Sensorbasierte, echtzeitfähige Online-Bahnplanung für die Mensch-Roboter-Koexistenz

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von

Thorsten Gecks

aus Limburg an der Lahn

1. Gutachter: Prof. Dr. Dominik Henrich (Universität Bayreuth)

2. Gutachter: Prof. Dr. Oliver Brock (Technische Universität Berlin)

Tag der Einreichung: 20.12.2010

Tag des Kolloquiums: 16.06.2011

Danksagung

Ich danke meinem Doktorvater Prof. Dr. Dominik Henrich für die langjährige Unterstützung und die intensive Betreuung. Ich danke meinen Kollegen für fruchtbare Diskussionen und Anregungen, insbesondere Antje Ober, Philipp Stolka, Stefan Kuhn, Marc Schütz, Markus Fischer, Jan Deiterding, Antoine Schlechter und Michel Waringo. Bedanken möchte ich mich auch bei meinen Studenten für geleistete Arbeit als Hiwi oder im Rahmen einer Bachelor-/Masterarbeit: Johannes Baumgartl, Ludwig Busse, Udo Pöhner, Tobias Werner und Oliver Münch. Der Welt insgesamt danke ich, dass größere Katastrophen bis zur Beendigung der Arbeit ausgeblieben sind. Meiner Freundin Antje Ober danke ich besonders für die moralische Unterstützung in der letzten Phase.

Zusammenfassung

Die Zusammenarbeit von Mensch und Roboter ist ein wichtiges Ziel der aktuellen Forschung in der Robotik, um die spezifischen Fähigkeiten von Mensch und Roboter zu verbinden. Die Voraussetzung für die Zusammenarbeit ist die sichere und effiziente Koexistenz in einem gemeinsamen Arbeitsraum. Hierzu muss die Sicherheit des Menschen gewährleistet sein und eine hohe Verfügbarkeit des Robotersystems erreicht werden. In dieser Arbeit wird ein bildbasierter Kollisionstest beziehungsweise eine Distanzberechnung entwickelt, um die Sicherheit des Menschen zu realisieren. Eine hohe Verfügbarkeit wird durch die Konzeption und Implementierung einer Bahnplanung erreicht, welche echtzeitfähig mit Ausweichbewegungen auf dynamische Hindernisse wie z.B. den Menschen reagiert, um gegenseitige Blockaden zu vermeiden.

Die Detektion des Menschen und anderer Objekte wird mit Hilfe eines Netzwerks stationärer Kameras erreicht, deren Befestigung den Einsatz effizienter Differenzbildverfahren („Change Detection“) ermöglicht und über die Kalibrierung der Kameras die Verwendung von Modellen bekannter Objekte (Roboter und Zellenaufbauten) erlaubt. Hierdurch werden Verdeckungen unbekannter Objekte (z.B. der Mensch) in einem bildbasierten Kollisionstestverfahren korrekt berücksichtigt. Weiterhin werden Konzepte zum Umgang mit dynamischen Hintergründen in realistischen Arbeitszellen erarbeitet. Die zusätzlich realisierte bildbasierte Distanzberechnung ermöglicht die Geschwindigkeitsregelung des Roboterarms für die sichere und ergonomische Mensch-Roboter-Koexistenz. Auch hierbei wird eine korrekte Behandlung der verdeckenden Geometrien berücksichtigt.

Die Bahnplanung wird in jedem Zyklus des Systems (100 ms) auf Basis aktueller Sensordaten (im bildbasierten Kollisionstest) durchgeführt, parallel zur Ausführung der jeweils im vorigen Zyklus berechneten Bahn. Das entwickelte Konzept verbindet dabei eine globale Bahnplanung auf randomisierten Netzen in hochdimensionalen Konfigurationsräumen mit der Begrenzung von teuren Kollisionstests bzw. Distanzberechnungen. Informationen über die nicht berechneten Teile des Graphen werden durch Schätzung aus bekannten Informationen erzeugt. Zusätzlich werden bereits berechnete Informationen über die Umwelt durch Speicherung im Graph über Systemzyklen hinweg beibehalten und somit sukzessive ausgeweitet. Ein Konzept wurde entwickelt und implementiert, um die persistenten Informationen bei detektierter Arbeitsraumdynamik zu adaptieren, wodurch flexibel auf Umweltveränderungen reagiert werden kann. Dieses grundlegende Rahmenwerk erzeugt einen Planer, welcher echtzeitfähig ist und somit auf dynamische Objekte reagieren kann und über Systemzyklen hinweg statistische Vollständigkeit in statischen Umgebungen erreicht.

Mit Hilfe des bildbasierten Kollisionstests wird dann ein wegoptimierender Planer

realisiert und mit Hilfe der bildbasierten Distanzberechnung ein zeitoptimierender Planer, welcher die Ausführungsdauer der Bewegung unter Maßgabe der distanzbasierten Geschwindigkeitsregelung optimiert. Letzterer ist für die Mensch-Roboter-Koexistenz aufgrund des vergrößerten Hindernisabstands besonders geeignet.

Abstract

Human-robot cooperation is a very active field of research, aiming to combine the specific capabilities of robots and humans to enable new forms of labor. One requirement for successful cooperation or coexistence is the safety of the human operator in the presence of heavy-weight and dangerous robot arms. As a second goal, high availability of the robot system is important to facilitate its adoption in industry. Thus two goals can be identified: first, detection of the human to enable safety measures, and second, path planning to avoid blockades by evasive behaviour of the robot arm. Both aspects are covered in this work.

Detection of the human operator and other obstacles is accomplished by a network of static cameras, monitoring the common workspace of the human and the robot. The static cameras allow to use an efficient background subtraction method and – through calibration – the use of models of known workspace object. The latter provides a means to detect occlusions of unknown objects (such as the human) by known objects. These are considered and handled correctly in the image-based collision detection and distance calculation methods presented in this work. Additionally, methods are described to cope with dynamic workspace backgrounds, which are common in industrial work spaces and impose difficulties on background subtraction methods. Using a distance calculation method, distance-based velocity control of the robot is realized to ensure safe human-robot coexistence.

The path planning method investigated here uses the available sensor data to react to current obstacle situations in each system cycle (100 ms). The developed concept combines global path planning on a randomized, static graph in a high-dimensional configuration space with the limitation of costly collision tests in the vertices and edges of the graph. Unknown collision or distance information in the graph is estimated from known information, which is stored over system cycles. A framework is presented to adapt this information according to the workspace activity. Thus, with this concept it is possible to meet real-time demands in the presence of dynamic obstacles while achieving statistical completeness over system cycles in static environments

Two path planner variants are investigated: a path-optimizing variant based on the collision test and a time-optimizing variant based on the distance measurement. The latter is especially well suited for human-robot cooperation as it also generates paths with extended clearance increasing the ergonomy of shared work spaces.

Inhaltsverzeichnis

Verwendete Formatierungen

Variablen	*a*
Mengen	$\boldsymbol{A}$
Vektoren	$\boldsymbol{a}$
Funktionen	*Func*()
Begriffsdefinitionen	*Begriff*
Pseudocode-Keyword/Funktionen	**keyword**

Liste der verwendeten Bezeichner

Name	**Domäne**	**Beschreibung**
C	$\mathbb{N}$	Anzahl der Kameras im System
$\boldsymbol{A}$	$\subseteq\mathbb{R}^3$	Arbeitsraumvolumen (beinhaltet die überwachten Volumina)
c	$\{1, \ldots, C\}$	Bezeichnet eine bestimmte Kamera
t	$\mathbb{R}$	Zeitpunkt
$I_c(t)$	Bild	Bild einer Kamera c zum Zeitpunkt t
$D_c(t)$	Bild	Differenzbild einer Kamera c zum Zeitpunkt t
g	$\mathbb{N}$	Identifiziert eine Kameragruppe
C_g	$\mathbb{N}$	Anzahl Kameras in einer Kameragruppe g
$\boldsymbol{C}_g$	$\subseteq\mathbb{N}$	Menge der Kameras c in der Gruppe g
$\boldsymbol{A}_g$	$\subseteq\mathbb{R}^3$	Teilmenge des Arbeitsraumes $\boldsymbol{A}$, der von allen Kameras der Gruppe g gemeinsam erfasst wird
$\boldsymbol{A}_c$	$\subseteq\mathbb{R}^3$	Teilmenge des Raumes, der von einer Kamera c erfasst wird
$\boldsymbol{Z}_k$	$\boldsymbol{Z} \subseteq\mathbb{R}^3$, $k \in\mathbb{N}$	Das k-te bekannte (modellierte) Objekt (z.B. der Roboterarm)
$\boldsymbol{H}_l$	$\boldsymbol{H} \subseteq\mathbb{R}^3$, $l \in\mathbb{N}$	Das l-te unbekannte (sensorisch erfasste) Objekt (Hindernis)
$\boldsymbol{F}_c$	$\subseteq\mathbb{N}^2$	Menge der Vordergrund-Pixel (Differenzbild) in Kamera c
$\boldsymbol{U}_c(g)$	$\subseteq\mathbb{N}^2$	Menge der verdeckten Pixel in Kamera c in Gruppe g (eine Teilmenge der Abbildung der bekannten Objekte $\boldsymbol{Z}_k$)

Name	Domäne	Beschreibung
$\boldsymbol{O}_c(g)$	$\subseteq \mathbb{N}^2$	Menge der Objektpixel in Kamera c in Gruppe g (eine Teilmenge der Abbildung der unbekannten Objekte $\boldsymbol{H}_l$)
$\boldsymbol{E}_c(g)$	$\subseteq \mathbb{N}^2$	Menge der mit „Frei“ markierten in Pixel in Kamera c in Gruppe g
$S_c(g)$	Bild	Szenenbild einer Kamera c in Gruppe g
θ_g	$\mathbb{N}$	Anzahl an Kameras, in denen ein unbekanntes Objekt $\boldsymbol{H}_l$ in $\boldsymbol{A}_g$ vollständig verdeckt sein kann
β_g	$\mathbb{N}$	Anzahl an Kameras, in denen jeder Teil eines unbekannten Objekts $\boldsymbol{H}_l$ in $\boldsymbol{A}_g$ vollständig verdeckt sein kann
r	$\mathbb{R}^d$	Roboterkonfiguration mit d als Anzahl der Gelenkwinkel
$\boldsymbol{T}(r)$	$\subseteq \mathbb{R}^3$	Volumen des Roboters an der Konfiguration r
$\boldsymbol{T}_c(r)$	$\subseteq \mathbb{N}^2$	Menge der Pixel, welche durch das projizierte Volumen $\boldsymbol{T}(r)$ des Roboters an Konfiguration r in Kamera c erzeugt wird
$\boldsymbol{P}_j$	$\boldsymbol{P} \subseteq \mathbb{R}^3, j \in \mathbb{N}$	j-tes Projektionsschnittvolumen (Conexel)
$\boldsymbol{V}_g$	Potenzmenge von $\mathbb{R}^3$	Menge aller Pj, die für den Kollisionstest sichtbar sind, Teilmenge von $\boldsymbol{A}_g$
$\overline{\boldsymbol{V}_g}$	Potenzmenge von $\mathbb{R}^3$	Komplementärmenge zu $\boldsymbol{V}_g$ in $\boldsymbol{A}_g$
$(\boldsymbol{p}_u, \boldsymbol{p}_o)$	$\boldsymbol{p}_u, \boldsymbol{p}_o$ in $\mathbb{N}^2$	Paar benachbarter Pixel, $\boldsymbol{p}_u$ aus $\boldsymbol{U}_c(g)$ und $\boldsymbol{p}_o$ aus $\boldsymbol{O}_c(g)$
$\boldsymbol{E}_i(c, \boldsymbol{p})$	$\subseteq \mathbb{N}^2, i \in \mathbb{N}$	Pixelmenge (die sogenannte Epipolarmenge) des Pixels $\boldsymbol{p}$ in Kamera c in Kamera i, $i \neq c$
$o_c(g)$	$\mathbb{R}$	Bildbasiert berechnete Distanz in einer Kamera c auf Basis der Pixelmenge $\boldsymbol{O}_c(g)$
$u_c(g)$	$\mathbb{R}$	Bildbasiert berechnete Distanz in einer Kamera c auf Basis der Pixelmenge $\boldsymbol{U}_c(g)$
d_{min}	$\mathbb{R}$	Distanz minimaler Robotergeschwindigkeit V_{min}
d_{max}	$\mathbb{R}$	Distanz maximaler Robotergeschwindigkeit V_{max}
$\boldsymbol{q}_i$	$\mathbb{R}^d, i \in \mathbb{N}$	Roboterkonfiguration im Konfigurationsraum, $i = s$ ist die Startkonfiguration, $i = z$ die Zielkonfiguration der Planung
$G(\boldsymbol{E}, \boldsymbol{V})$	Graph	Statischer Planungsgraph mit der Menge der Knoten $\boldsymbol{V}$ und der Menge der Kanten $\boldsymbol{E}$

Name	**Domäne**	**Beschreibung**
v_i	Knoten, $i \in \mathbb{N}$	Ein Knoten des Graphen G
e_i	Kante, $i \in \mathbb{N}$	Eine Kante des Graphen G
OPEN	Menge von Knoten	Eine für die Planung mittels A*-Varianten verwendete Knotenmenge
$g(v)$	$\mathbb{R}$	Funktion, welche die Wegkosten des Knoten v vom Startknoten v_s aus berechnet für eine konkrete Planung
$h(v)$	$\mathbb{R}$	Funktion, welche die heuristische geschätzten Wegkosten des Knoten v zum Zielknoten v_z berechnet
$rhs(v)$	$\mathbb{R}$	Funktion, welche die Wegkosten des Knoten v aus den Kosten $g(v_i)$ seiner Nachbarn v_i bestimmt
I	Intervall	Distanzintervall aus der Menge an Distanzintervallen $\boldsymbol{I}(v)$ pro Knoten v

1 Einleitung

1.1 Motivation

Das Leitmotiv in der Künstlichen Intelligenz ist eine intelligente Maschine, die dem Menschen in seinen Fähigkeiten mindestens ebenbürtig ist [Kurzweil90, Moravec90]. Die dabei betrachteten Fragestellungen und Konsequenzen reichen von einer Erforschung des menschlichen Geistes (Bewußtseinsdiskussion) durch Rekonstruktion, über die Überwindung der biologischen Grenzen (Befreiung des Geistes vom organischen Träger, Erweiterung aktuatorischer und sensorischer Grenzen) bis hin zur Diskussion des menschlichen Selbstverständnisses als Lebewesen, das sich von der toten und lebendigen (Pflanzen) Materie durch eine Seele unterscheidet. Die andere Seite dessen, was Robotik ausmacht, ist seit der Prägung des Begriffs durch den tschechischen Dichter Karel Capek [Capek17] in seinem Stück „Opilek“ aus dem Jahre 1917 die Entlastung des Menschen von schweren und monotonen Arbeiten, wie sie die heute im industriellen Umfeld eingesetzten Robotersysteme schon weitgehend erfüllen. Dies ist gleichzeitig und natürlich der Ursprung und Evolutionspfad der modernen Robotersysteme auf dem Weg zum (diskussionswürdigen) erstgenannten Ziel.

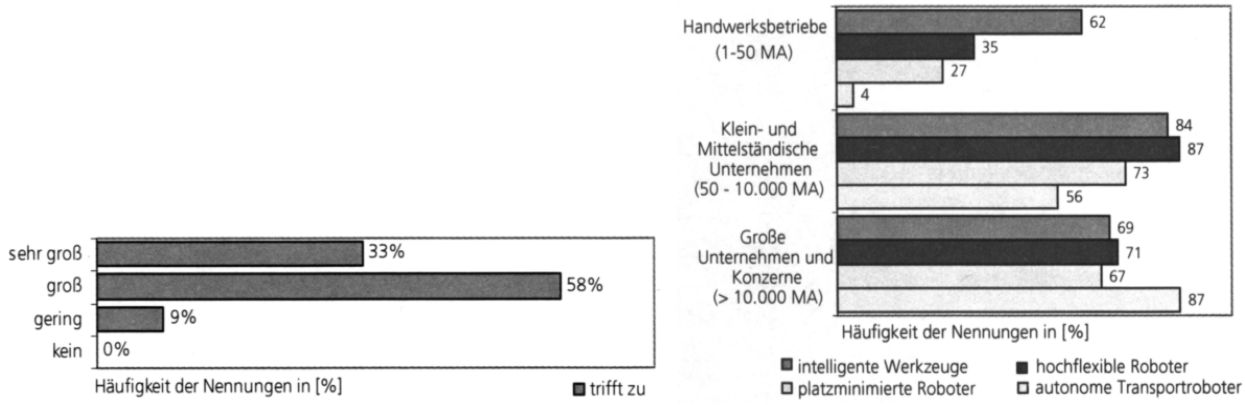


*Abbildung 1.1: Studie „Assistenzroboter in der Produktion“ [Breckweg05]. **Links**: Erfolgspotential für Assistenzroboter/ Roboter ohne trennende Schutzeinrichtungen sehen die Studienteilnehmer zu 91% groß bis sehr groß. **Rechts**: Das größte Potential sehen die Teilnehmer in mittelständischen bis großen Unternehmen für die genannten Teilgebiete der Assistenzrobotik.*

Robotersysteme haben sich im Laufe der Zeit von einfachen, schweren und

mechanisch programmierten Automaten zu komplexen Maschinen mit umfangreicher sensorischer Informationsverarbeitung entwickelt und entwickeln sich weiterhin. Mit zunehmenden sensorischen Verarbeitungsfähigkeiten verbessert sich damit einhergehend auch die Umweltwahrnehmung und Kommunikationsfähigkeit der Systeme. Dieses öffnet ein neues Feld der industriellen Produktion: die Verknüpfung der Fähigkeiten von Mensch und Roboter durch Kooperation. Den üblicherweise industriell eingesetzten Robotersystemen werden dabei die Attribute Präzision, Kraft, Ausdauer, Schnelligkeit, und Wiederholgenauigkeit zugeschrieben, dem menschlichen Kollegen komplexe Umweltwahrnehmung, Adaption an unvorhergesehene Situationen und Problemlösungsfähigkeiten. Die **Kombination der individuellen Fähigkeiten** ist ein wünschenswertes Ziel zur Verbesserung des industriellen Produktionsprozesses und Steigerung der Produktivität. Der Roboter kann hierbei z.B. als intelligentes Werkzeug betrachtet werden (siehe hierzu auch die Studie „Assistenzroboter in der Produktion“, Abbildung 1.1).

Abbildung 1.2: Beispiele der Mensch-Roboter-Kooperations-/Koexistenz-Anwendungen. ***Links****: Online-Schweißprozessüberwachung aus [Morhard02].* ***Mitte****: Kraftunterstützung zur Führung schwerer Werkstücke.* ***Rechts****: Positionierung von schweren Bauteilen zur Montage zusätzlicher Bauteile aus [Adams02].*

Eine Kooperation von Mensch und Roboter hat noch weitere Vorteile:

Ergonomie durch günstige Werkstückpositionierung für Bearbeitung und Kontrolle durch den Menschen, durch Kraftunterstützung für Montageoperationen, Untersetzung für Präzisionsaufgaben, Bewegungskontrolle bei Führungsaufgaben („Virtual Fixtures“), Entlastung von ermüdenden monotonen Aufgaben, die hohe Konzentration erfordern (zu den genannten Punkten siehe auch Teilbilder in Abbildung 1.2).

Zeit- und Raumersparnis durch Reduktion/Eliminierung der heute üblichen trennenden Schutzeinrichtungen zwischen Mensch und Roboteranlage. Hier kann durch die Zusammenführung der Arbeitsbereiche von Roboter und Mensch Fläche gespart werden und die Bewegungsfreiheit des Menschen wird weniger eingeschränkt. Auch können Zuführeinrichtungen wie Fließbänder und Rolltore eingespart werden, da keine Werkstücke vom Roboterarbeitsplatz zum Arbeitsplatz des Menschen notwendig sind. Durch die Reduktion der Zuführungsoperationen und Wege kann eine Zeitersparnis erreicht werden.

Motivation zur Entwicklung neuer Robotertechnologien. Durch die Kooperation

von Mensch und Roboter können Roboter in Bereichen eingesetzt werden, in denen sie nicht völlig autonom agieren können aufgrund ihrer Einschränkungen hinsichtlich Adaption und Problemlösungsfähigkeiten. Hierbei können Technologien und Werkzeuge entwickelt werden, die keine vollständige Autonomie ermöglichen, aber Wegmarken zu diesem Ziel bilden. Somit wird eine stetige Innovation ermöglicht, indem Roboterfähigkeiten erweitert werden und die fehlenden Lücken für die jeweilige Aufgabenstellung durch den menschlichen Kollegen ausgeglichen werden (siehe hierzu auch Argumentation in [Baerveldt92]).

Während die *Kooperation* von Mensch und Roboter so definiert ist, dass beide eine gemeinsame Aufgabenstellung verfolgen, ist die *Koexistenz* von Mensch und Roboter durch das Fehlen der gemeinsamen Aufgabenstellung definiert (siehe hierzu auch die Einteilungen in [Ferber99] und [Thiem05]). Die Koexistenz kann als Zwischenschritt auf dem Weg zur Kooperation betrachtet werden, da komplementäre und/oder unabhängige Zielstellungen für Mensch und Roboter vorgegeben sind, die auch individuell ohne Interaktion erreicht werden können. Die Fähigkeiten des Roboters müssten für die Kooperation entsprechend erweitert werden.

Die Gemeinsamkeiten von Koexistenz und Kooperation liegen bei den Ressourcenkonflikten bezüglich Raum und Zeit zwischen Mensch und Roboter, die durch die gleichzeitige Durchführung der jeweiligen Aufgabenstellung in einem gemeinsamen Arbeitsraum entstehen. Somit ist die Lösung dieser Ressourcenkonflikte ein wichtiger Grundbaustein und muss zunächst angegangen werden, um sowohl Koexistenz als auch Kooperation zu ermöglichen. Diese Arbeit konzentriert sich daher auf die Lösung bzw. Erkennung der Ressourcenkonflikte. Die Erweiterung der Roboterfähigkeiten zur Ermöglichung der Kooperation als nächstem Schritt werden nicht mehr betrachtet.

1.2 Ziele

Realisiert werden soll ein System, das die **Koexistenz** von Roboter und Mensch ermöglicht, also die gleichzeitige Durchführung individueller Aufgabenstellungen in einem gemeinsamen Arbeitsraum.

Da sich Mensch und Roboter bei der Koexistenz/Kooperation in einem gemeinsam genutzten Arbeitsraum bewegen, ensteht die Möglichkeit gefährlicher Verletzungen für den Menschen (siehe hierzu beispielsweise [Alvarado02]) und potentiell teure Beschädigung von anderen Hindernisobjekten. Dies ist daher durch eine Sicherheitsfunktion zu verhindern, bzw. in der Wirkung zu minimieren. (siehe hierzu auch die Normen [ISO 10218, EN 61496-1, EN 13849-1]). Die sich daraus ergebende Aufgabe beinhaltet also die **Verhinderung von Kollisionen** zwischen Roboter und Mensch (oder Hindernissen allgemein) durch die entsprechende Steuerung der beweglichen Teile des Roboters oder

alternativ die Entschärfung dieser Kollisionen. Es sind solche Verhinderung von Kollisionen ausgeschlossen, die der Mensch selbst herbeiführt, obwohl die Steuerung des Roboters aktiv versucht, diese zu vermeiden. Zur Verhinderung der **Kollisionen** müssen diese **detektiert** werden, bevor sie geschehen. Die Positionen von Mensch und Hindernissen ist dem System im Voraus unbekannt und muss daher **sensorisch erfasst** werden.

Neben dem Aspekt der Sicherheit ist die **Verfügbarkeit** eines Robotersystems ein wesentlicher Faktor. Mit Verfügbarkeit wird die Fähigkeit des Robotersystems beschrieben, eine Aufgabenstellung trotz äußerer Störungen (in diesem Fall die Hindernisse) fortzusetzen und somit ein kostenintensives Aussetzen der Aufgabe zu vermeiden. Die Verringerung der Ausfallzeiten des Robotersystems erhöht die Akzeptanz der Koexistenz/Kooperation von Mensch und Roboter. Für äußere Störungen der Aufgabenstellung durch Ressourcenkonflikte in Raum und Zeit zwischen Mensch und Roboter lässt sich eine hohe Verfügbarkeit mit Hilfe einer **Bahnplanung** erreichen. Diese ändert die ursprünglich vorgesehenen Bewegungen des Robotersystems so ab, dass Kollisionen verhindert werden und dennoch die Zielposition einer Bewegung erreicht werden kann. Die Verwendung dieser Methode schließt allerdings Aufgabenstellungen aus, bei denen die genaue Einhaltung der Roboterbewegung (Trajektorie) entscheidend ist (z.B. Schweißbahnen oder Kleberaupen). In diesen Fällen sind äußere Störungen jedoch generell ungünstig und diese Art von Aufgabenstellungen daher wenig für die Koexistenz geeignet.

Der Kollisionstest soll kostengünstig möglichst viel Informationen über Hindernisse im überwachten Raum liefern, um effiziente Bahnplanung zu ermöglichen. Eine adäquate Reaktion durch Bahnplanung soll **echtzeitfähig** erfolgen, um in einer Umgebung mit dynamischen Hindernissen wie dem Menschen sinnvoll anwendbar zu sein. Hierzu ist der Rechenaufwand von Kollisionstest und Bahnplanung und die Latenz zwischen sensorischer Erfassung des Arbeitsraumes und Reaktion gering zu halten. Gleichzeitig soll die Bahnplanung die geplante Bahn des Roboterarms weitestgehend optimieren, um die Verfügbarkeit zu erhöhen.

Das realisierte System soll sich zudem durch möglichst geringe Einschränkungen bezogen auf die Verwendung in unterschiedlichen **Anwendungsdomänen** auszeichnen. Dazu zählt bespielsweise keine oder nur geringe Einschränkungen der Geometrie und des Verhaltens aller Objekten innerhalb des Arbeitsraumes. Auch die Erledigung von sinnvollen Aufgaben durch das Robotersystem, wie z.B. der Transport von Objekten (keine Einschränkungen auf reinen Roboterarm) soll möglich sein. Der Bahnplaner soll eine Bahn von Start zum Ziel finden, falls eine solche existiert. Hierzu ist eine dreidimensionalen Erfassung des Arbeitsraumes durch den Kollisionstest hilfreich.

Der Mensch empfindet im Allgemeinen schnelle Bewegungen des Robotersystems bei gleichzeitig großer Nähe als unangenehm. Durch eine **distanzbasierte Geschwindigkeits-**

regelung soll daher ein ergonomisches Verhalten des Robotersystems erreicht werden, indem in der Nähe des Menschen durch eine niedrige Geschwindigkeit das empfundene und reale Gefahrenpotential verringert wird. Dies ist unter anderem motiviert durch die entsprechenden Voruntersuchungen in [Thiem05].

1.3 Stand der Forschung

In diesem Kapitel werden Systeme vorgestellt, die als Gesamtsysteme die Mensch-Roboter-Kooperation/Koexistenz zum Ziel haben. Auf den Stand der Forschung zu den Teilkomponenten des im Rahmen dieser Arbeit untersuchten Systems wird in den jeweiligen Kapiteln eingegangen. Grundlage für die Beurteilung der Systeme sind die schon im Kapitel Ziele genannten Kriterien:

- K1: Sicherheitsfunktion (Kollisionserkennung/-verhinderung/-entschärfung)
- K2: Verfügbarkeit
- K3: Echtzeitfähigkeit
- K4: Anwendungsdomäne

Die distanzbasierte Geschwindigkeitsregelung ist nicht mit aufgeführt, da sie im Wesentlichen durch alle Systeme realisiert wird, bis auf jene mit reiner Kontaktsensorik und Kollisionsentschärfung (s.u.). Bei diesen Systemen ist jedoch schon durch den üblichen leichten Roboterarm und die geringen Bewegungsgeschwindigkeiten ein starke Reduktion des Gefahrenpotentials gegeben und damit eine ausreichende Ergonomie erreicht.

Die im Folgenden dargestellten Systeme sind zunächst grundsätzlich nach der eingesetzten Sensorik zur Erfassung der Umwelt eingeteilt, da diese prinzipbedingt die möglichen Reaktionsstrategien (Stichwort Verfügbarkeit) beschränkt. Umgekehrt ist dies im Allgemeinen nicht der Fall. Zu den möglichen Reaktionsstrategien zählen (siehe hierzu auch [Ebert03]):

- R1: Kollisionsentschärfung: Begrenzung der kinetischen Energie/der Roboter-geschwindigkeit, Polsterung der Roboteroberfläche, Nachgiebigkeit des Roboterarms in Gelenken oder Gliedern
- R2: Geschwindigkeitsregelung, stufenweise oder kontinuierlich (distanzbasiert)
- R3: Lokale und Globale Bahnplanung

Da die Reaktionsstrategien nicht gut geeignet sind, den Stand der Forschung einzuteilen, erfolgt die Kategorisierung des Stands der Forschung in den folgenden Abschnitten anhand der Sensorik. Die Bewertung erfolgt anhand der oben genannten Kriterien und nach einem einfachen „Notenschema“: „+“ (erfüllt Kriterium), „−“ (erfüllt Kriterium nicht), „±“ (erfüllt Kriterium teilweise).

Kontaktsensorik

Allen Systemen dieser Klasse gemeinsam ist, dass sie Kollisionen von Mensch und Roboter lediglich bei direkter Berührung und damit bei schon erfolgter Kollision detektieren. Diese Systeme arbeiten mit einer Kollisionsentschärfung und detektieren Mensch-Roboter-Kollisionen durch interne Sensorik wie Motorströme und Encoderinformationen, Kraftsensorik zwischen Gelenken oder am Greifer in Verbindung mit dem gegriffenen Objekt. Die Sicherheitsfunktion ist somit prinzipiell realisiert (K1 +). Zur Kollisionserkennung werden auch berührungsempfindliche künstliche Häute eingesetzt. Die Systeme verwenden im Allgemeinen sehr simple oder gar keine Bahnplanungsalgorithmen, sodass sie in lokalen Minima oder rein mechanisch blockieren können. Oftmals ist daher der Nutzer für das Erreichen des Ziels verantwortlich, z.B. durch Führung. Diese Probleme reduzieren die Verfügbarkeit (K2 −). Es sind aufgrund der recht einfachen Sensordatenverarbeitung meist sehr kurze Regelschleifen realisierbar, sodass die Reaktion echtzeitfähig ist (K3 +). Durch die notwendige Begrenzung der kinetischen Energie ist der Einsatzbereich beschränkt; im Allgemeinen auf kleinteilige Montageaufgaben/Objekttransport oder Ähnliches (K4 −).

In [Heinzmann99] wird ein Leichtbaurobotersystem zur Kollisionsentschärfung beschrieben, welches passiv bewegt wird (Zero-G-Regelung). Die auftretenden Momente werden nach Abzug der für die Zero-G-Regelung notwendigen Momente begrenzt und der Roboter wird bei Überschreitung einer maximalen Geschwindigkeit abgeschaltet. Die maximale Geschwindigkeit wird anhand eines dynamischen Modells bestimmt und basiert auf der Begrenzung der Einschlagskraft. In [Lim00] wird eine Kollisionsentschärfung durch eine nachgiebige Roboterbasis erreicht. Kraftsensorik in der Basis ermöglicht die Limitierung von Kontaktkräften. Das System aus [Heilig02] enthält zwar auch lokale Sensorik (und wird daher weiter unten noch einmal genauer dargestellt), jedoch kommt hier auch eine Polsterung des Roboters zum Einsatz und eine Kollision mit dem Menschen wird bei sofortiger Abschaltung der Roboters zugelassen. In [Göger06] und als Teilprojekt des Sonderforschungsbereiches 588 [SFB588] wurde eine taktile Sensorhaut (leitfähiges Gummi, zweidimensionales Sensorfeld) für den Roboter entwickelt. Die gemessenen Kontaktkräfte werden in die Kategorien Kollision, Kommando und Aufgabenkontakt eingeteilt. Die Behandlung detektierter Kollisionen ist nicht genauer beschrieben. In [DeLuca06] werden die internen Kraft-Momenten-Sensoren in den Gelenken eines KUKA Leichtbauroboters verwendet, um die Position und den Vektor einer Krafteinwirkung zu bestimmen und dann eine reflexhaftes Zurückweichen auszulösen. In [Oberer-Treitz08] wurden die Auswirkungen von Kollisionen zwischen Mensch und Roboterarm mittels Crash-Test-Dummy-Experimenten untersucht.

Detektion von Schutzzonenverletzung

Unter Schutzzonenverletzung ist die Detektion der Anwesenheit von Objekten in einem definierten Raumbereich zu verstehen. Dies ist schon lange Zeit Standard im industriellen Bereich, z.B. durch Einzäunungen mit Sicherheitsschalter an den Türen, Laserzäunen, Fußmatten, welche ein darauf platziertes Objekt durch sein Gewicht erkennen, und integrierten Systemen mit 1,5D-Laserscannern, welche die Verletzung des Bereichs signalisieren. Im Allgemeinen reagieren die Systeme mit Stillstand oder Reduzierung der Robotergeschwindigkeit (250 mm/s) und sind daher nur wenig verfügbar (K2 −). Die realisierte Sicherheitsfunktion ist durch die Konstruktion der Schutzzone (K1 +) gegeben, die den gesamten Mensch immer umfassen muss, wenn er detektiert wird. Die Systeme sind echtzeitfähig (K3 +), was aufgrund der einfachen Sensordatenverarbeitung gegeben ist. Die Anwendungsdomäne ist nicht eingeschränkt (K4 +).

In [Zettl02] werden Laserscanner zur Überwachung einer Schutzzone um ein Robotersystem eingesetzt, welches mit Hochdruckwasserstrahlen Teppiche schneidet. Die Laserscanner erfassen eine Ebene parallel zum Boden, welche die ganze Zelle umfasst und etwas darüber hinaus geht. Die Schutzzone ist zusätzlich in Unterzonen eingeteilt, welche bezogen auf die aktuelle Roboterposition aktiviert und deaktiviert werden. Bei Verletzung einer Zone werden die Robotersysteme abgeschaltet. In [Morhard02] (siehe auch Abbildung 1.2) werden Laserzäune zur Detektion der Schutzzonenverletzung verwendet. Die Anwendung liegt im Bereich der Kontrolle von automatischen Schweißvorgängen und schaltet den Roboter in Schleichfahrt bei Detektion einer Schutzzonenverletzung.

Lokale Sensorik

Lokale Sensorik wird hier so definiert, dass zu einem Aufnahmezeitpunkt durch den Sensor nicht der gesamte Arbeitsraum des Roboterarms erfasst wird, wobei Nichterfassung durch Abschattung explizit nicht gemeint ist. Die begrenzte Reichweite kann durch Exploration auf den gesamten Arbeitsraum ausgedehnt werden, wie dies z.B. häufig in der mobilen Robotik erfolgt, dies setzt jedoch eine statische Umgebung voraus, was eine erhebliche Einschränkung für die Anwendbarkeit bedeuten würde (K4 −). Daher werden im Folgenden alle Systeme ausgeblendet, die iterativ Umgebungskarten konstruieren und dann auf diesen planen.

Die eingesetzte lokale Sensorik erfordert häufig eine Einschränkung der erkannten Objekte auf spezifische Typen (z.B. bei kapazitiver oder induktiver Sensorik). Es wird typischerweise auch keine Unterscheidung von erwünschten und unerwünschten Umweltkontakten geleistet. Dies betrifft somit sowohl die Kriterien K1 als auch K4 negativ (K1 −, K4 −). Stehen nur Daten der aktuellen Aufnahme zur Verfügung, sind Reaktionsstrategien bis zur lokalen Bahnplanung üblich, die dann anfällig für lokale

Minima sind (K2 −). Der Rechenaufwand ist aufgrund des geringen Datenverarbeitungsaufwands und der lokalen Planungsalgorithmen recht gering und Echtzeitfähigkeit daher leicht zu erreichen (K3 +). Eine weitere Problematik lokaler Sensorik spezifischen Typs wird am Ende dieses Abschnitts genauer erläutert.

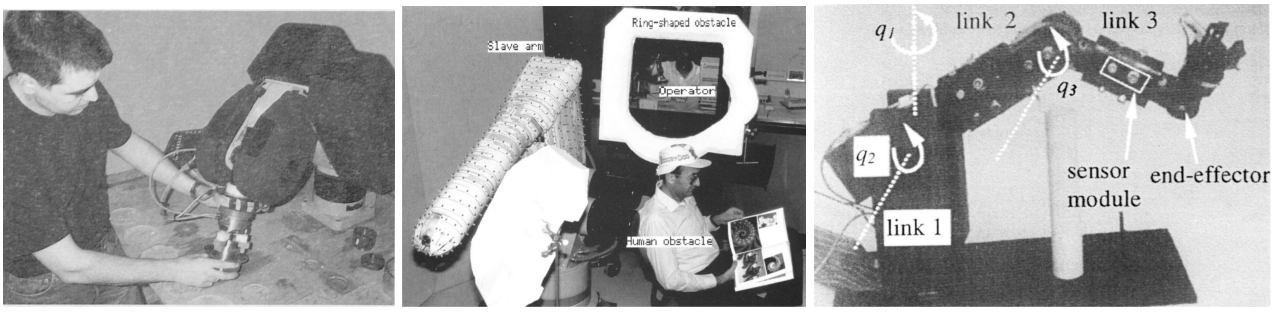


Abbildung 1.3: Systeme mit lokaler Distanzsensorik in Form von sogenannten künstlichen Häuten, von links nach rechts: [Heilig02], [Lumelsky93], [Martin99]

Die lokale Sensorik zur Absicherung von Roboterbewegungen wird im Allgemeinen auf dem Roboterarm selbst angebracht, da dies am sinnvollsten ist, denn sonst wäre der abgesicherte Arbeitsraum durch die Reichweite der Sensorik eingeschränkt (bei statischer Befestigung). Die verwendeten Sensoren bestehen aus Feldern einfacher Distanzsensoren, die einen skalaren oder binären Distanzwert pro Befestigungspunkt auf der Roboteroberfläche liefern (siehe Abbildung 1.3). Kapazitive Distanzsensoren finden bei [Novak92, Feddema94, Heilig02] Anwendung, Infrarotsensoren bei [Lumelsky93, Wegerif92] und Ultraschallsensoren bei [Martin99]. Die Systeme in [Heilig02, Novak92, Feddema94] realisieren eine Geschwindigkeitsregelung, die Systeme in [Wegerif92, Lumelsky93, Martin99] eine lokale Bahnplanung.

Grundsätzlich sind Systeme, die Distanzsensoren auf dem Roboter platzieren dadurch eingeschränkt, dass sie auf den lokalen Daten planen müssen (und damit im Allgemeinen leicht in Sackgassen geraten). Eine weitere Einschränkung der Anwendbarkeit hängt direkt mit der Anforderung an die Sicherheit zusammen (K1 ±): wenn Objekte im Greifer des Roboters transportiert werden sollen, so dürfen diese eine maximale Größe nicht überschreiten, da aus praktischen Gründen keine Distanzsensoren auf den transportierten Objekten platziert werden können und daher eine Kollision der transportierten Objekte mit der Umgebung nicht abgesichert werden kann (siehe Abbildung 1.4). Sind die Objekte klein genug, mag die Abschattung der Sensoren kein erhebliches Problem darstellen. Ein weiteres Problem ist die Empfindlichkeit der künstlichen Häute gegen mechanische (und sonstige) Einwirkungen, wodurch die Anwendungsdomäne eingeschränkt wird (K4 ±). Die Kosten der Spezialhardware sind im Allgemeinen hoch und bieten relativ geringe Ortsauflösungen im Vergleich zu beispielsweise Kamerasystemen. Eine beschränkte Messdistanz kann eine prinzipielle Geschwindigkeitsbeschränkung zur Folge haben. Aus den oben aufgeführten

Gründen wird daher die lokale Sensorik für den in den Folgekapiteln dargestellten Lösungsansatz ausgeschlossen.

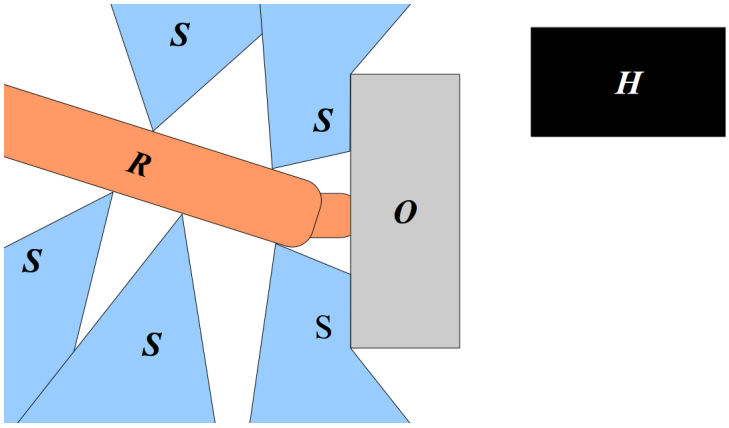


*Abbildung 1.4: Sicherheitsprobleme lokaler Distanzsensorik bei durch den Roboter **R** transportierten, großen Objekten **O** durch deren Abschattung des Sensorsmessfelds **S**. Das Hindernis **H** wird nicht detektiert, wodurch Kollisionen auftreten können.*

Globale Sensorik

Globale Sensorik erfasst zu einem Aufnahmezeitpunkt den gesamten Arbeitsraum des Roboters und aus den Daten wird die Belegung des Raumes mit Objekten berechnet. Hierbei kann sehr unterschiedliche Sensorik zum Einsatz kommen, die im Folgenden anhand der Dimensionalität (1,5D bis 3D) der gelieferten Rauminformation unterschieden wird. Die Rauminformation besteht üblicherweise aus einem binären Zustand: „Belegt“ bzw. „Nicht Belegt“, kann aber auch eine Wahrscheinlichkeit angeben, für die an der gegebenen Stelle ein Hindernisobjekt existiert („Occupancy Grid“). Die Erfassung ist durch die unterschiedliche Dimensionalität mehr oder weniger vollständig, sodass sich auch recht unterschiedliche Sicherheitsniveaus erzielen lassen oder umgekehrt die Anwendung stark eingeschränkt werden muss. Auch die beiden anderen Kriterien hängen stark von der tatsächlich eingesetzten Sensorik ab, weshalb keine generellen Aussagen getroffen werden, sondern eine einzelne Beurteilung der Systeme erfolgt.

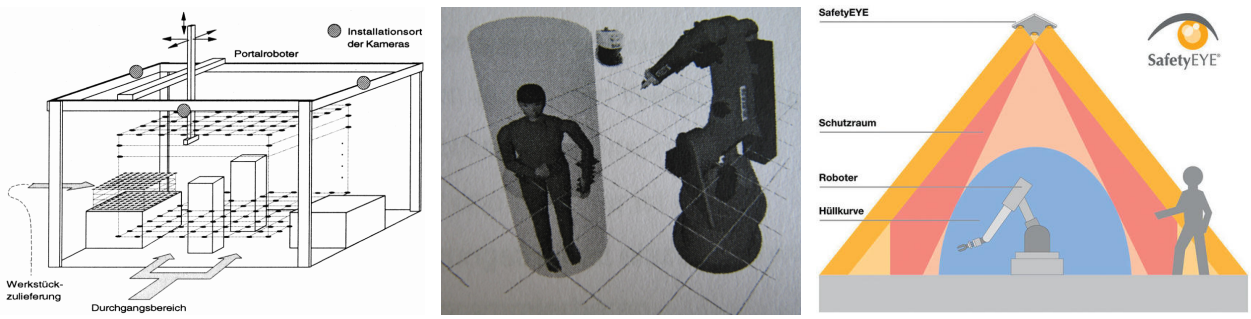


Abbildung 1.5: Globale Sensorik [Meisel94][Som05][SafetyEye]

In [Som05] (Abbildung 1.5 Mitte) detektiert ein Laserscanner (1,5D) Objekte in einem Freiraum vor dem zu überwachenden Roboter. Die Objekte werden von diesem

Laserscanner in einer Ebene erfasst, die parallel zum Boden orientiert ist in einer geringen Höhe, sodass die Füße von Personen erfasst werden. Für die erfassten Personen/Objekte wird angenommen, dass es sich um aufrecht stehende Personen handelt, die mit einem Zylinder approximiert werden können. Gebeugte Personen oder ausgestreckte Arme werden nicht erfasst (K1 ±). Die Distanz des Zylinders zur aktuellen Robotergeometrie wird lediglich zur Geschwindigkeitsregelung verwendet (K2 ±). Die Echtzeitfähigkeit ist durch den geringen Sensordatenverarbeitungsaufwand gegeben (K3 +). Die Einschränkungen bezüglich der Anwendungsdomäne sind gering (K4 +).

