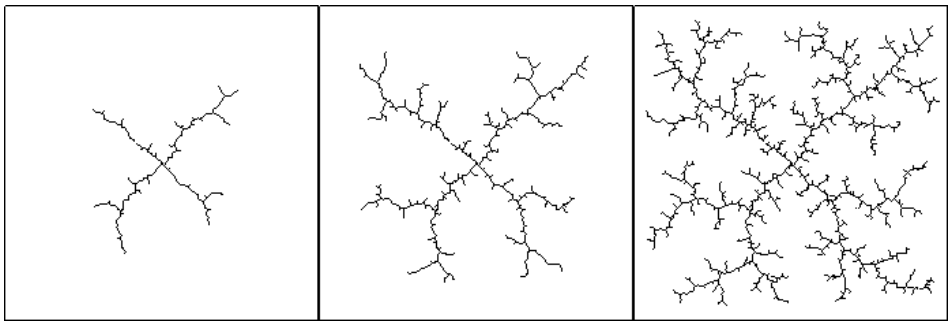

Abbildung 9: Ein RRT

gewählt. Ob sie in $\mathcal{C}_{obs}$ oder $\mathcal{C}_{free}$ liegt, ist nicht weiter wichtig. Dann wird im Baum der Baumknoten, also die Konfiguration, $q_{near}$ gesucht, der am nächsten an $q_{rand}$ liegt. Dafür ist wieder eine geeignete Metrik erforderlich. Mittels Konvergenz wird dann eine Konfiguration $q_{next}$ bestimmt, die ein Stück näher an $q_{rand}$ liegt als $q_{near}$. Diese wird als Baumknoten eingefügt, $q_{next}$ und $q_{near}$ werden mittels einer Kante verbunden. Es entsteht so ein Baum, der langsam in alle Richtungen des Konfigurationsraums wächst. Die Wurzel kann beliebig aus $\mathcal{C}_{free}$ gewählt werden. Im Beispiel sehen wir einen RRT, mit $C = [0, 100] \times [0, 100] \subset \mathbb{R}^2$ Die Wurzel wurde als $q_{init} = (50, 50)$ gewählt.

Da die Konfigurationen uniform zufällig im ganzen Konfigurationsraum gewählt werden, wird dieser auf lange Sicht gut abgedeckt. Gibt es beispielsweise eine enge Passage so, erzeugen alle zufällig gewählten Konfigurationen auf der noch unentdeckten Seite ein Streben des Baumes in Richtung des unentdeckten Konfigurationsraums hinter der Passage. Sobald dann ein Weg durch die Passage gefunden wurde, strebt der Baum wieder auseinander. PRM scheitern in diesem Fall wesentlich öfter, da der lokale Planer meist keinen Weg durch die Passage findet, vorallem wenn nicht passende Konfigurationen auf beiden Seiten gewählt wurden. In der Praxis werden meist zwei RRTs gleichzeitig konstruiert, einer hat die Wurzel im Ziel und einer im Start. Dieser bidirektionale Ansatz ist effizienter als der hier vorgestellte, läuft aber im Prinzip gleich ab.

# 3 Zusammenfassung

## 3.1 Vergleich der Verfahren

Die gut 30 Jahre der Entwicklung der Wegfindeverfahren haben deutliche Spuren hinterlassen. Während die ersten Ansätze wie A* noch auf 2- oder höchstens 3-dimensionale, statische Räume beschränkt waren, können moderne Verfahren auch höherdimensionale, dynamische Probleminstanzen bewältigen. Auch nicht-holonome Einschränkungen stellen keine unüberwindbaren Hindernisse mehr da. Es ergibt sich folgende Tabelle, die die Verfahren und ihre Eigenschaften kurz zusammenfasst:

| Verfahren | Raum | lokal | dynamisch | Sonstiges |
|---|---|---|---|---|
| Voronoi | $\mathbb{R}^2$ | - | - | einfach |
| HGVG | $\mathbb{R}^3$ | $\surd$ | - | kompliziert |
| Sichtgraph | $\mathbb{R}^2$ | $\surd$ | $\surd$ | sehr schnell |
| Potential | $\mathbb{R}^3$ | - | - | schlägt ev. fehl |
| Verbessertes Potential | $\mathbb{R}^n$ | - | - | Wahl der Funktion schwer |
| A* | $\mathbb{R}^3$ | - | - | braucht Diskretisierung |
| PRM | $\mathbb{R}^n$ | $\surd$ | $\surd$ | schnell |
| RRT | $\mathbb{R}^n$ | $\surd$ | - | sehr schnell |

Die erfolgversprechendsten Verfahren sind sicher RRT für höherdimensionale Räume und HGVH für niedrigere Dimensionen. PRM ist bei wiederkehrenden Anfragen effizienter als RRT. Gerade bei "einfachen" Anwendungen, wie zum Beispiel Navigationsproblemen sind aber auch klassische Ansätze, A* mit der Luftline als Heuristik, völlig ausreichend. Die neuste umfassende Arbeit kommt von LaValle [LaV06], sie ist komplett online verfügbar. Dort werden weitgehend alle Aspekte der Wegeplanung vorgestellt.

## 3.2 Anwendungen für den Virtual Human

Die obigen Ausführungen sind relativ allgemein gehalten. Es folgend einige Bewertungen zur Anwendung der Algorithmen im Umfeld des Virtual Human. Sollen menschenähnliche Roboter oder virtuelle Menschen visualisiert werden, muss neben der Wegeplanung auch die Schritt- bzw. Gelenkplanung berücksichtigt werden. Algorithmen, die nur im zwei- oder dreidimensionalen Raum agieren können, sind daher hier nur zum Teil von Nutzen. So kann ein Sichtgraph Algorithmus beispielsweise zur Grobplanung des Weges eingesetzt werden. Die holonomen Einschränkungen, denen ein Mensch unterliegt muss er aber ebenso wie die Schrittplanung anderen Verfahren überlassen. Mit einem PRM oder RRT Ansatz hingegen ist es möglich die komplette Berechnung der Bewegung inklusive aller einzelnen Gelenke gleichzeitig durchzuführen. Kallmann [KAAT03] hat erfolgreich ein 22 dimensionales Problem zur kollisionsfreien Bewegung beim Greifen eines Menschen gelöst. In einer neueren Arbeit [KM] kombiniert er eine Zellzerlegung des Raumes (ähnlich zu der Octree Zerlegung) mit einer Probabilistic Roadmap, um eine dynamische Roadmap (DRM) zu erhalten, die sehr effizient auf lokale Veränderungen reagieren kann. Ein lokaler Planer wird verwendet, um fehlenden Lösungen zu errechnen. Dies ist bei wenig Dynamik schneller als jedesmal einen RRT zu konstruieren.

Für die grobe Vorplanung des Weges wird es zu langsam sein, RRT oder PRM Ansätze zu wählen. Hier bietet sich ein mehrstufiger Ansatz an, wo zunächst mit einem einfach Algorithmus (Sichtgraph, HGVG o.ä.) eine grobe Wegeplanung vorgenommen wird. Derart eingeschränkt kann dann ein RRT oder DRM Ansatz verwendet werden, um Detaillösungen zu errechnen.

Da das Problem NP-Vollständig ist, ist es bei allen vorgestellten Algorithmen möglich, dass sie keine Lösung finden. Da die randomisierten Verfahren PRM und RRT beides Monte-Carlo Algorithmen sind, ist es natürlich möglich sie in Las-Vegas Algorithmen zu überführen. Dies kann bei schwierigen Problemen extrem auf die Laufzeit schlagen. Bei A* mit Quadtrees, Sichtgraph, Voronoi und dem einfach Potentialverfahren ist nicht einmal das nicht möglich. Findet einer von

denen keine Lösung, muss man andere Möglichkeiten in Betracht ziehen. Beispielsweise kann bei Quadtrees die Diskretisierungsschranke kleiner gewählt werden.