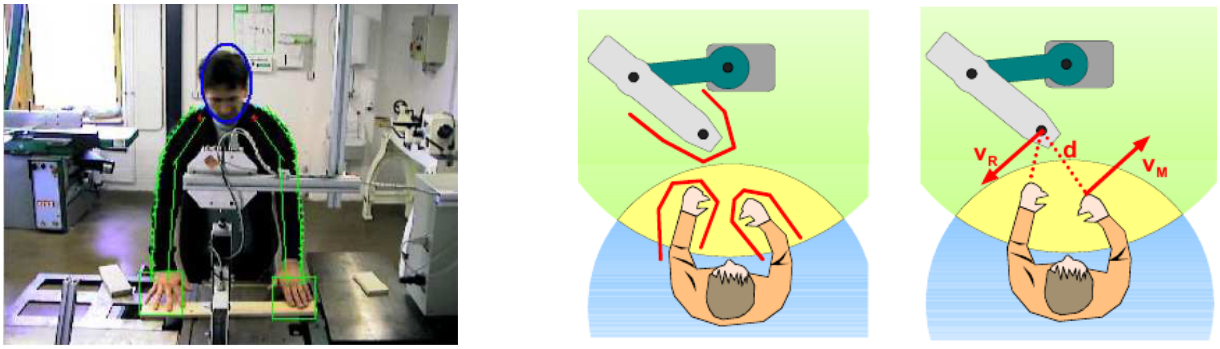


Abbildung 1.6: Bildbasierte Systeme zur Kollisionsdetektion Links: [Vieth08] Rechts: [Thiem05]

In [Vieth08] (Abbildung 1.6 links) werden einzelne Körperteile (Hände, Kopf über Hautfarben, usw.) des Menschen in zwei Kameraansichten (Überblick und Detail) segmentiert und deren Eindringen in eine Schutzzone detektiert aus der Sicht der Detailkamera, was zum Notaus der überwachten Maschine führt (in diesem Fall ein Sägeblatt). Die Verfügbarkeit des Systems ist gut, da die Domäne sehr stark eingeschränkt wird. Jedoch muss hierbei beachtet werden, dass es zu Verringerungen der Verfügbarkeit kommen kann, wenn Gegenstände änlicher Farbe in den Überwachungsbereich eingebracht werden. Daher ist insgesamt die Verfügbarkeit neutral zu werten (K2 ±). Die Sicherheitsfunktion ist eingeschränkt, da lediglich eine Hand im Gefahrenbereich detektiert wird (K1 ±). Die relativ aufwendige Sensordatenverarbeitung ist bezüglich der Echtzeitfähigkeit problematisch (K3 −). Da lediglich der Mensch als Hindernis wahrgenommen wird und lediglich die Geschwindigkeit eines Sägeblattes geregelt wird, ist die Anwendungsdomäne der Lösung stark eingeschränkt (K4 −).

Die Verwendung von Hautfarben spielt auch in der Arbeit von [Spingler02, Thiem02] (Abbildung 1.6 rechts) eine Rolle. Hier wird in mehreren Kameras die Distanz von Händen und Kopf des Menschen zu einem Punkt des Roboters (SCARA-Typ) berechnet. Die Details dieses Verfahrens sind nicht näher spezifiziert. Die Geschwindigkeit des Roboters wird entsprechend der Distanz von Mensch und Roboter und der aktuellen Roboterbewegung geregelt. Andere Objekte oder Teile des Menschen werden nicht erfasst, obwohl der Arbeitsraum des Roboters relativ umfangreich ist (K1 −). In [Schulz03] und [Thiem05]

wurde das System durch 2,5D-Stereokameras verbessert. Hier bleibt allerdings weiterhin unklar, ob andere Objekte außer der menschlichen Hautoberfläche erkannt werden. Diese Systeme erfahren aufgrund ihrer Ähnlichkeit zu [Vieth08] eine identische Beurteilung nach den weiteren Kriterien.

In [Steinhaus99, Hoover99, Lee03] werden mobile Roboter durch ein System fest installierter Kameras überwacht und auf Kollision mit detektierten Objekten überprüft. Hierbei werden keine Verdeckungen berücksichtigt, d.h. es wird angenommen, dass der gesamte Überwachungsraum sichtbar ist. Da die Bahn der mobilen Systeme im Allgemeinen in einer zweidimensional Ebene durchgeführt wird, die durch die Kameras direkt erfasst wird, können die detektierten Objekte ohne aufwendige Transformationen direkt in der Planung verwendet werden. Der Kollisionstest findet somit in einem zweidimensionalen Raum statt und ist daher für die hier vorgesehene Anwendung nicht ausreichend (K4 −). Der Aufwand für die Bildverarbeitung und den Kollisionstest ist mäßig, sodass Echtzeitfähig erreichbar scheint (K3 +). Die Sicherheit hängt von der Leistungsfähigkeit der Bildverarbeitung ab (K1 ±). Die Verfügbarkeit ist aufgrund der globalen Bahnplanung hoch (K2 +).

Das zertifizierte (K1 +) Produkt [SafetyEye06] (Abbildung 1.5 rechts) der Firma Pilz überwacht mittels eines tri-okularen Kamerasystem den Arbeitsraum eines Roboters. Durch ein Stereoverfahren wird für die detektierten Objekte eine Entfernung von den Kameras berechnet, sodass sich eine 2,5-dimensionale Umweltinformation ergibt bei einer Zykluszeit von 200 bis 300ms (K3 ±). Bei Verletzung von benutzerdefinierten Schutzzonen wird das Robotersystem lediglich in Schleichfahrt versetzt oder gestoppt, wodurch die Verfügbarkeit deutlich eingeschränkt ist (K2 −). Objekte werden lediglich vor einem statischen Hintergrund erkannt. Die Anwendungsdomäne ist daher beschränkt (K4 ±).

Time-of-Flight-Tiefenkameras sind ebenfalls durch die gelieferte 2,5D-Rauminformation zur Überwachung geeignet, ihr hohes Rauschen, die geringe Auflösung (im Vergleich zu konventionellen Kameras) und die generellen Probleme aktiver Verfahren ist sowohl die Sicherheit (K1 ±) als auch die Verfügbarkeit (K2 ±) eingeschränkt. Objekte dürfen sich nur in einem bestimmten Bereich vor der Kamera aufhalten, da es ansonsten zu Fehlmessungen kommt (K4 ±). Die Echtzeitfähigkeit ist je nach Integrationszeitraum gegeben (K3 ±), schränkt dann jedoch durch hohes Rauschen bei reduziertem Integrationszeitraum die beiden ersten Kriterien weiter ein. Im Rahmen der Mensch-Roboter-Koexistenz/Kooperation existieren Systeme, die eine Kamera einsetzen [Graf08, Schiavi09 und Winkler07] und hierdurch dem Problem der Abschattung ausgesetzt sind, was sie nur für wenige Anwendungen geeignet macht (K4 ±). Systeme, wie in [Mure-Dubois08] und [Kim09] dargestellt, verwenden mehrere TOF-Kameras zur Raumrekonstruktion, sind aber zu rechenaufwendig [Kim09] (K3 −) oder bieten nicht die

notwendige Semantik zur Realisierung eines Kollisionstests zwischen bekannten Objekten (Roboterarm) und Hindernissen (K4 −). Dieser und weitere Punkte betreffend allgemeiner Rekonstruktionsverfahren soll im nächsten Absatz kurz erläutert werden. Zu erwähnen bleibt noch ein Mensch-Roboter-Koexistenz-System, welches auf der Detektion einer punktbasierten Raumbelegung arbeitet: [Meisel94] (Abbildung 1.5 links). In diesem System wird der dreidimensionale Arbeitsraum eines Portalroboters mittels mehrerer Kamerasysteme überwacht. Im Arbeitsraum werden dazu grob Raumpunkte verteilt und für jeden Punkte in den Kamerabildern bestimmt, ob an der projizierten Stelle eine Änderung zum erwarteten Bildhintergrund vorliegt (einfaches Differenzbildverfahren), welches recht schnell berechnet werden kann (K3 +). Resultierend ist eine Belegung der Punkte des Arbeitsraumes, der den Roboter jedoch nicht enthalten darf, da dieser nicht herausgerechnet werden kann. Verdeckungen durch bekannte Objekte werden nicht berücksichtigt. Durch diese Voraussetzungen ergibt sich nur eine stark eingeschränkte Verwendbarkeit und Verfügbarkeit (K2 − und K4 −). Die Sicherheit hängt von der Leistungsfähigkeit der Bildverarbeitung ab (K1 ±).

In der Literatur sind sehr viele reine Rekonstruktionsverfahren zu finden (siehe auch [Ladikos08]), die aus (Tiefen-)Kamerabilddaten Belegungsinformationen des überwachten 3D-Raums berechnen. Diese bieten jedoch nicht die für einen Kollisionstest notwendig Semantik. Hierfür sind *Kollisionsklassen* für alle Objekte zu bestimmen, die festlegen, welche Objekte zur Realisierung der Sicherheitsfunktion untereinander auf Kollision getestet werden müssen und welche nicht, da letztere beispielsweise Teil des Prozesses sind, der vom Roboter ausgeführt wird und kollidieren können müssen. Die Objekte jeder Klasse müssen dazu in der Rekonstruktion sicher erkannt und separiert werden. Für Bahnplanungsaufgaben besteht ein Teil der Objekte aus parametrisierten Modellen (Roboterarm) und kann daher in der Rekonstruktion platziert werden (unter der Voraussetzung kalibrierter Kameras). Darauf aufbauend müssen Algorithmen zur Trennung der Modelle von realen Hindernisobjekten eingesetzt werden, die nicht modelliert sind. Da die Rekonstruktion der visuellen Hülle von Objekten aus Kameradaten auch Hüllen für die modellierten Objekte erzeugt, die immer größer-gleich dem jeweiligen Modell sind, ist eine einfache Subtraktion des Modells von den rekonstruierten Hüllen nicht möglich. Es würde in diesem Fall eine Menge von belegten Raumbereich übrig bleiben, welche als Hindernisobjekt erkannt werden würden, was aufgrund der Nähe zu den modellierten Objekten zu einer hohen Anzahl an falsch erkannten Kollisionen führen würde.

Eine reine Rekonstruktion kann daher einen sinnvollen Kollisionstest nicht realisieren ohne die oben dargestellten Probleme zu behandeln. Dies beinhaltet auch Betrachtungen von Raumbereichen, die durch bekannte Objekte wie den Roboterarm in den Kamerasichten verdeckt sind und deren Interpretation als „belegt“ bzw. „nicht belegt“. Die Betrachtung der

(tiefen-)bildbasierten Rekonstruktion mit Definition einer solchen Semantik und deren Erstellung wird durch zwei parallel laufende Promotionsprojekte am betreuenden Lehrstuhl bearbeitet. Die im Folgenden dargestellten Systeme zur Erfassung globaler Umwelt-information sind daher auf rein bildbasierte Verfahren ohne Raumrekonstruktion beschränkt.

In [Baerveldt92] wird ein Multi-Kamerasystem zur Überwachung von Roboterarbeitsbereichen dargestellt, welches eine dreistufige Geschwindigkeitsregelung des Roboters realisiert (siehe Abbildung 2.1a). Die tatsächliche Fusion der Kameradaten und Kalibrierung der Kamerasysteme ist nicht beschrieben, es wird lediglich die Funktion einer Überkopfkamera dargestellt. Durch eine Differenzbildberechnung auf Spezialhardware („Data Cube“) werden Hindernisse mit einer Latenz von 20ms detektiert (K3 +) und durch einen nicht genauer spezifizierten Algorithmus in „Kein Hindernis“, „Fernes Hindernis“ und „Nahes Hindernis“ eingeteilt und darauf eine dreistufige Geschwindigkeitsregelung (bis zum Stillstand) für den überwachten Roboter gesetzt (K2 − und K4 −). Die Realisierung von Ausweichbewegungen wurde vorgeschlagen, jedoch nicht realisiert. Verdeckungen (Abschattungen) der Kameraperspektiven wurden nicht berücksichtigt (K1 ±).

Die hier dargestellte Arbeit hat durch das System von [Ebert03] (ebenfalls entwickelt im SIMERO-Projekt) einen Vorläufer, dessen Konzepte jedoch in hier stark abgeändert, erweitert oder ersetzt werden. Details zu [Ebert03] finden sich in Kapitel 2.1. Generell besteht bei [Ebert03] das Problem, dass relativ hohe Anforderungen an die reale Arbeitszelle (statischer Bildhintergrund) und die darin enthaltenen Geometrien (Roboter) gestellt werden und die dargestellte Bahnplanung auf drei Freiheitsgrade und einen kleinen Arbeitsraum reduziert wurde, sodass sich Einschränkungen bei den Kriterien K1, K2 und K4 ergaben, die den Bedarf einer Weiterentwicklung weckten, die mit dieser Arbeit präsentiert wird.

Schlussfolgerungen

Die dargestellten Verfahren haben jeweils eine oder mehrere Schwächen in den genannten Kriterien Sicherheitsfunktion, Verfügbarkeit, Echtzeitfähigkeit und Anwendungsdomäne. Durch das in dieser Arbeit präsentierte Gesamtkonzept sollen alle Kriterien zugleich besser gelöst werden. In Tabelle 1.1 sind alle dargestellten Referenzen noch einmal übersichtlich bezüglich der Kriterien dargestellt.

Referenz	**K1**	**K2**	**K3**	**K4**
[Heinzmann99]	+	−	+	−
[Lim00]	+	−	+	−
[Heilig02]	+	−	+	−

Referenz	**K1**	**K2**	**K3**	**K4**
[Göger06]	+	−	+	−
[DeLuca06]	+	−	+	−
[Oberer-Treitz08]	+	−	+	−
[Zettl02]	+	−	+	+
[Morhard02]	+	−	+	+
[Novak92]	±	−	+	±
[Feddema94]	±	−	+	±
[Heilig02]	±	−	+	±
[Lumelsky93]	±	−	+	±
[Wegerif92]	±	−	+	±
[Martin99]	±	−	+	±
[Som05]	±	±	+	+
[Vieth08]	±	±	−	−
[Spingler02]	−	±	−	−
[Thiem02]	−	±	−	−
[Schulz03]	−	±	−	−
[Thiem05]	−	±	−	−
[Steinhaus99]	±	+	+	−
[Hoover99]	±	+	+	−
[Lee03]	±	+	+	−
[SafetyEye06]	+	−	±	±
[Graf08]	±	±	±	±
[Schiavi09]	±	±	±	±
[Winkler07]	±	±	±	±
[Mure-Dubois08]	±	±	±	−
[Kim09]	±	±	−	±

Referenz	K1	K2	K3	K4
[Meisel94]	±	−	+	−
[Baerveldt92]	±	−	+	−

Tabelle 1.1: Übersichtstabelle Stand der Forschung in der Reihenfolge der Referenzierung in Kapitel 1.3. In den Spalten sind die Kriterien Sicherheitsfunktion (K1), Verfügbarkeit (K2), Echtzeitfähigkeit (K3), Anwendungsdomäne (K4) aufgelistet.

1.4 Abgrenzung

In dieser Arbeit werden keine mobilen Roboter betrachtet, da hier die Mensch-Roboter-Koexistenz bereits hinreichend gelöst ist, durch Ausnutzung des stark reduzierten Planungsraum (2 bis 3 DOF) und Verwendung kostengünstiger Kollisionstests (im Allgemeinen in 2D).

Für das Robotersystem werden keine Fähigkeiten (Skills) entwickelt, die für die Kooperation notwendig wären, da die Realisierung der Koexistenz Vorraussetzung für die Kooperation bildet und daher zuerst bearbeitet werden muss. Prototypische Experimente in Richtung Kooperation wurden jedoch im Rahmen des SIMERO-Projekts durchgeführt [Kuhn06, Awais10], unter Verwendung der hier dargestellten Methoden zur Ermöglichung der Koexistenz.

Es ist weiterhin nicht Ziel dieser Arbeit Hardware- oder Software-Fehler zu berücksichtigen, wie dies für eine Erfüllung der einschlägigen Normen notwendig wäre, um ein sicheres System im Sinne dieser Normen zu konstruieren. Dieses Thema wird unter dem Stichwort „Redundanz“ in einem parallelen Promotionsprojekt im Rahmen des Industrie-Projekts SafetyVision erörtert. Auch soll keine Hardware selbst entwickelt werden, sondern auf Standard-Hardware zurückgegriffen werden.

Die 3D-Rekonstruktion des Raumes mittels mehrerer (Tiefen-)Kamerasysteme wird, wie bereits in Kapitel 1.3 erwähnt, in parallelen Promotionsprojekten am Lehrstuhl verfolgt [Kuhn09, Fischer09]. Die zu Beginn getroffene Entscheidung gegen die Rekonstruktion beruhte auch auf der Erwartung eines hohen Rechen- und Speicheraufwands für die Rekonstruktion.

1.5 Aufgabenstellung und Systemkonzept

1.5.1 Aufgabenstellung

Die im Folgenden gegebene Aufgabenstellung ist ein Überblick und wird in den jeweiligen Kapiteln in ihren Einzelheiten genauer spezifiziert.

Gegeben (siehe auch Abbildung 2.4) ist ein zu überwachendes Arbeitsraumvolumen $\boldsymbol{A}$ einer Roboterzelle mit enthaltenen bekannten Objekten $\boldsymbol{Z}_k$, deren Geometrie durch ein gegebenes Modell berechnet werden kann. Dies schließt ein Robotersystem mit ein (hier eine serielle Kinematik mit sechs Freiheitsgraden). Weiterhin sind mehrere stationäre (Farb-)Kamerasysteme gegeben, deren Sichtvolumen sich mit $\boldsymbol{A}$ schneiden. Die Bilder dieser Kameras sind synchronisiert und an einer zentralen Stelle vorverarbeitet verfügbar, die Systeme also vernetzt. Weiterhin ist ein zyklischer Prozess und ein zyklisches Roboterprogramm gegeben. Für das Roboterprogramm soll gelten: die durch das Programm definierten Anfahrpunkte müssen erreicht werden; zwischen diesen Punkten kann eine beliebige Bahn gewählt werden, welche von der direkten Verbindung abweicht, z.B. aufgrund von Ausweichbewegungen. Die Inhalte des Roboterprogramms sind, soweit nötig, dem System bekannt (insbesondere die Anfahrpunkte). Die Roboterkonfiguration zu einem beliebigen Zeitpunkt t vor der aktuellen Zeit steht dem System zur Verfügung. Der Roboterarm ist durch das System steuerbar. Ein Teil der Objekte $\boldsymbol{Z}_k$ wird im Rahmen des Roboterprogramms bewegt, die Zeitpunkte von Aufgreif- und Ablageoperationen sind dem System bekannt.

Gesucht ist die Detektion unbekannter Objekte $\boldsymbol{H}_l$ und deren Berücksichtigung während der Abarbeitung des Roboterprogramms durch die kollisionsfreie Bahnplanung mit Realisierung einer distanzbasierten Geschwindigkeitsregelung. Die Kollisionsfreiheit schließt bekannte Objekte ein. Auch sollen Kollisionen des Roboters mit sich selbst (insbesondere mit transportierten Objekten) verhindert werden.

1.5.2 Demonstrator

Zur Verifikation der Praxistauglichkeit des hier entwickelten Gesamtsystems wird eine prototypische Roboterzelle aufgebaut, die durch die Vereinigung verschiedener Komponenten einen gleichzeitigen Test der einzelnen Softwaremodule ermöglicht. Zu diesem Zweck wird ein Prozess realisiert, der den Transport von Objekten durch den Roboter beinhaltet mit Ablage- und Aufnahmepositionen sowie den Transport dieser Objekte auf einem Fließband, um kontinuierliche Bildveränderungen zu erzeugen und einen endlosen Prozesszyklus auf einefache Art und Weise zu ermöglichen.

Wie im linken Bild in Abbildung 1.7 dargestellt, befindet sich in der Demonstratorzelle eine „Bearbeitungsstation“ auf der linken Seite, ein Fließband im Hintergrund und Ablagemöglichkeiten auf dem im Vordergrund platzierten Tisch. Mit roten Ellipsen markiert sind die Positionen der verwendeten Farbkameras zur Überwachung, befestigt an einem Metallrahmen. Die roten Polygone markieren die Positionen der Bildverarbeitungsrechner, mit denen die Kameras verbunden sind. Diese sind vernetzt mit einem zentralen Rechner (grüner Quader) zur Fusion und Verarbeitung der Einzelbilder und

Steuerung des Roboterarms. Zu diesem Zweck ist der zentrale Rechner mit der Robotersteuerung (blauer Quader) vernetzt.

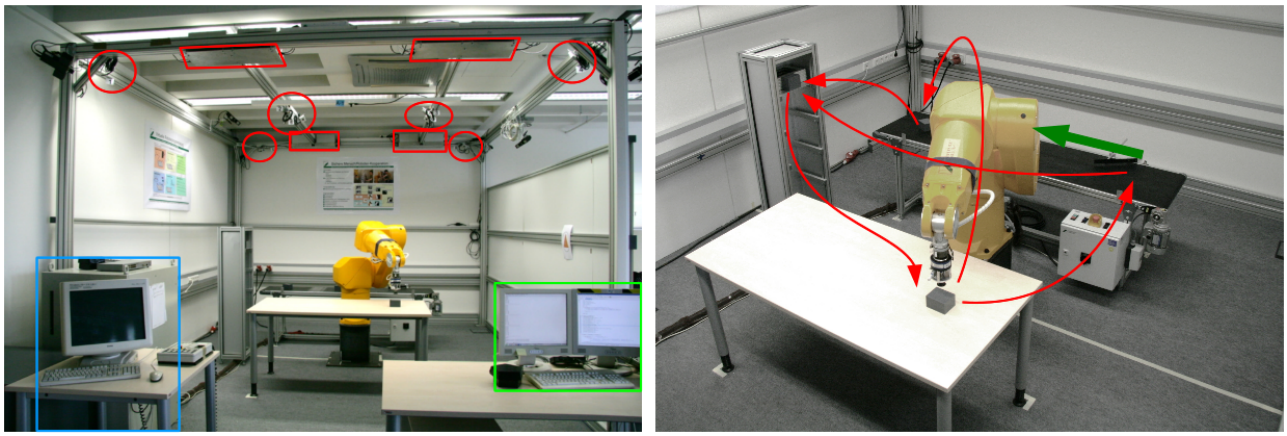


Abbildung 1.7: Demonstratorzelle und Ablauf des prototypischen Prozesses. ***Links****: Foto der gesamten Demonstratorzelle mit sechsachsigem Industrieroboterarm und markierten Hardwarekomponenten.* ***Rechts****: Schematischer Ablauf des Prozesses mit Aufnahme- und Ablagepositionen.*

Der gewählte, beispielhafte Prozess (Abbildung 1.7 rechts) läuft prinzipiell endlos, um beliebig lange Testsequenzen zu ermöglichen. Dazu wird über das Fließband eine Schleife im Prozess erzeugt, indem die durch den Roboter von der linken Seite auf die rechte Seite des Fließbands tranportierten Objekte (graue Plastikquader) wieder von rechts nach links (grüner Pfeil) tranportiert werden. Die dazwischen liegenden Transportstrecken vom Fließband zur „Bearbeitungsstation“ zur Ablage auf dem Tisch und wieder zum Fließband und den Zwischenbewegungen sind durch unterschiedliche Länge gekennzeichnet, um für die Bahnplanung eine breitere Basis an Aufgabenstellungen zu bieten.

1.5.3 Systemkonzept

Das grundlegende Systemkonzept ist in Abbildung 1.8 dargestellt. Die Demonstratorzelle wird mit Hilfe von Kamerasystemen überwacht. Hierzu sind diese Kameras statisch an einem Rahmen befestigt sind. Diese statische Befestigung ermöglicht die Vorteile kalibrierter Kameras und den Einsatz effizienter Differenzbildverfahren (*Change Detection*) zur Detektion von unbekannten Objekten $\boldsymbol{H}_l$. Die Kameras sind mit einem eigenen Bildverarbeitungsrechner verbunden, um den Aufwand bzw. Rechenanteil der Bildverarbeitung flexibel festlegen zu können. Die von der Bildverarbeitung erstellten Differenzbilder werden an den zentralen Rechner versendet. Dieser steuert den Roboterarm auf der Basis eines vorgegebenen Prozessablaufs (entspricht dem Roboterprogramm).

In jedem Zyklus iteriert das System die folgenden Schritte: Auf der Basis eines Differenzbildverfahrens werden die unbekannten Objekte $\boldsymbol{H}_l$ als Vordergrund in den Kameras segmentiert, soweit sie nicht verdeckt sind. Eine Weiterverarbeitung der entstandenen Differenzbilder unter Berücksichtigung von bekannten Verdeckungen erzeugt

Szenenbilder, die im Kollisionstest verwendet werden. Dieser wiederum wird von der Bahnplanung verwendet. Die Bahnplanung erzeugt und übermittelt eine aufgrund der Hindernissituation aktualisierte Bahn an die Steuerung des Roboterarms. Diese ist fähig die zuvor abgefahrene Bahn abzubrechen und die neue weiter zu verfolgen.

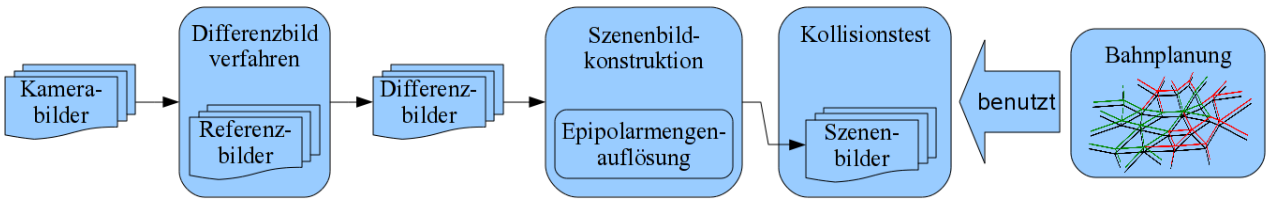


Abbildung 1.8: Systemzyklus. Die funktionalen Softwarekomponenten (in abgerundeten Rechtecken) werden in jedem Zyklus von links nach rechts ausgeführt.

Der Kollisionstest realisiert auch eine Abstandsberechnung, die zur Regelung der Geschwindigkeit des Roboter verwendet wird, siehe hierzu auch [Kuhn06]. Dies wird einmal pro Zyklus durchgeführt. Die Bewegung des Roboterarm erfolgt parallel zu diesen Berechnungen aufgrund der durch den Bahnplaner bestimmten Trajektorie und der vorgegebenen Geschwindigkeit. Die Geschwindigkeitsfunktion ist dabei über Sicherheitsparameter bestimmt, wie gemessene Distanz zu Objekten, maximale negative Beschleunigung (inklusive transportierter Lasten), Zykluszeit und maximale Geschwindigkeit unbekannter Objekte.

1.6 Kapitelübersicht

Die Anordnung Kapitel ist am Datenfluss des Gesamtsystems orientiert (siehe Abbildung 1.8). Daher wird zunächst in Kapitel 2 der bildbasierte Kollisionstest dargestellt, gefolgt von den Bahnplanervarianten in den Kapiteln 3 und 4. In Kapitel 3 ist dies der wegoptimierende Planer, da dieser die Basis für den zeitoptimierenden Planer bildet, welcher in Kapitel 4 dargestellt wird. In Kapitel 5 werden experimentelle Ergebnisse der Bahnplanervarianten vorgestellt. Kapitel 6 beinhaltet eine Zusammenfassung und einen Ausblick.

2 Bildbasierter Kollisionstest in dynamischen Arbeitszellen

In diesem Kapitel wird die Grundlage der in den Kapiteln 3 und 4 dargestellten Bahnplaner konstruiert, der bildbasierte Kollisionstest. Nach der Darstellung des Stands der Forschung im Bereich der bildbasierten Kollisionstests und der Abgrenzung zu alternativen Ansätzen in Kapitel 2.1 wird in Kapitel 2.2 die Aufgabenstellung präzisiert. Daran anschließend wird zunächst der bildbasierte Kollisionstest in Kapitel 2.3 und die Distanzberechnung in Kapitel 2.4 beschrieben ausgehend von ideal detektierten Differenzbildern unter Berücksichtigung von Verdeckungen durch bekannte Objekte. Die Erzeugung von Differenzbildern in Arbeitszellen mit dynamischen Bildhintergründen wird im darauf folgenden Kapitel 2.5 dargestellt, da hierbei Ergebnisse des Kollisionstests verwendet werden, beziehungsweise auf diesen verwiesen wird.

Der im folgenden häufig verwendete Begriff *Differenzbildverfahren* (im englischen auch *„Change Detection“*) lässt sich für diesen Kontext wie folgt definieren: Gegeben sei ein bildgebendes Verfahren zur Erfassung der Umweltinformation. Für diese Umweltinformation existiert ein Modell, welches den erwarteten Inhalt bzw. die erwarteten Werte (z.B. für jeden einzelnen Pixel) definiert. Ziel eines Differenzbildverfahrens ist es nun, Abweichungen von diesen Erwartungen zu definieren und ein sogenanntes *Differenzbild* zu produzieren, welches in jedem Pixel durch die Markierung *Vordergrund* eine Abweichung vom Modell anzeigt und im Falle einer Bestätigung des Modells den Pixel mit *Hintergrund* markiert.

2.1 Stand der Forschung

Kollisionsdetektion für Roboter mittels durch Kamerasysteme erfasster Umweltinformationen wurde in verschiedenen Systemen entwickelt mit sehr unterschiedlichen Zielrichtungen und Anwendungsbereichen. Es gibt Systeme aus der mobilen Robotik mit mobilen Kameras und stationären Kamerasystemen und beide Varianten auch für stationäre (Industrie)-Roboter (siehe auch Tabelle 2.1).

	Mobiler Roboter	**Stationärer Roboter**
Mobile Kamera	**(M/M)** Navigation	**(M/S)** Visual Servoing, Grasping
Stationäre Kamera	**(S/M)** Planung, Kollisionsvermeidung	**(S/S)** Planung, Kollisionsvermeidung

Tabelle 2.1: Beispielanwendungen für mobile/stationäre Roboter und mobile/stationäre Kamerasysteme. Die Aufgabenstellung dieser Arbeit liegt im Feld stationär/stationär

Mobile Roboter mit mobilen (subjektiven, also auf dem Roboter befestigten) Kameras können durch ihre Eigenbewegung kein einfach berechenbares Differenzbildverfahren verwenden, welches eine stationäre Kamera voraussetzt. Zudem ist durch den subjektiven Standpunkt die Umweltinformation lokal eingeschränkt und Verdeckungen können nur durch Fortbewegung erforscht (exploriert) werden. Planung in dynamischen Umgebungen ist daher nur auf der lokalen Umweltinformation möglich.

Der Nachteil der subjektiven Umweltinformation bei der Kombination M/M kann überwunden werden, wenn der Aufenthaltsbereich des mobilen Roboters bekannt und beschränkt ist und vollständig mit stationären Kameras überwacht werden kann. Die in diesem Rahmen entwickelten Systeme fallen in die Kategorie S/M. Sie entwickeln dabei unterschiedliche Strategien der Erkennung von Robotern (Differenzbildansätze, Farbmarkierungen etc.) und Hindernissen und der Art und Weise, wie die Umweltinformation den Robotern zur Verfügung gestellt wird (lokale oder globale Informationen) [Steinhaus99, Hoover99, Lee03]. Da der Planungsraum der mobilen Systeme im Allgemeinen zweidimensional ist, sind diese Ansätze daher für die hier vorgesehene Anwendung nicht ausreichend (siehe auch Kapitel 1.3).

Mobile Kameras für stationäre (Industrie-)Roboter (Kategorie M/S) werden unter anderem im Rahmen des Visual Servoing eingesetzt. Hierbei werden für objektrelative Positionierung beispielsweise des Robotergreifers visuelle Merkmale des (zu greifenden) Objekts erfasst und durch entsprechende Regelung mit einer Zielausrichtung in Deckung gebracht. Die Kamera(s) sind dabei im Allgemeinen fest mit einem Roboterglied (der Hand) verbunden, wodurch auch hier aufgrund des bewegten Hintergrundes einfache Differenzbildverfahren für die Hindernisdetektion nicht anwendbar sind. Die Kollisionsdetektion geschieht daher wenn überhaupt mit Hilfe eines 3D-Modells des

Arbeitsraumes und muss mit der Zielstellung des Visual Servoing in Einklang gebracht werden (z.B. [Mezouar02]).

Die Systeme im Feld S/S aus Tabelle 2.1 sind aufgebaut aus stationären Kamera- und Robotersystemen. Die dadurch mögliche Registrierung der Kamerakoordinatensysteme (und evtl. Roboterkoordinaten-systeme) als Teilresultat einer Kamerakalibrierung (wie z.B. [Svoboda05]) vereinfacht die Rekonstruktion von Rauminformationen z.B. mittels der shilouetten-basierten Erstellung von visuellen Hüllen [Ameling96, Meisel91, Meisel94, Niem97, Guan06, Ladikos08, Keck08] oder aufwendigeren Algorithmen wie z.B. sogenannten Photohüllen definiert über Farbkorrelationen [Magnor04]. Alternativ können Objektmodelle oder geometrische Primitive in die detektierten Hindernisse eingepasst werden [Braune05]. Verdeckungen durch bekannte Objekte werden bei diesen Verfahren nicht betrachtet. Die Rekonstruktion des Arbeitsraumes wird in dieser Arbeit nicht betrachtet (siehe auch Argumentation in Kapitel 1.3), um den Rechenaufwand einzusparen und den Kollisionstest lediglich auf Bilddaten zu rechnen. Die Untersuchung von Rekonstruktionsalgorithmen unter Berücksichtigung von Verdeckungen durch bekannte Objekte wird in einem parallelen Promotionsprojekt am selben Lehrstuhl im Rahmen des SIMERO-Projekts durchgeführt [Kuhn09]. Nicht unerwähnt bleiben sollen hier die seit einiger Zeit Verbreitung findenden Überwachungssysteme auf Basis von 2,5-D Time-Of-Flight (TOF)-Kameras. Diese erfassen mittels spezieller Sensorik zu jedem Pixel auch die Raumtiefe bis zur ersten Oberfläche. Werden die Sichten mehrerer kalibrierter Kameras kombiniert, sind auch hier Methoden der Rekonstruktion verwendbar. Die Untersuchung dieser Kameratypen wird ebenfalls in einem parallelen Promotionsprojekt innerhalb des SIMERO-Projekts durchgeführt (siehe [Fischer09]). Hierzu ähnlich sind Stereokamera-systeme (kommerzielle Systeme mit einer Perspektive wie [SafetyEye06]). Im Folgenden werden nun Arbeiten dargestellt, die Kollisions- und Distanzaussagen lediglich auf Bildinformationen berechnen.

In [Vieth08] und [Spingler02, Thiem02] (siehe auch Abbildung 1.6 und detaillierte Beschreibung in Kapitel 1.3) werden einzelne Körperteile detektiert und deren Eindringen in Schutzzonen [Vieth08] oder Abstände zum Roboterkörper [Spingler02, Thiem02] zum Anlass einer (stufigen) Geschwindigkeitsregelung des kontrollierten Geräts verwendet. In diesen Arbeiten werden somit nur Teile des Menschen und keine weiteren Hindernisse detektiert.

In der Arbeit von [Bischoff99] wird der TCP (Tool-Center-Point) eines Roboterarms in einer 2D-Welt geplant. Die Hindernisse werden über eine Farbkodierung der Umwelt mittels zweier Grauwertkameras erfasst. Somit werden alle Hindernisse erfasst, aber die Farbkodierung ist in der Praxis eine deutliche Einschränkung für die Anwendbarkeit.

Die Verwendung mehrerer Kamerasichten ist Basis für die Arbeiten von [Zeller97]

und [Okada09]. Bei diesen Arbeiten wird eine Projektionsfunktion gelernt mittels selbstorganisierender Neuronennetze, die die Roboterkonfiguration auf die Position des TCP in den Bildern abbildet (was daher auch mit unkalibrierten Kameras möglich ist). Dabei werden Hindernisse über eine Kollisionsdetektion in mehreren Bildern berücksichtigt (nur für den TCP bei [Zeller97] und für den gesamten Arm bei [Okada09], für letzteres siehe Abbildung 2.1b) und c)). Dabei reicht eine freie Kamera für die Detektion der Kollisionsfreiheit einer Roboterkonfiguration aus. Verbindungen zwischen den Knoten des Graphen werden nicht getestet, bzw. die Autoren führen hierzu nichts aus. Die Hindernisse müssen für die Dauer des Lernens (10 min bei [Okada09]) und der Planung statisch bleiben. Eine Start- und Zielkonfiguration einer zu planenden Bewegung werden auf die nächstliegenden Neuronen abgebildet und dann mittels Diffusion bzw. Potentialfeld-methoden ein Pfad innerhalb des Neuronennetzes geplant.

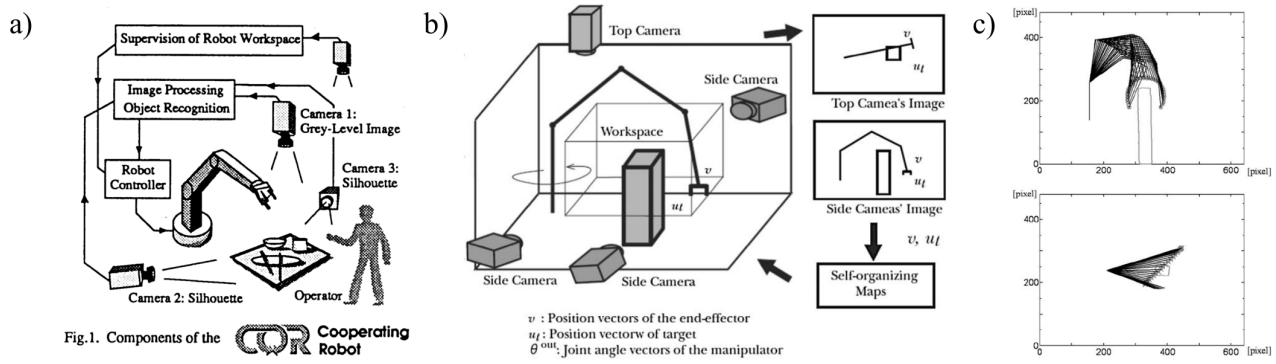


Abbildung 2.1: Bildbasierte Kollisionsdetektionssysteme a) [Baerveldt92] b) und c) [Okada09].

In [Hosoda95] wird ein bildbasierte Planung/Kollisionstest präsentiert, der für den TCP eines Roboters mittels zweier kalibierter Kameras einen kollisionsfreie Bahn plant. Hierbei darf es zu keinen Verdeckungen kommen. Eine lineare Verbindung von Start und Ziel wird in einer Kamera in eine Menge von Zwischenpunkten aufgeteilt und diese auf Kollision getestet. Im Falle einer Kollision werden diese Punkte in der zweiten Kamera auf Epipolarlinien verschoben, die mit den Punkten der ersten Kamera assoziiert sind. Wird durch diese Verschiebung Kollisionsfreiheit der Punkte erreicht, gilt die gesamte Bahn als kollisionsfrei und wird ausgeführt. Die Kanten zwischen den Punkten werden nicht getestet.

In [Quick96] wird ähnlich zu [Hosoda95] der TCP des Roboters auf Kollision mit mittels Differenzbildverfahren detektierten Hindernissen getestet, bzw. die Distanz zu jenen berechnet. Darauf aufbauend wurde eine Bahnplanung mittels Potentialfeldmethoden verwendet. Eine Betrachtung von Verdeckungen wird nicht diskutiert.

In [Leou92] wird ein System zur bildbasierten Detektion der Kollisions von mehreren Robotern in einem gemeinsamen Arbeitsraum beschrieben, bei dem die mehrachsigen Roboterarme an ihren Kanten so markiert werden, dass diese Kanten im Bild extrahiert

werden können. Eine Kollision mit weiteren Hindernissen und eine Betrachtung von Verdeckungen findet nicht statt.

In [Baerveldt92] wird ein Multi-Kamerasystem zur Überwachung von Roboterarbeitsbereichen dargestellt, welches eine dreistufige Geschwindigkeitsregelung des Roboters realisiert (siehe Abbildung 2.1a)). Die tatsächliche Fusion der Kameradaten und Kalibrierung der Kamerasysteme ist nicht beschrieben, es wird lediglich die Funktion einer Überkopfkamera dargestellt. Durch eine Differenzbildberechnung auf Spezialhardware (Data Cube) werden Hindernisse mit einer Latenz von 20 ms detektiert und durch einen nicht genauer spezifizierten Algorithmus in „Kein Hindernis“, „Fernes Hindernis“ und „Nahes Hindernis“ eingeteilt und darauf eine dreistufige Geschwindigkeitsregelung (bis zum Stillstand) für den überwachten Roboter gesetzt.

Für die bisher vorgestellten Systeme zur bildbasierten Kollisionserkennung lässt sich feststellen, das wenigstens eines der folgenden Kriterien nicht erfüllt ist:

- Betrachtung von Verdeckungen durch bekannte Objekte
- Detektion von a priori unbekannten Objekten
- Berechnung der Kollision/der Distanz mit einem vollständigen Robotermodell
- Keine Ausnutzung mehrerer Kameraansichten zur Planung auf 3D-Umweltdaten
- Möglichst geringe Anforderung an den überwachten Raum (Farbkodierungen etc.)
- Echtzeitfähigkeit in dynamischen Umgebungen

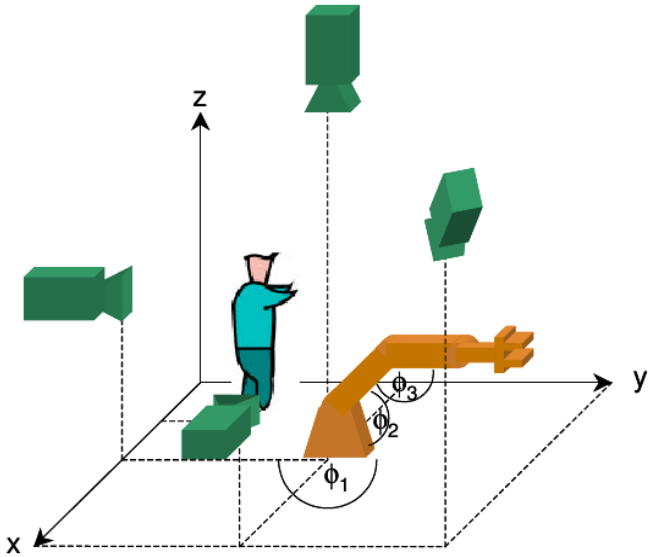


Abbildung 2.2: Bildbasiertes Überwachungssystem aus [Ebert03].

Diese Aspekte werden in einem kamerabasierten Überwachungssystem angesprochen [Ebert03], welches für das hier vorgestellte Gesamtsystem den Vorgänger darstellt. Die in diesem Kapitel entwickelten Algorithmen verbessern die Anwendbarkeit des bildbasierten Kollisionstests durch eine Reduktion der in [Ebert03] gestellten hohen Anforderungen und

Verbesserung der Kollisionsberechnung für zukünftige Roboterpositionen (im Rahmen der Planung). Weiterhin wird das System um eine Distanzberechnung erweitert.

Das System aus [Ebert03] soll im Folgenden kurz umrissen werden, um einen Vergleich zu erleichtern und einen Einstieg in den anschließenden, inhaltlichen Teil zu bieten. Das Verfahren aus [Ebert03] fällt in die Kategorie S/S und verwendet mehrere kalibrierte Kameras, die gemeinsam einen Arbeitsraum überwachen. Über ein Differenzbildverfahren werden unbekannte Objekte detektiert. Zu den als Vordergrund detektierten unbekannten Objekten gehört aufgrund des als statisch angenommenen Bildhintergrundes auch immer der bewegte Roboterarm. Für diesen existiert daher ein Modell, welches in die kalibrierten Kameras projiziert werden kann, wodurch die durch den Roboter verursachten Vordergrundpixel maskiert werden und aus den Kollisionsberechnungen ausgenommen werden.

Diese Roboterpixel stellen die einzige (dynamische) Verdeckung von unbekannten Objekten dar, da sowohl vor als auch hinter dem Roboterarm aus Sicht der jeweiligen Kamera kein Objekt detektiert werden kann. Dieser verdeckte Bereich wird daher mithilfe der anderen Kameras auf mögliche Objekte überprüft (siehe hierzu auch Abbildung 2.15 für eine ähnliche Vorgehensweise). Dazu wird jeder Roboterpixel mithilfe von Epipolarlinien in die anderen Kameras rückprojiziert und die jeweilige Epipolarlinie auf Schnitt mit den Pixeln unbekannter Objekte getestet. Damit ergibt sich für jeden Roboterpixel einer Kamera eine Anzahl an anderen Kameras, in denen ein solcher Schnitt auftritt. Um nun zu entscheiden, ob an der Stelle des Roboterpixels ein verdecktes Objekt vorliegt, wird ein Schwellwert über die Anzahl an Kameras mit Schnitt angewendet und dann je nach Entscheidung der Pixel in Vordergrund oder Hintergrund umgewandelt.

Bei [Ebert03] wird über den vorgenannten Schwellwert hinaus ein weiterer Schwellwert θ_{cam} definiert, der eine Annahme an die Verdeckung von Objektvolumina durch den Roboter definiert und daher mit dem oben genannten Schwellwert zusammenhängt. Dies wird allerdings nicht explizit ausgearbeitet, sondern über den Kollisionstest beschrieben. θ_{cam} definiert in [Ebert03] die Anzahl an Kameras, in denen die Projektion eines beliebigen Testvolumens als frei erkannt werden muss (also nicht mit Vordergrundpixeln eine gemeinsame Schnittmenge hat), um Kollisionsfreiheit anzuzeigen. Dies ist äquivalent zu der Aussage, dass $(\theta_{cam} - 1)$ die Anzahl an Kameras beschreibt, in denen ein einzelnes Arbeitsraumvolumen (dort als Voxel bezeichnet) vollständig durch die Projektion (dynamischer) bekannter Objekte verdeckt sein kann. Diese Annahme ist insofern sehr einschränkend, als dass es solche Volumina immer in erheblichem Umfang und Größe gibt (siehe Beispiel in Abbildung 2.3). Die in dieser Arbeit vorgenommene Redefinition des Parameters verbessert die Sicherheit und Verfügbarkeit des Überwachungssystems, die entscheidend von der Korrektheit dieses Parameters abhängt.

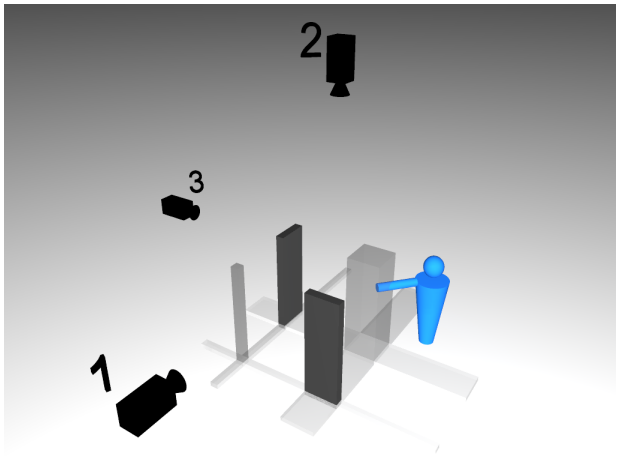


Abbildung 2.3: Schematische Beispielsituation für voluminöse Verdeckungen. Im Bild sind drei Kameras 1-3 in Schwarz positioniert und zwei (dynamische) Objekte in Dunkelgrau, welche eine Verdeckung erzeugen, in die ein Mensch (in Blau) mit dem Arm hineinragt. Auf der Bodenfläche der Szenerie sind durch den Schatten die parallel projizierten Verdeckungen aus den Kameras angedeutet, um die Entstehung des Verdeckungsvolumens zu erläutern. Das verdeckte Volumen, in welches der Mensch hineinragt ist in den Kameras 1 und 3 verdeckt.

Schlussfolgerung

Das wesentliche Problem des Ansatzes aus [Ebert03] ist die Annahme der Verdeckung eines beliebigen Arbeitsraumvolumens in maximal ($\theta_{cam} - 1$) Kamerabildern. Dies ist in der Realität oft nicht gegeben, selbst wenn vernachlässigbar kleine Volumina aus der Betrachtung ausgenommen werden. In den Volumina, die durch die Definition von θ_{cam} als nicht überwacht gelten müssen, dürfen sich daher bei diesem Ansatz keine Objekte aufhalten, beziehungsweise auch keine Teile von Objekten. Diese werden durch den Kollisionstest nicht erkannt, wodurch eine Kollision des Roboterarms nicht verhindert werden kann, falls die Planung den Roboter in die entsprechende Position bewegen möchte. Das Verfahren ist daher in diesem Lichte betrachtet als nicht sicher einzustufen. Die Sicherheit kann nur durch Anhebung des Schwellwertes θ_{cam} gewährleistet werden, was allerdings die Verfügbarkeit stark einschränkt, da dann wesentlich mehr Kollisionen erkannt werden, welche real nicht vorhanden sind. Im diesem Kapitel wird daher eine Methode erarbeitet, die dieses Problem behandelt und dazu den Blick von der Verdeckung einzelner Voxel auf die Ebene der Verdeckung ganzer Hindernisobjekte hebt (womit eine Mindestobjektgröße impliziert wird). Weiterhin werden bei [Ebert03] durch die Umwandlung verdeckter Pixeln in Vordergrund/Hintergrund mehr Kollisionen mit Objekten angezeigt, als tatsächlich vorliegen (in Abbildung 2.8 die Position *B*), auch dies wird durch die im Folgenden beschriebenen Algorithmen durch Weiterführung der Verdeckungsinformation verbessert.

Die hier gestellte Aufgabe besteht daher darin, eine praktikable Fusion von Einzelkamerakollisionstest für richtungsdiversitäre Kameras zu entwickeln, die unter den vorgegebenen, nicht zu stark einschränkenden Randbedingungen korrekt ist und bekannte

Verdeckungen berücksichtigt. Die Behandlung dynamischer Bildhintergründe (Ablagebereiche, Fließbänder, etc.) soll prototypisch untersucht werden, um die Einschränkungen durch einen statischen Bildhintergrundes für reale Anwendungen zu verringern. Im direkt anschließenden Kapitel wird eine formale Spezifizierung der Aufgabe präsentiert.

2.2 Aufgabenstellung

Gegeben ist eine Anzahl von insgesamt C stationären (Farb-)Kameras und ein Arbeitsraumvolumen $\boldsymbol{A}$. Die Kameras sind synchronisiert, somit erzeugt jede Kamera $c \in 1\ldots C$ ein Bild $I_c(t_k)$ des Arbeitsraumes zu den Zeitpunkten $t_k = t_0 + k * t_f$ mit t_f als (*f*rame-)Zykluszeit der Kameras und $k \in \mathbb{N}$. Die einzelnen Kameras müssen dabei nicht das gesamte Arbeitsraumvolumen $\boldsymbol{A}$ überblicken. Die extrinsischen und intrisischen Parameter jeder Kamera sind durch eine Kalibrierung bekannt (siehe z.B. [Svoboda05]). Es existiert eine Einteilung der Kameras in Gruppen g (siehe auch Abbildung 2.4), $g \in \mathbb{N}$ mit jeweils C_g Kameras in Gruppe g, $C_g \in 1\ldots C$, und mit einem pro Gruppe definierten Teil-Volumen $\boldsymbol{A}_g \subseteq \boldsymbol{A}$ des Arbeitsraumes, welches von allen zur Gruppe gehörigen Kameras vollständig eingesehen wird. Diese Teilvolumina müssen nicht überschneidungsfrei sein, ebenso müssen sich die C_g nicht zu C aufaddieren, es können Kameras in verschiedenen Gruppen „wieder verwendet“ werden. Für reale Anwendungen ist es sogar sinnvoll, dass die $\boldsymbol{A}_g$ sich überschneiden, um unüberwachte Lücken zu vermeiden. Die zu einer Gruppe g gehörigen Kameras werden über eine Menge $\boldsymbol{C}_g$ definiert, welche die zugehörigen Kameras aufzählt.

Im Arbeitsraum gibt es *bekannte Objekte* $\boldsymbol{Z}_k$, $k \in \mathbb{N}$, die modelliert sind und für die ein schneller Kollisionstest existiert (siehe hierzu [Henrich92] oder [Lin04]). Diese Objekte können beweglich sein wie z.B. der Roboterarm. Für diesen Fall sind alle zeitveränderlichen Parameter zu jedem Zeitpunkt t bekannt, die benötigt werden, um das jeweilige Modell vollständig zu beschreiben. Soweit die Objekte $\boldsymbol{Z}_k$ Verdeckungen in den einzelnen Kameras erzeugen können, sind diese Verdeckungen als Pixelmengen $\boldsymbol{U}_c(g)$ für jede Kamera c einer Gruppe g bekannt, bzw. können zur Laufzeit aus den gegebenen Modellen und einer Projektionsfunktion pro Kamera berechnet werden. Wenn im Folgenden zur Vereinfachung lediglich von $\boldsymbol{U}_c$ die Rede ist, so wird ein beliebiges, aber festes g impliziert.

Die bekannten Objekte $\boldsymbol{Z}_k$ können auch prinzipiell unendliche Objekte sein, wie z.B. Halbräume (Wände, Decke, Boden, …). Diese sind jedoch im Allgemeinen nicht Teil des Modells, welches nur Objekte enthalten muss, die Verdeckungen oder Kollisionen erzeugen können. Es gibt daher üblicherweise Arbeitsraumelemente, die zum Prozess gehören, aber nicht modelliert sind.

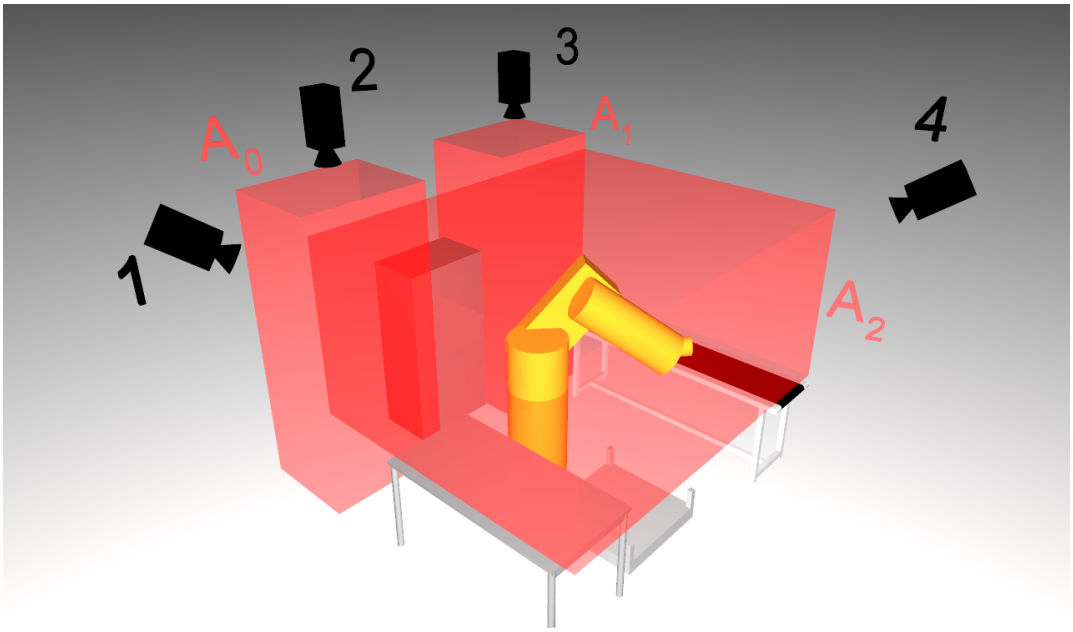


Abbildung 2.4: Definition mehrerer Überwachungsvolumina $\boldsymbol{A}_g$ durch den Benutzer mit zugehörigen Kameragruppen. Die Volumina sind als transparente rote Quader dargestellt (Volumen $\boldsymbol{A}_0$, $\boldsymbol{A}_1$ und $\boldsymbol{A}_2$). Weiterhin im Bild sind Zellaufbauten zu sehen: ein Tisch vorne links, ein Fließband hinten rechts, ein sechsachsiger Industrieroboter mittig, eine Palette als Ablagebereich unten rechts, eine (schematische) Werkzeugmaschine hinten links. Das Volumen $\boldsymbol{A}_2$ wird durch die Kameras 1 und 4 überwacht, diese sind als Kameragruppe G_2 definiert. Die Volumen $\boldsymbol{A}_0$ und $\boldsymbol{A}_1$ werden jeweils durch die Gruppe G_0 mit den Kameras 2 und 4 und die Gruppe G_1 mit den Kameras 3 und 4 überwacht.

Als *unbekannte Objekte* $\boldsymbol{H}_l$, $l \in \mathbb{N}$, seien alle Objekte definiert, die dem System nicht im Voraus bekannt sind („Hindernisse“), also zur Laufzeit detektiert werden müssen, wie z.B. der Mensch oder abgestellte Werkzeugkoffer. Pro Volumen $\boldsymbol{A}_g$ ist durch den Benutzer ein Parameter θ_g, $\theta_g < C_g$ definiert, der die maximale Anzahl an Kameras beschreibt, in denen ein unbekanntes Objekt $\boldsymbol{H}_l$ **vollständig** durch bekannte Objekte $\boldsymbol{Z}_k$ verdeckt sein kann (Details hierzu finden sich in den folgenden Abschnitten). Dieser Parameter ist sicherheitsrelevant und kann z.B. durch Simulation des Prozesses unter Annahme einer minimalen Größe der unbekannten Objekte automatisch bestimmt werden.

Da die unbekannten Objekte über einen recheneffizienten Differenzbildansatz („Change Detection“-Verfahren) detektiert werden sollen, steht dem Überwachungssystem darüber hinaus eine Bildsequenz des hindernisfreien Arbeitsraumes zur Verfügung, der mindestens einen Prozesszyklus des Roboters mit ausreichender Messfrequenz (Framerate) in Gänze erfasst und somit geeignet ist, alle *erwarteten* Bildzustände zu beschreiben. Die genaue Definition von „ausreichend“ ist in diesem Fall stark abhängig vom eingesetzten Differenzbildansatz und den Details des Prozesses.

Gesucht ist zunächst eine Detektion der unbekannten Objekte $\boldsymbol{H}_l$ in allen Kameras, die diese sehen können. Grundlage hierfür ist ein Differenzbildverfahren mit entsprechendem Modell des Hintergrunds. Diese Verfahren liefert pro Kamera c die Menge $\boldsymbol{F}_c$ der Vordergrundpixel, welche die Projektion der unbekannten Objekte darstellt, soweit diese

nicht (teilweise) verdeckt sind. Aufbauend auf der Detektion der unbekannten Objekte in den einzelnen Kameras ist ein Kollisionstest gesucht, der sich auf die Kollision der *kontrollierten Systemteile* (Teilmenge von $\boldsymbol{Z}_k$, hier der Roboterarm) mit den unbekannten Objekten bezieht. Dieser Test wird für einen gegebenen Parametersatz, in diesem Fall für eine gegebene Roboterkonfiguration r, durchgeführt. Weitergehend ist eine Distanzberechnung von den kontrollierten Systemteilen zu dem nächsten unbekannten Objekt gesucht, um den zeitoptimierenden Bahnplaner zu ermöglichen (Kapitel 4). Dabei soll der Kollisionstest oder die Distanzberechnung die Verdeckungen der unbekannten Objekte durch bekannte Objekte berücksichtigen.

Nicht gesucht im Rahmen dieser Arbeit ist die Absicherung von Roboterbewegungen in nicht-konvexen Arbeitsbereichen (z.B. Montage), die nur schwierig kamerabasiert überwacht bzw. nur mit wenigen Kameras eingesehen werden können. Denn daraus resultieren viele Möglichkeiten zu statischen und dynamischen Verdeckungen, was eine Kollisionsdetektion stark erschwert oder unmöglich macht. Auch reicht die für diese Aufgaben nötige Auflösung der Kameras bei praktikabler Anordnung im Allgemeinen nicht aus, um eine Planung für die Montage zu ermöglichen. Hierfür müssten die Kameras sinnvoller auf dem Roboterarm befestigt werden und würden damit eine einfache Anwendung eines Differenzbildansatzes unmöglich machen. Daher werden diese Anwendungsbereiche hier ausgeblendet.

Übersicht

Die Darstellung des Lösungsansatzes ist im Folgenden in drei Kapitel aufgeteilt. Mit einem idealen Differenzbildverfahren als Basis, welches unverdeckte, unbekannte Objekte in jeder Kamera erfasst wird in Kapitel 2.3 der bildbasierte Multi-Kamera-Kollisionstest beschrieben. Auf den dort erarbeiteten theoretischen Grundlagen und Verfahren setzt die bildbasierte Distanzberechnung auf, welche in Kapitel 2.4 dargestellt wird. Die Untersuchungen zur Behandlung von dynamischen Bildhintergründen wie z.B. Fließbänder im Rahmen des Differenzbildverfahrens werden in Kapitel 2.5 präsentiert. Dabei werden Methoden aus Kapitel 2.3 wieder verwendet, was die Sortierung der Kapitel in dieser Reihenfolge motiviert.

2.3 Bildbasierter Multi-Kamera-Kollisionstest

Das Ziel dieses Kapitels ist es, die Berechnung der Kollisionsaussage für eine Roboterkonfiguration r zu beschreiben bezüglich der sensorisch erfassten unbekannten Objekte $\boldsymbol{H}_l$. Nach einer einführenden Festlegung der verwendeten Terminologie (Kapitel 2.3.1) sowie der Einführung der für den Kollisionstest grundlegenden Bildinformationen und dessen prinzipieller Konstruktion aus dem Differenzbild (Kapitel 2.3.2) wird der

Kollisionstest beschrieben (Kapitel 2.3.3). Die daraus folgenden Konsequenzen für die Überwachung werden in den Kapiteln 2.3.4 und 2.3.5 erörtert. Die Leistungsfähigkeit der beschriebenen Vorgehensweisen werden durch den Aufbau des Arbeitsraumes und der Positionierung der Kamerasysteme beeinflusst. Wenngleich dies nicht zur Aufgabenstellung dieser Arbeit gehört, sollen einige wesentliche Punkte in Kapitel 2.3.6 angesprochen werden. Teile dieses Kapitels wurden in [Gecks06] und [Henrich08] veröffentlicht.

2.3.1 Terminologie

Gegeben sind *C* Kameras (s.o.). Die kameraspezifischen, unendlichen Überwachungsvolumina $\boldsymbol{A}_c$ für jede Kamera *c* sind über eine hier verwendete Abwandlung der Projektionsfunktion $Proj_c(\boldsymbol{x})$ des Lochkameramodells definiert, welche für einen Raumpunkt $\boldsymbol{x}$ einen Pixel $\boldsymbol{p}$ der Kamera liefert, wenn der Raumpunkt durch das Lochkameramodell auf einen Pixel in der Projektionsebene projiziert, ansonsten die leere Menge:

$$Proj_c(\boldsymbol{x}) = \begin{Bmatrix} \boldsymbol{p} & \text{, wenn } \boldsymbol{x} \text{ auf den Pixel } \boldsymbol{p} \text{ projiziert} \\ \emptyset & \text{, sonst} \end{Bmatrix} \tag{2.1}$$

Damit ergibt sich $\boldsymbol{A}_c$ zu der Menge $\{\boldsymbol{x} | Proj_c(\boldsymbol{x}) \neq \emptyset\}$. Weiterhin sei eine Funktion $Proj_c(\boldsymbol{V})$ definiert, die zu einem Volumen $\boldsymbol{V}$ die Menge aller Pixel $\boldsymbol{p}$ liefert, auf welche die Punkte $\boldsymbol{x}$ aus $\boldsymbol{V}$ abbilden:

$$Proj_c(\boldsymbol{V}) = \{\boldsymbol{p} | Proj_c(\boldsymbol{x}) = \boldsymbol{p},\ \boldsymbol{x} \in \boldsymbol{V}\} \tag{2.2}$$

Betrachtet man umgekehrt die Rückprojektion der Pixel $\boldsymbol{p}$ einer Kamera in den dreidimensionalen Arbeitsraum, so ergibt sich eine endliche Menge von unendlichen Pyramiden, die einander berühren (bei idealen Pixeln ohne Zwischenraum). Betrachten mehrere Kameras einen gemeinsamen Arbeitsraum, so schneidet sich ein Teil der Pyramiden unterschiedlicher Kameras. Dadurch wird eine Raumaufteilung in eine endliche Menge von nichtleeren Projektionsschnittvolumina $\boldsymbol{P}_j$ (Abbildung 2.5) erzeugt.

Für jedes dieser Volumina gilt, dass die Projektionen beliebiger Punkte $\boldsymbol{x}$ und $\boldsymbol{y}$ aus dem Volumen identisch sind, das heißt auf den gleichen Pixel oder die leere Menge in der jeweiligen Kamera *c* fallen:

$$\forall\ \boldsymbol{x} \in \boldsymbol{P}_j \forall\ \boldsymbol{y} \in \boldsymbol{P}_j \left(\forall\ c \left(Proj_c(\boldsymbol{x}) = Proj_c(\boldsymbol{y}) \right) \right) \tag{2.3}$$

Zur Vervollständigung der Definition der $\boldsymbol{P}_j$ muss noch beschrieben werden, dass die Komplementärmenge sich in mindestens einer Kamera unterscheidet, denn sonst könnten $\boldsymbol{P}_j$ konstruiert werden, die aus nur zwei Raumpunkten bestehen würden. Daher müssen die $\boldsymbol{P}_j$ auch die folgende Bedingung erfüllen:

$$\forall\ \boldsymbol{x} \in \boldsymbol{P}_j \forall\ \boldsymbol{y} \notin \boldsymbol{P}_j \left(\exists\, c\ Proj_c(\boldsymbol{x}) \neq Proj_c(\boldsymbol{y}) \right) \tag{2.4}$$

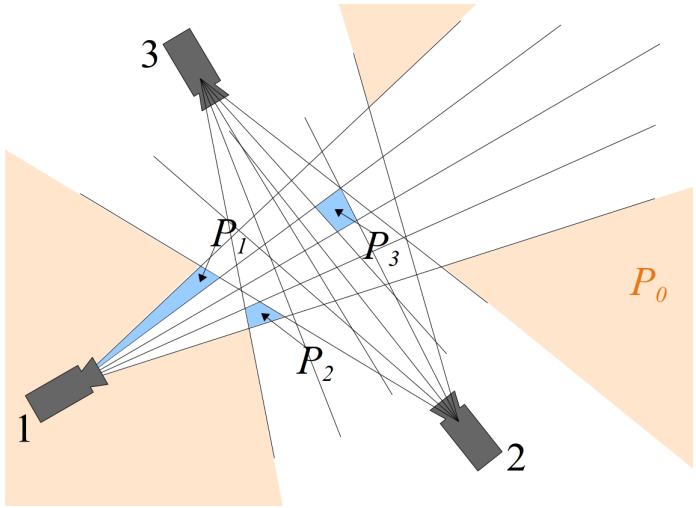


Abbildung 2.5: Schematische 2D-Darstellung der entstehenden Projektionsschnittvolumina $\boldsymbol{P}_j$ durch Rückprojektion der Pixel der Kameras. Das nicht zusammenhängende Volumen $\boldsymbol{P}_0$ wird von keinem Pixel einer Kamera erfasst und ist durch die orange eingefärbten Flächen dargestellt. Von den übrigen Volumina sind drei beispielhaft durch blauen Hintergrund hervorgehoben. Das Volumen $\boldsymbol{P}_1$ wird von Kamera 1 gesehen, das Volumen $\boldsymbol{P}_2$ von Kamera 1 und 3, das Volumen $\boldsymbol{P}_3$ von allen Kameras.

Diese Definition entspricht den sogenannten „Conexeln“ aus [Casas06]. Es entsteht hierbei auch ein nicht notwendigerweise zusammenhängendes spezielles Volumen $\boldsymbol{P}_0$ (Abbildung 2.5), das durch die Eigenschaft ausgezeichnet ist, dass es von keiner Kamera überwacht wird: $\boldsymbol{P}_0 = \{\boldsymbol{x} | \forall\, c\, Proj_c(\boldsymbol{x}) = \emptyset\}$. Für die folgenden Betrachtungen wird eine weitere Funktion definiert, die die Menge aller Kameras liefert, in die alle $\boldsymbol{x}$ aus einem $\boldsymbol{P}_j$ projizieren. Diese Funktion kann auch als Signaturfunktion für das Projektionsschnittvolumen $\boldsymbol{P}_j$ bezeichnet werden:

$$Sig(\boldsymbol{P}_j) = \{c | Proj_c(\boldsymbol{x}) \neq \emptyset,\ \boldsymbol{x} \in \boldsymbol{P}_j\} \tag{2.5}$$

Das Volumen $\boldsymbol{P}_2$ in Abbildung 2.5 hat beispielsweise die Signatur {1,3}. Für alle Mengen $\boldsymbol{P}_j$ mit $j \neq 0$ gilt, dass die Projektion der gesamten Menge in mindestens einer Kamera ein gültiges Pixel trifft: $|Sig(\boldsymbol{P}_j)| \geq 1,\ \boldsymbol{P}_j$, mit $j \neq 0$. Für alle Mengen $\boldsymbol{P}_j$ innerhalb eines Überwachungsbereiches $\boldsymbol{A}_g$ muss gelten, dass alle Kameras der Gruppe jeden Punkt aus diesen Mengen sehen können müssen: $\boldsymbol{A}_g = \{\boldsymbol{P}_j | Sig(\boldsymbol{P}_j) = \boldsymbol{C}_g\}$, also die Signaturfunktion die Menge $\boldsymbol{C}_g$ liefert, welche alle zugehörigen Kameras der Gruppe enthält.

Weiterhin existieren wie in der Aufgabenstellung formuliert eine Menge von L (dynamischen, zusammenhängenden) *unbekannten Objekten* $\boldsymbol{H}_l$, die mit den bekannten Volumina $\boldsymbol{Z}_k$ keine gemeinsame Schnittmenge haben (können): $\forall\, l \forall\, k : \boldsymbol{H}_l \cap \boldsymbol{Z}_k = \emptyset$. Sie werden als unbekannte Objekte bezeichnet, da sie nicht modelliert sind, sondern lediglich sensorisch erfasst werden können. Der Schnitt mit den Projektionsschnittvolumina $\boldsymbol{P}_j$ ist nichtleer für jedes Objekt $\boldsymbol{H}_l$: $\forall\, l : (\exists\, j : \boldsymbol{P}_j \cap \boldsymbol{H}_l \neq \emptyset)$.

Der in der Aufgabenstellung als gegeben beschriebene Parameter θ_g für jede

Kameragruppe g lässt sich mit den obigen Definitionen nun genauer fassen: Dieser besagt, dass für jedes unbekannte Objekt $\boldsymbol{H}_l$ gilt, dass unter allen erwarteten Umständen für Kameras die Projektion dieses Volumens in maximal θ_g vollständig verdeckt ist:

$$\forall\, l : \left(\left| \left\{ c \,|\, c \in \boldsymbol{C}_g, \left(\forall\, \boldsymbol{V} \subset \boldsymbol{H}_l \left(Proj_c(\boldsymbol{V}) \subseteq \boldsymbol{U}_c(g) \right) \right) \right\} \right| \;\leq\; \theta_g \right) \tag{2.6}$$

Umgekehrt betrachtet, kann $\boldsymbol{V}$ ein einziges Teilvolumen von $\boldsymbol{H}_l$ sein, welches in mindestens $(C_g - \theta_g)$ Kameras als Objekt detektiert wird. Es können aber auch mehrere $\boldsymbol{V}$ in $\boldsymbol{H}_l$ existieren, die jeweils in weniger als $(C_g - \theta_g)$ Kameras gesehen werden, wobei jedoch insgesamt die Bedingung 2.6 erfüllt ist. Bespielsweise kann in einer Drei-Kamera-Szenerie in einer Kamera der Kopf des Menschen sichtbar sein, in einer anderen die Füße und in der dritten ist er vollständig verdeckt. Dann gilt für diesen Fall $\theta_g = 1$.

Szenenbilder und Pixelmengen

Aus den (Farb-)Bildern I_c jeder Kamera c werden Differenzbilder D_c mif Hilfe eines Change-Detection-Verfahrens mit zwei Markierungsmöglichkeiten pro Pixel {Vordergrund, Hintergrund} erzeugt (siehe auch Abbildung 2.6 links und Mitte). Die (nicht notwendigerweise zusammenhängende) Menge der Vordergrundpixel repräsentiert die unbekannten Objekte und ist mit $\boldsymbol{F}_c$ bezeichnet. Die komplementäre Menge $\boldsymbol{B}_c$ entspricht den Hintergrundpixeln. Für die Betrachtungen in den folgenden Kapiteln wird ein idealer Differenzbildansatz angenommen, der die gesamten sichtbaren und nicht verdeckten Teile der Volumina $\boldsymbol{H}_l$ in jeder Kamera vollständig und konservativ erfasst:

$$Proj_c(\boldsymbol{H}_l) \setminus \boldsymbol{U}_c(g) \subseteq \boldsymbol{F}_c \tag{2.7}$$

Aus den Differenzbildern D_c werden pro Kameragruppe g insgesamt C_g *Szenenbilder* $S_c(g)$ erzeugt (Abbildung 2.6 rechts) für alle c in der jeweiligen Indexmenge $\boldsymbol{C}_g$, in denen außer unbekannten Objekten und Freiräumen die bekannten Verdeckungen vermerkt sind.

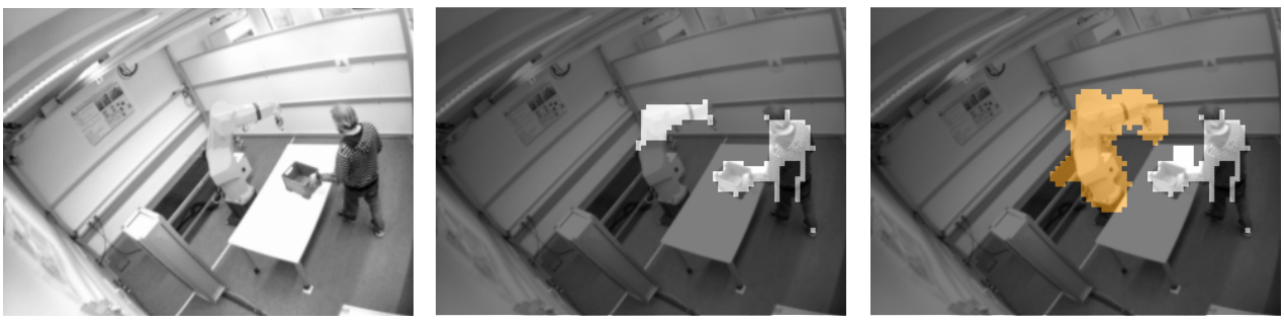


Abbildung 2.6: Kamerabildtransformation. ***Links:*** *Original-Kamerabild I_c (Farb- oder Grauwertbild).* ***Mitte:*** *Differenzbild D_c dem Originalbild überlagert mit den detektierten unbekannten Objekten als weiße Vordergrundpixel $\boldsymbol{F}_c$ und Hintergrund $\boldsymbol{B}_c$ als schwarze Pixel.* ***Rechts:*** *Szenenbild $S_c(g)$ mit Verdeckungspixeln $\boldsymbol{U}_c(g)$ in Orange, die aus dem projizierten Robotermodell erzeugt werden, Objektpixeln $\boldsymbol{O}_c(g)$ in Weiß und Freiraum $\boldsymbol{E}_c(g)$ in Schwarz.*

In jedem Szenenbild existieren Pixel mit drei unterschiedlichen Markierungen: {*Frei*,

Objekt, *Verdeckt*} und damit die zugehörigen Mengen $\boldsymbol{E}_c(g)$ von freien Pixeln („*Empty*“), $\boldsymbol{O}_c(g)$ von Objektpixeln („*Object*“) und $\boldsymbol{U}_c(g)$ von verdeckten Pixeln („*Undiscovered*“, Verdeckungspixel). Die Konstruktion der Szenenbilder aus den Differenzbildern der einzelnen Kameras wird im folgenden Kapitel beschrieben.

2.3.2 Szenenbildkonstruktion

Das Szenenbild muss alle Informationen enthalten, die für spätere Fusion der Kollisionstestdaten notwendig sind. Dazu gehören Informationen über den Freiraum, detektierte unbekannte Objekte und Verdeckungen. Die Menge der verdeckten Pixel $\boldsymbol{U}_c(g)$ kann aus statisch festgelegten Bildbereichen stammen (Arbeitsraumanalyse) oder dynamisch über ein projiziertes, parametrisiertes Modell (z.B. das des Roboterarms) erzeugt werden.

Konstruktion von $\boldsymbol{U}_c(g)$

Verdeckungsursachen können vielgestaltig sein und beziehen sich auf die jeweilige Kamera und das zugehörige Überwachungsvolumen $\boldsymbol{A}_g$ (Abbildung 2.7). Statische bekannte Objekte (Zellenaufbauten) haben ein bekanntes Erscheinungsbild, welches im Hintergrundbild des Differenzbildverfahrens enthalten ist. Sie können dennoch Verdeckungen erzeugen, wenn sich hinter ihrer von der Kamera aus sichtbaren Oberfläche im überwachten Raum $\boldsymbol{A}_g$ Volumina befinden, in denen sich ein unbekanntes Objekt vollständig oder teilweise aufhalten kann. Bei Projektion dieser Volumina kann es vorkommen, dass nur ein Teil des bekannten Objekts in den Bildern als Verdeckung gekennzeichnet werden muss (z.B. Objekt $\boldsymbol{Z}_0$ in Abbildung 2.7).

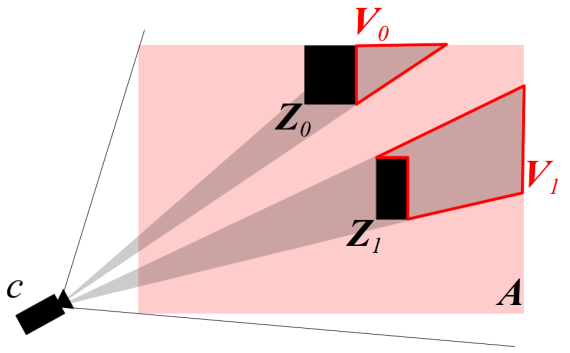


Abbildung 2.7: Vereinfachte Darstellung der Konstruktion von $\boldsymbol{U}_c(g)$ aus statischen bekannten Objekten $\boldsymbol{Z}_0$ und $\boldsymbol{Z}_1$. Das Objekt $\boldsymbol{Z}_0$ erzeugt aus der Sicht der Kamera c innerhalb des überwachten Volumens $\boldsymbol{A}$ die Verdeckung $\boldsymbol{V}_0$, in der unbekannte Objekte verdeckt sein können. Hier wird nur ein Teil des Objekts $\boldsymbol{Z}_0$ abgebildet, sodass der rückprojizierte Kegel der entsprechenden Verdeckungspixel $\boldsymbol{V}_0$ umfasst. $\boldsymbol{Z}_1$ wird vollständig als Verdeckung abgebildet, da das verdeckte Volumen $\boldsymbol{V}_1$ entsprechend groß ist.

Dynamische bekannte Objekte (Roboterarm, dynamische Hintergründe etc.) können unbekannte Objekte aus Sicht der Kamera vor und und hinter sich verdecken, da sie kein bekanntes Erscheinungsbild haben und somit in einfachen Differenzbildverfahren

Vordergrundpixel erzeugen. Dynamische bekannte Objekte werden daher vollständig projiziert und zu den Verdeckungspixeln $\boldsymbol{U}_c(g)$ hinzu genommen.

Alle Geometrien, die Verdeckungen erzeugen, müssen konservativ abgebildet werden, um keine teilweisen Verdeckungen von Pixeln zu übersehen und dadurch Sicherheitsprobleme zu erzeugen. Weitere Details zur Konstruktion von $\boldsymbol{U}_c(g)$ werden in Kapitel 2.3.6 betrachtet.

Szenenbildfusion

Die Szenenbildfusion verarbeitet die Differenzbilder mit zusätzlichen Informationen über die Verdeckungen und den überwachten Raum zu Szenenbildern basierend auf der Abbildungsvorschrift aus Tabelle 2.2. Die Abbildungsvorschrift arbeitet auf Pixelpositionen. Die Größe (BxH) von Differenz- und Szenenbild ist daher gleich.

Markierung in D_c	**Pixel in $\boldsymbol{U}_c$(g) ?**	**Pixel in Projektion von $\boldsymbol{A}_g$?**	**Markierung in $S_c(g)$**
Hintergrund	ja	ja	Verdeckt
Vordergrund	ja	ja	Verdeckt
Hintergrund	nein	ja	Frei
Vordergrund	nein	ja	Objekt
*	*	nein	Frei

Tabelle 2.2: Funktionstabelle für die Konstruktion eines Szenenbildes aus einem Differenzbild. Abbildungsvorschrift pro Pixel (die letzte Zeile expandiert zu 4 Zeilen durch die „Dont Care"-Terme in den ersten beiden Spalten).

Die Verdeckungen aus bekannten statischen Bereichen und aus dynamischen Bereichen wie dem Roboterarm werden dabei in einem Zwischenbild gespeichert und dienen als Eingabe für die zweite Spalte. Das Differenzbild D_c enthält alle detektierten Objekte für die Kamera c. Von diesen sind jedoch lediglich die Pixel relevant, die in dieser Kamera innerhalb der Projektion des von der Kameragruppe gemeinsam überwachten Volumens $\boldsymbol{A}_g$ liegen (Spalte 3), weshalb lediglich diese Werte übertragen werden und alle anderen Pixel im Szenenbild als Frei markiert werden (letzte Zeile). Die Projektion des Volumens $\boldsymbol{A}_g$ und Teile der verdeckten Pixel können offline berechnet werden.

2.3.3 Bildbasierter Kollisionstest

Im Folgenden wird abweichend von der tatsächlichen Abfolge der Berechnungen in Software (siehe Abbildung 1.8), zunächst der bildbasierte Kollisionstest dargestellt, um die weiteren Behandlungsschritte des Szenenbildes zu motivieren und das Verständnis zu erleichtern. Der Kollisionstest wäre ohne diese Schritte nicht korrekt, da er auf einer semantischen Umdefinition (s.u.) des Parameters θ_g beruht. Aus der Funktionsweise des Kollisionstest werden die weiteren Behandlungsschritte klarer. Der tatsächliche Ablauf in

Software ist in der Abbildung 2.26 auf Seite 73 in der Übersicht dargestellt. Die Korrektheit des gesamten Verfahrens wird durch Methoden sichergestellt, die in Kapitel 2.3.4 beschrieben sind.

Im folgenden Kapitel ist die Bedeutung des Parameters θ_g (vergleiche Formel 2.6) umdefiniert zu einem neuen Parameter β_g, da durch diese Definition die Konstruktion des Kollisionstests erheblich vereinfacht wird. β_g ist hier festgelegt als maximale Anzahl an Kameras, in denen jedes Projektionsvolumen $\boldsymbol{P}_j$, welches mit einem unbekannten Objekt $\boldsymbol{H}_l$ schneidet, verdeckt sein kann, also dessen Projektion zu den verdeckten Pixeln $\boldsymbol{U}_c(g)$ gehört:

$$\forall\, l \,\forall\, \boldsymbol{P}_j \cap \boldsymbol{H}_l \neq \emptyset : \left(\left| \left\{ c \middle| Proj_c(P_j) \in \boldsymbol{U}_c(g) \right\} \right| \leq \beta_g \right) \tag{2.8}$$

β_g hat dabei denselben Wert wie das durch den Benutzer vorgegebene θ_g, also $\beta_g = \theta_g$, lediglich die dahinter stehende Semantik ist eine andere. Der Parameter β_g ist deutlich restriktiver definiert im Vergleich zu θ_g und in der Praxis nicht zu erreichen, es sei denn es gilt $\beta_g = C_g$, was gleichermaßen zu einem nicht benutzbaren System führt. Kapitel 2.3.4 beschreibt, wie die restriktive Semantik von β_g erreicht wird, aus einem System heraus, welches lediglich θ_g erfüllt.

Aufgabenstellung

Gegeben ist ein zu testendes Volumen $\boldsymbol{T}(r)$ des Roboterarm an Konfiguration r, sowie eine Projektion dieses Volumens in jede Kamera c der Gruppe g, woraus eine Pixelmenge $\boldsymbol{T}_c(r)$ für jede Kamera resultiert, die jeweils nichtleer ist (daraus resultiert eine Mindestgröße für Testvolumina).

Gesucht ist die Kollisionsaussage für dieses Volumen, also: kollidiert dieses Volumen mit einem Hindernis oder nicht?

Für jede Kamera können nun zwei Prädikate definiert werden, die beschreiben, ob sich die Pixelmenge $\boldsymbol{T}_c(r)$ mit den Mengen $\boldsymbol{U}_c(g)$ oder $\boldsymbol{O}_c(g)$ schneidet:

$$\begin{aligned} Coll_O(g,c,r) &= \left(\left| \boldsymbol{O}_c(g) \cap \boldsymbol{T}_c(r) \right| \neq \emptyset \right) \\ Coll_U(g,c,r) &= \left(\left| \boldsymbol{U}_c(g) \cap \boldsymbol{T}_c(r) \right| \neq \emptyset \right) \end{aligned} \tag{2.9}$$

Aus diesen Prädikaten wird der Kollisionstest wie folgt definiert:

$$Coll(g,r) = \left(\forall\, c : \left(Coll_U(g,c,r) \vee Coll_O(g,c,r) \right) \right) \;\wedge\; \left(\left| \left\{ c \middle| Coll_O(g,c,r) \right\} \right| \geq \left(C_g - \beta_g \right) \right) \tag{2.10}$$

In Worten: Eine Roboterkonfiguration r ist kollisionsbehaftet, wenn in allen Kameras ein Schnitt mit den Objekt- oder Verdeckungspixeln existiert und die Zahl der Objektpixelschnitte größer oder gleich dem für die Gruppe gegebenen Schwellwert ($C_g - \beta_g$) ist. Negativ formuliert gibt es keine Kollision für die Konfiguration r, falls mindestens eine Kamera völlig frei von jeglichen Pixelschnitten ist oder die Anzahl der Objektpixelschnitte unter dem Schwellwert liegt:

$$Nocoll(g,r)=\left(\neg\left(\exists c:\left(\left(\neg Coll_U(g,c,r)\right)\wedge\left(\neg Coll_O(g,c,r)\right)\right)\right)\right)\ \vee \\ \left(\left|c|Coll_O(g,c,r)\right|<\left(C_g-\beta_g\right)\right) \tag{2.11}$$

Dies soll im Folgenden anhand einer vereinfachten schematischen Darstellung erläutert werden. In Abbildung 2.8 ist eine Situation mit unbekanntem Objekt ***H*** (visuelle Hülle: blau umrandeter Bereich) und Roboter ***R*** (visuelle Hülle: grün umrandeter Bereich) schematisch dargestellt. Der Parameter β_g ist für dieses Beispiel auf eins festgesetzt, was bedeutet, dass kein Teil eines Objekts in mehr als einer Kamera durch den Roboter verdeckt sein kann. Dies ist in diesem Beispiel nicht korrekt, denn es gibt in der visuellen Hülle des Roboters Volumina, welche in beiden Kameras verdeckt sind. Diese Problematik wird jedoch wie schon erwähnt im anschließenden Kapitel 2.3.4 behandelt, hier soll von diesen Fehlern abgesehen werden, um die prinzipielle Funktionsweise zu erklären.

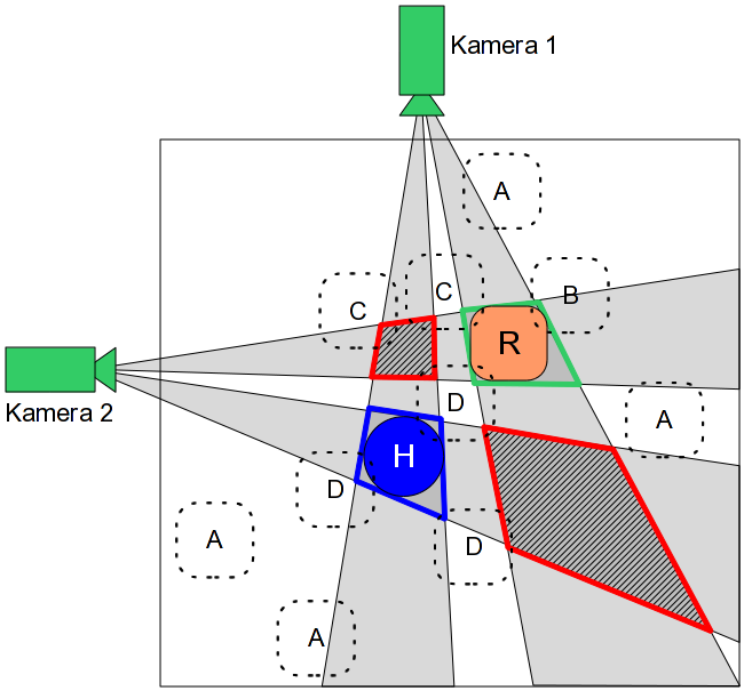


*Abbildung 2.8: Veranschaulichung des Kollisionstests mit einem Zwei-Kamera-System: Dargestellt ist eine prototypische Szene mit **R**oboter (visuelle Hülle: grün umrandeter Bereich) und unbekanntem Objekt (**H**indernis, visuelle Hülle: blau umrandeter Bereich). Rot umrandete, schraffierte Bereiche: Aufenthaltsorte möglicher unbekannter Objekte. Gestrichelt umrandete Bereiche: getestete, zukünftige Roboterkonfigurationen.*

Die Szenenbildfusion markiert die projizierten Bereiche des Roboters in den Kamerabildern als Menge $\boldsymbol{U}_c(g)$; es existieren keine weiteren Verdeckungen. In den rot umrandeten, schraffierten Bereichen könnte sich in Realität ein Objekt aufhalten, ohne dass sich die Abbildung der Szene in den Kameras verändern würde, Objekt- und Verdeckungspixel wären an gleicher Stelle. Die gestrichelt umrandeten Bereiche repräsentieren das Modell des Roboters an verschiedenen Konfigurationen, die auf Kollision getestet werden sollen.