### 3.3 Ausblick

Es gibt immer noch viel aktive Forschungsarbeit auf dem Gebiet, das Optimum ist sicher noch nicht erreicht. Da ständig neue Anforderungen dazukommen - beispielsweise die Proteinfaltung, oder Skelettanimation - müssen auch permanent neue Verfahren entwickelt werden. Erste Ansätze versuchen neuronale Netzwerke einzusetzen [GKG95], auch Juidette und Youlal [JY00] sind von der Natur inspiriert und verwendet dort genetische Algorithmen. [LK01] gibt weitere Bewertungen für RRTs, in [PHLM00] findet sich ein Ansatz, um mit spezieller (Grafik-)Hardware Wegefindungsprobleme zu lösen. Gerade die Herausforderung in Echtzeit mit Menschen zu interagieren, stellt neue Anforderungen an die Wegefindung der (virtuellen) Roboter.

## Literaturverzeichnis

[Bec01] H. Becker. Der A*-Algorithmus im Einsatz zur Bahnplanung am Beispiel eines mobilen Roboters. September 2001.

[Cho96] H. Choset. *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. PhD thesis, California Institute of Technology, 1996.

[CR00] S. Caselli and M. Reggiani. ERPP: An experience-based randomized path planner. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 2, 2000.

[Dij59] EW Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[GKG95] R. Glasius, A. Komoda, and S. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133, 1995.

[HA92] YK Hwang and N. Ahuja. A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, 8(1):23–32, 1992.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[HNR72] P.E. Hart, N.J. Nilsson, and B. Raphael. Correction to A formal basis for the heuristic determination of minimum cost paths. *SIGART Newsletter*, 37:28–29, 1972.

[IYM00] M. Iwamura, M. Yamamoto, and A. Mohri. Near-optimal motion planning for nonholonomic systems usingtime-axis transformation and gradient method. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 2, 2000.

[JM00] K.H.I.I.I.T.C. John and K.M.C.L.D. Manocha. Interactive Motion Planning Using Hardware-Accelerated Computation of Generalized Voronoi Diagrams. 2000.

[JY00] H. Juidette and H. Youlal. Fuzzy dynamic path planning using genetic algorithms. *Electronics Letters*, 36(4):374–376, 2000.

[KAAT03] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping. *Computer Graphics Forum*, 22(3):313–322, 2003.

[KM] M. Kallman and M. Mataric. Motion planning using dynamic roadmaps. *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 5.

[KO03] J. Kim and JP Ostrowski. Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints. *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, 2, 2003.

[LaV98] S.M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Computer Science Dept, Iowa State University, Tech. Rep. TR*, pages 98–11, 1998.

[LaV06] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[LF00] A. Lambert and T. Fraichard. Landmark-based safe path planning for car-like robots. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 3, 2000.

[LK01] S.M. LaValle and J.J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[LK04] A.M. Ladd and L.E. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 20(2):229, 2004.

[MM02] SA Masoud and AA Masoud. Motion planning in the presence of directional and regional avoidance constraints using nonlinear, anisotropic, harmonic potential fields: a physical metaphor. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 32(6):705–723, 2002.

[PHLM00] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated Voronoi sampling. *Proc. Workshop on Algorithmic Foundation of Robotics*, 2000.

[PSL02] J. Pettre, T. Simeon, and JP Laumond. Planning human walk in virtual environments. *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, 3:3048?3053, 2002.

[rei] *Complexity of the mover?s problem and generalizations*.

[Str04] Morten Strandberg. *Robot Path Planning: An Object-Oriented Approach*. PhD thesis, Department of Signals, Sensors and Systems, Royal Institute of Technology, Stockholm, 2004. ISSN 1404-2150, ISBN 91-7283-868-X.

[WC00] Y. Wang and GS Chirikjian. A new potential field method for robot path planning. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 2, 2000.