Die Konfigurationen sind in Klassen A-D eingeteilt, die jeweils eine bestimmte , prinzipielle Kombination der Prädikate aus Gleichung 2.9 repräsentieren. Für jede Klasse ist eine Anzahl von prototypischen Roboterkonfigurationen in Abbildung 2.8 eingetragen. Tabelle 2.3 listet die möglichen Kombinationen von $Coll_U(g,c,r)$ und $Coll_O(g,c,r)$ für jede Kamera sowie die Zuordnung zu der entsprechenden Klasse auf. Der Gruppenindex g ist beliebig aber fest, und r ist jeweils eine Konfiguration für die die angegebenen Wahrheitswerte der angegebenen Prädikate zutreffen.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$Coll_O(g, 1, r)$	f	f	f	f	f	f	f	f	w	w	w	w	w	w	w	w
$Coll_U(g, 1, r)$	f	f	f	f	w	w	w	w	f	f	f	f	w	w	w	w
$Coll_O(g, 2, r)$	f	f	w	w	f	f	w	w	f	f	w	w	f	f	w	w
$Coll_U(g, 2, r)$	f	w	f	w	f	w	f	w	f	w	f	w	f	w	f	w
Klasse	A	A	A	A	A	B	C	C	A	C	D	D	A	C	D	D

Tabelle 2.3: Kollisionstestklassen aus Abbildung 2.8 (f = falsch, w = wahr). Aufgezählt sind alle möglichen Fälle von Belegungen der Prädikate in der ersten Spalte.

Die Klasseneinteilung dient dazu verschiedene, sich grundsätzlich unterscheidende Fälle zu illustrieren:

- Die Klasse **A** beinhaltet alle Fälle, in denen mindestens eine Kamera keinerlei Schnittmenge von $\boldsymbol{T}_c(r)$ anzeigt. In diesen Fällen ist das Testvolumen aus Sicht mindestens einer Kamera ganz im freien, unverdeckten Bereich des überwachten Raumes lokalisiert. Da das gesamte Volumen der rückprojizierten Pixel freien Raum repräsentiert ist auch jedes wie auch immer geartetes Testvolumen **kollisionsfrei** mit den unbekannten Objekten, da es immer vollständig in dem rückprojizierten Volumen liegen muss und damit im freien Raum. Hierzu reicht für die jeweilige Gruppe eine kollisionsfreie Kamera unabhängig von der Gesamtanzahl an Kameras. Die Klassen B-D haben im Unterschied mindestens ein wahres Prädikat pro Kamera durch Schnitte von $\boldsymbol{T}_c(r)$ mit $\boldsymbol{U}_c(g)$ oder $\boldsymbol{O}_c(g)$.
- Die Klasse **B** enthält die Fälle, in denen für jede Kamera mindestens ein Prädikat wahr ist. Die Anzahl der Schnitte ($Coll_O(g,c,r)$) von $\boldsymbol{T}_c(r)$ mit den Objektpixeln $\boldsymbol{O}_c(g)$ bleibt jedoch unterhalb des Schwellwerts ($C_g - \beta_g$). Dieser Schwellwert sagt hier aus, dass ein Schnitt mit Objektpixeln mindestens in dieser Anzahl an Kameras auftreten

muss, falls das Testvolumen $\boldsymbol{T}(r)$ einen Schnitt mit einem unbekannten Objekt aufweist. Für dieses Beispiel ist dieser Schwellwert 1, da $C_g = 2$ und $\beta_g = 1$. Der Fall B tritt daher hier nur einmal auf, da die Anzahl der $Coll_O(g,c,r)$ null sein muss und damit lediglich der Fall übrig bleibt, dass in beiden Kameras Schnitte mit verdeckten Pixeln angezeigt werden. Insgesamt ist dieser Fall als **kollisionsfrei** einzustufen.

- In Klasse **C** wird der Schwellwert an Schnitten mit Objektpixeln erreicht. Es gibt zwar eine Kamera, die keine Kollision mit Objekten, sondern lediglich mit verdeckten Bereichen anzeigt, in diesen Fällen muss jedoch eine **Kollision angenommen** werden, da eine Kollision mit einem Objekt in β_g Kameras nicht angezeigt würde, da das gemeinsame Volumen von $\boldsymbol{T}(r)$ und unbekanntem Objekt in dieser Anzahl an Kameras potentiell verdeckt ist. In dem gegebenen Beispiel müssen alle Konfigurationen als kollisionsbehaftet angenommen werden, in denen mindestens eine von beiden Kameras eine Kollision mit einem Objekt anzeigt und die andere Kamera eine Kollision mit Verdeckungen.
- In Klasse **D** gibt es in allen Kameras Schnitte von Objektpixeln und Testpixeln. Wenngleich dies definitiv **als Kollision bewertet** wird, muss in der Realität keine tatsächliche Kollision auftreten, da bei mehreren unbekannten Objekten im Raum potentiell Volumina $\boldsymbol{P}_j$ existieren, die in allen Kameras auf einen Objektpixel projizieren, jedoch kein reales Objekt enthalten (sogenannte „Pseudoobjekte“).

Die Korrektheit des Kollisionstests lässt sich aus den Voraussetzungen herleiten: Wenn ein Testvolumen $\boldsymbol{T}(r)$ mit einem unbekannten Objekt $\boldsymbol{H}_l$ kollidiert, so ist das Schnittvolumen $\boldsymbol{S}$ nichtleer: $\boldsymbol{S} = \boldsymbol{T}(r) \cap \boldsymbol{H}_l \neq \emptyset$. Für alle $\boldsymbol{P}_j$, die mit $\boldsymbol{S}$ schneiden, gilt, da sie Teil von $\boldsymbol{H}_l$ sind, die Bedingung aus 2.8. Durch das Differenzbildverfahren, welches die Bedingung 2.7 erfüllt, ist sichergestellt, das diese $\boldsymbol{P}_j$ auf Objektpixel $\boldsymbol{O}_c(g)$ abbilden, wenn sie nicht verdeckt sind. Mit diesen Voraussetzungen ist das Prädikat des Kollisionstest für jedes $\boldsymbol{P}_j$ erfüllt, welches $\boldsymbol{T}(r)$ und $\boldsymbol{H}_l$ schneidet. Somit wird eine Kollision angezeigt, falls eine vorliegt. Der umgekehrte Fall, dass sicher eine Kollision von $\boldsymbol{T}(r)$ mit einem unbekannten Objekt $\boldsymbol{H}_l$ vorliegt, falls das Kollisionstestprädikat erfüllt ist, gilt nicht (siehe obige Beispiele in Klasse D).

Mehrere Kameragruppen

Die obigen Absätze sind auf die Betrachtung einer Kameragruppe bezogen. Existieren mehrere Kameragruppen für einen Arbeitsraum so werden die Kollisionstestaussagen verodert: $Coll_{ges}(r) = \left(Coll(1,r) \vee \ldots \vee Coll(G,r)\right)$ für G Gruppen.

2.3.4 Behandlung nicht-sichtbarer Volumina

In diesem Abschnitt werden die Vorverarbeitungsschritte dargestellt, welche zur zuverlässigen Funktion des obigen Kollisionstests nötig sind. Im Folgenden gilt daher auch die Semantik des Parameters θ_g, im Gegensatz zum Kollisionstest, wo dann die Semantik von β_g gilt (s.o.). Zur Wiederholung: Der Parameter θ_g besagt, dass ein unbekanntes Objekt $\boldsymbol{H}_l$ unter allen Umständen Teilmengen besitzt, die für das gesamte Objekt betrachtet in mindestens $(C_g - \theta_g)$ Kameras auf einen Objektpixel abbilden.

Ohne die im Folgenden beschriebene Behandlung von Szenenbildern ergeben sich durch ein nach obiger Semantik bestimmtes θ_g Projektionsschnittvolumina $\boldsymbol{P}_j$, in denen nach obigem Kollisionstestalgorithmus und $\beta_g = \theta_g$ keine Kollision festgestellt würde, in die die unbekannten Objekte allerdings hineinragen können, da sie nicht mit einem bekannten Objekt $\boldsymbol{Z}_k$ belegt sind (Beispiele in den Abbildungen 2.10 und 2.11). Dieses „Hineinragen" muss dementsprechend sicher detektiert werden und die entsprechenden Bildbereiche markiert werden. Im Folgenden wird eine mögliche Behandlung dieser Volumina beschrieben. Nach dieser Behandlung können alle nachfolgenden Softwareprozesse (also Kollisionstest oder Epipolarlinientests) die Semantik von β_g verwenden.

Einteilung der Projektionsschnittvolumina $\boldsymbol{P}_j$

Durch die Verdeckungspixel $\boldsymbol{U}_c(g)$ und die Semantik des Parameters β_g entsteht eine Unterteilung der Projektionsschnittvolumina $\boldsymbol{P}_j$, die in dem jeweiligen Überwachungsvolumen $\boldsymbol{A}_g$ liegen. Die Menge der (für den Kollisionstest) sichtbaren $\boldsymbol{P}_j$ kann definiert werden als:

$$\boldsymbol{V}_g = \left\{ \boldsymbol{P}_j \mid \left(\boldsymbol{P}_j \cap \boldsymbol{A}_g = \boldsymbol{P}_j\right) \wedge \left(\left| \left\{ c \mid \left(c \in \boldsymbol{C}_g\right) \wedge \left(Proj_c\left(\boldsymbol{P}_j\right) \notin \boldsymbol{U}_c(g) \right) \right\} \right| \geq \left(C_g - \beta_g\right) \right) \right\} \qquad (2.12)$$

In Worten: Die Menge aller Projektionsschnittvolumina $\boldsymbol{P}_j$, deren Projektion in mehr als $(C_g - \beta_g)$ Kameras nicht mit einem Verdeckungspixel schneidet, die also in mindestens dieser Anzahl von Kameras nicht verdeckt sind. Zur Erinnerung: Die Verdeckungspixel werden konservativ aus den verdeckten Volumina bestimmt.

In der komplementären Menge zu $\boldsymbol{V}_g$ sind alle Volumina $\boldsymbol{P}_j$ enthalten, die in mehr als β_g Kameras verdeckt sind. Diese Menge ist ebenfalls einzuschränken auf das überwachte Volumen $\boldsymbol{A}_g$:

$$\overline{\boldsymbol{V}_g} = \left\{ \boldsymbol{P}_j \mid \left(\boldsymbol{P}_j \cap \boldsymbol{A}_g = \boldsymbol{P}_j\right) \wedge \left(\left| \left\{ c \mid \left(c \in \boldsymbol{C}_g\right) \wedge \left(Proj_c\left(\boldsymbol{P}_j\right) \notin \boldsymbol{U}_c(g) \right) \right\} \right| < \left(C_g - \beta_g\right) \right) \right\} \qquad (2.13)$$

Die unbekannten Objekte $\boldsymbol{H}_l$ können prinzipiell beide Mengen schneiden, es sei denn $\overline{\boldsymbol{V}_g}$ ist vollständig von den bekannten Objekten $\boldsymbol{Z}_k$ ausgefüllt, was jedoch unwahrscheinlich ist. Um die Korrektheit des Kollisionstests zu garantieren, müssen alle $\boldsymbol{P}_j$ in $\boldsymbol{V}_g$ liegen, was mit Hilfe der im Folgenden beschriebenen Methodik erreicht werden soll.

Teilweise Verdeckung unbekannter Objekte

Ein Objekt, das teilweise verdeckt ist, kann, wie in Abbildung 2.9 dargestellt, vielgestaltige Formen innerhalb der Verdeckung annehmen. Diese teilweisen Verdeckungen müssen detektiert werden und können dann entsprechend behandelt werden.

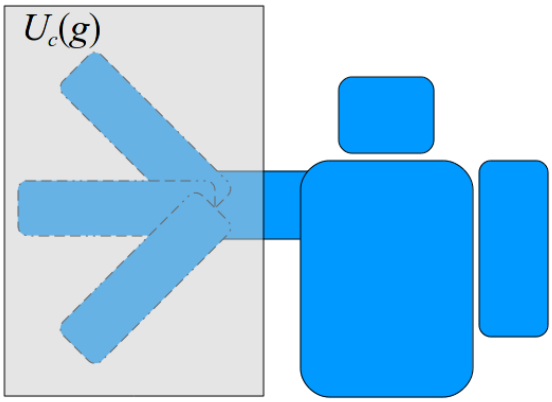


Abbildung 2.9: Unsicherheiten der tatsächlichen Objektgestalt in verdeckten Bereichen. Schematisches Kamerabild mit einem Menschen in Blau, Verdeckungsbereich $\boldsymbol{U}_c(g)$ in transparentem Grau. Verschiedene Unterarmpositionen sind mit gestrichelten Rändern angedeutet.

Die Detektion einer teilweise Verdeckung basiert auf einer pixelbasierten Nachbarschaft. Die Projektion des Objektvolumens $\boldsymbol{H}_l$ bildet für zusammenhängende Volumina (wovon hier ausgegangen wird) eine zusammenhängende Pixelmenge $Proj_c(\boldsymbol{H}_l \cap \boldsymbol{A}_g)$ in jeder Kamera c. Für diese Menge können drei Fälle auftreten, vorausgesetzt, die Schnittmenge mit $\boldsymbol{E}_c(g)$ ist leer, was die Bildverarbeitung und die Modellierung der Verdeckungen sicher stellen muss: (1) sie liegt vollständig in $\boldsymbol{O}_c(g)$, (2) sie liegt vollständig in $\boldsymbol{U}_c(g)$ und (3) sie hat nichtleere Schnittmengen mit $\boldsymbol{O}_c(g)$ und $\boldsymbol{U}_c(g)$. Für höchstens θ_g Kameras gilt Fall (2) nach Definition von θ_g. Für die anderen Kameras ($\geq C_g - \theta_g$) gilt (1) oder (3). Fall 1 beschreibt den Idealfall, dass das Objekt vollständig erkannt wird, in Fall 2 ist die Projektion des Objekts vollständig verdeckt. Da in beiden Fällen die Markierung des Objekts schon einheitlich ist, werden diese Fälle nicht weiter bildbasiert behandelt. Für die hier dargestellte Vorgehensweise ist somit Fall 3 relevant.

Wenn eine zusammenhängende Pixelmenge $Proj_c(\boldsymbol{H}_l \cap \boldsymbol{A}_g)$ nach Fall 3 existiert, so muss es mindestens ein Paar von Pixeln $(\boldsymbol{p}_u, \boldsymbol{p}_o)$ geben, von denen der eine, $\boldsymbol{p}_u$, in $\boldsymbol{U}_c(g)$ liegt und der andere, $\boldsymbol{p}_o$, in $\boldsymbol{O}_c(g)$. Zusammenhängend sei hier über die 4er-Nachbarschaft definiert, da ausgeschlossen werden soll, dass Objekte $\boldsymbol{H}_l$ so „schmal“ werden können, dass die resultierende Pixelmenge $Proj_c(\boldsymbol{H}_l \cap \boldsymbol{A}_g)$ an manchen Stellen nur über einen diagonalen Pixelschritt zusammenhängt. Alle Pixelpaare $(\boldsymbol{p}_u, \boldsymbol{p}_o)$ von benachbarten Pixeln mit der Eigenschaft, dass ein Pixel in $\boldsymbol{U}_c(g)$ und einer in $\boldsymbol{O}_c(g)$ liegt, werden im Folgenden als *Kontaktpixelpaare* bezeichnet. Eine Beispielsituation für das Auftreten solcher Kontaktpixelpaare ist in Abbildung 2.10 dargestellt.

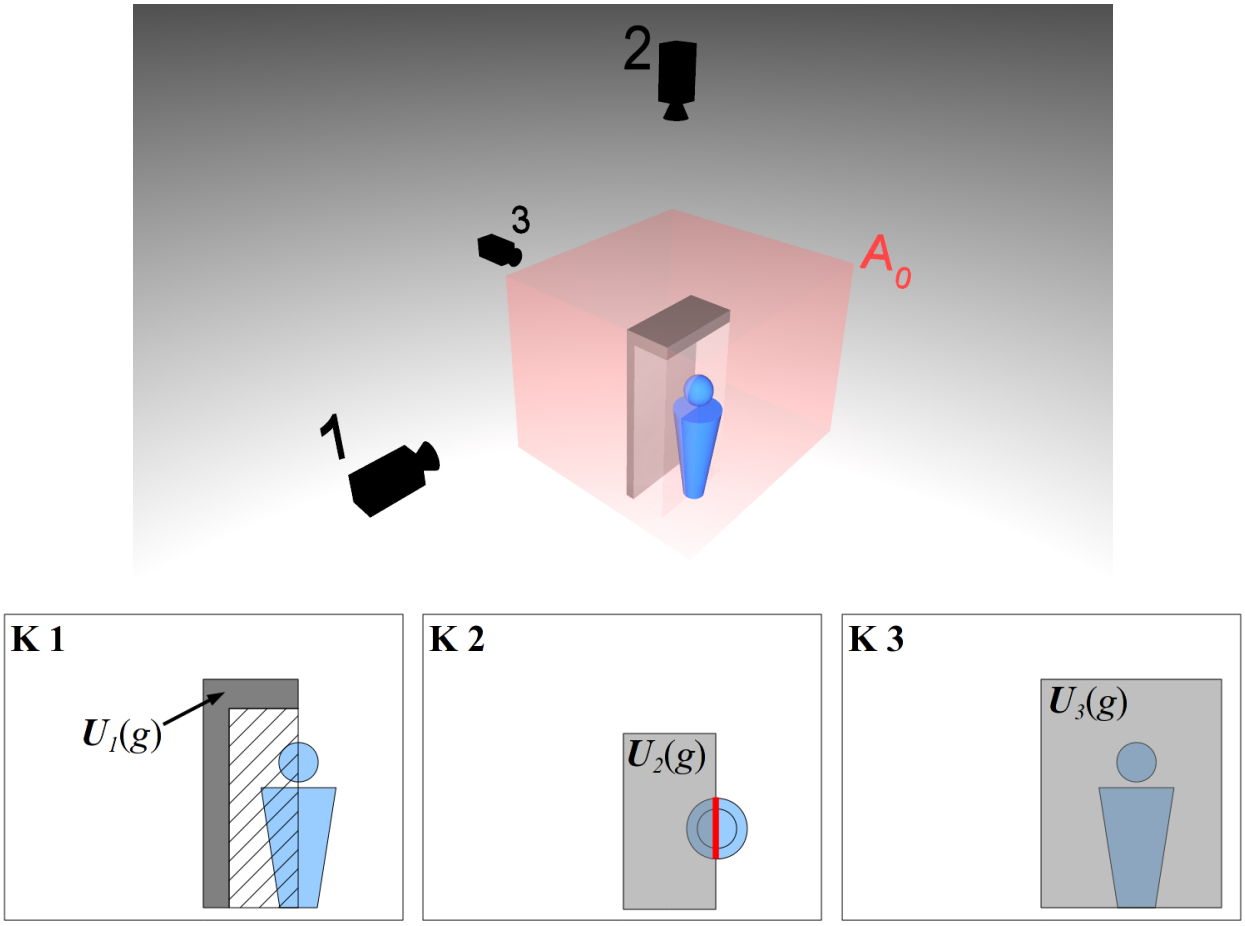


*Abbildung 2.10: Beispielsituation für das Auftreten nicht sichtbarer Volumina nach der Semantik von β_g. **Oben**: 3D-Situation mit einem Überwachungsvolumen $\boldsymbol{A}_0$ in transparentem Rot, drei Kameras (hier zur Vereinfachung als Parallelprojektion), einem unbekannten Objekt in Blau und verdeckenden Aufbauten in Dunkelgrau (beispielsweise ein Durchgangsbereich in umgedrehter L-Form), in transparentem Weiß (schwach sichtbar) markiert der nicht-sichtbare Bereich $\overline{\boldsymbol{V}}_g$ für ein hier gesetztes $\beta_g = \theta_g = 1$. **Unten**: Ansichten der drei eingezeichneten Kameras (**K**amera **1** bis **3**) mit den jeweils in Dunkelgrau (zum Teil transparent) dargestellten Verdeckungsbereichen $\boldsymbol{U}_c(g)$. In K1 zusätzlich schraffiert dargestellt das projizierte Volumen, das in zwei Kameras (K2 und K3) verdeckt ist. In K2 in Rot markiert: alle sogenannten Kontaktpixelpaare ($\boldsymbol{p}_u$, $\boldsymbol{p}_o$), die ein benachbartes Paar von Pixeln mit jeweils einem Pixel aus $\boldsymbol{U}_2(g)$ und $\boldsymbol{O}_2(g)$ darstellen. In den Kameras 1 und 3 existieren solche Paare nicht.*

Die Detektion dieser Paare kann effizient während der Szenenbildfusion (Kapitel 2.3.2) geschehen, da zu diesem Zeitpunkt sowohl die Verdeckungspixel $\boldsymbol{U}_c(g)$ als auch die Objektpixel $\boldsymbol{O}_c(g)$ erstellt werden und somit eine Nachbarschaft bei Iteration über die Pixel leicht festgestellt werden kann.

Behandlung durch Regionenfüllen

Die verdeckten Regionen, an die die Kontaktpixelpaare angrenzen, werden mit der Markierung Objekt gefüllt. Dies geschieht ebenfalls über die Betrachtung der 4er-Nachbarschaft, da die 8er-Nachbarschaft diagonal verbundene Verdeckungsbereiche (Berührung lediglich an einer Pixelecke) füllen würde, was bei einer Objektbreite von mindestens einem Pixel fehlerhaft sein muss. Es würde zudem potentiell zum Füllen größerer Mengen von Verdeckungspixel führen, als konservativ notwendig. Durch das

Füllen muss der gesamte zusammenhängende verdeckte Bereich im Bild erfasst werden, der mit dem jeweiligen Kontaktpixelpaar assoziiert ist, selbst wenn sich dieser Bereich aus mehreren einzelnen statischen und dynamischen Verdeckungen zusammensetzt, da die Rückprojektion der verdeckten Pixel für den bildbasierten Kollisionstest entscheidend ist. Durch die Kombination mehrerer Projektionskegel verdeckter Pixel aus verschiedenen Kameras einer Gruppe können beispielsweise auch Volumina aus $\overline{\boldsymbol{V}_g}$ entstehen, die im Raum nicht direkt an ein Verdeckungsobjekt angrenzen (siehe hierzu Abbildung 2.11 rechts).

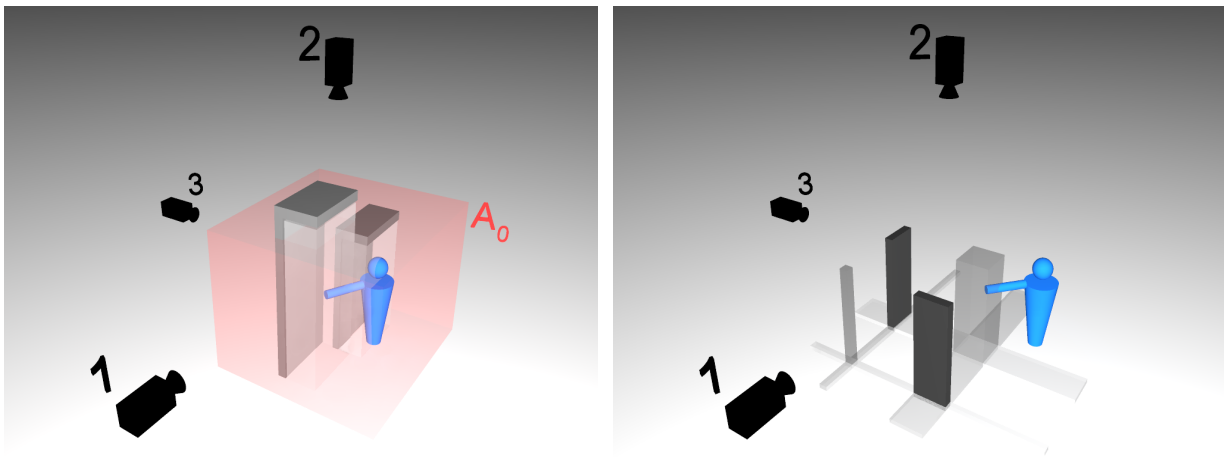


*Abbildung 2.11: Beispielsituation für assoziierte Verdeckungen (in beiden Beispielen gilt wie oben $\beta_g = \theta_g = 1$). **Links**: Beispielsituation für eine notwendige Verknüpfung von Verdeckungsbereichen, hier in Kamera 2. Das unbekannte Objekt grenzt in dieser Kamera lediglich an den Verdeckungsbereich, der durch das hintere statische Aufbauelement verursacht wird, ragt jedoch bis in den Verdeckungsbereich des zweiten, vorderen Aufbauelements hinein. **Rechts**: Beispielsituation für beliebig im Raum entstehende, nicht sichtbare Volumina $\overline{\boldsymbol{V}_g}$. Im Bild zu sehen sind zwei quaderförmige Verdeckungen (solid dunkelgrau) und die Projektion der rückprojizierten Verdeckungspixel aus den Kameras 1 und 3 ist auf den „Boden" der Zelle ist mit transparent hellgrauen Flächen dargestellt. Es ergeben sich durch die Rückprojektion zwei Volumina im Raum, die in beiden Kameras 1 und 3 verdeckt sind (kenntlich gemacht durch die senkrechte Quader in transparentem Dunkelgrau, das unbekannte Objekt $\boldsymbol{H}_l$ ragt in den Rechten mit dem „Arm" hinein). Die Quader grenzen nicht an ein bekanntes Objekt $\boldsymbol{Z}_k$.*

Das Regionenfüllen behandelt ein teilweise verdecktes unbekanntes Objekt konservativ korrekt, da alle Pixel auf die Markierung Objekt gesetzt werden und somit die Projektion des Objekts mindestens in dieser Kamera vollständig erfasst sein muss. Nach Behandlung aller Kontaktpixelpaare in allen Kameras auf diese Weise kann kein $\boldsymbol{P}_j$ mehr in $\overline{\boldsymbol{V}_g}$ enthalten sein.

Widerspruchsbeweis. Mindestens ein $\boldsymbol{P}_j$ eines unbekannten Objekts $\boldsymbol{H}_l$ sei nach Behandlung aller Kontaktpixelpaare in allen Kameras immer noch in $\overline{\boldsymbol{V}_g}$: Zunächst ist zu bemerken, dass nach Auflösung der Pixelpaare die Projektion eines zusammen-hängenden Objekts in allen Kameras entweder vollständig in $\boldsymbol{O}_c(g)$ oder $\boldsymbol{U}_c(g)$ liegen muss, da es durch die Behandlung keine Kontaktpixelpaare mehr geben kann und unbekannte Objekte in den Kameras konservativ vollständig aufgelöst werden, in denen sie zum Teil verdeckt sind. In

allen anderen Kameras ist die Projektion vollständig Teil von $\boldsymbol{O}_c(g)$ oder $\boldsymbol{U}_c(g)$ schon nach den Fällen 1 und 2. Also gilt für alle Projektionsschnittvolumina $\boldsymbol{P}_j$, die mit dem unbekannten Objekt $\boldsymbol{H}_l$ schneiden, für eine einzelne Kamera die Zugehörigkeit der Projektionen der Volumina vollständig entweder zu $\boldsymbol{O}_c(g)$ oder $\boldsymbol{U}_c(g)$. Für das unbekannte Objekt $\boldsymbol{H}_l$ existiert eine Indexmenge $\boldsymbol{I}_l$ aller Indices der $\boldsymbol{P}_j$, welche das Objekt schneiden:

$$\boldsymbol{I}_l = \left\{ j \,\middle|\, \left(\boldsymbol{P}_j \cap \boldsymbol{H}_l \cap \boldsymbol{A}_g \right) \neq \emptyset \right\} \tag{2.14}$$

Mit dieser Indexmenge gilt also nach obiger Behandlung durch Regionenfüllen:

$$\forall\, c \in \boldsymbol{C}_g : \quad \left(\left(\forall\; j \in \boldsymbol{I}_l : Proj_c\left(\boldsymbol{P}_j\right) \cap \boldsymbol{U}_c(g) \neq \emptyset \right) \;\vee\; \left(\forall\; j \in \boldsymbol{I}_l : Proj_c\left(\boldsymbol{P}_j\right) \cap \boldsymbol{O}_c(g) \neq \emptyset \right) \right) \tag{2.15}$$

Wäre dies nicht der Fall, so müsste eine Kamera ein Kontaktpixelpaar enthalten, da ein zusammenhängendes Objekt dann teilweise in der Objektpixelmenge und teilweise in der Menge der verdeckten Pixel liegen würde. (Fall 2). Sei nun für ein j aus $\boldsymbol{I}_l$ das zugehörige $\boldsymbol{P}_j$ in $\overline{\boldsymbol{V}_g}$, so müsste gelten:

$$\left(\left| \left\{ c \,\middle|\, \left(c \in \boldsymbol{C}_g \right) \wedge \left(Proj_c\left(\boldsymbol{P}_j\right) \cap \boldsymbol{O}_c(g) \neq \emptyset \right) \right\} \right| < \left(C_g - \theta_g \right) \right) \tag{2.16}$$

Da für alle j aus $\boldsymbol{I}_l$ die Projektion von $\boldsymbol{P}_j$ für jede Kamera c in jeweils die gleiche Menge $\boldsymbol{O}_c(g)$ oder $\boldsymbol{U}_c(g)$ fällt, würde die Bedingung aus Gleichung 2.16 für alle $\boldsymbol{P}_j$ mit j aus $\boldsymbol{I}_l$ gelten. Nach der Definition von θ_g gilt jedoch für die oben aufgeführten Schnittvarianten von $Proj_c(\boldsymbol{H}_l \cap \boldsymbol{A}_g)$ für maximal θ_g Kameras der Fall 2. Für die restlichen Kameras gilt entweder Fall 1 oder Fall 3 und nach der obigen Behandlung von Fall 3 gilt für alle restlichen Kameras (konservativ) Fall 1. Es gibt also mindestens $(C_g - \theta_g)$ Kameras, in denen die $\boldsymbol{P}_j$ aus $\boldsymbol{I}_l$ auf einen Pixel aus der Menge $\boldsymbol{O}_c(g)$ abbilden müssen, was ein Widerspruch zu Gleichung 2.16 darstellt. Ansonsten wäre θ_g inkorrekt gewählt.

Alle Volumina $\boldsymbol{P}_j$ in $\boldsymbol{A}_g$ sind somit nach dieser Behandlung in mindestens $(C_g - \theta_g)$ Kameras als Objekt markiert, wenn sie sich mit einem $\boldsymbol{H}_l$ schneiden. Dadurch gilt im Effekt die Semantik von β_g. Für alle darauf aufsetzenden Algorithmen wird daher im Weiteren immer von diesem Parameter die Rede sein, obwohl θ_g und β_g den gleichen Wert haben.

2.3.5 Kollisionsreduktion durch Epipolarmengen-Auflösung

Die Menge der als „Objekt“ oder „Verdeckt“ markierten Pixel in den Kamerabildern bestimmt die Verfügbarkeit des Systems durch die damit implizit definierten kollisionsbehaften Bereiche des Konfigurationsraumes.

Es ist daher wünschenswert, nicht jeden verdeckten Bereich zu füllen, der durch ein Kontaktpixelpaar dazu vorgesehen ist. Auch ist es sinnvoll verdeckte Pixel in Pixel mit der Markierung „Frei“ umzuwandeln, wenn sich kein Objekt im Projektionskegel befindet. Es besteht die Möglichkeit, beides mithilfe von Epipolargeometrie zu erzielen. Die folgende

Darstellung gliedert sich in drei Abschnitte: Erstellung von Epipolarmengen, Verifikation der Verletzung verdeckter Bereiche sowie die Auflösung verdeckter Pixel.

Erstellung von Epipolarmengen

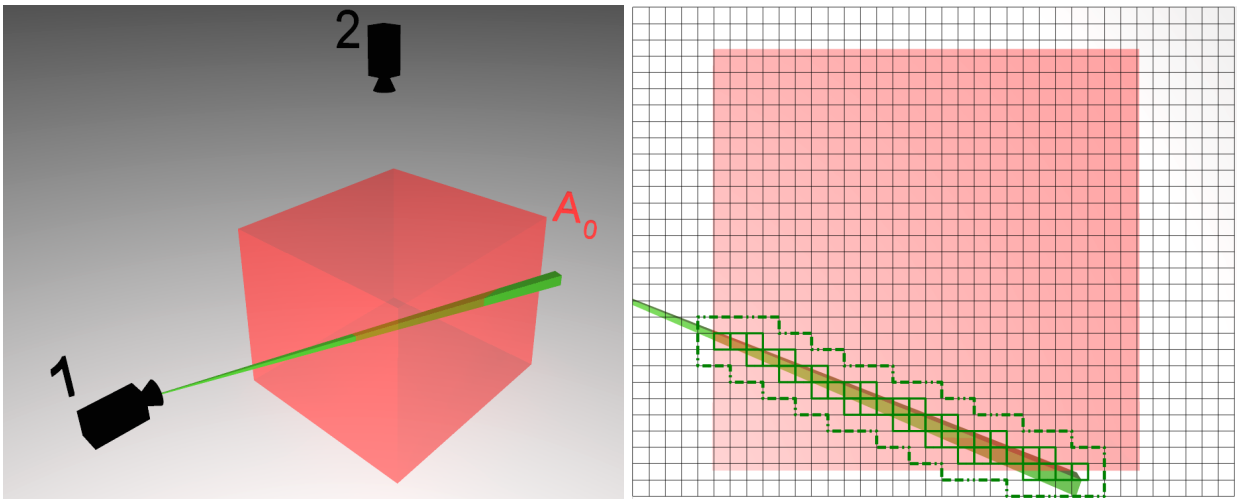


Abbildung 2.12: Epipolarmengenkonstruktion. ***Links****: Projektion eines Pixels aus Kamera 1 als (prinzipiell unendlich ausgedehnte) Pyramide.* ***Rechts****: Bild von Kamera 2 mit projizierter Pyramide. Durchgehend hellgrün umrandet: Beispielabdeckung durch Sampling mit fehlenden Pixeln. Gestrichelt hellgrün: Nach Expansion durch morphologischen Operator sind alle Pixel sicher abgedeckt.*

In der klassischen Multi-Kamera-Geometrie werden Epipolarlinien definiert, die von Punkt- und Linienkorrespondenzen zwischen Pixel und projiziertem Strahl ausgehen („Multiple View Geometry“, [Hartley04]). Pixel einer Kamera sind jedoch ausgedehnt und erzeugen genau genommen eine Pyramide im Raum, wenn sie rückprojiziert werden. Damit einher geht, dass diese Pyramide im Allgemeinen nicht auf eine Linie in den anderen Kameras abbildet, sondern im schlimmsten Fall auf eine unter Umständen das gesamte Kamerabild erfassende Pixelmenge (im Falle einer direkten Sichtverbindung zweier Kameras). Diese Menge wird im Folgenden als *Epipolarmenge* bezeichnet. Auch Pixel, die durch die Projektion der Pyramide nur teilweise überstrichen werden müssen aus Sicherheitsgründen zu der Epipolarmenge hinzugenommen werden. Formal muss für jeden Pixel $\boldsymbol{p}$ einer Kamera c eine Epipolarmenge $\boldsymbol{E}_i(c, \boldsymbol{p})$ von Pixeln in jeder anderen Kamera i ($i \neq c$) berechnet werden, die folgende Bedingung erfüllt:

$$\forall\, i \in \boldsymbol{C}_g, i \neq c, c \in \boldsymbol{C}_g, \boldsymbol{E}_i(c, \boldsymbol{p}) = \left\{ Proj_i(\boldsymbol{x}) \middle| \boldsymbol{x} \in \boldsymbol{A}_g : Proj_c(\boldsymbol{x}) = \boldsymbol{p} \right\} \tag{2.17}$$

Diese Mengen müssen in einem Vorverarbeitungsschritt erzeugt werden, da die Online-Berechnung zu aufwendig wäre. Die Berechnung der Epipolarmengen ist auf vielfältige Arten und Weisen möglich, die nicht Gegenstand dieser Arbeit sind; hier sei nur kurz eine simple Abtast-Methode dargestellt.

Die Abtast-Methode beruht auf einer regelmäßigen Abtastung des überwachten Arbeitsraumes $\boldsymbol{A}_g$, dessen Geometrie bekannt ist. Für jeden Abtastpunkt $\boldsymbol{x}$ werden die Pixel

aller Kameras berechnet, auf die dieser projiziert. Diese lokale, von $\boldsymbol{x}$ abhängige Pixelmenge wird pro Kamera in einer Datenstruktur abgespeichert, die für jeden Pixel die Menge an zugehörigen Pixeln der anderen Kameras enthält (parallele Konstruktion aller $\boldsymbol{E}_i(c, \boldsymbol{p})$). Dies ermöglicht zur Laufzeit einen wahlfreien Zugriff in O(1).

Die Abtastung muss fein genug sein, um in einer Kamera jeden Pixel, der vollständig von der Projektion der Pyramide überdeckt wird, mindestens einmal abzutasten. Für nicht vollständig überdeckte Pixel wäre jedoch prinzipiell eine unendlich feine Abtastung notwendig, um eine teilweise Überdeckung festzustellen. Um diese Komplexität zu vermeiden und dennoch konservativ sicher zu sein, werden die Mengen $\boldsymbol{E}_i(c,\boldsymbol{p})$ durch einen morphologischen Operator um einen Pixel expandiert. Um sicherzustellen, dass die Expansion von einem Pixel ausreicht, wird die notwendige Abtastung aus den ex- und intrinsischen Parametern der Kameras sowie der Geometrie des überwachten Arbeitsraumes konservativ geschätzt. Da so die Anzahl der Abtastpunkte sehr groß werden kann, kann die Berechnung entsprechend lange dauern. Die Auflösung und Anzahl der Kameras trägt wesentlich zur Speicherkomplexität bei. Für eine Bildauflösung von Breite W und Höhe H und C_g Kameras für die jeweilige Gruppe ergibt sich mit der Näherung $n=\sqrt{WH}$ eine Komplexität von $O(n^3 C_g^2)$. Jedem Pixel einer Kamera wird im Durchschnitt eine Menge $\boldsymbol{E}_i(c, \boldsymbol{p})$ von Pixeln aller anderen Kamera zugeordnet, deren Kardinalität im Bereich von n liegt, da im Allgemeinen eine Linie angenähert wird. Der schlimmste Fall ist allerdings $O(n^4 C_g^2)$, da eine Epipolarmenge ein gesamtes Bild umfassen kann wie oben beschrieben. Dies betrifft jedoch im Allgemeinen nur wenige Pixel, sodass diese Komplexität nicht praxisrelevant ist.

Konstruktionsexperiment Epipolarmengen

In diesem Experiment wurden Epipolarmengen für vier Kameragruppen berechnet, die für die Demonstratorzelle (Kapitel 1.5) definiert sind. In jeder Gruppe werden vier kalibrierte Kameras verwendet. Die berechnete Epipolarmengen wird pro Gruppe in einer Datei gespeichert. Die Größe der Dateien für die jeweilige Kameragruppe sind im Diagramm in Abbildung 2.13 dargestellt. Pro Gruppe wurde die Epipolarmenge für Bildgrößen von 80x60 und 160x120 Pixeln berechnet. Das Verhältnis der Dateigrößen pro Gruppe liegt im Bereich von 6,19 bis 6,42. Nach der obigen, abschätzenden Komplexitätsberechnung wird ein maximaler Faktor von 8 erwartet.

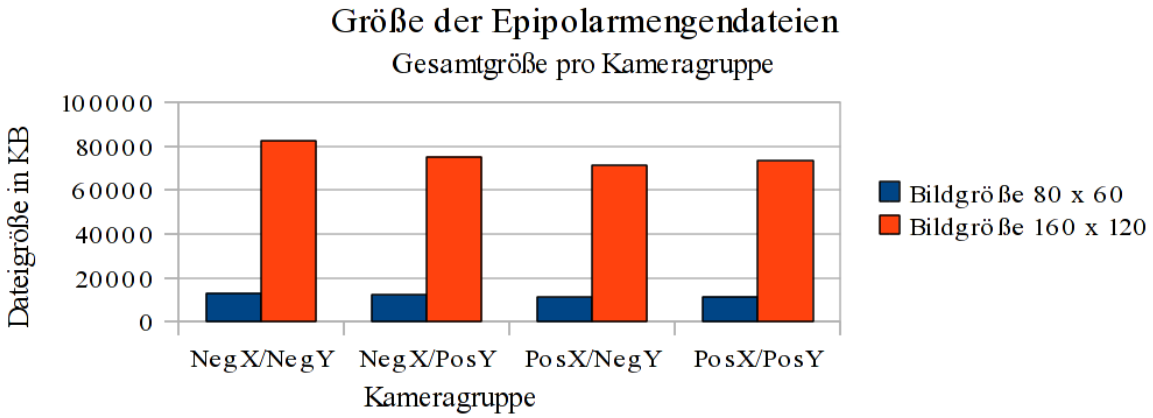


Abbildung 2.13: Dateigrößen der abgespeicherten Epipolarmengen für vier Kameragruppen (mit den Bezeichnern „NegX/NegY“ bis „PosX/PosY“), die für die Demonstratorzelle (Kapitel 1.5.2) definiert sind und jeweils ein Raumvolumen von 2000³ mm³ erfassen. Die Berechnung der Epipolarmengen aller vier Kameragruppen benötigt 3 min 37 sec auf der Hardwarekonfiguration S1 (siehe Anhang). Bei einer Bildgröße von 160x120 entstehen ca. 75 MB große, unkomprimierte Dateien im Textformat. Bei einer Bildgröße von 80x60 werden 1 min 48 sec benötigt mit Dateigrößen um 12 MB. Die Dateigrößen können aufgrund der dichten Kodierung mit einem geringen Anteil Fülltext für die Vergleiche der Speicherkomplexität approximativ verwendet werden.

Verifikation der Verletzung nicht sichtbarer Volumina bzw. der Kontaktpixelpaare

Mithilfe der oben definierten Epipolarmengen können Bedingungen für die Kontaktpixelpaare überprüft und dadurch für bestimmte Fälle das Füllen verhindert werden, wodurch die daraus resultierende Vergrößerung der Objektpixelmenge $\boldsymbol{O}_c(g)$ vermieden wird. Für die folgenden Darstellungen gilt die Semantik von θ_g, da sie vor der Behandung durch das Füllen stattfindet. Daher wird dieser Parameter in den Beschreibungen verwendet (siehe hierzu Abschnitt 2.3.4).

Die Abbildung 2.14 zeigt ein schematisches Beispiel für die folgende Erläuterung. Es sei in Kamera c ein Kontaktpixelpaar $(\boldsymbol{p}_u, \boldsymbol{p}_o)$ gewählt, mit $\boldsymbol{p}_u \in \boldsymbol{U}_c(g)$ und $\boldsymbol{p}_o \in \boldsymbol{O}_c(g)$, wobei $\boldsymbol{p}_u$ und $\boldsymbol{p}_o$ benachbart sind. Für den Pixel $\boldsymbol{p}_u$ wird nun überprüft, ob er die Abbildung eines Projektionsschnittvolumens $\boldsymbol{P}_j$ aus der Menge nicht-sichtbarer Volumina $\overline{\boldsymbol{V}_g}$ ist. Dies geschieht mit Hilfe der Epipolarmengen $\boldsymbol{E}_i(c, \boldsymbol{p}_u)$. Wenn es ein $\boldsymbol{P}_j$ in $\overline{\boldsymbol{V}_g}$ gibt, welches auf $\boldsymbol{p}_u$ abbildet, so muss dieses $\boldsymbol{P}_j$ in insgesamt mehr als θ_g Kameras verdeckt sein (vgl. Gleichung 2.13). Abzüglich der Verdeckung von $\boldsymbol{P}_j$ in der aktuellen Kamera c muss dieses $\boldsymbol{P}_j$ also in mindestens θ_g anderen Kameras verdeckt sein. Alle Epipolarmengen $\boldsymbol{E}_i(c, \boldsymbol{p}_u)$ in den anderen Kameras enthalten die Projektion dieses $\boldsymbol{P}_j$. Die Epipolarmengen schneiden daher in mindestens θ_g Kameras die Menge der verdeckten Pixel, wenn $\boldsymbol{P}_j$ in $\overline{\boldsymbol{V}_g}$ liegt:

$$\left|\left\{ i \mid \left(i \in \boldsymbol{C}_g, i \neq c, \exists\, \boldsymbol{p} \in \boldsymbol{E}_i(c, \boldsymbol{p}_u) : \boldsymbol{p} \in \boldsymbol{U}_i(g) \right) \right\}\right| \geq \theta_g \qquad (2.18)$$

Dies ist eine notwendige Bedingung für die Kamera c, denn durch diese Bedingung wird nicht garantiert, dass ein auf $\boldsymbol{p}_u$ abgebildetes $\boldsymbol{P}_j$ in $\overline{\boldsymbol{V}_g}$ liegt, jedoch wenn es ein solches $\boldsymbol{P}_j$ gibt, trifft die Bedingung zu.

Zusätzlich zu dieser strikten Betrachtung ist noch eine heuristische Reduktion der gefüllten Bereiche denkbar, die bei der Behandlung durch Regionenfüllen direkt angesetzt werden kann, indem ein Mindestdurchmesser für unbekannte Objekte in Pixeln angenommen wird und Flächen in den Verdeckungsbereichen nicht weiter gefüllt werden, die diesen Durchmesser unterschreiten. Auch ein Maximaldurchmesser für die Objekte kann die Ausbreitung des Regionenfüllens begrenzen.

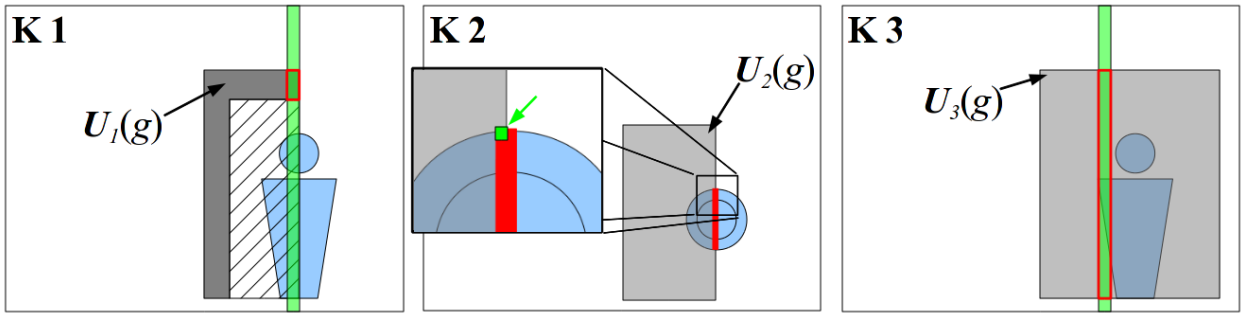


*Abbildung 2.14: Verifikation der Verletzung nicht-sichtbarer Volumina bzw. der Kontaktpixelpaare anhand der Situation aus Abbildung 2.10 (θ_g = 1). Die Darstellung zeigt parallel-projizierte Ansichten der drei eingezeichneten Kameras (**K**amera **1** bis **3**) mit den jeweils in Dunkelgrau (zum Teil transparent) dargestellten Verdeckungsbereichen $\boldsymbol{U}_c(g)$. In **K2** ist ein vergrößerter Ausschnitt der Kontaktpixelpaare mit Umgebung zu sehen. In Hellgrün (mit Pfeil) ist in der Vergrößerung der beispielhaft ausgewählte Pixel $\boldsymbol{p}_u$ in $\boldsymbol{U}_2(g)$ markiert. In **K1** und **K3** ist die jeweils zugehörige Epipolarmenge $\boldsymbol{E}_1(2, \boldsymbol{p}_u)$ und $\boldsymbol{E}_3(2, \boldsymbol{p}_u)$ in hellgrün (halbtransparent) eingezeichnet. Es sind jeweils mehrere Pixel aus $\boldsymbol{U}_1(g)$ und $\boldsymbol{U}_3(g)$ Teil von $\boldsymbol{E}_1(2, \boldsymbol{p}_u)$ bzw. $\boldsymbol{E}_3(2, \boldsymbol{p}_u)$. Demnach ist die Projektion von $\boldsymbol{p}_u$ in zwei weiteren Kameras verdeckt, was gleich der minimalen Anzahl von θ_g = 1 Kamera ist. Daher ist $\boldsymbol{U}_2(g)$ mit der Markierung Objekt zu füllen.*

Die Komplexität dieser Methode liegt bei $O(PnC_g^2)$ mit P als durchschnittliche Anzahl an Kontaktpixelpaaren pro Kamera und $n=\sqrt{WH}$ wie oben, da für jedes Paar in jeder Kamera eine Epipolarmenge in jeweils allen anderen Kameras betrachtet werden muss.

Auflösung verdeckter Pixel

Der Szenenbildrekonstruktion und Behandlung nicht sichtbarer Volumina nachfolgend ist eine Umwandlung von verdeckten Pixeln aus $\boldsymbol{U}_c(g)$ in die Markierung „Frei“ im Szenenbild möglich, falls zulässig. Damit wird im Kollisionstest die Detektion von einer zusätzlichen Zahl von nicht kollisionsbehafteten Konfigurationen ermöglicht, bzw. die Messwerte der Distanzberechnungen verbessert. Da diese Auflösung nach der Behandlung nicht-sichtbarer Volumina erfolgt, gilt im Folgenden die Semantik von β_g (siehe hierzu Abschnitt 2.3.4).

Hierzu wird zu jeder Kamera c für jeden Pixel $\boldsymbol{p}$ in $\boldsymbol{U}_c(g)$ überprüft, ob er die Projektion eines unbekannten Objekts $\boldsymbol{H}_l$ sein kann, also ob es ein $\boldsymbol{P}_j$ gibt, welches Teil eines $\boldsymbol{H}_l$ ist unter Maßgabe des Parameters β_g. Falls nicht, wird dieser Pixel in Freiraum umgewandelt. Er wird im Fall, dass er die Projektion eines unbekannten Objekts sein könnte, jedoch nicht in die Markierung Objekt umgewandelt, da sonst aufgrund der Konstruktion des Kollisionstests zusätzliche Kollisionen erzeugt würden, wie z.B. in

Abbildung 2.8 die getestete Roboterposition der Klasse *B*. Zur Entscheidung bezüglich der Umwandlung wird folgende Bedingung für die Kamera *c* überprüft:

$$\left|\left(i \middle| i \in \boldsymbol{C}_g, i \neq c, \exists\, \boldsymbol{p}_e \in \boldsymbol{E}_i(c, \boldsymbol{p}), \boldsymbol{p}_e \in \boldsymbol{O}_i(g)\right)\right| \geq \left|\left(C_g - 1\right) - \left(\beta_g - 1\right) = \left(C_g - \beta_g\right)\right| \qquad (2.19)$$

In Worten: Die Anzahl der Kameras *i* exklusive *c* in denen die Epipolarmenge von $\boldsymbol{p}$ die unverdeckten Teile eines projizierten unbekannten Objekts $\boldsymbol{H}_l$ schneidet ist größergleich $C_g - \beta_g$. Dies entspricht der Mindestanzahl an Kameras mit Objektpixelschnitten bei der Kollisionsdetektion. In der Gleichung steht „−1“ bei C_g und β_g , da die aktuelle Kamera *c* nicht untersucht wird und $\boldsymbol{p}$ in dieser Kamera dieser Kamera schon Teil der verdeckten Pixel ist. Falls die Bedingung aus 2.19 nicht zutrifft, wird das jeweilige Pixel von der Markierung „Verdeckt“ zu „Frei“ gewandelt. Dieser Test stellt sicher, dass diese Umwandlung nicht stattfindet, falls ein $\boldsymbol{P}_j$ in einem $\boldsymbol{H}_l$ existiert, welches auf $\boldsymbol{p}_u$ projiziert, denn dann muss es in den Epipolarmengen Objektpixel in einer ausreichenden Anzahl von Kameras geben. Dies ist eine Konsequenz aus β_g und somit der Behandlung nicht-sichtbaren Volumina. Diese Behandlung muss daher zwingend vor dieser Auflösung stattfinden (siehe hierzu auch Abbildung 2.26 auf Seite 73).

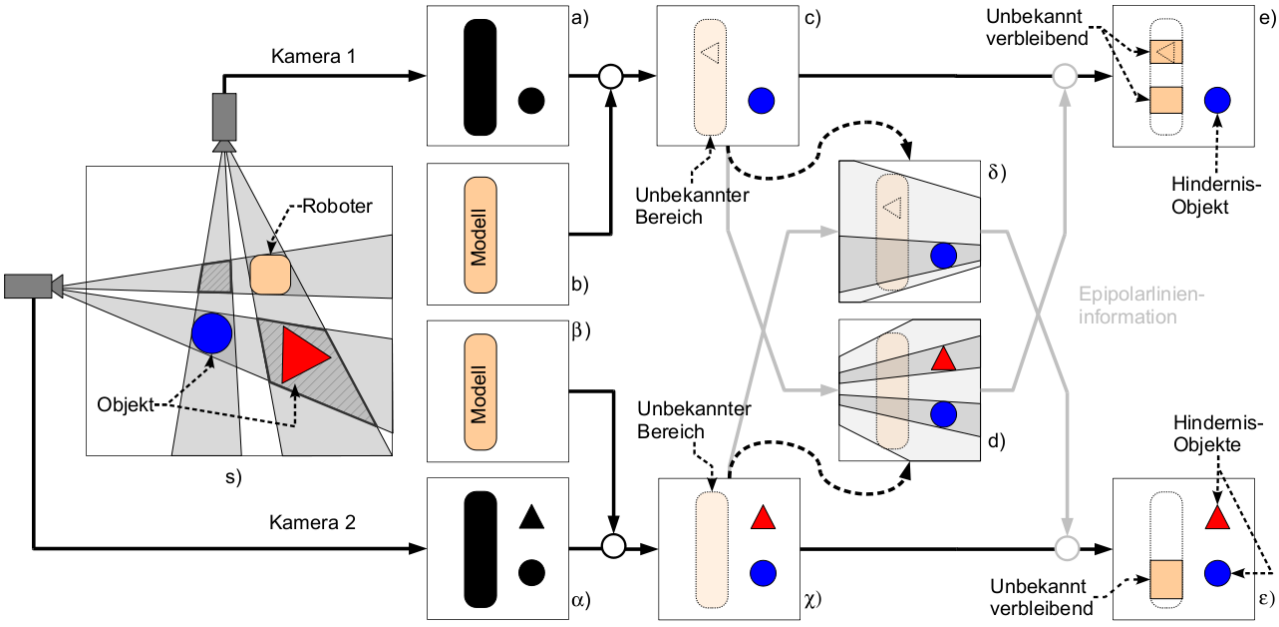


Abbildung 2.15: Auflösung verdeckter Bereiche zu Freiraum in der schematischen Übersicht für eine Beispielsituation (β_g ist 1) ohne Behandlung nicht-sichtbarer Volumina.

In Abbildung 2.15 ist die oben erläuterte Auflösung von verdeckten Pixeln für eine 2D-Beispielsituation mit den resultierenden Szenenbildern dargestellt. Die schematisch reduzierte Szene (Abb. 2.15s) wird durch die Kameras 1 und 2 zu den Differenzbildern Abb. 2.15a und 2.15α verarbeitet. In diesem Fall ist keine Behandlung von nicht sichtbaren Volumina notwendig, da es keine Kontaktpixelpaare gibt. Die Menge der Verdeckungspixel $\boldsymbol{U}_c(g)$ besteht in diesem Szenario lediglich aus dem Roboter, der über sein Modell in die

Kamerasichten abgebildet wird (Abbildung 2.15b und 2.15β)). Diese Modellbilder werden über die Abbildungsvorschrift aus Kapitel 2.3.2 mit den Differenzbildern zu Szenenbildern fusioniert (Abb. 2.15c und 2.15χ). In diese Szenenbilder werden nun die Epipolarmengen aller unbekannten Pixel der jeweils anderen Kamera auf Schnitt mit Objekten untersucht, hier dargestellt in Abb. 2.15d und 2.15δ. Abb. 2.15d liegt also Abb. 2.15χ zugrunde. Die Epipolarmengen der unbekannten Pixel aus $\boldsymbol{U}_l(g)$ (Abb. 2.15c) sind in transparentem Hellgrau eingetragen. Die Teilmenge der Epipolarmengen, die einen Schnitt mit einem detektierten Objekt aufweisen ist transparent dunkelgrau dargestellt. Die Schnitte in Abb. 2.15d führen dazu, dass die unbekannten Pixel in Abb. 2.15c die Markierung „Verdeckt“ behalten, während alle anderen Pixel in „Frei“ umgewandelt werden. Analog gilt dies für Abb. 2.15δ, es resultieren die Szenenbilder in Abb. 2.15e und ε.

Der Rechenaufwand für den oben dargestellten Algorithmus entspricht der Speicherkomplexität der Epipolarmengen: $O(n^3 C_g^2)$. Die unter realistischen Bedingungen benötigten Laufzeiten und die durch die Auflösung der verdeckten Pixel erzielten Verbesserungen des Kollisionstests werden in Kapitel 2.6 aufgeführt.

2.3.6 Kamerapositionierung und Konstruktion der Kameragruppen

Das Ziel der Kamerapositionierung und Konstruktion von Kameragruppen ist es, mit wenigen Kameras den Arbeitsraum möglichst gut zu überwachen. Gut kann beispielsweise definiert werden über die Größe des überwachten Bereiches, die Anzahl und Größe der nicht sichtbaren Bereiche, ein niedriges θ_g und die räumliche Auflösung (was die minimal erkannte Objektgröße impliziert). Weitere Optimierungsmöglichkeiten bestehen in der Betrachtung von typischen Bewegungsszenarien mit gegebener Ortsverteilung für die unbekannten Objekte und den Roboter; hier sollten Kameras so platziert werden, dass sie in Ebenen liegen, die eine Trennung von unbekannten Objekten und Roboter in möglichst vielen Positionen in der Darstellungsebene ermöglichen. Da die Optimierung der Kamerapositionierung nicht Teil dieser Arbeit ist, sollen im Folgenden lediglich einige Hinweise ohne Anspruch auf Vollständigkeit und tiefer gehende Untersuchungen gegeben werden, die vor allem das gewählte θ_g beeinflussen, bzw. die durch das gewählte θ_g entstehenden nicht sichtbaren Bereiche reduzieren.

Für die Positionierung der Kameras sind unter anderem die statischen bekannten Objekte von Bedeutung, die Verdeckungen erzeugen können. In Abbildung 2.16 ist links die zweidimensionale Parallel-Projektion der Situation aus Abbildung 2.11 rechts dargestellt. Hierbei entstehen einige Volumina, die einfach und auch zweifach verdeckt sind. Abhilfe kann geschaffen werden, indem die Kameras so positioniert werden, dass der überblickte Arbeitsraum bis zu den verdeckenden Objekten möglichst groß wird, was durch die Repositionierung der Kameras in Abbildung 2.16 rechts erreicht wird. Dann ist es

möglich, ein Überwachungsvolumen $\boldsymbol{A}_g$ so zu definieren, dass die Sicht der Kameras bis an die Verdeckungen heran reicht (und diese deshalb nicht als Verdeckungspixel aufgenommen werden müssen). Zwei weitere Überwachungsvolumina $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ erweitern den gesamten überwachten Bereich zu der ursprünglichen Größe (links). Diese Aufteilung führt im schlimmsten Fall zu einer Verdreifachung des Rechenaufwands für Szenenbildfusion, Nachbearbeitungsalgorithmen und Kollisionstest (Der Aufwand wächst proportional zur Anzahl der Kameragruppen). Diesem Aufwand muss daher der Nutzen gegengerechnet werden.

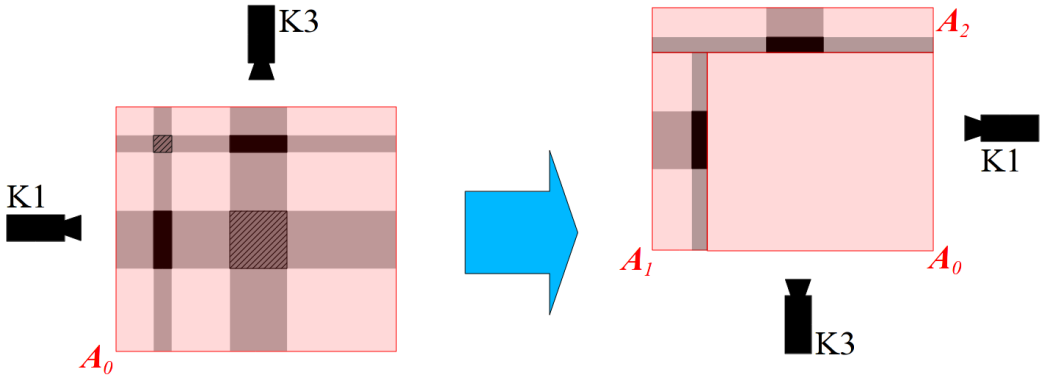


Abbildung 2.16: Reduktion verdeckter und nicht sichtbarer Volumina (für θ_g = 1) durch Kamera-positionierung und Konstruktion geeigneter Kameragruppen mit zugehörigen Überwachungsvolumina. Diese Abbildung referiert auf die Situation aus Abbildung 2.11 rechts. ***Links*** *abgebildet ist die Draufsicht aus Kamera 2 auf* ***eine*** *Kameragruppe mit Überwachungsvolumen $\boldsymbol{A}_0$ (transparente rote Fläche, rot umrandet). Die schraffierten Flächen markieren die nicht-sichtbaren Volumina $\overline{\boldsymbol{V}}_g$.* ***Rechts*** *abgebildet ist eine Umpositionierung der Kameras und die Konstruktion von drei separaten Überwachungsvolumina $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, $\boldsymbol{A}_2$. Mit dieser Konstruktion werden die nicht-sichtbaren Volumina in $\boldsymbol{A}_0$ eleminiert. Die Rückprojektions-volumina verdeckter Pixel reduzieren sich ebenfalls deutlich und sind auf die Überwachungsvolumina $\boldsymbol{A}_1$ und $\boldsymbol{A}_2$ beschränkt.*

Als Grundregeln aus diesem Beispiel lassen sich also ableiten:

- Maximierung des sichtbaren Arbeitsraumvolumens bis zum ersten statischen bekannten Objekt, das Verdeckungen auslöst durch Umpositionierung der Kameras
- Aufteilung der Überwachungsvolumina (Rechenaufwand berücksichtigen!)

Ein weiterer Aspekt der Kamerapositionierung betrifft besondere Arten von Verdeckungen, die an den Rand des überwachten Bereiches „heranragen“. Hierbei kann es zu einer Verbindung mit großen unüberwachten Volumina kommen, die so das erforderliche θ_g massiv erhöhen und damit die Verfügbarkeit des Systems stark einschränken.

In Abbildung 2.17 links ist eine solche Situation und eine Möglichkeit zur Behebung des Problems dargestellt. Grundsätzlich ist es problematisch, wenn Volumina aus $\overline{\boldsymbol{V}}_g$ an Bereiche grenzen, die außerhalb des Sichtbereichs aller Kameras liegen (welche daher automatisch zu $\overline{\boldsymbol{V}}_g$ gehören, also explizit $\boldsymbol{P}_0 \in \overline{\boldsymbol{V}}_g$, siehe Kapitel 2.3.1). Diese Bereiche

verschmelzen dann miteinander und können leicht so groß werden, dass ganze Objekte hinein passen und somit die Bedingung von θ_g nicht erfüllt werden kann.

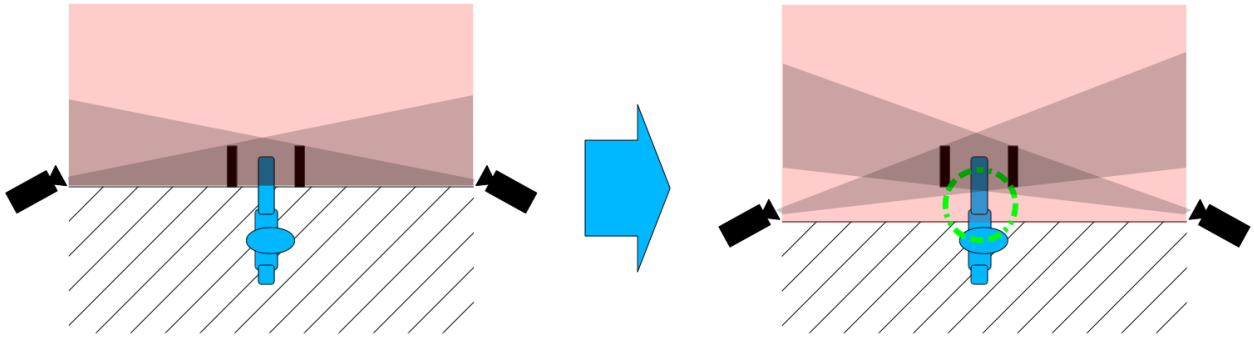


Abbildung 2.17: Problematik von verdeckten Volumina (transparent grau und dunkelgrau), die an den Rand des Überwachungsvolumens (transparent rot) heran ragen. Links müsste $\theta_g = 2$ gelten, da sich ein unbekanntes Objekt (blau) im unüberwachten Bereich (schraffiert) aufhalten kann und in den überwachten Bereich hineinragt, hierbei jedoch keinen Bereich berührt, der durch mindestens eine Kamera sichtbar wäre. Ein θ_g von 2 bedeutet jedoch bei insgesamt zwei Kameras für die Kameragruppe, dass das System keine Aussage über Kollisionsfreiheit machen kann. Rechts wäre hingegen durch Verschiebung der Kamerapositionen eine Reduktion von θ_g auf 1 möglich, da Objekte immer aus den zweifach verdeckten Bereichen herausragen (bei entsprechender Mindestgröße).

Eine mögliche Behebung besteht in der Verschiebung der Kameras und damit Erweiterung des sichtbaren Bereiches, sodass Volumina aus $\overline{\boldsymbol{V}_g}$ vollständig von überwachtem Volumen umschlossen sind (mit einem ausreichenden Abstand zu dessen Rand), so wie in Abbildung 2.17 rechts dargestellt.

Eine Alternative zu den Verschiebungen der Kameras ist die Definition zusätzlicher Kameragruppen, falls einzelne Kameras aus der gegebenen Gruppe in den Bereich hineinsehen können. Diese Kameragruppen sind dann im Allgemeinen Untermengen der ursprünglichen Kameragruppe mit angepasstem θ_g.

Eine weitere Alternative sind bauliche Maßnahmen wie Wände, die ein Eindringen von Objekten an den kritischen Stellen verhindern. Diese Lösung ist jedoch weniger wünschenswert, da die Zielsetzung dieser Arbeit im Überwinden der Einzäunung von Robotersystemen besteht.

2.4 Bildbasierte Multi-Kamera-Distanzberechnung

Nach der Behandlung von nicht-sichtbaren Bereichen (optional mit Epipolarmengentests) und der optionalen Reduktion der verdeckten Pixel schließt sich im einfachen Fall der Kollisionstest mit einem projizierten Testvolumen $\boldsymbol{T}(r)$ an, wie in Kapitel 2.3.3 beschrieben. Alternativ dazu ist eine Berechnung von Distanzen der Testvolumina $\boldsymbol{T}(r)$ zu den detektierten unbekannten Objekten möglich und wird im Folgenden beschrieben. Diese basiert auf einer Fusion von Distanzberechnungen in den einzelnen Kamerabildern. Die Inhalte dieses Kapitels wurden zum Teil in [Kuhn06] veröffentlicht.

2.4.1 Einzelkamera-Distanzberechnung

Die schnelle bildbasierte Berechnung von Distanzen beruht auf einem Verfahren, welches in [Henrich08b] beschrieben wird. Hierbei wird $\boldsymbol{T}(r)$ in Form von Kugelmodellen approximiert, also einer Menge von Kugeln $\boldsymbol{K}(r)$, die im Falle des Roboters durch die Roboterkonfiguration bestimmt im Raum angeordnet sind. Durch die Kalibrierung ist für jedes Pixel eine Gerade durch seinen Mittelpunkt und den Fokuspunkt der Kamera bekannt.

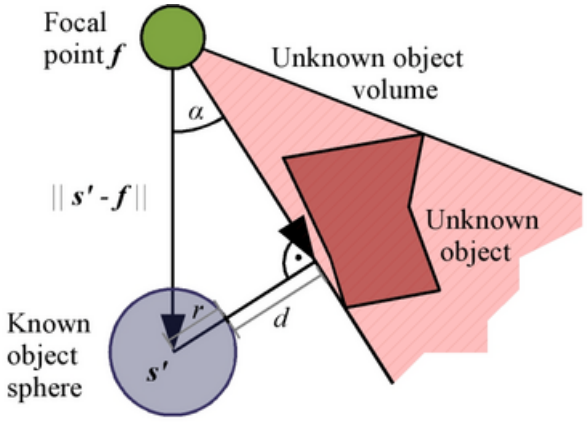


Abbildung 2.18: Schematische Darstellung der Einzelkameradistanzberechnung mit Kugelmodellen (Bild aus [Henrich08b]): Alle Pixel aus der Menge der verdeckten Pixel $\boldsymbol{U}_c(g)$ oder Objektpixel $\boldsymbol{O}_c(g)$ werden als Strahlen in den Raum projiziert (beispielsweise durch den Pixelmittelpunkt). Für jeden Strahl wird die minimale Distanz d zu einer Kugel mit Mittelpunkt $\boldsymbol{s}$ und Radius r bestimmt. Hierzu muss auch der Fokuspunkt f der Kamera aus der Kalibrierung bekannt sein.

Über eine einfache geometrische Beziehung kann damit die minimal mögliche Distanz (schlimmster Fall) zwischen einer Kugel und der Gerade bestimmt werden und um den maximal möglichen Fehler korrigiert werden, der durch die Wahl der Pixelmittelpunkte entsteht. Somit lässt sich für jede Kamera c aus $\boldsymbol{C}_g$ der minimale Distanzwert $u_c(g)$ aller Pixel aus $\boldsymbol{U}_c(g)$ und der minimale Distanzwert $o_c(g)$ aller Pixel aus $\boldsymbol{O}_c(g)$ zu den Kugeln des Modells bestimmen zu:

$$\begin{aligned} o_c(g,r) &= \min_{p \in \boldsymbol{O}_c(g)} \left(\min_{k \in \boldsymbol{K}(r)} \left(d(p,k) \right) \right) \\ u_c(g,r) &= \min_{p \in \boldsymbol{U}_c(g)} \left(\min_{k \in \boldsymbol{K}(r)} \left(d(p,k) \right) \right) \end{aligned} \tag{2.20}$$

mit $d(\boldsymbol{p}, k)$ als Distanzfunktion von Pixel $\boldsymbol{p}$ zu Kugel k aus $\boldsymbol{K}(r)$. Die Distanzberechnung ist schematisch in Abbildung 2.18 dargestellt.

2.4.2 Multi-Kamera-Distanzberechnung

In diesem Abschnitt wird die Fusion der in den einzelnen Kameras berechneten Distanzen (siehe voriges Kapitel) unter Beachtung von Verdeckungen und Berücksichtigung des Parameters β_g beschrieben. Prinzipiell muss bei der Fusion auch die Distanz zum Rand des überwachten Bereiches betrachtet werden, wenn sich dort Hindernisse aufhalten können. Dies wird jedoch bei den folgenden Betrachtungen zunächst aussen vor gelassen

und zum Ende des Abschnitts bei der Bestimmung der Gesamtdistanz dargestellt.

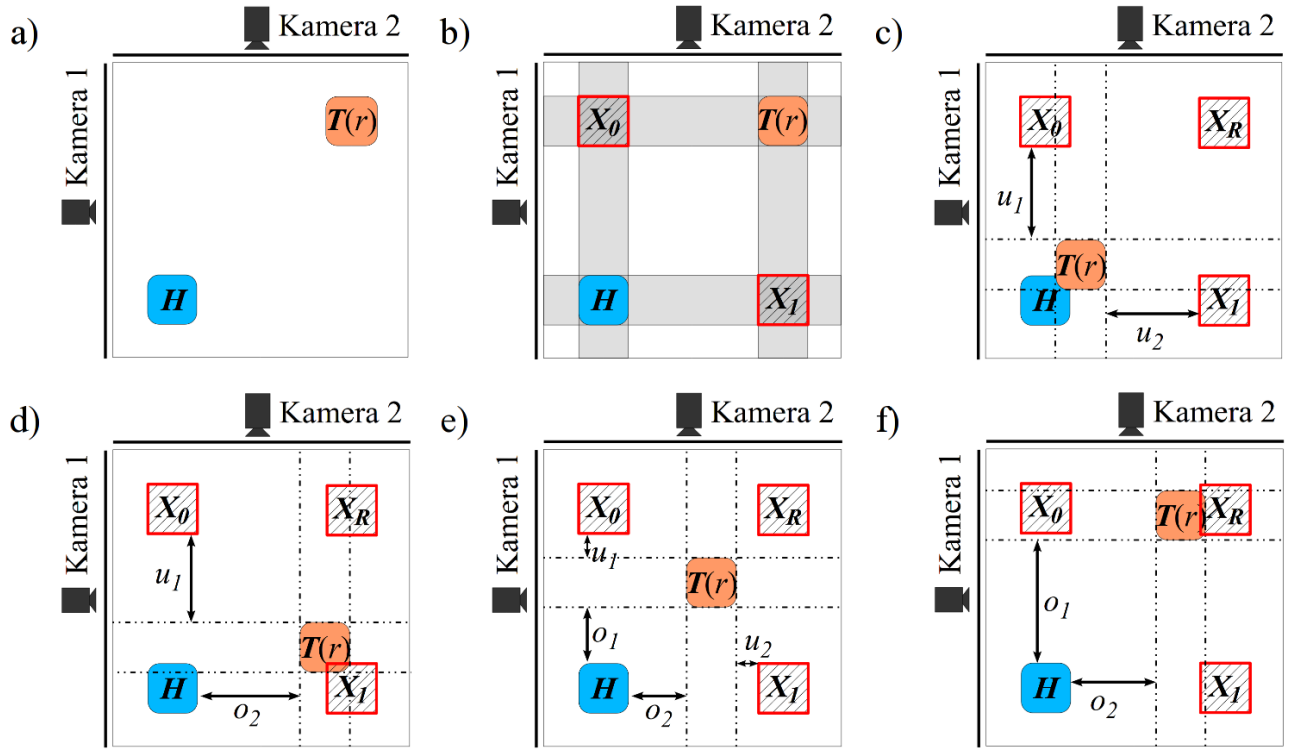


*Abbildung 2.19: Draufsicht einer 2D-Beispielszene für die Multi-Kamera-Distanzfusion mit verschiedenen Kollisionstestfällen c)-f). Für die Szene gilt $\beta_g = 1$: **a)** Szene mit Robotervolumen **T**(r) (dynamische Verdeckung), Objekt **H** und parallel projizierenden Kameras 1 und 2. **b)** Rückprojektion (hellgrau) von $\mathbf{U}_1(g)$ bzw. $\mathbf{U}_2(g)$ und $\mathbf{O}_1(g)$ bzw. $\mathbf{O}_2(g)$ in die Szene mit den nicht-sichtbaren Schnittmengen $\mathbf{X}_0$ und $\mathbf{X}_1$ (schraffiert), es wird keine Auflösung verdeckter Pixel berechnet, bzw. es sei in diesem Fall keine möglich. $\mathbf{X}_0$ und $\mathbf{X}_1$ können daher wegen $\beta_g = 1$ prinzipiell Objekte enthalten. **c)** Kollisionstest mit getestetem Robotervolumen **T**(r) (zusätzlich eingezeichnete Schnittmenge $\mathbf{X}_R$ der rückprojizierten Pixel $\mathbf{U}_1(g)$ bzw. $\mathbf{U}_2(g)$ und in diesem Fall mit resultierender Distanz $d_{min}(g) = d(g) = 0 = max(min(o_1,u_1), min(o_2,u_2))$ wegen $o_1 = o_2 = 0$. **d)** Kollisionstest mit Szenario „Schnitt mit möglichem Objekt“. Die Distanz $d_{min}(g)$ ergibt sich zu 0 wegen $o_1 = u_2 = 0$ und $s = o_1 = 0$. **e)** Kollisionstest ohne Berührung eines kritischen Bereiches (alle Distanzwerte > 0), es sei $o_1 > o_2$ und $u_1 < u_2 < o_2$. Resultierend ist $d_{min}(g) = o_2$, da $s = o_2$. **f)** Kollisionstest mit Berührung des originalen Robotervolumens (visuelle Hülle) **T**(r). Hierbei sind $u_1 = u_2 = 0$. Da $s = o_2$ ergibt sich $d_{min}(g) = o_2$, was der korrekten Distanz zu $\mathbf{X}_0$ entspricht, wo potentiell ein unbekanntes Objekt existiert.*

Konservativ kann angenommen werden, dass verdeckte Bereiche unbekannte Objekte enthalten, womit sich die Distanz pro Kamera c zu dem Minimum aus den Distanzen $u_c(g, r)$ und $o_c(g, r)$ ergibt. Die insgesamt resultierende Minimaldistanz $d_{kons}(g, r)$ zum nächsten Objekt ist dann die maximale Distanz der Minimaldistanzen aller Kameras, da jede Kamera einzeln eine konservativ korrekte Aussage macht:

$$d_{cons}(g,r) = \max_{c \in \mathbf{C}_g} \left(\min \left(o_c(g,r), u_c(g,r) \right) \right) \tag{2.21}$$

Nun ist es jedoch möglich, dass bei allen Einzelkameradistanzberechnungen die Distanz zu verdeckten Bereichen 0 beträgt (beispielhaft in Abbildung 2.19b,f dargestellt).

Das Problem der Nulldistanz gilt insbesondere dann, wenn die verdeckten Pixel nicht mithilfe von Epipolarmengen-Tests aufgelöst wurden. Die Berechnung der Distanz 0 würde

für eine distanzgeregelte Robotergeschwindigkeit im Allgemeinen ein Verfügbarkeitsproblem bedeuten, da der Roboter dann stillstehen müsste. Nun ist dieser Fall jedoch häufig keineswegs mit einer tatsächlichen Hinderniskollision verbunden, was auch Abbildung 2.19f) beispielhaft zeigt.

Hierauf kann begegnet werden, indem der Parameter β_g Anwendung findet. Zur Rekapitulation: Dieser beschreibt die maximale Anzahl an Kameras, in denen Teilvolumen $\boldsymbol{P}_j$ verdeckt sein kann. Betrachtet man nun lediglich die Abstände $o_c(g, r)$, so können darunter im schlimmsten Fall β_g Abstände sein, die zu groß sind, da ein Objekt den entsprechenden Kameras vollständig verdeckt ist. Die Distanz zu den unbekannten Objekten hat daher eine untere Schranke, wenn die β_g größten Distanzen ignoriert werden. Sei $\boldsymbol{S}_g$ die aufsteigend sortierte Menge der $o_c(g, r)$, so sei die sortierte Distanz $d_{sort}(g, r)$ das $(C_g - \beta_g)$-te Element aus $\boldsymbol{S}_g$. Dann ergibt sich die minimale Distanz $d_{min}(g, r)$ für die Kameragruppe g zu:

$$d_{min}(g, r) = \max\left(d_{cons}(g, r), d_{sort}(g, r)\right) \tag{2.22}$$

Dies gilt, da jede einzelne Distanz $d_{cons}(g, r)$ und $d_{sort}(g, r)$ sicher unter der realen Minimaldistanz $d_{realmin}(g, r)$ von allen unbekannten Objekten $\boldsymbol{H}_l$ in $\boldsymbol{A}_g$ zum Testvolumen $\boldsymbol{T}(r)$ liegt. Für die konservative Distanz $d_{cons}(g, r)$ ergibt sich dies offensichtlich, da alle Information ausser dem detektierten Freiraum in die Berechnung miteinbezogen wird. Für die sortierte Distanz $d_{sort}(g, r)$ wird im Folgenden ein Beweis skizziert:

Wenn Hindernisse in $\boldsymbol{A}_g$ existieren, so gibt es mindestens einen Punkt auf diesen Hindernissen mit minimaler Distanz $d_{realmin}(g, r)$ zum Testvolumen $\boldsymbol{T}(r)$. Dann gibt es auch mindestens ein $\boldsymbol{P}_j$, welches diese minimale Distanz zum Roboter besitzt. Für ein solches $\boldsymbol{P}_j$ gilt nach der β_g-Bedingung, dass es in mindestens $(C_g - \beta_g)$ Kameras sichtbar ist und Teil von $\boldsymbol{O}_c(g)$. Da jede im Bild berechnete Distanz kleinergleich der realen Distanz $d_{realmin}(g, r)$ ist, existieren somit $(C_g - \beta_g)$ Distanzen, welche kleinergleich der realen Distanz $d_{realmin}(g, r)$ sind. Für die übrigen β_g berechneten Distanzen können zwei Fälle unterschieden werden: Sie sind größer als $d_{realmin}(g, r)$ oder sie sind kleinergleich $d_{realmin}(g, r)$. In beiden Fällen wird nach der Sortierung eine Distanz $d_{sort}(g, r)$ gewählt, welche kleinergleich $d_{realmin}(g, r)$ ist.

Verbesserung der Distanzberechnung durch Multi-Kugel-Distanztest

Die auf Kugeln basierten Testvolumina $\boldsymbol{T}(r)$ bieten die Möglichkeit die Distanzberechnung zu minimalen Mehrkosten in bestimmten Fällen zu verbessern. Dazu wird zu jeder Kugel k_i in $\boldsymbol{K}(r)$ (siehe Abbildung 2.20) einzeln die Distanz bestimmt und anschließend über alle diese Distanzen minimiert. Dies verbessert die Distanzberechnung insbesondere in Fällen, wie sie in Abbildung 2.20 dargestellt sind. Dies geschieht analog zur Verbesserung des Kollisionstests aus Kapitel 2.3.3.

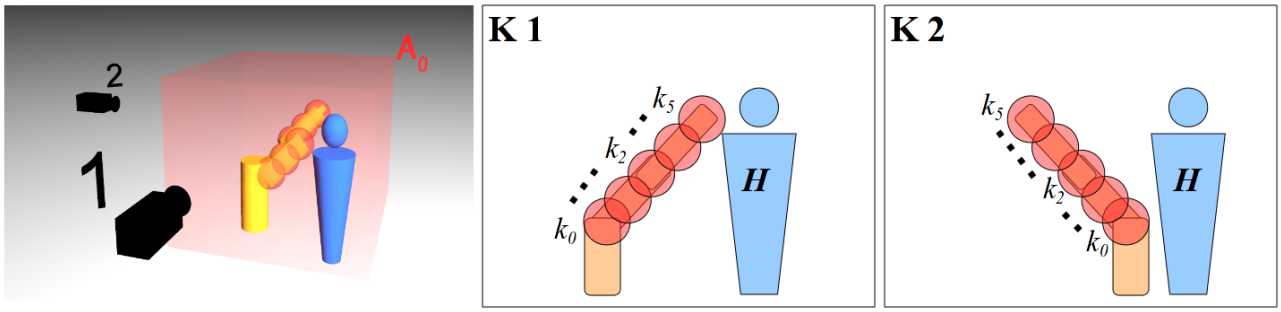


*Abbildung 2.20: Multi-Kugel-Distanztest (sei $\beta_g = 0$): **Links:** Beispielszene mit Roboter in Gelb, **T**(r) als Kugeln um den Roboterkörper in transparentem Rot, unbekanntes Objekt in Blau, Überwachungsvolumen $\boldsymbol{A}_0$ als transparenter roter Quader, Kameras 1 und 2 in Schwarz. **Mitte:** Sicht von Kamera 1. Das unbekannte Objekt **H** berührt Kugel k_5. Zu k_0 ist die Distanz maximal. **Rechts:** Sicht von Kamera 2. Die minimale Distanz zum Objekt hat Kugel k_0, die maximale Kugel k_5. In beiden Kameras hat die Kugel k_2 die mittlere, resultierende Gesamtdistanz d_{is}.*

Wie in Abbildung 2.20 schematisch dargestellt, kann es vorkommen, dass eine bestimmte Kugel in einer Kamera eine große Distanz und in einer anderen Kamera eine kleine Distanz besitzt. Wird nun für jede Kamera zunächst die minimale Distanz des unbekannten Objekts zu allen Kugeln berechnet (wie in Gleichung 2.20), resultiert in beiden Kameras die minimale Distanz, wodurch auch die fusionierte Gesamtdistanz minimal wird, obwohl eine größere Distanz berechenbar wäre. Die größere Distanz erhöht die Verfügbarkeit des Systems und ist daher wünschenswert. Die größere Distanz $d_{is}(g, r)$ (**i**ndividual **s**pheres) durch Multi-Kugel-Distanztest ergibt sich zu:

$$d_{is}(g,r) = \min_{k \in \boldsymbol{K}(r)} \left(\max \left(\max_{c \in \boldsymbol{C}_g} \left(\min \left(o_c(g,k), u_c(g,k) \right) \right), s(g,k) \right) \right)$$
$$\text{mit } o_c(g,k) = \left(\min_{p \in \boldsymbol{O}_c(g)} d(p,k) \right) \text{ und } u_c(g,k) = \left(\min_{p \in \boldsymbol{U}_c(g)} d(p,k) \right) \qquad (2.23)$$

mit $s(g, k)$ als $(C_g - \beta_g)$-tem Element der Menge aufsteigend sortierter Distanzen $o_c(g, k)$. Die hiermit erreichten Verbesserungen werden im Kapitel 2.6 dargestellt.

Mehrere Kameragruppen bzw. Überwachungsbereiche

Bei mehreren Kameragruppen ist als Gesamtdistanz $d_{min}(r)$ das Minimum der für die einzelnen Gruppen berechneten Distanzen zu nehmen, da diese im Allgemeinen nicht das gleiche Volumen $\boldsymbol{A}_g$ überwachen:

$$d_{min}(r) = \min_{g \in \{1 \ldots G\}} \left(d_{min}(g,r) \right) \text{ oder } d_{min}(r) = \min_{g \in \{1 \ldots G\}} \left(d_{is}(g,r) \right) \qquad (2.24)$$

Die linke Seite gilt für Tests des gesamten Robotervolumens und die rechte für Multi-Kugel-Distanztests (siehe oben).

Randdistanz

Zur Berechnung der insgesamt minimalen Distanz $d_{min}(g, r)$ muss noch die Distanz $d_{border}(r)$ des Roboterstestvolumens zum Rand des überwachten Bereiches miteinbezogen werden, falls sich hier nicht überwachte Hindernisse befinden können. Dabei sind zunächst

alle Ränder der Kameragruppen gemeint, die für die Berechnung fusioniert werden und welche nicht innerhalb des durch diese Kameragruppen überwachten Bereichs liegen. Weiterhin können Ränder ausgeschlossen werden, an denen sich kein Hindernis befinden kann, welches überwacht werden muss, da z.B. durch bauliche Maßnahmen (Zäune) keine Hindernisse durch diesen Rand in die überwachte Zone eindringen können. Insgesamt ist $d_{border}(r)$ somit nicht pro Kameragruppe zu berechnen sondern beschreibt die Distanz des Testvolumens $\boldsymbol{T}(r)$ zu den relevanten Rändern des überwachten Bereichs. Die resultierende Distanz $d_{res}(r)$ ergibt sich dann zu dem Minimum aus der bildbasiert berechneten Distanz $d_{min}(r)$ und der modellbasiert berechneten Distanz $d_{border}(r)$:

$$d_{res}(r) = \min\left(d_{min}(r), d_{border}(r)\right) \tag{2.25}$$

Eine konservative Abschätzung für $d_{border}(r)$ wäre das Minimum der Minima des Testvolumens zum Rand der jeweiligen Überwachungsbereiche $\boldsymbol{A}_g$ der einzelnen Kameragruppen. Weitere Ansätze zur Ermittlung von $d_{border}(r)$ können sein: Spezifikation der relevanten Ränder des überwachten Raumes durch den Benutzer und Online-Berechnung der Distanz zu diesen, Spezifikation einer Distanz für den schlimmsten Fall durch den Benutzer (die dann alle berechneten Distanzen begrenzt), Lernen der gesuchten Distanz durch Online-Beobachtung von wo und mit welcher Distanz unbekannte Objekte erscheinen, etc. Die Untersuchung dieser Möglichkeiten war nicht Teil dieser Arbeit.

2.5 Differenzbildberechnung in dynamischen Arbeitszellen

Die Differenzbildberechnung wurde für die in den vorangegangenen Kapiteln erläuterten Algorithmen als ideal angenommen, was heißt, dass alle sichtbaren Teile jedes unbekannten Objekts als Vordergrund erkannt werden und alle anderen Pixel als Hintergrund. In realen Umgebungen beeinflussen eine Vielzahl von Faktoren die Qualität des Differenzbildes. Eine Reihe unterschiedlicher Verfahren wird z.B. in [Radke04] dargestellt. Einfache Varianten der Differenzbilderzeugung erzeugen pro Pixel ein Modell der erwarteten Werte auf Basis einer Normalverteilung mit konstantem Mittelwert, deren Standardabweichung durch das Rauschen der Kamera definiert ist. Hierbei wird die Zugehörigkeit des Pixelwertes des aktuellen Bildes zu Vordergrund bzw. Hintergrund mit Hilfe dieser Verteilung bestimmt. Diese Modelle gehen von einem statischen Bildhintergrund aus. In der Realität und insbesondere in industriellen Fertigungszellen existieren jedoch häufig Bildbereiche, die nicht statisch sind und beispielsweise Bewegungen enthalten. Diese dynamischen Bildhintergründe sollen dennoch sicher als Hintergrund erkannt werden. Die hier eingesetzten, komplexeren Modelle zur Modellierung des Bildhintergrunds sind ein aktives Feld der Forschung. Die Ursachen für Bilddynamik können vielgestaltig sein und sind in der folgenden Liste knapp und ohne Anspruch auf

Vollständigkeit aufgeführt:

- Starre, semi-statische Objektbewegungen
  - Bearbeitung in geschlossenen Werkzeugmaschinen, Montage, Palettierung, Pick-and-Place-Aufgaben im Allgemeinen
- Wiederholte Muster mit klar abgegrenzten Zuständen
  - Blinklichter, Statusmonitor, Maschinenzustände
- Wiederholte Muster ohne abgegrenzte Zustände
  - Flüssigkeiten, Schüttgut, zufällig verteiltes Stückgut auf Förderbändern
  - Schmutz auf Boden und Objekten, sonstige Ablagerungen (z. B. durch Lackierung)
  - Gerätevibrationen
- Chaotische Störungen
  - Atmosphärische Störungen (Dampf, Staub, Rauch, etc.)
  - Funken beim Schleifen und Schweißen, Gasflammen, Feuer
  - Spanender Abtrag, Schnittstücke
  - Beleuchtungsänderungen, z.B. Schatten (kann auch regelmäßig sein)

Im industriellen Umfeld ergibt sich durch die definierte Umgebung spezielle Behandlungsmöglichkeiten für ausgewählte Punkte der obigen Liste, welche in den folgenden Kapiteln dargestellt werden sollen. Für diese Arbeit wird eine Unterteilung der Behandlungsmöglichkeiten vorgenommen, die zu den in den vorherigen Kapiteln entwickelten Konzepten passt und diese zum Teil nutzt: die Behandlung durch Maskierung (Kapitel 2.5.1), durch Referenzbildaktualisierung(Kapitel 2.5.2) und durch die Hintergrundmodellierung(Kapitel 2.5.3).

2.5.1 Maskierung

Das Prinzip der Maskierung dynamischer Hintergründe ist inspiriert von den dynamischen Verdeckungsbereichen: Bildbereiche, für die kein Differenzbild berechnet werden kann, werden durch Deklaration als verdeckte Bereiche behandelt. Dies ist beispielsweise für den Roboter der Fall, dessen Erscheinungsbild in einer beliebigen Konfiguration nicht bekannt ist und kann prinzipiell auf alle nicht-statischen Bildbereiche erweitert werden. Abbildung 2.21 zeigt den Bildausschnitt einer Kamerasicht in die Demonstratorzelle mit markierten Verdeckungsbereichen, die zum Zwecke der Maskierung dynamischer Bildteile verwendet werden.

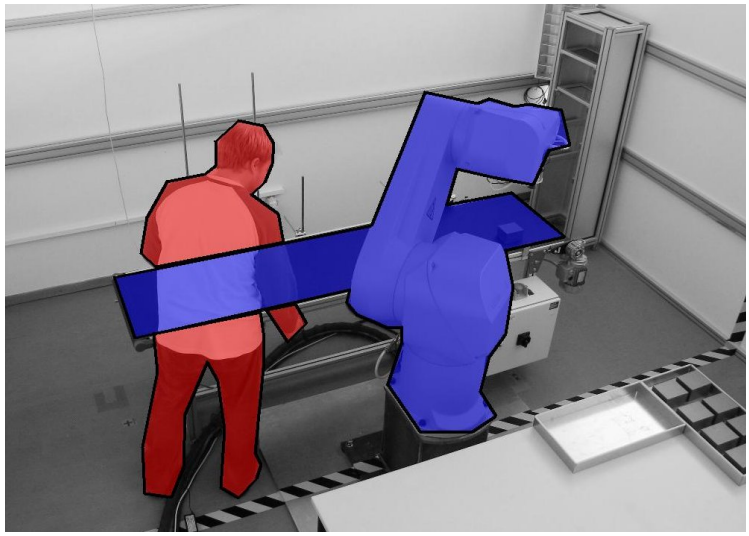


Abbildung 2.21: Verwendung der Menge verdeckter Pixel $\boldsymbol{U}_c(g)$ zur Maskierung dynamischer Bildbereiche. In transparentem Blau ist die Menge $\boldsymbol{U}_c(g)$ markiert, welche den Roboter und ein Fließband für den Werkstücktransport in die bzw. aus der Roboterzelle umfasst. In transparentem Rot ist die Pixelmenge $\boldsymbol{O}_c(g)$ markiert, welche hier einen Arbeiter erfasst, der teilweise durch $\boldsymbol{U}_c(g)$ verdeckt wird.

Während die Maskierung zunächst den Vorteil bietet, mit Methoden des bildbasierten Kollisionstests eine Behandlung jeglicher Bilddynamik zu ermöglichen, entstehen Nachteile durch die zusätzlichen Verdeckungen. Es werden möglicherweise neue, nicht überwachte Volumina erzeugt, wodurch zusätzliche Kollisionen und Pseudoobjekte entstehen können, welche die Verfügbarkeit reduzieren. In Abbildung 2.21 führt beispielsweise die Maskierung des Fließbandes bei dieser Ansicht dazu, dass eine Berührung von $\boldsymbol{O}_c(g)$ und $\boldsymbol{U}_c(g)$ festgestellt wird, weshalb alle blauen Bereiche inklusive Roboterarm mit der Markierung Objekt zu fluten wären (Behandlung nicht-sichtbarer Volumina in Kapitel 2.3.4). Weiterhin kann die Maskierung von Bereichen dem gewählten θ_g widersprechen, sodass die Sicherheit des gesamten Überwachungssystems in Frage gestellt wird oder θ_g in unpraktikable Bereiche erhöht werden muss. In Abbildung 2.21 beispielsweise könnte der blau markierte Bereich im Zusammenspiel mit gewissen Roboter- und Menschpositionierungen zu groß werden und dann den Menschen vollständig verdecken. In Konsequenz ist diese Methode stark abhängig von der Geometrie der Zelle sowie der Position der Kamerasysteme und kann gegebenenfalls nicht angewendet werden, wenn dadurch die Sicherheit gefährdet wird und eine Erhöhung von θ_g nicht möglich ist.

2.5.2 Referenzbildaktualisierung

Die prinzipielle Unterscheidung statischer und dynamischer Hintergründe kann durch eine für die Praxis sehr relevante dritte Klasse erweitert werden, die sogenannten semi-statischen Hintergründe, die aus der dynamischen Klasse abgespaltet werden kann. Während *statische Hintergründe* keine Veränderung der Pixelwerte aufweisen, die nicht durch Kamerarauschen bedingt sind, sind *semi-statische Hintergründe* dadurch gekennzeichnet, dass für längere Zeitintervalle die Pixelwerte als statisch im

vorhergenannten Sinne betrachtet werden können und sich nur zu bestimmten Zeitpunkten diskret verändern. Alle verbleibenden dynamischen Hintergründe fallen dann in den Bereich der *dynamischen Hintergründe*.

Das Anwendungsfeld dieser Kategorisierung ist das weite Spektrum der Pick-und-Place-Aufgaben, also beispielsweise Werkstückbeschickung einer Fräßmaschine, Palettierung und Montage. Hierbei werden starre Körper bekannter Geometrie durch den Roboter zwischen bekannten Positionen transportiert und dort jeweils aufgenommen oder abgelegt. Für diese Anwendungen kann durch den im Folgenden beschriebenen Algorithmus eine Verbesserung der Verfügbarkeit erreicht werden, indem Maskierungen der entsprechenden Bildbereiche vermieden werden.

Die Basis für den Algorithmus bilden Modelle der transportierten Objekte sowie Signale des Roboterprogramms, die Informationen darüber liefern, wann die Objekte abgelegt oder aufgenommen werden. Die beiden Fälle müssen durch die Konstruktion des weiter unten dargestellten Algorithmus nicht unterschieden werden. Die Modelle der transportierten Objekte liegen dem System vor, da sie für den Kollisionstest relevant sind und zwar sowohl für den bildbasierten Kollisionstest als auch für den modellbasierten Kollisionstest, der auf Kollisionen mit bekannten Objekten testet (Zur Erinnerung: der modellbasierte Kollisionstest wird in der Aufgabenstellung als gegeben vorausgesetzt, siehe Kapitel 2.2).

Für die Dauer des Transports müssen die Werkstücke Teil des Robotermodells sein, um die korrekte Funktion des Kollisionstests zu gewährleisten. Sie werden daher Teil des maskierten Bereiches im Bild. Daher sind lediglich die Aufnahme- und Ablagebereiche von einer Veränderung des Bildhintergrunds betroffen, der als semi-statisch bezeichnet werden kann und nicht maskiert werden soll.

Die Grundidee des Algorithmus besteht darin, im Fall einer Aufnahme oder Ablage eines Werkstücks das Modell des Objekts im Robotergreifer zu verwenden, um über die Projektionsfunktion der Kameras und die zum jeweiligen Zeitpunkt bekannte Roboterkonfiguration r einen Bereich von zu aktualisierenden Pixeln $\boldsymbol{R}_c(g)$ zu bestimmen und das jeweilige Pixelmodell zu aktualisieren, falls nötig und möglich. Nötig ist eine Aktualisierung dann, wenn nicht dynamische ($\boldsymbol{U}_{c,dynamic}(g)$) oder statische ($\boldsymbol{U}_{c,static}(g)$) bekannte Verdeckungen den entsprechenden Bereich verdecken (siehe Abbildung 2.22) und möglich ist sie dann, wenn der Bereich im Rahmen der Szenenbildkonstruktion zu Freiraum umgewandelt wurde. Die Mengen $\boldsymbol{U}_{c,dynamic}(g)$ und $\boldsymbol{U}_{c,static}(g)$ sind Teilmengen von $\boldsymbol{U}_c(g)$.

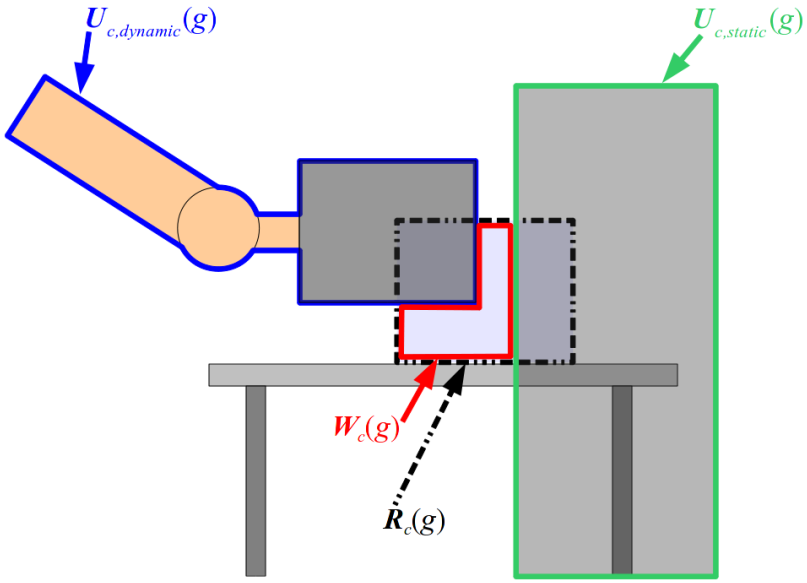


Abbildung 2.22: Subtraktion von statischen ($\boldsymbol{U}_{c,static}(g)$) und dynamischen ($\boldsymbol{U}_{c,dynamic}(g)$) Mengen verdeckter Pixel von der Menge zu aktualisierender Pixel $\boldsymbol{R}_c(g)$. Die verbleibenden Pixel $\boldsymbol{W}_c(g)$ werden nach Abzug der durch andere unbekannte Objekte verursachten verbleiben Pixelmengen zur Aktualisierung des Referenzbildes verwendet. [Gecks04][Gecks05]

Zur Bestimmung der zu aktualisierenden Pixelmengen wird ein Signal des Roboterprogramms benötigt, welches den Objekttyp und dessen Position bei der Aufnahme bzw. Ablage beschreibt (Abbildung 2.23). Die Position des Objekts ist durch die Roboterkonfiguration *r* bestimmt.

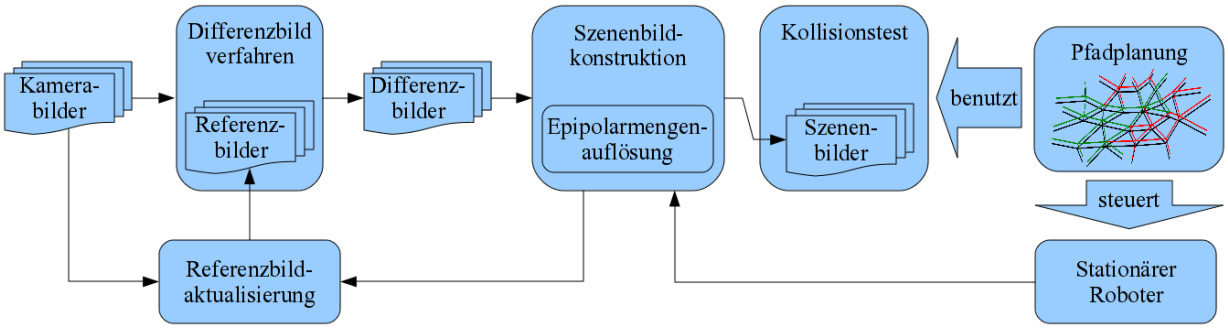


Abbildung 2.23: Erweiterung des Systemdatenflusses (siehe auch Abbildung 1.8) für die Referenzbildaktualisierung. Die Erweiterung geschieht durch Rückkopplung von Steuerinformationen des Roboterprogramms und Aktualisierung der Referenzbilder im Differenzbildverfahren mit Informationen aus der Szenenbildkonstruktion.

Diese Daten werden dann verwendet, um die Projektion des aufgenommenen bzw. abgelegten Objekts zu bestimmen und diese Pixel der Aktualisierungsmenge $\boldsymbol{R}_c(g)$, welche initial leer ist, zuzuschlagen (siehe Algorithmus 2.1 Zeile (2-4)) unter Abzug der Menge statisch verdeckender Pixel $\boldsymbol{U}_{c,static}(g)$. Die Menge $\boldsymbol{R}_c(g)$ wird dann als gewöhnliche, bekannte Verdeckung (Maskierung) behandelt (Zeile (5)), da das Referenzbild an dieser Stelle ungültig ist und somit keine sinnvolle Vordergrund- bzw. Hintergrund-Klassifikation vorgenommen werden kann. In Zeile (6) erfolgt mit dem Aufruf der Funktion

constructSceneImages() die Konstruktion der Szenenbilder. Hierbei wird die Auflösung verdeckender Pixel benötigt (Kapitel 2.3.5), da verdeckte Pixel zu Pixeln mit der Markierung „Frei“ umgewandelt werden müssen, denn sonst wäre die Menge der zu aktualisierenden Pixel $\boldsymbol{W}_c(g)$ leer, welche in Zeile (7) aus dem Schnitt der Aktualisierungsmenge $\boldsymbol{R}_c(g)$ und der Menge der freien Pixel $\boldsymbol{E}_c(g)$ (siehe Kapitel 2.3.2) konstruiert wird. Die notwendige Auflösung zu freien Pixeln garantiert, dass sich kein unbekanntes Objekt im zu aktualisierenden Bildbereich befindet.

Von $\boldsymbol{W}_c(g)$ müssen im Abschluss noch diejenigen Pixel subtrahiert werden (Zeile (8)), die zu dynamischen Verdeckungen $\boldsymbol{U}_{c,dynamic}(g)$ gehören, denn für diese macht eine Aktualisierung keinen Sinn, da sich definitionsgemäß in diesen Bereichen kein (semi-)statischer Bildhintergrund befindet.

Nachdem $\boldsymbol{W}_c(g)$ dann die tatsächlich zu aktualisierenden Pixel jeder Kamera enthält, wird die Aktualisierung in der Funktion *copyRefImageParts*(...) (Zeile (9)) vollzogen, indem die entsprechenden Pixel(-merkmale) des aktuellen Kamerabildes $\boldsymbol{I}_c(g)$ in das Referenzbild übertragen werden (Abbildung 2.23).

(1) **updateReferenceImage**()
(2) **if**(known object picked or placed)
(3) $\forall\ c\ \ \boldsymbol{R}_c(g) = \boldsymbol{R}_c(g)\ \cup$ {projection of known object at known position}
(4) $\forall\ c\ \ \boldsymbol{R}_c(g) = \boldsymbol{R}_c(g) \setminus \boldsymbol{U}_{c,static}(g)$
(5) $\forall\ c\ \ \boldsymbol{U}_c(g) = \boldsymbol{U}_c(g)\ \cup\ \boldsymbol{R}_c(g)$
(6) *constructSceneImages*()
(7) $\forall\ c\ \ \boldsymbol{W}_c(g) = \boldsymbol{R}_c(g) \cap \boldsymbol{E}_c(g)$ // only use pixels marked as empty for update
(8) $\forall\ c\ \ \boldsymbol{W}_c(g) = \boldsymbol{W}_c(g) \setminus \boldsymbol{U}_{c,dynamic}(g)$ // skip known dynamic occlusions from update
(9) $\forall\ c$ *copyRefImageParts*$(\boldsymbol{W}_c(g), \boldsymbol{I}_c(g))$
(10) $\forall\ c\ \ \boldsymbol{R}_c(g) = \boldsymbol{R}_c(g) \setminus \boldsymbol{W}_c(g)$

Algorithmus 2.1: Referenzbildaktualisierung für Aufnahme- und Ablageoperationen (der Algorithmus ist für beide Fälle identisch)

Dieser Algorithmus ist identisch für Aufnahme- und Ablageoperationen, da in beiden Fällen eine Änderung im entsprechenden Bildbereich stattfindet. Lediglich das Robotermodell und damit die Menge $\boldsymbol{U}_{c,dynamic}(g)$ unterscheiden sich. In Abbildung 2.24 ist der Ablauf des Algorithmus für einen Zyklus für eine Aufnahmeoperation dargestellt.

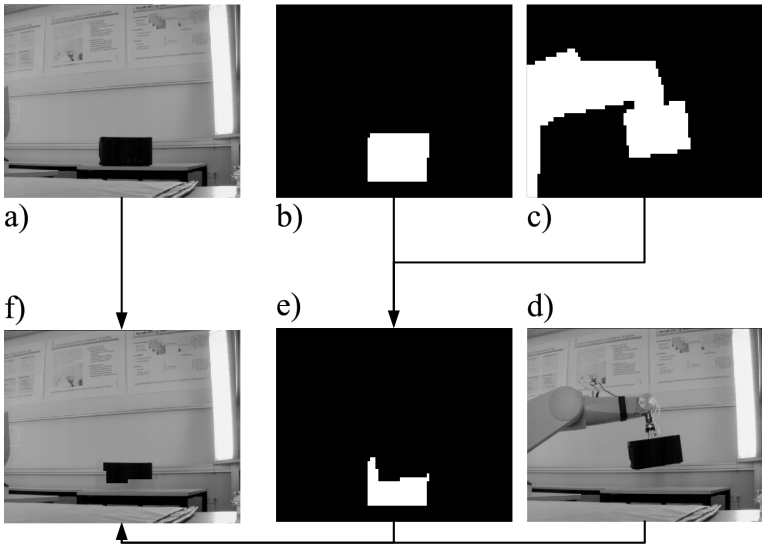


Abbildung 2.24: Referenzbildaktualisierung anhand eines realen Objekts. ***a)*** *Referenzbild mit schwarzer Kiste.* ***b)*** *Projektion des Objekts an der Aufnahmeposition (dies konstituiert hier die Menge* $\boldsymbol{R}_c(g)$*).* ***c)*** *Projektion des aktuellen Robotermodells (konstituiert die Menge* $\boldsymbol{U}_{c,dynamic}(g)$*).* ***d)*** *Aktuelles Kamerabild.* ***e)*** *Da keine unbekannten Objekte im Bild sichtbar sind, bleibt die Menge* $\boldsymbol{R}_c(g)$ *aus b) vollständig und wird lediglich durch die Menge* $\boldsymbol{U}_{c,dynamic}(g)$ *aus c) zur Menge* $\boldsymbol{W}_c(g)$ *reduziert.* ***f)*** *Das resultierende, aktualisierte Referenzbild. Die Pixelmenge* $\boldsymbol{W}_c(g)$ *aus e) definiert die Pixel, die aus dem aktuellen Kamerabild d) in das aktualisierte Referenzbild übertragen werden.*

Im folgenden Absatz wird der vorhergehende Algorithmus diskutiert. Durch die Akkumulation von zu aktualisierenden Bereichen $\boldsymbol{R}_c(g)$ in der Zeile (3) des Algorithmus kann die Größe der verdeckten Bereiche $\boldsymbol{U}_c(g)$ stark anwachsen (Zeile (5)), wenn nicht ausreichend schnell die Aktualisierung durchgeführt werden kann oder ein statisches unbekanntes Objekt dies dauerhaft verhindert. Dieses Anwachsen verdeckter Bereiche kann zur Ungültigkeit des Parameters θ_g führen. Um dies zu verhindern, ist gegebenenfalls eine Größenbeschränkung der Bereiche $\boldsymbol{R}_c(g)$ notwendig und somit in Konsequenz eine Blockierung des Prozesses, bis eine ausreichende Pixelmenge aktualisiert wurde. Alternativ kann der Parameter θ_g temporär erhöht werden, dies reduziert jedoch gleichfalls die Verfügbarkeit des Systems.

2.5.3 Hintergrundmodellierung

Die Modellierung des Bildhintergrundes wird unter den Stichworten „Change Detection“, „Background Modelling or Subtraction“ und „Novelty Detection“ in der digitalen Bildverarbeitung erforscht und bildet die Grundlage für viele weitere Verfahren, die eine Segmentierung von Regions-Of-Interest benötigen (z.B. Tracking). Das Spektrum reicht dabei von simplen Verfahren, die pro Pixel lediglich einen Referenzwert speichern bis zu komplexen Modellen für einzelne Pixel oder Regionen. Die Modelle beschreiben die erwarteten Eigenschaften dieser Bildprimitive. Ein Überblick über aktuelle Verfahren bieten [Elhabian08, Markou03, Piccardi04, Radke04]. Komplexe Modelle können eine Vielzahl der für Industriezellen relevanten Bildhintergründe erfassen. Da die Arten der

Bildhintergründe jedoch innerhalb einer Ansicht eines Arbeitsraumes sehr unterschiedlich sein können (eine Ansicht ist typischerweise eine Mischung von statischen, semi-statischen und dynamischen Hintergründen mit einem Schwerpunkt auf den statischen Anteilen), ist ein besonderes Augenmerk bei der Auswahl der Modelle auf die effiziente Handhabung der unterschiedlichen Hintergrundarten zu legen. Häufig hat die Verteilung der Bildeigenschaften in dynamischen Hintergründen recht komplexe, nicht normalverteilte Formen (siehe Abbildung 2.25c)).

Im Rahmen einer studentischen Arbeit [Busse06] wurden verschiedene Ansätze zur Klassifikation von Vordergrund vor dynamischen Hintergründen untersucht. Dabei standen Tests der Leistungsfähigkeit der Verfahren im Vordergrund unter Berücksichtigung der Klassifikationsqualität wie auch der Rechen- und Speichereffizienz.[1]

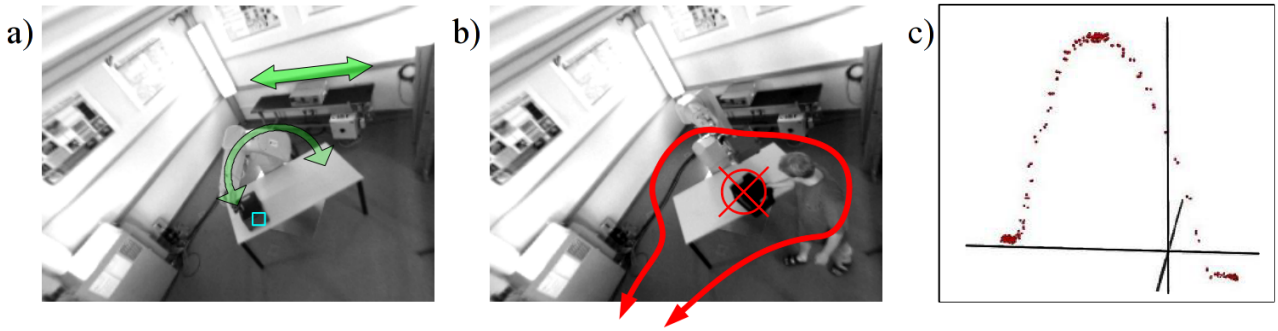


Abbildung 2.25: Experimenteller Aufbau zur Untersuchung von dynamischen Hintergünden. ***a)*** *Prototypische Arbeitszelle mit einer Auswahl von semi-statischen und dynamischen Hintergünden. Semi-statische Bereiche durch Pick-und-Place einer Kiste (vorne). Dynamischer Hintergrund durch den Roboter selbst und durch ein Fließband im Reversierbetrieb mit Verpackungskarton (hinten). Quadrat in Türkis: Bildkachel, deren Merkmalsvektoren über die gesamte Bildsequenz Grundlage für die Darstellung in c) bilden.* ***b)*** *Testphase der Klassifikation mit Bewegung von Hindernisobjekten. Roter Doppelpfeil: Bewegung des Menschen. Rot markierte Position: Temporäre Platzierung eines Hindernisobjekts.* ***c)*** *Verteilung von drei-dimensionalen Merkmalsvektoren der Kachel aus Bild a) mit erkennbaren Häufungspunkten.*

Die Grundannahmen für die hier untersuchten Methoden unterscheiden sich von einigen klassischen Ansätzen zur „Change Detection“, die eine Assimilation von Veränderungen im Hintergrundbild zur Klassifikationszeit zulassen, beziehungsweise explizit vorsehen (z.B. [Pilger05]). Insbesondere aus Sicherheitsgründen ist es jedoch nicht sinnvoll, unbekannte Objekte in den Hintergrund zu assimilieren, wenn sie nur lange genug an einer Stelle stehen. Als vereinfachende Annahmen für diese Voruntersuchungen wurden Beleuchtungsveränderungen in der Zelle ausgeschlossen und ein zyklischer Prozess vorausgesetzt. Dies ermöglicht eine Phase unüberwachten Lernens des Hintergrundbildes und zur Laufzeit eine Klassifikation des aktuellen Bildes durch Vergleich mit dem daraus erstellten Modell, wobei auf die Laufzeit des Vergleichs und die Größe der benötigten Datenstrukturen besonderen Wert gelegt wurde. Zur Geschwindigkeitssteigerung wurde wie

1 Eine über diese Voruntersuchungen hinausgehende Betrachtung alternativer Verfahren zur Behandlung dynamischer Hintergründe wird in einem parallelen Promotionsprojekt am Lehrstuhl fortgeführt und ist daher nicht Teil dieser Arbeit.

in [Ebert03] eine feste Aufteilung der Bilder in Kacheln verwendet und eine Transformation der Bilddaten in einen *n*-dimensionalen Merkmalsraum pro Kachel zur Erhöhung der Robustheit und Informationsreduktion.

Zur Untersuchung der vorgenannten Methoden wurde eine prototypische Zelle aufgebaut, die sowohl semi-statische als auch dynamische Hintergründe enthält (Abbildung 2.25a). Die semi-statischen Bereiche wurden durch eine Pick-and-Place-Aufgabe realisiert, die durch den Roboter ausgeführt wird. Der von der Roboterbewegung überstrichene Bildbereich wurde im Rahmen des Experiments auch als dynamischer Hintergrund aufgefasst und mit einer Maskierung durch Verdeckungspixel verglichen. Dynamischer Hintergrund wurde ebenfalls durch einen Verpackungskarton auf einem Fließband im Reversierbetrieb realisiert. Für alle Bildbereiche konnte so eine für die Untersuchungen praktikable kurze Zykluszeit erreicht werden. In der Testphase wurden Hindernisse (Mensch, Kiste) über einige Bildbereiche bewegt (Abbildung 2.25b). Die Klassifikationsqualität in der Testphase wurde durch einen Vergleich mit einer händisch bestimmten Klassifikation in der Bildsequenz bestimmt.

Verschiedene Verfahren des unüberwachten Lernens kamen zum Einsatz: Clustering-Methoden (Hierarchisches und Graph-basiertes Clustering) [Duda01] sowie ART [Moore88, Carpenter91]. Ziel war es vor allem, auch komplexe Verteilungen von Merkmalsvektoren zu erfassen (siehe auch Abbildung 2.25c). Für die Cluster wurde anschließend eine effiziente Repräsentation mittels Ellipsen und regulären Raumaufteilungen (Octrees etc.) erstellt, um zur Laufzeit eine möglichst schnelle und speichereffiziente Klassifikation zu ermöglichen trotz nahezu beliebiger Wahrscheinlichkeitsverteilungen für den Hintergrund. Der Speicherverbrauch und Rechenaufwand ist proportional zur Auflösung des daraus errechneten Differenzbildes. Für das Beispielszenario wurde im Mittel eine zu 96 % korrekte Klassifikation der Kacheln über alle Bilder der Sequenz erreicht.

Ausblick

Neben der Vielzahl an Modellierungsmethoden für dynamische Hintergründe, die noch untersucht werden können, kann durch erweiterte Informationen über Bildfolgen über die Zeit (Tracking) und Methoden zur Objekterkennung und -vervollständigung die Sicherheit und Leistungsfähigkeit der Detektion von unbekannten Objekten verbessert werden. Die domänenspezifischen Eigenheiten industrieller Prozesse wie zyklische Wiederholung und definierte Umgebungen können dabei vorteilhaft ausgenutzt werden. Für eine möglichst weitgehende Anwendbarkeit sind möglichst parameterfreie Methoden zu untersuchen, um den Einrichtbetrieb zu vereinfachen.

2.6 Experimentelle Ergebnisse

Im Folgenden werden Experimente dargestellt, die in Testläufen des Roboterprogramms in der in Kapitel 1.5.2 beschriebenen Demonstratorzelle für reale Hindernisse aufgezeichnet wurden.

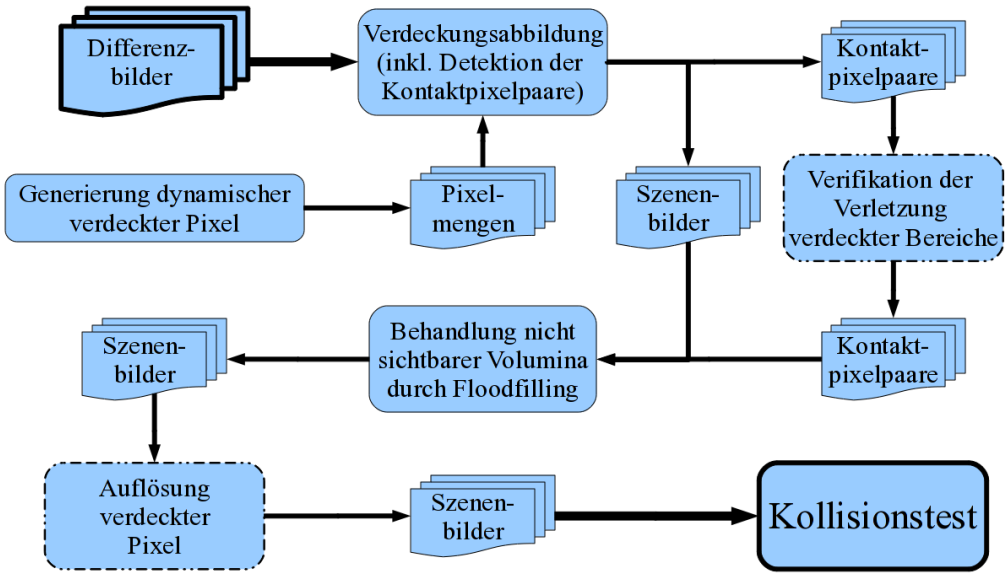


Abbildung 2.26: Komponenten der Szenenbilderstellung ausgehend von den aktuellen Differenzbildern der Szene (oben links). Die Generierung dynamischer verdeckter Pixel erfolgt aus dem Robotermodell und der aktuellen Roboterposition. Die so erstellten Pixelmengen werden mit den statischen verdeckten Pixeln vereinigt. Über die Szenenbildfusion werden dann Szenenbilder mit den Pixelmengen $\boldsymbol{E}_c(g)$, $\boldsymbol{O}_c(g)$ und $\boldsymbol{U}_c(g)$ erstellt. Aus Optimierungsgründen werden gleichzeitig Kontaktpixelpaare identifiziert. Anschließend kann optional eine Reduktion der Kontaktpixelpaare durch Verifikation der Verletzung verdeckter Bereiche mittels Epipolarmengen erfolgen. Aus den Kontaktpixelpaaren und den Szenenbildern erfolgt die Behandlung nicht sichbarer Volumina mittels Floodfilling. Daran kann optional eine Auflösung verdeckter Pixel mittels Epipolarmengen erfolgen. Die resultierenden Szenenbilder werden dann durch den Kollisionstest bzw. die Distanzberechnung verwendet, die für die Geschwindigkeitsregelung oder Bahnplanung eingesetzt werden.

Versuchsdurchführung

Die Geschwindigkeit des Roboterarms wurde basierend auf der berechneten Distanz zum nächsten unbekannten Objekt geregelt. Ein Mensch vollführte Bewegungen in der Zelle, die für eine Reihe von unterschiedlichen Roboterpositionierungen ein breites Spektrum von Entfernungen zum Roboterarm abdecken. Die berechneten Distanzen variierten über ein breites Spektrum, sind jedoch softwareseitig auf das Intervall von der Distanz minimaler Geschwindigkeit d_{min} bis zur Distanz maximaler Geschwindigkeit d_{max} beschränkt, was heißt, dass Distanzen ausserhalb dieses Intervalls nicht exakt berechnet werden. Alle Laufzeitmessungen wurden mit der Systemkonfiguration S1 (siehe Anhang) durchgeführt.

Laufzeitmessungen

Die Darstellung der Laufzeiten ist aufgeteilt in die Vorbereitung der Distanz-

berechnung und die eigentliche, anschließende Distanzberechnung (Kollsionstest). In der Vorbereitung der Distanzberechnung können verschiedene Phasen unterschieden werden (Abbildung 2.26), die in den obigen Kapiteln im Detail erläutert sind. Die Laufzeiten der einzelnen Phasen sind in den nachfolgenden Diagrammen für verschiedene Variationen der Parameter Kamera- und Gruppenanzahl und Bildauflösung dargestellt.

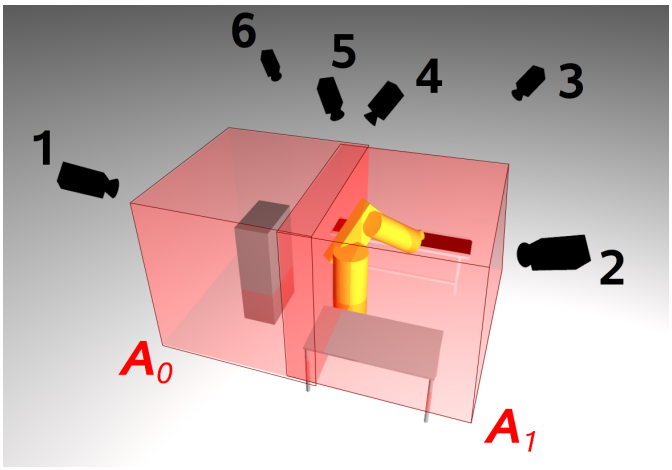


Abbildung 2.27: Schematische Darstellung der Konfiguration der Überwachungsbereiche und Kamerapositionen für den Experimentaufbau in der Prototypzelle aus Abbildung 1.7. Es sind zwei Überwachungsvolumina (mit zugeordneten Kameragruppen) A_0 und A_1 definiert. Für das Volumen A_0 sind dies die Kameras 1,2,6 und 4, für das Volumen A_1 sind dies die Kameras 1,2,3 und 5. Die Überwachungsvolumina überschneiden sich und haben jeweils ein Volumen von ca 2000x2000x1500 mm³.

Die Bildverarbeitungskomponente zur Erstellung des Differenzbildes in jeder Kamera ist hier von der Darstellung ausgespart, da durch die hohe Anzahl an Ansätzen für die Problemstellung sehr unterschiedliche Laufzeiten zu erwarten sind. Das hier verwendete Verfahren aus [Ebert03] hat Laufzeiten im Bereich von 5 ms bei einer Auflösung von 80x60 Differenzbildpixeln pro Kamera auf der Hardwarekonfiguration S2 (siehe Anhang).

Die verschiedenen betrachteten Softwarekonfigurationen variieren die Anzahl der Überwachungsvolumina bzw. Kameragruppen und die Auflösung der Kameras. Die betrachteten Volumina sind in Abbildung 2.27 dargestellt. Die unten aufgeführten Softwarekonfigurationen sind nummeriert, die Konfiguration 1 umfasst das Volumen A_1 und eine Auflösung von 80x60 je Kamera, die Konfiguration 2 die Volumina A_0 und A_1 bei der gleichen Auflösung, die Konfiguration 3 das Volumen A_1 bei einer Auflösung von 160x120 und die Konfiguration 4 die Volumina A_0 und A_1 bei dieser Auflösung.

Komponenten der Szenenbilderstellung

Die Diagramme in Abbildung 2.28 stellen links die durchschnittlichen Laufzeiten der Komponente Verdeckungsabbildung und rechts die der Komponenten Verifikation der Verletzung verdeckter Bereiche und der Kontaktpixelpaare und Behandlung nicht sichtbarer Volumina durch Regionenfüllen dar. Als konkreter Algorithmus für das Regionenfüllen wurde ein Floodfilling über die 4er-Nachbarschaft gewählt. Die Verdeckungsabbildung ist

hier die rechenaufwändigste Komponente mit Laufzeiten bis vier Millisekunden in der maximalen Konfiguration (Auflösung 160x120, insgesamt 8 verarbeitete Kamerabilder). Die (optionale) Auflösung verdeckter Pixel ist im Vergleich zur Verdeckungsabbildung etwa eine Großenordnung aufwändiger (siehe Abbildung 2.29).

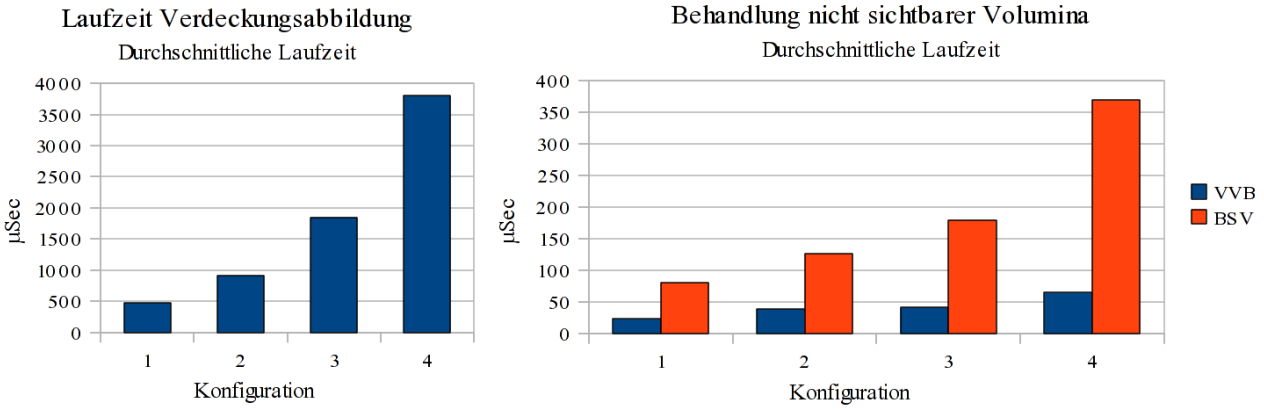


*Abbildung 2.28: Durchschnittliche Laufzeiten der Komponenten (**links**) Szenenbildfusion, (**rechts**) Verifikation der Verletzung verdeckter Bereiche (**VVB**, Kapitel 2.3.5) und Behandlung nicht sichtbarer Volumina (**BSV**, Kapitel 2.3.4) für die Softwarekonfigurationen mit Kameraauflösungen von 80x60 mit einem Überwachungsvolumen (Konfiguration 1) und zwei Volumina (Konfiguration 2) mit je vier Kameras pro Gruppe. Die Konfigurationen 3 und 4 umfassen eine Kameraauflösung von 160x120 für eine und zwei Überwachungsvolumina entsprechend.*

Die Relationen der Laufzeiten der Szenenbildfusion untereinander entsprechen der erwarteten Komplexität von $O(n^2C)$ mit n als Wurzel aus Höhe mal Breite eines Bildes und C der Gesamtanzahl an verarbeiteten Bildern (Kameras), da sich zwischen den Konfigurationen 1 und 2 sowie 3 und 4 jeweils die Kameraanzahl verdoppelt (Faktor 2) und zwischen den Szenarien 1 und 3 sowie 2 und 4 die Auflösung verdoppelt (Faktor 4). Die Bearbeitung der Kontaktpixelpaare hat einen deutlich geringeren Rechenaufwand, da ihre Anzahl im Allgemeinen sehr beschränkt ist. Das Floodfilling (rote Balken) nimmt dabei den größeren Anteil ein, was zu erwarten ist.

Die Laufzeit für die Verifikation der Verletzung verdeckter Bereiche ist vernachlässigbar gering und kann dennoch Verbesserungen der berechneten Distanzen erzeugen. Dies wird weiter unten in einem weiteren Experiment sichtbar. Sie wird daher im realisierten Prototyp immer eingesetzt.

Auflösung verdeckter Pixel

Die Komponente zur Auflösung verdeckter Pixel ist die in der Praxis rechenaufwändigste Komponente, da alle Pixel aus $\boldsymbol{U}_c(g)$ für jede Kamera c jeder Gruppe g auf mögliche Auflösbarkeit untersucht werden. Sie ist daher für hohe Auflösungen als optional zu betrachten und sollte nur verwendet werden, wenn die erforderlichen Zykluszeiten dies zulassen. Die Anzahl der Pixel in $\boldsymbol{U}_c(g)$ variiert zur Laufzeit im Wesentlichen mit der aktuellen Stellung des Roboterarms, da dessen Projektion die Menge

$\boldsymbol{U}_c(g)$ für den hier getesteten Demonstrator definiert.

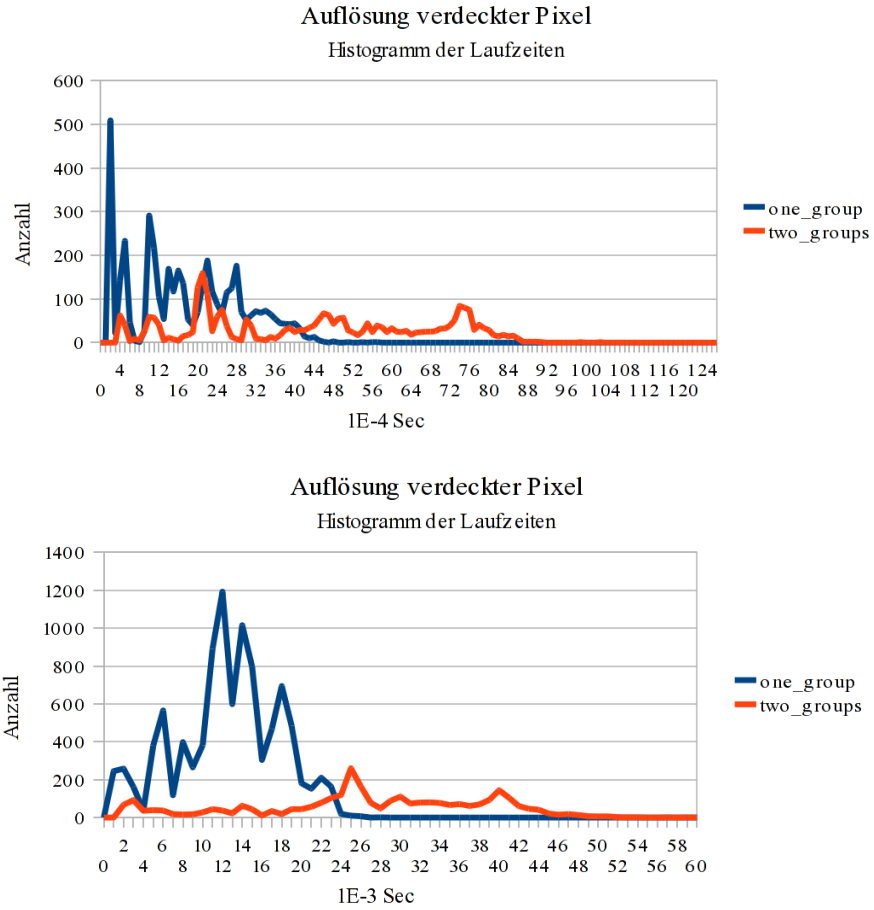


*Abbildung 2.29: Laufzeit-Histogramme der Auflösung verdeckter Pixel (Kapitel 2.3.5). Die getesteten Softwarekonfigurationen 1-4 verwenden eine Auflösung von 80x60 (**oben**) beziehungsweise 160x120 Pixeln (**unten**) im Szenenbild und eine bzw. zwei Kameragruppen zu je vier Kameras (Bezeichner „one_group“ und „two_groups“ in der jeweiligen Legende). Die Mittelwerte der Laufzeiten für die jeweilige Konfiguration sind **oben**: 1829 µSec für die Konfiguration „one_group“, 4472 µSec für die Konfiguration „two_groups“; **unten**: 13080 µSec für die Konfiguration „one_group“ und 27298 µSec für die Konfiguration „two_groups“. Bezogen auf die Skalierung der beiden Diagramme ist zu beachten, dass zur Verbesserung der Darstellung für das obere Diagramm eine Skalierung von 10^{-4} Sekunden, für das untere 10^{-3} Sekunden auf der Abszissenachse zur Anwendung kam.*

Die Ergebnisse aus Abbildung 2.29 bestätigen grob die prinzipielle Komplexität von $O(n^3 C_g^2)$, siehe hierzu Kapitel 2.3.5. Durch die ähnlich starke Abhängigkeit des Aufwands für den Kollisionstest von der Bildauflösung der Szenenbilder (siehe unten) wird für den realisierten Prototyp standardmäßig die Konfiguration 2 verwendet, also zwei Überwachungsvolumina bzw. Kameragruppen zu je vier Kameras mit einer Auflösung von 80x60 Pixeln, die zusammen ein überwachtes Volumen von 4 x 2 x 1,5 m abdecken.

Distanzverbesserung durch die Auflösung verdeckter Pixel

Um den Effekt der Auflösung verdeckter Pixel in der Praxis zu messen, wurde die Verbesserung berechneter Distanzen gemessen, indem die Distanzberechnung auf einer zufällig gewählten Roboterkonfiguration vor und nach der Auflösung aufgerufen wurde, mit

den Distanzen d_{vor} und d_{nach} als Ergebnis.

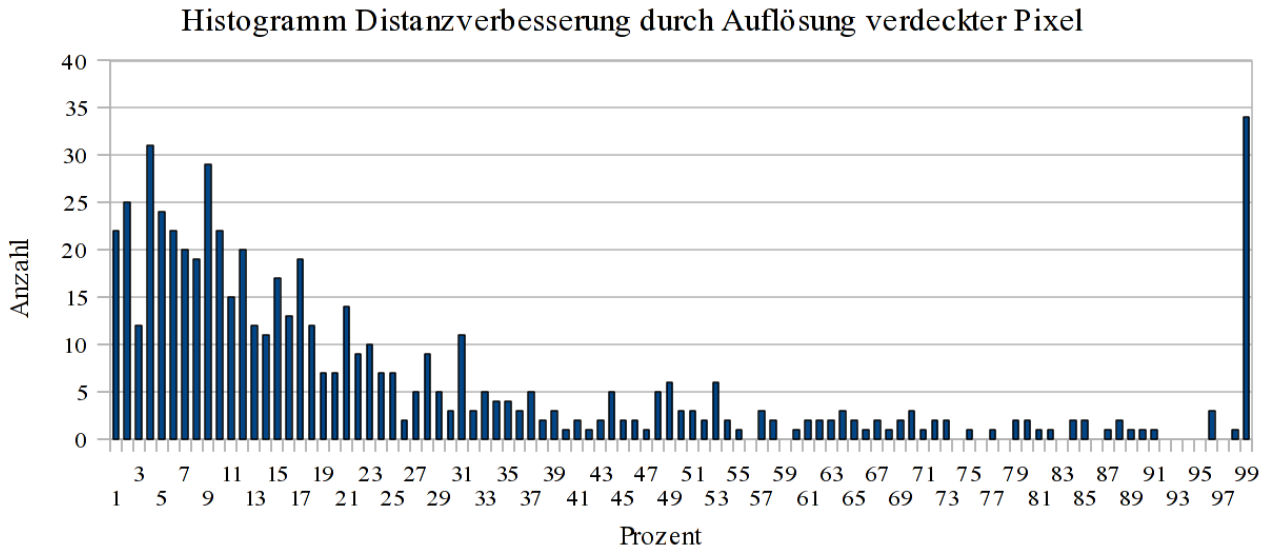


Abbildung 2.30: Histogramm über die prozentuale Verbesserung der Distanz zum nächsten unbekannten Objekt als Effekt der Auflösung verdeckter Pixel. Grundlage der dargestellten Ergebnisse sind die Distanzmessungen d_{vor} und d_{nach} für die Softwarekonfigurationen 1-4. Die sehr ähnlichen Histogrammwerte der einzelnen Testläufe wurden für obiges Histogramm aufaddiert. Für das Histogramm wurden insgesamt 5000 Messwert-Paare aufgezeichnet. Bei einem Großteil (2838) der Messwerte liegen beide Distanzen oberhalb der Distanz maximaler Geschwindigkeit d_{max}, sodass prinzipiell keine Verbesserung erreicht werden kann und sind daher nicht im Histogramm enthalten. Eine Kollision bei beiden Distanzen war für 1145 Messwerten angezeigt, weshalb auch diese nicht im Histogramm enthalten sind. Für 41 Distanzberechnungen zeigte d_{vor} eine Kollision an und d_{nach} nicht. Hier kann keine relative Verbesserung berechnet werden (Division durch 0). Von den verbleibenden 976 Fällen zeigt wiederum ein Teil gar keine Verbesserung (375 von 976) und ist daher nicht enthalten, um das Histogramm nicht zu verzerren und die Darstellung klarer zu gestalten.

Die Distanz d_{nach} ist prinzipbedingt immer größer als d_{vor}. Das Diagramm in Abbildung 2.30 stellt ein Histogramm der prozentualen Verbesserungen der Distanzen dar, berechnet nach folgender Formel:

$$p = \min\left(99, \left(\frac{d_{nach}}{d_{vor}} - 1\right) * 100\right) \tag{2.26}$$

Das Verbesserungshistogramm von p umfasst somit bei der Position „99 Prozent“ alle Distanzverbesserungen, die bei ≥ 99% liegen. Wie in Abbildung 2.30 zu sehen, ist eine hohe Bandbreite von Distanzverbesserungen möglich, der Fokus liegt jedoch auf den niedrigen Prozentwerten. Die nicht dargestellten 0% (keine Verbesserung der Distanz) machen jedoch in diesem Experiment einen Anteil von 38% aus. Betrachtet man lediglich die im Diagramm dargestellten Verbesserungen so ergibt sich eine durchschnittliche Verbesserung von 26% in der Distanz. Dieser Effekt muss mit den Laufzeitkosten (durchschnittlich kleiner als 5 ms für die Konfiguration 2, siehe oben) der Auflösung der verdeckten Pixel abgewogen werden, lohnt sich jedoch im Allgemeinen, wenn die angestrebte Gesamtzykluszeit des Systems betrachtet wird, welche die weiteren Komponenten Bildverarbeitung und

Robotersteuerung/Bahnplanung enthält und bei etwa 100 ms liegen soll für den realisierten Prototyp.

Distanzverbesserung durch multiple Testvolumina

Das hier verwendete Kugelmodell des Roboterarms bietet die Möglichkeit, die Distanzberechnung pro Kugel zu berechnen und die resultierende Distanz dadurch in bestimmten Fällen zu verbessern (siehe Kapitel 2.4.2). Für dieses Experiment wurde im Rahmen der oben dargestellten Experimentdurchführung die Distanz zu dynamischen unbekannten Objekten gemessen zum einen mit einer Aufteilung der Distanzberechnung auf die Kugeln des Robotermodells und zum anderen mit der Berechnung einer Distanz für den gesamten Roboterarm.

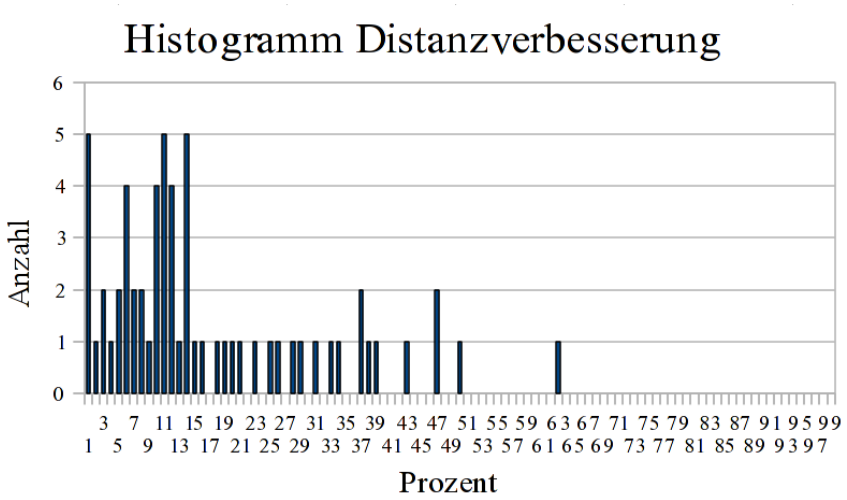


Abbildung 2.31: Histogramm über die prozentuale Verbesserung der Distanz zum nahesten unbekannten Objekt für die Berechnung mittels multipler Testvolumina im Vergleich zu der Berechnung der Objektdistanz für den Roboterarm als Ganzes (Details zum Verfahren in Kapitel 2.4.2). Grundlage sind 2048 x 2 Distanzberechnungen. Durch die Bewegungen des Objekts liegt ein Großteil (1349) der Berechnungen für beide Distanzen oberhalb der Distanz maximaler Geschwindigkeit d_{max}, sodass keine Verbesserung erreicht werden kann. Diese Berechnungen sind daher nicht in das Histogramm miteinbezogen, ebenso wie die Berechnungen, für die beide Distanzen unterhalb der Distanz 0 liegen, was für 322 Berechnungen der Fall war. Von den verbleibenden 377 Fällen zeigt wiederum ein Großteil keine Verbesserung (315 von 377) und ist daher nicht mitaufgeführt, um das Histogramm nicht zu verzerren. Die restlichen 62 Fälle zeigen Verbesserungen von bis zu 63 %.

In Abbildung 2.31 dargestellt ist das Histogramm der prozentualen Verbesserungen der minimalen Objektdistanzen durch multiple Testvolumina. Die Fälle, in denen keine Verbesserung möglich war (0 %), sind hier nicht mitaufgeführt, da sie einen sehr großen Anteil ausmachen (315 von 377 Fällen) und somit die Darstellung verzerrt hätten. Durch die multiplen Volumina kann für einen Teil (in diesem Experiment in etwa 15%) der real vorkommenden Situationen eine Distanzverbesserung erreicht werden, die hier bei etwa 10-15% liegen. Der Einsatz dieses Verfahrens lohnt sich daher, da der Mehraufwand unerheblich (nahezu nicht messbar) ist.

Laufzeit des bildbasierten Kollisionstests bzw. der Distanzberechnung

Im Systemprototyp wird der Kollisionstest durch die Distanzberechnung realisiert, da diese eine etwas günstigere Laufzeiten aufweist. Dies ist im realisierten Prototyp der Fall, da die bildbasierte Distanzberechnung mit einem Robotermodell durchgeführt wird, welches aus den Roboterarm mit lediglich 13 Kugeln approximiert (Details zur Berechnung siehe Kapitel 2.4.1). Der bildbasierte Kollisionstest, wie in Kapitel 2.3.3 beschrieben, projiziert hingegen (beliebige) Testvolumina in die Kamerabilder und testet die resultierenden Pixelmengen auf Schnitt mit den Pixelmengen $\boldsymbol{U}_c(g)$ und $\boldsymbol{O}_c(g)$. Dies ist für die hier gegebene Parametrisierung aufwendiger, kann aber je nach Modellierung (Anzahl der Kugeln, etc.) und Projektionsaufwand günstiger sein.

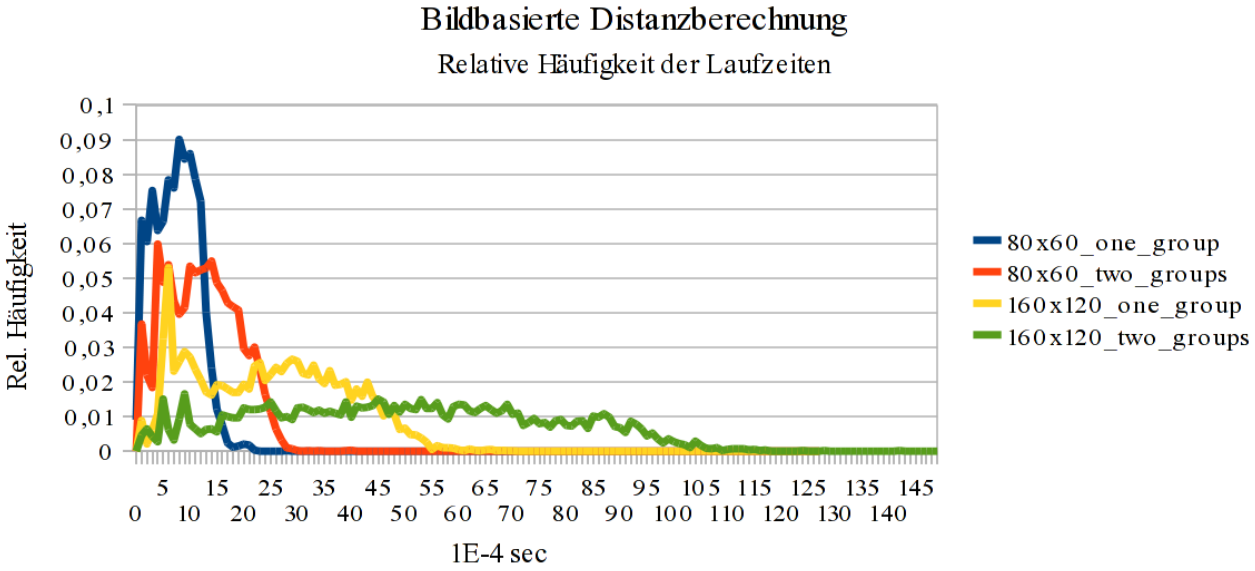


Abbildung 2.32: Histogramm der Laufzeiten der bildbasierten Distanzberechnung inklusive der Fusion der auf den Einzelbildern berechneten Distanzen (Kapitel 2.4.1). Es wurden alle Softwarekonfigurationen 1-4 getestet. Aus den Konfigurationen leiten sich die Bezeichner der Legende ab (z.B. der Bezeichner „80x60_one_group" = eine Kameragruppe ($\boldsymbol{A}_1$) mit Auflösung 80x60). Die Mittelwerte der Laufzeiten für die jeweilige Konfiguration sind: 790 µSec für die Konfiguration „80x60_one_group". 1273 µSec für die Konfiguration „80x60_two_groups", 2520 µSec für die Konfiguration „160x120_one_group" und 5017 µSec für die Konfiguration „160x120_two_groups".

In Abbildung 2.32 ist für die Softwarekonfigurationen 1-4 das jeweilige Histogramm der bildbasierten Distanzberechnung aufgetragen. Die Laufzeiten können bis zu einem durch die jeweilige Konfiguration gegebenen Maximum stark schwanken, da die Anzahl der Objektpixel und deren Position zum projizierten Testvolumen die Laufzeit entscheidend beeinflusst. Die Anzahl der Objektpixel schwankt, da sich das Objekt während des Tests häufig aus und in die Zelle bewegt und nicht permanent in allen Kameras vollständig sichtbar ist.

Für den Demonstrator wird die Konfiguration „80x60_two_groups" (siehe Abbildung 2.32) gewählt, diese bietet eine gute Raumabdeckung bei ausreichender Auflösung. Die

mittlere Laufzeit für eine Distanzberechnung beträgt in diesem Fall 1273 µSec.

Übersicht

In Tabelle 2.4 sind die durchschnittlichen Laufzeiten für die im Prototyp verwendete Softwarekonfiguration 2 aufgeführt. Diese umfasst die zwei Kameragruppen $\boldsymbol{A}_0$ und $\boldsymbol{A}_1$ aus Abbildung 2.27 mit einer Auflösung von 80x60.

Komponente	Mittlere Laufzeit
Verdeckungsabbildung	910,55 µSec
VVB	38,30 µSec
BSV	126,20 µSec
Auflösung verdeckter Pixel	4472,00 µSec
Summe Vorverarbeitung	ca. 5,5 mSec
Kollisionstest	ca. 1,3 mSec

Tabelle 2.4: Mittlere Laufzeiten der verwendeten Softwarekomponenten für den bildbasierten Kollisionstest.

2.7 Zusammenfassung

Dieses Kapitel präsentiert eine bildbasierte 3D-Überwachung eines gemeinsamen Arbeitsraumes von Mensch und Roboter basierend auf einem Netzwerk statisch montierter Kamerasysteme. Das System realisiert einen bildbasierten Kollisionstest und eine Abstandsberechnung zwischen a priori unbekannten Objekten wie dem Mensch und bekannten, modellierten Objekten wie einem industriellen Roboterarm. Dabei werden Verdeckungen der unbekannten Objekte durch die bekannten Objekte berücksichtigt. Die unbekannten Objekte werden mittels eines Differenzbildansatzes recheneffizient in den Kamerabildern detektiert. Es werden Verfahren dargestellt, um die Dynamik des Bildhintergrundes zu berücksichtigen, welche in üblichen Arbeitszellen auftritt (beispielsweise kontinuierliche Bewegungen durch Zuführeinrichtungen oder diskrete Veränderungen an Ablage-/Aufnahmepositionen). Zur Berechnung der Kollision oder Distanz wird die Kalibrierungsinformation der Kameras verwendet, um das Robotermodell in den Bildern zu berechnen. Basierend auf einem durch den Anwender vorgegebenen Sicherheitsparameter θ_g werden Objekte, die teilweise von bekannten Objekten verdeckt sind, im Kollisionstest und der Distanzberechnung konservativ korrekt berücksichtigt. Der Sicherheitsparameter ist dabei abhängig vom Aufbau der jeweiligen Arbeitszelle und der Anordnung der Kamerasysteme. Durch die Kamerakalibrierung bietet sich die Option, die Kollisions-/Distanzinformation mit Hilfe der Epipolargeometrie und einer Fusion aller Kamerabildinformationen zu verbessern.

Anhand eines Experiments konnte eine durchschnittliche Verbesserung der Distanzen durch die Verwendung der Epipolargeometrie von 15% gemessen werden und dies zu moderatem Speicher- und Rechenaufwand. Aus diesem Experiment ergab sich ein Gesamtaufwand für die Vorverarbeitung der Kamerabilder von ca. 5 ms im Schnitt für eine Konfiguration von acht Kameras zu einer Auflösung von 80x60 Pixeln. Für diese Konfiguration hatte der Kollisionstest bzw. die Distanzberechnung eine mittlere Laufzeit von 1,2 ms für eine beliebige zu testende Konfiguration der Robotergeometrie. Im Experiment wurde ein Raumvolumen von 4x2x1,5 m³ überwacht.

Der Aufwand für die Vorverarbeitung ohne Verbesserung durch die Epipolargeometrie und den Kollisionstest selbst sind proportional zu der Gesamtanzahl an Pixeln (abhängig von Kameraanzahl und -auflösung). Die Verbesserung durch die Epipolargeometrie hat einen Aufwand von $O(N^3C^2)$ mit der Anzahl Kameras C und N als Mittelwert aus Höhe und Breite des Bildes. Diese optionale Verbesserung muss daher bei höheren Auflösungen und Kameraanzahlen gegen den Rechenaufwand abgewogen werden. In Falle des Verzichts ist auch die Speicherkomplexität stark reduziert, weshalb sich dieses Verfahren auch für leistungsschwache Hardwarekonfigurationen im eingebetteten Bereich eignet. Durch die Epipolarmengen-Berechnungen kommt das Verfahren an die Leistungsfähigkeit von Rekonstruktionsverfahren [Kuhn09] heran. Hierbei büßt das Verfahren jedoch seine attraktive Berechnungs- und Speicherkomplexität ein. Der Algorithmus ist durch die unabhängigen Berechnungen auf Pixelebene jedoch sehr gut parallelisierbar. Ein Hauptaugenmerk für weitere Untersuchungen sollte auf der Verbesserung der Differenzbildberechnungen liegen, da hierdurch die Sicherheit des Gesamtsystems im Wesentlichen bestimmt wird. Auch könnte die weitere Untersuchung der effizienten Berechnung des Sicherheitsparameters θ_g sinnvoll sein, da hiervon auch maßgeblich die Sicherheit und der Einrichtungsaufwand des Systems abhängt.

3 Wegoptimierte Bahnplanung in dynamischen Arbeitszellen

Durch die Zielsetzung der Mensch-Roboter-Koexistenz ergibt sich ein dauerhafter oder temporärer Aufenthalt des Menschen im Arbeitsraum des Roboters. Der Mensch stellt dabei eine dynamische Schutzzone dar. Für diese soll Kollisionsfreiheit garantiert werden, soweit die vom System kontrollierten Zellenbestandteile kollidieren können, in diesem Fall der Roboterarm. Weiterhin steht die Forderung nach einer hohen Verfügbarkeit des Gesamtsystems, also eine Garantie der Zellenfunktionalität, soweit die Hindernissituation dies zulässt. Hindernisse (der Mensch und sonstige unbekannte Objekte) in der vorgesehenen Bahn des Roboters führen bei einer reinen Geschwindigkeitsregelung zur Blockade des Roboters und mithin des Prozesses. Sie erfordern daher eine Behandlung durch Umplanung der aktuellen Bahn. Daraus motiviert sich die Aufgabe, eine Bahnplanung zu realisieren, die das Ziel erreicht, wenn es eine kollisionsfreie Bahn gibt, die Kollisionsfreiheit garantiert und mit der Dynamik der Hindernisse umgehen kann, indem sie echtzeitfähig reagiert. Die Erweiterung dieser zunächst rein wegoptimierenden Bahnplanung zu einer zeitoptimierenden Planung wird in Kapitel 4 vorgestellt.

Dieses Hauptkapitel gliedert sich in die folgenden Abschnitte: Zunächst werden in Kapitel 3.1 die Anforderungen an die hier entwickelte Bahnplanung motiviert und formuliert. Kapitel 3.2 bietet einen Überblick über den relevanten Stand der Forschung auf diesem Gebiet. Daran anschließend wird in Kapitel 3.3 das grundlegende Konzept des Bahnplaners dargestellt, welches in [Gecks07] publiziert wurde, und in den Kapiteln 3.4-3.6 in seinen Teilaspekten erläutert. Experimentelle Ergebnisse werden in Kapitel 5 präsentiert.

3.1 Motivation und Aufgabenstellung

Wie in Kapitel 1.3 bereits dargelegt existiert schon heute eine Reihe von Systemen, die im industriellen Bereich eine Koexistenz bzw. Kooperation von Mensch und Roboter ermöglichen. Die hierbei eingesetzten Verfahren erlauben eine Definition von Schutzzonen oder eine distanzbasierte Geschwindigkeitsregelung. Um weiterhin einen effizienten und ununterbrochenen Prozessablauf zu garantieren, werden üblicherweise die Raumbereiche eingeschränkt, in denen sich Mensch und Roboter begegnen können. Dies geschieht durch entsprechende Zellenaufbauten, die durch Zäune und Abtrennungen die nicht für die Interaktion vorgesehenen Bereiche definieren, sodass in diesen der Prozess ohne Überwachung und daher mit voller Geschwindigkeit ablaufen kann. In den Interaktionsbereichen wird dann die Geschwindigkeit des Roboterarms geregelt. Die Zielstellung der Mensch-Roboter-Kooperation/Koexistenz ist es jedoch, diese Art von trennenden Schutzeinrichtungen komplett zu überwinden und eine freie und effiziente Zusammenarbeit von Mensch und Roboter zu erreichen.

Daher muss die Tatsache akzeptiert werden, dass der Mensch oder andere unbekannte Objekte häufig und indeterministisch an vielen Stellen die Bahn des Roboters blockieren kann. Der Forderung nach hoher Verfügbarkeit und geringen Einschränkungen für die Bewegungsfreiheit des Menschen kann dann nur begegnet werden, wenn der Roboter in der Lage ist, diese Blockaden sicher und schnell zu erkennen und baldmöglichst eine alternative Bahn zu planen. Die Bahnplanung soll daher aufgrund der Dynamik der Hindernisse echtzeitfähig sein, indem sie innerhalb eines Systemzyklus durch Modifikation der Roboterbahn auf die Umweltänderungen reagiert. Wäre die Planung zu zeitaufwendig und würde dann durch ein dynamisches Hindernis die geplante Bahn blockiert, so wäre nur ein Stop des Roboters in Betracht zu ziehen, bis eine neue Bahn gefunden wäre, da ein weiteres Abfahren der bisherigen Bahn ohne Neuplanung potentiell zum dauerhaftem Stillstand des Robotersystems führt. Dies tritt dann ein, wenn der Roboter sich bis auf die Distanz der Null-Geschwindigkeit an ein Hindernis angenähert hat und dieses sich im Anschluss nicht bewegt, also statisch bleibt. Durch das Anhalten des Roboters ist dann die Verfügbarkeit der Roboterfunktion nicht mehr gegeben. Die schnelle Reaktion auf die aktuelle Situation steht daher im Vordergrund. Die schnelle Reaktion alleine ist jedoch nicht ausreichend, um die Verfügbarkeit des Systems zu gewährleisten, der Planer muss auch in der Lage sein, schlussendlich einen kollisionsfreien Weg zum Ziel zu finden, falls ein solcher existiert. Das hier zu entwickelnde Konzept soll daher beides leisten.

Im Folgenden wird das gegebene Planungsproblem zunächst durch ein Beispiel aus dem Alltag motiviert und beschrieben. Hieraus lassen sich Teilaspekte der Problemstellung geeignet aufzeigen. Ein typisches Planungsproblem für die Kategorie der hier betrachteten

Probleme ist die Navigation eines Fahrzeugs durch den Feierabendverkehr einer Großstadt basierend auf einer bekannten Karte der fahrbaren Straßen, welche ebenfalls (evtl. implizit) die bekannten, nicht-befahrbaren Objekte (Gebäude, Gewässer, etc.) enthält. Diese entsprechen den bekannten Objekten $\boldsymbol{Z}_k$, die sich im Arbeitsraum befinden und während des Prozessablaufs statisch sind. Diese Karte bietet zunächst die Möglichkeit, die optimale Strecke in Länge oder Zeitdauer (Beachtung von Geschwindigkeitsbeschränkungen) zu bestimmen. Während der Ausführung des resultierenden Plans können jedoch unvorhergesehene Zwischenfälle auftreten, die eine Neuplanung erforderlich machen. Diese Blockierungen von Straßen durch beispielsweise Staus, Unfälle oder Bauarbeiten modifizieren temporär die Karte, indem diese Straßen von der Menge der fahrbaren Straßen entfernt werden müssen. Die Umgebung ändert sich im dargestellten Fall also schneller, als Erfassung, Planung und Ausführung des Plans an Zeit in Anspruch nehmen.

Das Bahnplanungsproblem ist daher in diesem Fall durch zwei Eigenschaften gekennzeichnet: Zum einen stellt sich die Frage, ob für die Planung nur ein im Voraus vollständig bekanntes Umweltmodell $U_{bekannt}$ verwendet wird (hier bestehend aus den bekannten Objekten $\boldsymbol{Z}_k$), oder ob die sensorisch erfasste, im Voraus unbekannte Umwelt $U_{unbekannt}$ auch eine Rolle spielt (hier bestehend aus den unbekannten Objekten $\boldsymbol{H}_l$). Zum anderen kann unterschieden werden, inwiefern die Dynamik der Umwelt ein Teil der Problemstellung bestimmt. Kann die Umwelt als statisch angenommen werden für die Dauer von Planung und Ausführung (und eventuell Erfassung der unbekannten Umwelt) oder müssen die Bewegungen der Objekte berücksichtigt werden? Aus den genannten Kriterien ergibt sich die in Tabelle 3.1 dargestellte Einteilung.

		Berücksichtigung der Umweltdynamik	
		Nein	ja
$U_{unbekannt}$ **verwendet**	nein	(1) Montageplanung, Sweeping	(2) Raum-Zeit-Bewegungsplanung („Kinodynamic Motion Planning“)
	ja	(3) Exploration (SLAM), Griff in die Kiste	(4) Mensch-Roboter-Kooperation bzw. Koexistenz, Service-Roboter

Tabelle 3.1:Kategorien 1-4 von Bahnplanungsproblemen und jeweiligen Beispielanwendungen

Kategorie 1 aus Tabelle 3.1 beinhaltet viele klassische Bahnplanungsprobleme, die Zugriff auf die vollständig bekannte Umweltinformation haben und die Umwelt während der gesamten Planung und Ausführung als statisch betrachten können. Dazu gehört bespielsweise die Planung der Montage von Bauteilen, da diese Offline erfolgen kann und alle Hindernisse durch die vollständige Modellierung der Bauteile gegeben sind. Die einmalige Planung wird dann im Produktionsablauf ständig wiederholt. Auch für das vollständige Abfahren eines begrenzten Raumes (Sweeping) gibt es Algorithmen, die auf

bekannten Umweltinformationen optimierend planen und die Umwelt dazu während Planung und Ausführung als statisch betrachten (denn sonst würde die optimierte Bahn in Frage gestellt).

Bei Problemstellungen aus Kategorie 2 ist ein vollständiges Umweltmodell gegeben, dieses beinhaltet aber explizit die Dynamik der Objekte. Ein typisches Planungsbeispiel ist die Planung eines Andock-Manövers von Raumgleitern an Raumstationen (Begriff: „Kinodynamic Motion Planing“, [Donald93]). Bei diesen Verfahren wird in einem Zustandsraum geplant, der den Konfigurationsraum um eine Zeitachse erweitert und zu jedem Punkt eine Kollisionsinformation besitzt. Dieser neue Raum ist wiederum statisch.

Kategorie 3 beinhaltet Problemstellungen, die die Umwelt zwar als statisch betrachten können, diese jedoch sensorisch erfassen müssen, da sie in Teilen oder gänzlich unbekannt ist. Die Exploration der Gangstruktur eines Gebäudes durch einen Mobilen Roboter (auch SLAM, „Simultaneous localization and mapping“) ist hierfür ein gutes Beispiel. Beim sogenannten „Griff in die Kiste“ wird die Lage von Objekten in einem Behälter sensorisch erfasst und dann eine Greifplanung durchgeführt, um das jeweilige Objekt aufzugreifen und einer Verarbeitung zuzuführen oder zu montieren. Es ist zwar auch hier wünschenswert, dass diese Planung schnell vonstatten geht, jedoch steht die Qualität der Planung (optimale Bahn, Vollständigkeit, etc.) im Vordergrund.

Die Problemstellung dieser Arbeit findet sich in Kategorie 4 der Tabelle 3.1 wieder. Hierbei existieren unbekannte Objekte, die in der Planung berücksichtigt werden müssen und die Umwelt kann für die Dauer von Planung und Ausführung nicht als statisch angenommen werden, da die unbekannten (und evtl. die bekannten) Objekte ihre Position zum Teil deutlich ändern, bis das Ziel der Planung erreicht ist. Dies ist typisch für mobile Roboter, die in Menschenmengen navigieren (Serviceroboter), als auch in der Mensch-Roboter-Kooperation/Koexistenz, in der Mensch und Roboter sich einen gemeinsamen Arbeitsraum teilen. Der Mensch kann sehr dynamische Bewegungen ausführen und muss in der Bahnplanung entsprechend berücksichtigt werden. In der Realität findet das System zwar häufig eine eher statische Umgebung vor, muss jedoch permanent auf eine Änderung der Umweltdynamik reagieren können.

Aufgabenstellung

Gegeben sei eine Startkonfiguration $\boldsymbol{q}_s$ und Zielkonfiguration $\boldsymbol{q}_z$ für einen Roboter im Konfigurationsraum ζ. Weiterhin gegeben seien zwei Kollisionstests, die für eine Konfiguration $\boldsymbol{q}$ oder eine Linearbahn im Konfigurationsraum von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ eine Aussage über die Kollisionsfreiheit liefern. Der Kollisionstest $Coll_S(...)$ liefert eine Kollisionsaussage (true = Kollision) für die statischen, bekannten Objekte $\boldsymbol{Z}_k$:

$$Coll_S(\boldsymbol{q}) \in \{\text{true, false}\} \qquad Coll_S(\boldsymbol{q}_a, \boldsymbol{q}_b) \in \{\text{true, false}\} \tag{3.1}$$

Der Kollisionstest $Coll_D(...)$ liefert eine Kollisionsaussage für die dynamischen, sensorisch erfassten und daher a priori unbekannten Objekte (Kapitel 2):

$$Coll_D(\boldsymbol{q}) \in \{\text{true, false}\} \qquad Coll_D(\boldsymbol{q}_a, \boldsymbol{q}_b) \in \{\text{true, false}\} \tag{3.2}$$

Gesucht ist eine Bewegung des Roboters $\boldsymbol{q}(t)$, t aus $[0, t_{max}]$, mit $\boldsymbol{q}(0) = \boldsymbol{q}_s$ und $\boldsymbol{q}(t_{max}) = \boldsymbol{q}_z$, für die gilt:

$$\left(\left(\neg Coll_S(\boldsymbol{q}(t))\right) \wedge \left(\neg Coll_D(\boldsymbol{q}(t))\right)\right) \quad \forall\, t \in [0, t_{max}] \tag{3.3}$$

Diese Anforderung ist nur insoweit zu erfüllen, als dass die Bewegung des Roboters ursächlich für die Kollision ist. Steht der Roboterarm still, können Kollisionen auch durch die Bewegungen der unbekannten Objekte $\boldsymbol{H}_l$ entstehen. **Nicht gesucht** ist daher die Berücksichtigung der Kollisionen aufgrund dieser Bewegungen, da sie nicht der Kontrolle des Systems unterliegt.

Zusätzlich werden an das Planungskonzept hier die folgenden Anforderungen (**Ax**) und Bedingungen (**Bx**) gestellt. Die Anforderungen sind die Zielvorgaben, die der Planer erreichen soll, bzw. Bewertungskriterien für die Leistungen des Planers, die Bedingungen stellen Gegebenheiten der Umgebung oder vorausgesetzte Annahmen dar.

A1 Echtzeitfähigkeit

Durch das Anwendungsszenario sind dynamische unbekannte Objekte gegeben, die Anforderungen an die Echtzeitfähigkeit des Algorithmus stellen. Dies sind weiche Echtzeitanforderungen, da lediglich die Verfügbarkeit des Systems leidet, falls die Planung nicht rechtzeitig ein Ergebnis liefert in dem Sinne, dass ein Ausweichen möglich ist. Die Sicherheit ist durch die Geschwindigkeitsregelung gegeben, siehe hierzu Kapitel 1.5.3. Das System soll in jedem Zyklus auf die Veränderung der Umgebung reagieren und eine angepasste Bahn liefern.

A2 Weg-Optimierung

Die Länge der Roboterbahn soll minimal sein unter Berücksichtigung der aktuellen Hindernissituation. Über die zukünftige Bewegung der unbekannten Objekte $\boldsymbol{H}_l$ werden keine Aussagen gemacht, da dies ein Bewegungsmodell der Objekte erfordern würde, dessen Berechnung sehr aufwendig ist und zudem nur eine Aussage über ein kurzfristiges Zeitfenster zulässt, sodass der Einfluss auf die Planausführung bis zum Ziel gering wäre (siehe hierzu auch Argumentation in [Berg08]). Die Bahnlänge ist daher auf einer während eines Systemzyklus als statisch angenommenen Umwelt zu minimieren, da auch der Kollisionstest aus Kapitel 2 pro Zyklus lediglich das aktuelle Abbild der Szene als Grundlage für die Berechnung von Kollisionen verwendet.

A3 Hohe Anzahl an Freiheitsgraden

Der Planer soll fähig sein, in hochdimensionalen Konfigurationsräumen zu planen, im konkreten Fall für einen Knickarm-Industrieroboter mit sechs Freiheitsgraden. Die Anzahl an Freiheitsgraden soll nicht beschränkt werden, um beispielsweise redundante Roboter mit mehr als sechs Freiheitsgraden zuzulassen.

A4 Parallelität von Planung und Ausführung

Die Planung einer angepassten Roboterbewegung soll parallel zur Ausführung des aktuellen Plans durchgeführt werden, wie beispielsweise in der mobilen Robotik üblich. Die Planung in einem Zyklus beginnt somit mit der Roboterkonfiguration, die am Ende des aktuellen Systemzyklus erreicht wird. Diese Position kann aus der bisher geplanten Bahn und der aktuellen Robotergeschwindigkeit projiziert werden. Die neue Bahn ersetzt dann ab dieser Position die Alte.

A5 Vollständigkeit des Planers

Der Planer soll mindestens statistisch vollständig sein. Diese Erfordernis ergibt sich vor allem aus dem Anwendungsbereich. In industriellen Anlagen ist Verfügbarkeit ein wichtiges Ziel und daher soll das Planungsziel auch erreicht werden, wenn eine kollisionsfreie Bahn existiert. Damit soll ein Stillstand des industriellen Prozesses vermieden werden und somit hohe Kosten, die mit einem Stillstand verbunden wären, insbesondere dann, wenn die Zelle Teil einer Fertigungslinie ist mit von ihr abhängigen Linien und eventuell vorhandene Puffer klein sind.

A6 Effiziente(r) Einsatz der Kollisionstests bzw. Erfassung der Umwelt

Es existiert keine Transformation von Arbeitsraum in den Konfigurationsraum des Roboters. Die Belegung des Konfigurationsraums ist implizit durch den Kollisionstest gegeben (siehe auch Bedingung B1 weiter unten). Der Kollisionstest ist rechenintensiv, wodurch sich in Kombination mit der Anforderung an die Echtzeitfähigkeit (A1) eine Einschränkung der Anzahl an berechenbaren Kollisionstests ergibt. Daraus folgt die Forderung, dass der Kollisionstest effizient eingesetzt werden muss, um für die Planung einen hohen Nutzwert zu erreichen.

Dies bedeutet insbesondere, dass (1) lediglich der für die Planung relevante Teil der Umwelt durch den Kollisionstest erfasst werden sollte, da weitere Umweltinformationen dann per Definition keine Änderung der Bahn erzeugen. Weiterhin wird durch diese Anforderung verlangt, dass (2) Kollisionstests in dynamischen Umgebungen auf die nähere Umgebung des Roboterarms konzentriert werden, soweit dies auch Forderung (1) erfüllt, denn in zeitlich begrenzt vorhersagbaren Umgebungen macht es Sinn Umweltinformationen auszuwerten, die in naher Zukunft benötigt werden, also hier für die unmittelbar nächsten

Bewegungen. Weiterhin (3) müssen keine Kollisionstests wiederholt berechnet werden, wenn die unbekannten Objekte statisch sind.

A7 Sicherheit

Der Planer soll garantieren, dass Kollisionen mit bekannten und unbekannten (dynamischen) Objekten nicht auftreten können. Diese nahe liegende Anforderung ist durch die Geschwindigkeitsregelung des Roboters für die unbekannten Objekte schon realisiert. Für die bekannten Objekte ist diese Sicherheit mit Hilfe des Kollisionstests $Coll_S(...)$ zu gewährleisten.

A8 Anwendungsdomäne

Der Planer sollte geringe oder keine Anforderungen an die Art und das Verhalten der unbekannten Objekte stellen, um möglichst breit einsetzbar zu sein. Die durch die Zykluszeit bestimmte Latenz des Systems impliziert jedoch eine maximale Bewegungsgeschwindigkeit der unbekannten Objekte, wenn das Robotersystem nicht aus Sicherheitsgründen stillstehen soll. Diese Anforderung ergibt sich auch schon aus der reinen Geschwindigkeitsregelung.

Im Folgenden sind Bedingungen dargestellt, welche die Gegebenheiten der Umgebung oder vorausgesetzte Annahmen darstellen.

B1 Implizite Konfigurationsraumdarstellung

Diese Bedingung ergibt sich aus der Berechnungsdauer für den bildbasierten Kollisionstest, wie in Kapitel 2 vorgestellt. Durch die hohe Anzahl an geplanten Freiheitsgraden (A3) und der damit einhergehenden Größe des Konfigurationsraumes (siehe hierzu auch [Henrich98]) steht eine Transformation der Arbeitsraumdarstellung in eine Konfigurationsraumdarstellung nicht zur Verfügung (siehe auch Abschnitt zu tabellenbasierten Planern im Stand der Forschung, Kap. 3.2). Es existieren zwei Kollisionstestvarianten einmal für die bekannten Objekte aus $\boldsymbol{Z}_k$ und weiterhin für unbekannte Objekte $\boldsymbol{H}_l$.

B2 Rigidität des Roboters und der transportierten Objekte

Jeder Punkt $\boldsymbol{p}$ auf der Oberfläche des Roboters und der optional mitbewegten (gegriffenen) Objekte ist nur durch die Konfiguration r des Roboters bestimmt. Dies garantiert, dass verlässlich eine maximale Bewegungsdistanz im Arbeitsraum (MAXMOVE, siehe [Henrich98]) für die Linearverbindung zwischen zwei Punkten im Konfigurationsraum abgeschätzt werden kann. Damit sind deformierbare Objekte von der Überwachung explizit ausgeschlossen, wenn sie durch den Roboter transportiert werden.

Dies folgt auch schon aus den Voraussetzungen für die Referenzbildaktualisierung (Kapitel 2.5.2), wenn diese verwendet wird.

B3 Roboterdynamik

Üblicherweise wird die Robotergeschwindigkeit durch endliche Beschleunigung und Überschleifeffekte während der Ausführung einer Bewegung im Verhältnis zur maximal möglichen Geschwindigkeit reduziert. Dies wird hier nicht betrachtet, d.h. die Beschleunigung wird als unendlich angenommen und die Geschwindigkeit ist allein abhängig von der jeweils aktuellen Distanz zu den unbekannten Objekten. Auch die durch Überschleifen entstehenden Abweichungen von der geplanten Bahn werden als gering angenommen und gegebenenfalls durch eine Expansion des Robotermodells für Kollisionstests berücksichtigt (in diesem Fall ist allerdings das Erreichen des Ziels durch Engpässe eingeschränkt).

B4 Umweltmodell für statische bekannte Objekte

Die bekannten Objekte $\boldsymbol{Z}_k$ sind modelliert und in Kollisionsklassen eingeteilt. Darauf setzt ein Kollisionstest für einzelne Konfigurationen und Linearbahnen im Konfigurationsraum auf. Die Objekte können bespielsweise aus einem CAD-Modell der Zelle hervorgehen.

3.2 Stand der Forschung

Der Begriff der echtzeitfähigen Online-Bahnplanung in hochdimensionalen Konfigurationsräumen ist häufig mit lokaler, reaktiver Bahnplanung mittels Potentialfeldmethoden [Khatib86] verbunden. Hierbei wird in einem Systemzyklus aus den aktuellen Sensordaten und weiteren Umweltinformationen der nächste Bewegungsschritt des Roboters berechnet und dies für jeden Systemzyklus iteriert. Diese Vorgehensweise erfordert keine globalen Umweltinformationen und kann die verwendeten lokalen Informationen echtzeitfähig verarbeiten (A1), bzw. ohne aufwendige Berechnungen direkt auf Sensordaten arbeiten, wie dies z.B. bei den künstlichen Häuten der Fall ist (vgl. [Lumelsky93]). Durch die Beschränkung auf lokale Informationen können jedoch lokale Minima im Konfigurationsraum des Roboters nicht überwunden werden, womit die Planer dieser Kategorie nicht vollständig sind (A5) und in „Sackgassen“ stecken bleiben. Die Gruppe der rein lokalen Planer wird daher im weiteren ausgeblendet und lediglich globale Planer oder Mischformen betrachtet. Diese nutzen auch die globalen Umweltinformationen, die die gegebenen Kollisionstests zur Verfügung stellen.

Durch eine geschickte Verknüpfung von lokalen und globalen Planern ist es möglich, lokale Minima zum überwinden, wie z.B. mittels der in [Brock00] beschriebenen Methode des „elastic strip“. Hierbei wird durch einen globalen Planer initial eine kollisionsfreie

Roboterbahn geplant. Dieser Bahn ist der sogenannte *elastic strip* (Abbildung 3.1). Verändert sich die Umgebung, wird die Bahn verschoben durch virtuelle Kräfte, die von den Hindernissen ausgehen (siehe Potentialfeldmethoden). Die Berechnung und Anwendung dieser Kräfte ist echtzeitfähig möglich, da sie lokal begrenzt ist (A1). Dabei werden auch neue Zwischenpunkte in den Pfad eingefügt, falls notwendig. Solange alle Punkte des Pfades kollisionsfrei verbunden sind, bleibt der Pfad gültig. Bei starken Änderungen der Umgebung muss der Pfad jedoch global neu geplant werden. Die Berechnung des initialen Pfades bzw. eines neuen globalen Pfades ist sehr aufwendig und erfordert viele Kollisionstests. Wenn die Umgebung eine hohe Dynamik aufweist, kann der globale Pfad nach seiner Berechnung schon wieder ungültig sein. Die Verschiebung der Bahnpunkte kann zu ineffizienten Bahnen führen. Eine Schwelle für die Neuplanung muss für diese Fälle durch den Benutzer gesetzt werden. Plötzlich frei werdende Bahnen können nicht genutzt werden, da der global geplante Pfad verfolgt wird (Beispiel: eine plötzlich geöffnete Tür).

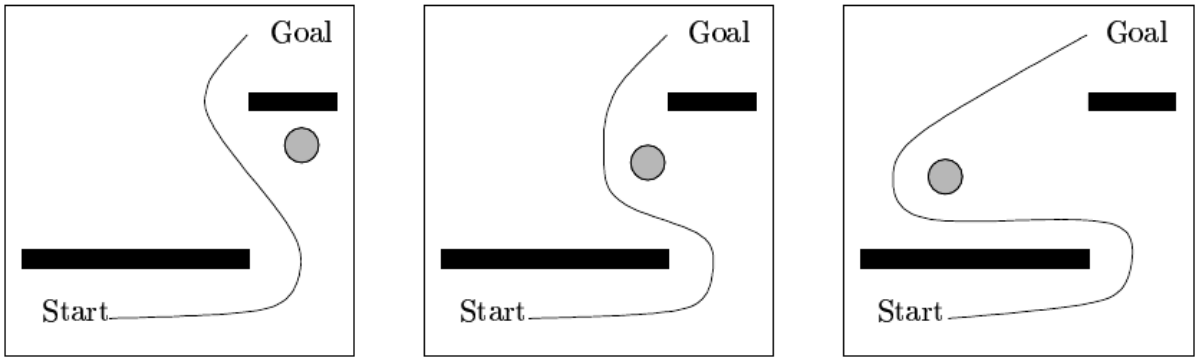


Abbildung 3.1: „Elastic Strips"-Verfahren aus [Brock00]. Ein initial global geplanter Pfad wird lokal durch abstoßende Kräfte deformiert, die von einem beweglichen Hindernis ausgehen.

Viele Bahnplaner sind im mobilen Bereich echtzeitfähig (siehe z.B. [Rohrmüller08] oder [Berg06]), weil die Kosten für den Kollisionstest minimal sind und weil der Raum, in dem geplant wird, im Allgemeinen dem sensorisch erfassten Arbeitsraum entspricht. Damit fallen die Kosten für eine Transformation von Arbeitsraumhindernissen in Konfigurationsraumhindernisse weg, welche erheblich sein können für Roboter mit vielen Freiheitsgraden. In diesen Fällen kann oft bereits durch eine geeignete Expansion der Arbeitsraumhindernisse mit einem punktförmigen Roboter geplant werden, welches die Kollisionstestkosten auf nahe Null bringt. Oder es können geometrische Eigenschaften der Hindernisse/des Freiraums für die effiziente Bestimmung von kollisionsfreien Bahnen verwendet werden (z.B. [Lindemann05]). Aufgrund dieser hier nicht erfüllten Voraussetzungen sind viele der als echtzeitfähig bezeichneten Konzepte aus der mobilen Robotik nicht auf die hier gegebene Aufgabenstellung übertragbar (A3, A6, A8, B1).

Die Transformation von Arbeitsraumhindernissen in den Konfigurationsraum des Roboters mit nachfolgender Planung mit punktförmigem Roboter in diesem Konfigurationsraum ist allerdings durch gestiegene Rechenleistung und größeren Speicherplatz in den Bereich möglicher Lösungen für die echtzeitfähige Bahnplanung in hochdimensionalen Konfigurationsräumen gerückt. Bei diesen Methoden wird jedoch nicht der gesamte Konfigurationsraum rekonstruiert, sondern lediglich eine Abbildung definiert, die Arbeitsraumhindernisse auf die Belegung von Knoten und Kanten eines Graphen im Konfigurationsraum abbildet. Im folgenden Abschnitt sollen diese Ansätze kurz gesondert dargestellt und diskutiert werden.

Tabellenbasierte Ansätze

Die hier als „Tabellenbasierte Planungsalgorithmen“ bezeichneten Ansätze arbeiten mit einem Graphen im Konfigurationsraum, der aus einer Menge von Knoten $\boldsymbol{V}$ und Kanten $\boldsymbol{E}$ besteht, die diese Knoten verbinden. Dieser Graph ist unveränderlich für die gestellten Planungsaufgaben (Multi-Query-Ansatz). Daher wird meist ein lokaler Planer verwendet, um eine konkrete Start- und Zielkonfiguration mit dem Graph zu verbinden. Weiterhin werden Abbildungen $\Phi_V{:}\boldsymbol{A}\rightarrow\boldsymbol{V}$ und/oder $\Phi_E{:}\boldsymbol{A}\rightarrow\boldsymbol{E}$ bestimmt, die jeden Voxel des Arbeitsraumes $\boldsymbol{A}$ auf die Menge von Knoten und/oder Kanten des Graphen abbildet, für die das (überstrichene) Volumen des Roboters für den jeweiligen Knoten/die jeweilige Kante eine nichtleere Schnittmenge mit den Voxel hat. Wie in [Leven00, Ebert03, Liu06] beschrieben, wird im Allgemeinen zunächst die Umkehrabbildung Φ^{-1} von Knoten und Kanten auf Voxel (vollständig) berechnet und dann die entstehende Tabelle invertiert. Diese Abbildungen können auch für einen bildbasierten Kollisionstest (Kapitel 2) berechnet werden, wenngleich dies etwas mehr Aufwand für die Datenstrukturen erfordert (siehe hierzu auch [Ebert03]). Das Ergebnis wäre in diesem Fall eine Abbildung von den Pixeln jeder Kamera auf die Knoten und Kanten des Graphen und die anschließende Berechnung des Kollisionstestprädikats auf diesen Informationen.

Alternativ wird für bekannte dynamische Objekte eine Abbildung definiert von deren diskretisierten Positionen auf die dadurch betroffenen Kanten des Graphen [Jaillet04, Berg06], wozu vollständige Informationen über die Hindernisse vorliegen müssen und lediglich deren Konfiguration zur Laufzeit sensorisch bestimmt wird.

In der Praxis ergibt sich bei den tabellenbasierten Planern für bestimmte Hindernis-Konstellationen eine äußerst schnelle Berechnung der Belegung des gesamten Graphen und damit eine schnelle Bahnplanung, die eine Planung mit dynamischen Hindernissen in Echtzeit ermöglicht. (Daten aus [Liu06]: 40×61×34 Gitter, 5000 Knoten im Graphen, Planungsdauer 25 bis 359 ms). Während die Schnelligkeit der Ansätze sie für die dynamische Bahnplanung interessant macht, ergeben sich doch einige Nachteile, darunter

vor allem hoher Speicherbedarf, Inflexibilität und mangelnde Echtzeitfähigkeit. Eine Distanzberechnung ist nur mit deutlich größerem Aufwand zu integrieren und in den genannten Referenzen nicht realisiert. Diese wird jedoch für den zeitoptimierenden Planer in Kapitel 4 benötigt. Die folgende Liste geht im Detail auf die problematischen Punkte ein:

- **Speicherbedarf**: Die Datenstrukturgröße ergibt sich aus der Auflösung des Voxelraumes und der Anzahl an Knoten/Kanten im Graph. Die erforderlichen Datenstrukturen kommen je nach (realistischer) Wahl der Parameter auf mehrere Gigabyte und sind somit für Systeme mit begrenzten Ressourcen problematisch. Wenn aus Platzgründen die Abbildung von Voxeln auf Kanten Φ_E nicht mit abgespeichert wird (wie in [Liu06]), müssen die Kanten vor der Ausführung auf Kollision getestet werden, um Sicherheit zu garantieren. Hierdurch entsteht weiterer Rechenbedarf für den Kollisionstest und die Vorteile der tabellenbasierten Planer kommen kaum noch zur Geltung.
- **Inflexibilität**: Es erzeugt zur Laufzeit einen erheblichem Aufwand, wenn neue Knoten und Kanten zum Graphen hinzugefügt werden, was den Aufwand für die Suche nach einer Lösung des Planungsproblems für PRM-basierte Planer stark erhöht, wenn zu diesem Zweck Punkte hinzugefügt werden müssen. Eine dynamische Änderung der Arbeitsraumauflösung (z.B. lokal) ist nicht vorgesehen, beziehungsweise zu teuer. Durch die fortwährende Abbildung der Objekte auf den gesamten Graphen werden unnötige Informationen berechnet, da nie der ganze Graph für die Planung verwendet wird. Durch die Dynamik des Arbeitsraums sind diese Daten vor Erreichen der entsprechenden Knoten bzw. Kanten mit hoher Wahrscheinlichkeit ungültig geworden. In der Grundvariante des Algorithmus darf der Roboterarm nicht in seiner Form variieren, wie dies z.B. bei dem Transport von Objekten im Greifer der Fall wäre. Die Verwendung mehrerer Tabellen wäre in diesem Fall eine Lösung, würde jedoch zusätzliche Kosten erzeugen.
- **Echtzeitfähigkeit**: Tabellenbasierte Planer sind nicht echtzeitfähig, denn für viele Objekte ergibt sich eine lange Laufzeit, bei wenigen eine kurze, denn für die Berechnung einer kollisionsfreien Bahn muss die gesamte Umweltinformation transformiert werden, sodass hier nicht im Sinne einer Echtzeitfähigkeit abgebrochen werden kann, der Algorithmus also nicht anytime-fähig ist.

Ein Vergleich von tabellenbasierter Methode und RRTs („Rapidly exploring random trees“, [LaValle98]) in [Kallmann04] kommt zu dem Fazit, dass die tabellenbasierte Methode benachteiligt ist mit zunehmender Anzahl an Knoten und feinerer Diskretisierung des Arbeitsraumes, von dessen Gittergrößen Speicherplatzbedarf, Vollständigkeit und Laufzeit abhängen. Von diesen Parametern kann allerdings das Auffinden einer Lösung für ein Planungsproblem abhängen. Ein Cache-basierter Ansatz (ähnlich zu [Jaillet04]), der

lediglich die häufig verwendeten Relationen enthält, wäre günstiger, da er in der Größe begrenzt werden kann; er wäre jedoch nicht vollständig und müsste daher mit einem nicht tabellenbasierten Kollisionstest kombiniert werden.

Nicht tabellenbasierte Ansätze

Im Folgenden sollen nun einige Ansätze zur echtzeitfähigen Planung in hochdimensionalen Räumen vorgestellt werden, die nicht mit einer tabellenbasierten Transformation von Arbeitsraumhindernissen arbeiten, sondern die Belegung des Konfigurationsraums implizit über die Ausführung des Kollisionstests berechnen. Grundlage der im Folgenden vorgestellten Ansätze ist ein Graph im Konfigurationsraum. Die Knoten repräsentieren die Roboterkonfigurationen an diesen Stellen, die Kanten verbinden die Knoten und repräsentieren je nach Ansatz die kollisionsfreie direkte Verbindung zwischen den Knoten über eine Linearbewegung im Konfigurationsraum oder die prinzipielle Möglichkeit, kollisionsfrei zwischen den jeweiligen Knoten verfahren zu können. Im letzteren Fall planen lokale Planer die Bewegung zwischen den Knoten.

Abseits von konkreten graph-basierten Algorithmen soll im Folgenden kurz die Suche des kürzesten Weges auf den jeweiligen Graphen resümiert werden durch Darstellung der Standard-Graphensuche mittels A*-Algorithmus ([Hart68], siehe Algorithmus 3.1). Je nach Hindernissituation und Graphgröße (gemessen in der Anzahl der Knoten und Kanten) kann die Anzahl der durch diesen Algorithmus ausgeführten Kollisionstests für die such-relevanten Teile des Graphen sehr hoch sein, wodurch dieser Algorithmus sehr rechenaufwendig ist. Die Kollisionstests fallen bei der Berechnung von Kantenkosten an ((Wieder-)Einfügen in die OPEN-Liste, Zeilen 10, 12 und 14). Bei Platzierung der Knoten auf einem regelmäßigen Gitter im Konfigurationsraum kann je nach Dimensionalität und Abtastung dieses Raumes der Graph eine hohe Anzahl von Knoten enthalten. Für diese festen Gitter können hierarchisierende Raumaufteilungen verwendet werden, welche je nach Hindernissituation die Auflösung des Gitters anpassen. Diese Ansätze benötigen jedoch Distanzberechnungen zum nächsten Hindernis und reduzieren die zugrundeliegende Komplexität nicht ausreichend, sodass diese Verfahren für die Offline-Planung eingesetzt werden [Hein03]. Die Reduktion der Kollisionstestanzahl spielt also für die Verwendung der graphbasierten Planungsverfahren eine wichtige Rolle.

```
(1)  searchGraphAStar(v_s, v_z)
(2)      CLOSED = {v_s}  // Startknoten ist automatisch in CLOSED
(3)      OPEN = {v_s}  // Startknoten muss als erstes expandiert werden
(4)      while OPEN is not empty
(5)          v = first(OPEN)  // kostengünstigsten Knoten aus OPEN holen
(6)          pop(OPEN)        // und aus OPEN entfernen
(7)          if v == v_z exit(CLOSED)   // SUCCESS case
(8)          forall v_i in Succ(v)  // Nachbarknoten expandieren
(9)              if v_i is not in CLOSED  // Nachbarknoten noch nicht bearbeitet?
(10)                 Insert(v_i, OPEN)
(11)                 Insert(v_i→v, CLOSED)
(12)             elseif  g(v_i) + k(v, v_i) < g(v_i)  // Günstigeren Weg gefunden?
(13)                 Redirect(v_i→v, CLOSED)
(14)                 Reopen(v_i, OPEN)
(15)     return FAILURE
```

Algorithmus 3.1: A-Graphensuche in einem Graphen mit Startknoten* $\mathbf{v}_s$ *und Zielknoten* v_z*. Der Algorithmus baut eine Baumstruktur („Suchbaum") **CLOSED** mit* v_s *als Wurzel auf, bei der jeder Knoten des Baumes auf seinen Vater zeigt. Zum Ende der Suche wird vom kostengünstigsten Blatt aus, das den Zielknoten* v_z *repräsentiert, über die Vaterknoten bis zur Wurzel iteriert und somit ein Pfad generiert. **OPEN** ist eine nach den Kosten der Knoten aufsteigend sortierte Menge von Knoten. Die Kosten eines Knotens bestimmen sich zu:* $f(v)=w\cdot g(\mathbf{v})+(1-w)\cdot h(\mathbf{v})$*, mit den bisherigen Wegkosten g(v), die rekursiv berechnet werden können, und den geschätzten Kosten h(v) zum Ziel, und einer heuristischen Gewichtung w. Succ(v) liefert die Nachbarknoten von v im Graph. Die Funktion Insert(*v_i*→v, **CLOSED**) fügt den Knoten* v_i *in den Suchbaum **CLOSED** ein mit einer Referenz auf den Vaterknoten v. Redirect(*v_i*→v, **CLOSED**) ändert den Vaterknoten des Knotens* v_i*. Reopen(...) löscht den gegebenen Knoten* v_i *aus der Knotenmenge **OPEN**, falls dieser darin enthalten ist und fügt ihn neu ein (Die Details des Algorithmus finden sich in [Hart68]).*

Eine Reduktion des Kollisionstestsaufwands kann prinzipiell durch Auftrennung der Kollisionstests in einen offline berechneten Teil und einen online berechneten Teil geschehen. Hierbei ist Wissen über den statischen Teil der Umgebung notwendig und eine Repräsentation, in der dieses Wissen assoziativ oder implizit gespeichert werden kann, wie dies beispielsweise in den Knoten und Kanten eines Graphen geschehen kann, der nur Knoten und Kanten enthält, die in Bezug auf die offline bekannten Objekte kollisionsfrei sind (siehe beispielsweise [Jaillet04]).

Durch Randomisierung des zugrundeliegenden Graphen kann eine Reduktion von Konfigurationstests erreicht werden, da die räumliche Abtastung des Konfigurationsraumes mittels der Knoten des Graphen lokal reduziert werden kann, wenn große zusammenhängende kollisionsfreie Raumbereiche mit geringerer Auflösung abgetastet werden können (siehe hierzu auch Kapitel 3.6). Die Knoten des Graphen werden (pseudo-)zufällig im Konfigurationsraum gewählt und dann mit einer Auswahl von Nachbarknoten verbunden, soweit die jeweilige Verbindung kollisionsfrei ist. Dieser Ansatz wird realisiert durch die grundlegenden Bahnplanungsmethoden Probabilistic RoadMap (PRM) [Kavraki96] für multiple Anfragen (Multi-Query-Planer) und Rapidly-exploring Random Trees (RRT) [LaValle98, Lavalle00] für einmalige Anfragen (Single-Query-Planer). PRMs bauen dazu einen statischen Graphen auf und verbinden für eine Anfrage Start- und Ziel-Knoten mit diesem Graphen und suchen dann mittels Standard-Graphensuchverfahren wie z.B. A* nach dem kürzesten Weg.

RRTs konstruieren für jede Anfrage zwei kollisionsfreie Bäume ausgehend von Start- und Zielknoten bis diese kollisionsfrei verbunden werden können, woraus die Lösung direkt bestimmt werden kann. Die erst-mögliche Verbindung der beiden Bäume ist dabei im Allgemeinen nicht die „Beste“ bezogen auf die Weglänge oder andere Kriterien. Die Single-Query-Methode RRT kann durch ihre Vorgehensweise prinzipiell nicht die Information über die statischen bekannten Objekten (Teilmenge von $\boldsymbol{Z}_k$) ausnutzen, da für jede Planung die Baumstrukturen größtenteils neu aufgebaut werden. Multi-Query-Planer (PRM) hingegen ermöglichen den Aufbau und die Wiederverwendung einer Roadmap, welche die Kollisionsinformationen mit bekannten statischen Objekten kodiert (B4).

Für die graphbasierten Verfahren lässt sich die Anzahl der notwendigen Kollisionstests noch weiter reduzieren, wenn die Techniken des sogenannten „Lazy PRM“ [Bohlin00] Anwendung finden. „Lazy PRM“ wird im folgenden auch als „**L**azy **C**ollision **C**hecking“ bezeichnet, da LCC auch mit RRTs verwendet werden kann [Bekris07]. Während der Suche auf dem Graphen oder während der Expansion der Bäume werden keine Kollisionen getestet. Wenn ein Pfad durch den Graphen/Baum von Start zu Ziel gefunden wurde, wird der Pfad auf Kollision getestet und dann kollidierende Kanten bzw. Knoten markiert bzw. entfernt und die Planung von neuem angestoßen. Dies iteriert bis ein kollisionsfreier Pfad zum Ziel gefunden wird. Hierbei sind deutliche Reduktionen der Anzahl an Kollisionstests die Folge. Es findet insgesamt eine Reduktion der Planungszeit statt, wenn der Rechenaufwand für den Planungsschritt im Verhältnis zum Aufwand für den Kollisionstest gering ist. Dennoch ist durch die hohe Anzahl an Iterationen die Rechenzeit für die Graphensuche nicht zu vernachlässigen. Eine Verbesserung wird hierbei durch A*-Varianten („D* lite“) erreicht, die den Suchbaum im Graphen nicht in jeder Iteration neu aufbauen, sondern lediglich Anpassungen durchführen, die sich durch veränderte

Kantengewichte ergeben [Koenig02]. Bei Kollision werden Kantengewichte dabei auf unendlich gesetzt und die dadurch verursachte Änderung von Wegkosten durch den Graphen propagiert. Rein prinzipiell ist das LCC nicht echtzeitfähig, da der Algorithmus in seiner Reinform erst dann terminiert, wenn ein vollständig kollisionsfreier Pfad vom Start zum Ziel gefunden wurde.

Diskussion und Schlussfolgerungen

Trotz Echtzeitfähigkeit sind lokale Planer im Hinblick auf Verfügbarkeit und Anwendungsdomäne hier nicht anwendbar. Die Verknüpfung mit globalen Planern löst das Problem der langsamen globalen Planung nicht und kann zu stark suboptimalen Pfaden führen, wenn sie nicht an die Arbeitsraumdynamik angepasst wird. Für globale Planer muss grundlegend dem Problem der Kollisionstestanzahl begegnet werden, welches durch randomisierte Graphen, Auftrennung des Kollisionstest in einen Offline- und Online-Teil und die Anwendung von Lazy-Collision-Checking geleistet wird. Die Vorausberechnung aller nötigen Kollisionstests durch Transformation des Arbeitsraums über Tabellen ist aufgrund des hohen Rechenaufwands für höhere Auflösungen und Graphengrößen nicht verwendbar und kann zudem keine Distanzinformationen liefern.

Für die PRM-basierten LCC-Algorithmen muss für die Realisierung der Echtzeitfähigkeit eine Modifikation entwickelt werden, welche die Planung mit dem jeweils vorliegenden Pfad abbricht. Dies ist möglich, da der Algorithmus am Ende jeder Iteration zu einem Pfad kommt, für dessen Kollisionsfreiheit auf der Gesamtlänge jedoch nicht garantiert wird. Diese Überlegung fließt in Teile des hier entwickelten Konzepts ein (siehe Kapitel 3.3) und noch weitergehende Überlegungen, die die weiter oben erwähnten Kritikpunkte von LCC-Planern angehen.

Wenn Planung und Ausführung des Plans parallel ablaufen sollen, teilt sich eine Roboterbewegung von Start v_s zu Ziel v_z aus der Perspektive des Bahnplaners betrachtet in eine Sequenz von Planungen von Zwischenpunkten v_i nach v_z ein. Hier muss insbesondere für statische Hindernissituationen (A6 (3)) eine Persistenz der Kollisionstestinformation in den Knoten und Kanten von Planung zu Planung gewährleistet werden, um unnötige Mehrfachberechnungen für neue Planungssituationen zu vermeiden. Gleichzeitig müssen in dynamischen Hindernissituationen vor jeder Planung Kollisionstestinformationen verworfen werden, um die Verfügbarkeit zu gewährleisten und keine Kanten zu blockieren, die aufgrund der veränderten Umwelt nicht mehr kollidieren. Bestehende Ansätze betrachten dies nicht, es sei denn, es findet eine vollständige Transformation des Arbeitsraums in den Konfigurationsraum statt wie bei tabellenbasierten Planern. Im Allgemeinen werden die Informationen aus vorherigen Planungszyklen vollständig verworfen zum jeweiligen Start eines Planungszyklus. Das hier entwickelte Konzept soll auch hierfür Ansätze entwickeln

und untersuchen. Single-Query-Planer sind aus dieser Überlegung prinzipiell nicht verwendbar, da sie aus ihrer Funktionsweise heraus zumindest einen großen Teil der gesammelten Information für einen neuen Planungszyklus komplett verwerfen und durch den fehlenden statischen Graph auch aus der Information über statische bekannte Objekte keinen Gewinn ziehen können.

3.3 Konzept

In den folgenden Abschnitten werden die grundlegenden Aspekte des in [Gecks07] dargestellten Planerkonzepts aufgeführt. Einzelaspekte wie Revalidierungsstrategien (Kapitel 3.4), Objekttransport (Kapitel 3.5) und Sampling-Strategien (Kapitel 3.6) werden in den jeweiligen Kapiteln detaillierter behandelt.

Statischer Graph

Um das Wissen über die statischen bekannten Objekte $\boldsymbol{Z}_k$ zu verwerten (Bedingung B4), wird ein statischer Graph für einen Multi-Query-Planer konstruiert. Die Vorgabe der hohen Anzahl an Freiheitsgraden legt die Verwendung von randomisierten Verfahren nahe, da diese mit Erfolg in Online-Planungsverfahren unter dieser Voraussetzung eingesetzt werden (siehe Stand der Forschung und Anforderung A3).

```
(1)  initGraph(G)
(2)      while( number of vertices in G < N )
(3)          v = sampleVertex()
(4)          if( not Coll_S(v) ) addVertex(G, v)
(5)      forall v in V
(6)          connect(v, k)
(7)          if(v has no edges) removeVertex(G, v)
```

Algorithmus 3.2: Konstruktion des statischen Graphen G mit einer maximalen Anzahl an Knoten N

Die Basis für den hier dargestellten Planer bildet daher ein Graph G mit einer Menge an Kanten $\boldsymbol{E}$ (edges) und Knoten $\boldsymbol{V}$ (vertices), der mittels des Kollisionstests auf statischen bekannten Objekten $\boldsymbol{Z}_k$ erstellt wird (siehe Algorithmus 3.2). Für das Sampling der Knoten in Zeile 3, Funktion *sampleVertex*(), können verschiedene Verfahren verwendet werden, die im Überblick in [Geraerts03] beschrieben werden. Hierbei ist für die gegebene Aufgabenstellung zu beachten, dass widerstrebende Ziele existieren: Einerseits ist für Pick-und-Place Aufgaben eine Notwendigkeit zur Planung in engen Passagen gegeben (bezogen auf die statischen bekannten Objekte), andererseits sollen auch große Freiräume in der statischen Umwelt mit einer ausreichenden Menge an Knoten abgetastet werden, um für die

Umplanungen um dynamische unbekannte Objekte genügend Pfad-Alternativen zur Verfügung zu stellen, da diese sich im diesem Freiraum aufhalten müssen. Daher sind nicht-uniforme Sampling-Strategien, die Knoten in der Nähe von Konfigurationsraum-Hindernissen platzieren ungünstig, da die dynamischen unbekannten Objekte zum Zeitpunkt der Konstruktion des Graphen noch nicht bekannt sind. Zudem sind auch die Besonderheiten des Sampling von Knoten während der Planung zu berücksichtigen (siehe hierzu Kapitel 3.6).

$Coll_S(v)$ (Zeile 4) repräsentiert den Kollisionstest mit statischen bekannten Objekten. Der für die Konstruktion des Graphen verwendete Kollisionstest greift auf ein dreidimensionales Umweltmodell zurück, wie es beispielsweise aus CAD-Daten der Roboterarbeitszelle erstellt werden kann. Die Laufzeit des Kollisionstest ist für die initiale Erzeugung des Graphen zwar weitgehend irrelevant, da diese offline geschieht, während der Online-Planung kann es jedoch vorkommen, dass neue Punkte in den Graphen eingefügt werden müssen, wie z.B. für unbekannte Start- und Zielpunkte der Roboterbewegung oder falls durch dynamische unbekannte Objekte kein Weg zum Ziel existiert. Daher ist es sinnvoll für diesen Kollisionstest bekannte schnelle Verfahren zu verwenden [Lin04].

Während des Graphenaufbaus können auch bekannte Objekte berücksichtigt werden, die während des Prozesses durch den Roboter transportiert werden und damit den Kollisionstest $Coll_S(v)$ verändern. Die Details hierzu werden in Kapitel 3.5 näher erläutert.

Die Funktion *connect*(v, k) verbindet den Knoten v mit maximal k seiner Nachbarknoten v_i. Die Kollisionsfreiheit der so entstehenden Kanten wird mittels $Coll_S(v, v_i)$ getestet, hierbei können die transportierten Objekte berücksichtigt und vermerkt werden. Alle Knoten, die nicht mit den Nachbarknoten verbunden werden konnten (Zeile 7), werden nicht in den Graphen aufgenommen.

Dynamischer Graph

Als semantische Schicht liegt über dem statischen Graphen ein dynamischer Graph (Abbildung 3.2). Dieser besteht aus einer Markierung an jeder Kante, die für diese die Auskunft gibt, ob sie für die Planung verwendet werden darf (die Kante ist *validiert*), oder nicht (die Kante ist *invalidiert*), weil sie als mit einem unbekannten Objekt $\boldsymbol{H}_l$ kollidierend betrachtet wird. Diese Markierung hängt nicht direkt mit der tatsächlichen Kollision der Kante mit einem unbekannten Objekt zusammen, dass heißt, dass auch tatsächlich keine Kollision vorliegen kann, obwohl die Kante invalidiert ist, und dass auch eine Kollision vorliegen kann, obwohl die Kante validiert ist. Diese Information wird zur Laufzeit durch entsprechende Kollisionstests bestimmt. Initial sind alle Kanten validiert.

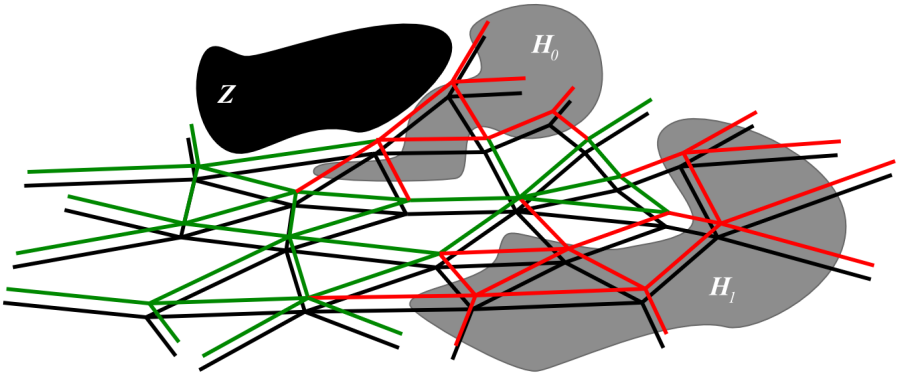


Abbildung 3.2: Illustration der semantischen Schichten des Planungsgraphen mit statischer Schicht in Schwarz und dynamischer, überlagerter Schicht in Grün (Freiraum, validiert) und Rot (kollisionsbehaftet, invalidiert). Dargestellt sind auch ein die Konfigurationsraumrepräsentation eines statischen bekannten Objekts $\boldsymbol{Z}$ *und von unbekannten Objekten* $\boldsymbol{H}_0$ *und* $\boldsymbol{H}_1$.

In die Graphensuche werden nur die validen Kanten einbezogen. Die Invalidierung einer Kante ist über Systemzyklen hinweg persistent, bis sie durch Revalidierungsmechanismen (Kapitel 3.4) wieder zurückgesetzt (validiert) wird. Die Speicherkomplexität für diese Schicht ist linear in der Anzahl der Kanten (O(|$\boldsymbol{E}$|)) mit kleinen Konstanten.

Parallele Planung und Ausführung

In Algorithmus 3.3 wird der Ablauf der parallelen Planung und Ausführung (Anforderung A4) in Kurzform dargestellt. Hier ist dabei der Ausschnitt aus dem gesamten Systemzyklus dargestellt, der in Abbildung 1.8 als Bahnplanung gekennzeichnet ist. Die Vorverarbeitungstufen für den bildbasierten Kollisionstest (Kapitel 2) seien bereits durchgeführt, bevor die Bahnplanung begonnen wird.

```
(1)  planMain(q_z)
(2)      checkEdgeRevalidation()
(3)      calcAndSetRobotSpeed()
(4)      if speed is zero return
(5)      v_ps = projectRobotPosition(P_i)  // project robot along path P_i,  i∈ℕ
(6)      P_{i+1} = plan(v_ps, v(q_z), P_i)
(7)      if P_{i+1} is not empty
(8)          commandPath(P_{i+1})
(9)      i = i+1
```

Algorithmus 3.3: Hauptschleife der Planungskomponente des Systems, $\boldsymbol{q}_z$ *wird durch das Roboterprogramm vorgegeben*

Die Funktion *planMain*($\boldsymbol{q}_z$) wird in jedem Systemzyklus genau einmal ausgeführt. Als Argument erhält sie die aktuelle Zielkonfiguration $\boldsymbol{q}_z$ aus dem laufenden Roboterprogramm. Zu Beginn werden invalidierte Kanten des dynamischen Graphen daraufhin überprüft, ob

sie revalidiert werden können (Zeile 2). Anschließend folgt die Berechnung und das Setzen der Robotergeschwindigkeit basierend auf der Entfernung zum nächsten unbekannten Objekt (Zeile 3). Bei Geschwindigkeit Null findet keine weitere Planung statt (Zeile 4). Andernfalls (Zeile 5) wird der Startknoten v_{ps} für die Planung auf dem Graph G durch die Funktion *projectRobotPosition*(P_i) mit Hilfe des aktuellen Pfades P_i berechnet, wie im Folgenden beschrieben. Initial (für $i = 0$) ist P_i leer.

Wie in Abbildung 3.3 dargestellt, befindet sich der Roboter zum Zeitpunkt des Aufrufs der Funktion *projectRobotPosition*(P_i) an der Konfiguration $\boldsymbol{q}_c$. Falls der Roboter keinen Pfad aus vorhergehenden Planungszyklen verfolgt, P_i also leer ist, steht er an $\boldsymbol{q}_c$ still und damit ist $v_{ps} = v(\boldsymbol{q}_c)$ zu wählen. $v(\boldsymbol{q})$ ist hierbei eine Funktion, die für eine gegebene Konfiguration $\boldsymbol{q}$ einen existierenden Knoten des Graphen liefert oder diesen erzeugt, falls sich an dieser Stelle im Graphen kein Knoten befindet (oder in einer ε-Umgebung). Die Funktion $v(\boldsymbol{q})$ verbindet außerdem einen eventuell erstellten neuen Knoten mit den Nachbarknoten und testet ihn und die zugehörigen Kanten auf Kollisionsfreiheit, wie in der Konstruktion des statischen Graphen beschrieben.

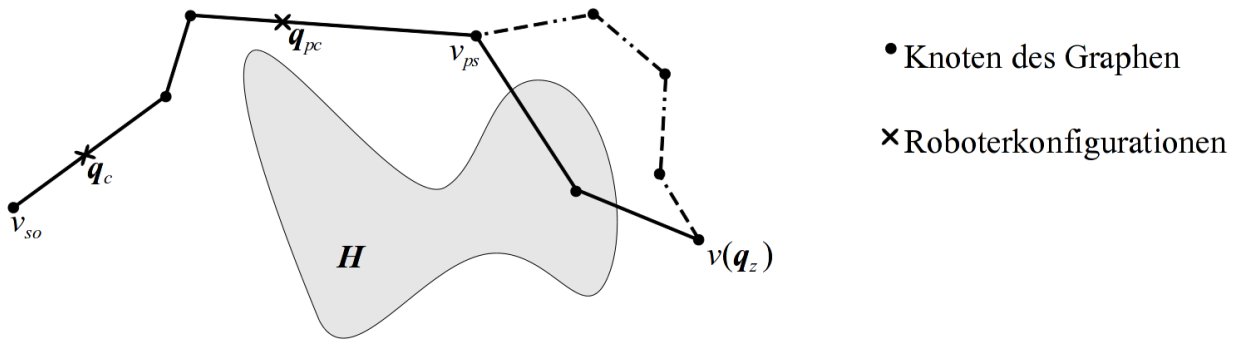


Abbildung 3.3: Beispielsituation mit Konsequenzen der parallelen Ausführung von Planer und Roboterbewegung mit Hindernis ***H****. Die durchgehende Linie beschreibt den aktuell vorliegenden und vom Roboter verfolgten Pfad von Knoten v_{so} nach $v(\boldsymbol{q}_z)$ (der Roboter ist an Position $\boldsymbol{q}_c$), die gestrichelte Linie kennzeichnet den veränderten Pfad von v_{ps} nach $v(\boldsymbol{q}_z)$ nach der Planung.*

Falls ein Pfad P_i aus Aufrufen des Planers in vorhergehenden Systemzyklen existiert von v_{so} nach $v(\boldsymbol{q}_z)$, fährt der Roboter aktuell diesen ab, mit der auf dem aktuellen Abstand zu unbekannten Objekten beruhenden Geschwindigkeit. Nach Ausführung der aktuellen Planung (Zeile 6) mit der Berechnungszeitdauer t_{plan} wird sich der Roboter von $\boldsymbol{q}_c$ nach $\boldsymbol{q}_{pc}$ bewegt haben. $\boldsymbol{q}_{pc}$ liegt in diesem Beispiel allerdings nicht auf einem Knoten des Graphen. Für den Startknoten der Graphensuche wird daher der nächstmögliche Knoten des Graphen im gegebenen Pfad P_i gewählt, dies ist v_{ps}. Es muss lediglich ab dieser Position geplant werden, denn zum Abschluss der Berechnungen hat der Roboter alle Graphknoten vor v_{ps} bereits erreicht und somit ist eine Veränderung der abgefahrenen Bahn erst ab diesem Graphknoten möglich, wenn die Bahn des Roboters auf dem Graph verlaufen soll. Die Funktion *projectRobotPosition*(P_i) behandelt im realisierten Prototyp eine Reihe von hier

nicht dargestellten Spezialfällen, die z.B. den Übergang von einem Planungsziel zum nächsten regeln.

In Zeile 6 von Algorithmus 3.3 wird die eigentliche Planung auf der dynamischen Ebene des statischen Graphen G durchgeführt von v_{ps} nach $v(\boldsymbol{q}_z)$, siehe auch Algorithmus 3.4. Die Funktion *commandPath*(P_{i+1}) übermittelt den Pfad P_{i+1} zur Ausführung an den Roboter, falls der Pfad nicht leer ist. Alle Zeilen außer Zeile 6 haben vernachlässigbare Laufzeiten, weshalb das Hauptaugenmerk im Folgenden auf der effizienten Realisierung der Funktion *plan*(...) liegt.

```
(1)  plan(v_ps, v_z, P_i)
(2)      P_local = searchGraph(v_ps, v_z)
(3)      if P_local is not empty
(4)          modifyPath(P_local, P_i, v_ps)
(5)          testPath(P_local)
(6)      else
(7)          if(not Coll_D(v_z)) sampleVertex()
(8)      return P_local
```

Algorithmus 3.4: Details der plan(v_{ps}, v_z, P_i)-Funktion

Die Funktion *plan*(v_{ps}, v_z, P_i) in Algorithmus 3.4 führt zunächst eine Graphensuche (Zeile 2) auf G durch mittels der Funktion *searchGraph*(v_{ps}, v_z) und liefert den Pfad P_{local}. Diese Graphensuche verwendet nur Kanten, welche validiert sind (s.o.).

Falls im Graph kein Pfad zum Zielknoten gefunden werden konnte, ist der Pfad P_{local} leer. In diesem Fall können neue Knoten durch die Funktion *sampleVertex*() eingefügt werden, um einen Weg zum Ziel zu finden (Zeile 7, siehe hierzu auch Kapitel 3.6). Das Einfügen neuer Knoten wird hier jedoch auf die Situationen beschränkt, in denen das Ziel selbst nicht kollisionsbehaftet ist (Bedingung in Zeile 7), da in diesem Fall selbst das Einfügen beliebig vieler neuer Knoten nicht die Erzeugung eines kollisionsfreien Pfad ermöglicht. Eine grundsätzliche Alternative zur Erzeugung neuer Knoten ist das Warten auf Revalidierung von Kanten beispielsweise durch Objektbewegungen (Kapitel 3.4).

Wenn ein Pfad im Graphen gefunden wurde, wird der neue Pfad durch die Funktion *modifiyPath*(P_{local}, P_i, v_{ps}), Zeile 4, erzeugt aus dem Knoten des aktuellen Pfades P_i unmittelbar vor v_{ps}, falls dieser Knoten existiert und dem Pfad P_{local} von v_{ps} nach v_z. Dieser neue Pfad wird zur weiteren Verarbeitung wieder P_{local} zugewiesen.

Der Pfad P_{local} wird dann mittels des Kollisionstests $Coll_D$(...) auf Kollisionen mit unbekannten Objekten $\boldsymbol{H}_l$ überprüft in der Funktion *testPath*(P_{local}) in Zeile 5. Die hierfür

nötigen Tests von Kanten können z.B. mit einer Adaption der Techniken aus [Henrich04] und [Baginski99] durchgeführt werden basierend auf einzelnen bildbasierten Kollisionstests. Die Prüfung der (Teilstücke der) Kanten beginnt ab $\boldsymbol{q}_{pc}$, der projizierten Roboterposition am Ende der aktuellen Planung, denn von hier aus bewegt sich der Roboter nach dem Ersetzen des alten Pfades durch den Neuen. Die Details und Spezialfälle der Prüfung in *testPath*(P_{local}) sollen im Folgenden kurz dargelegt werden (siehe hierzu auch Abbildungen 3.4 und 3.5 und Algorithmus 3.5).

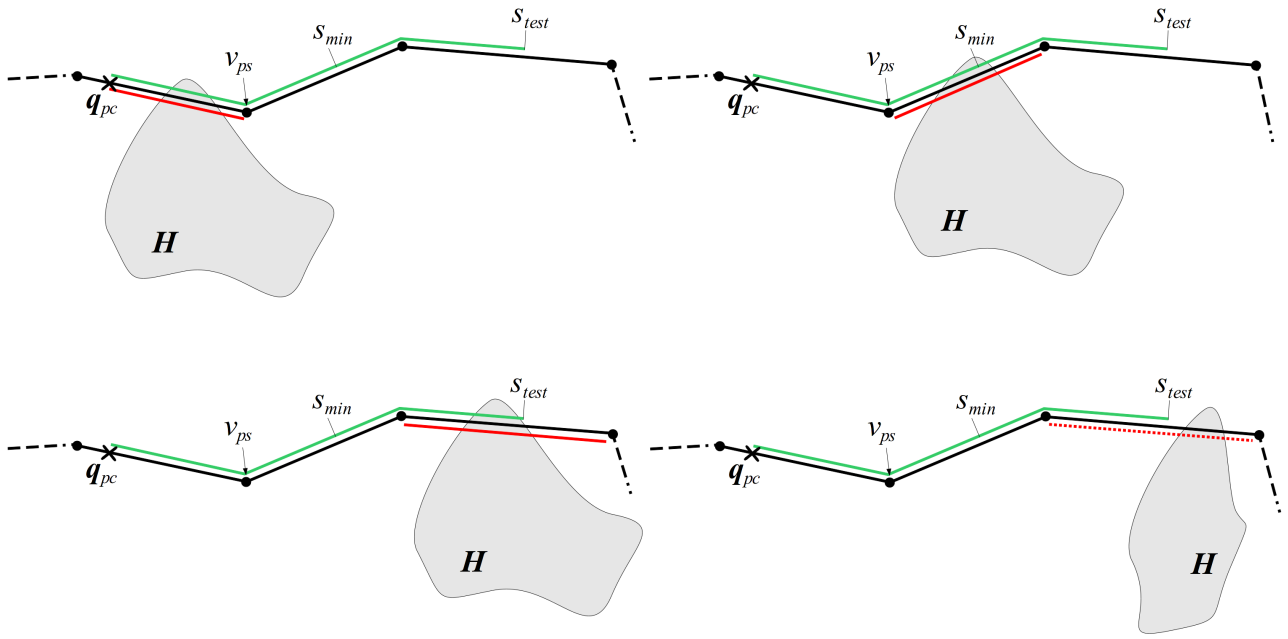


Abbildung 3.4: Grundlegende Fälle der Kollision mit einem unbekannte Objekt auf einer Kante des aktuellen Pfades von der Roboterkonfiguration $\boldsymbol{q}_{pc}$ aus. Diese Fälle werden in der Funktion testPath(P) entsprechend behandelt. Die grüne Linie markiert die getesteten Teile des Pfades von $\boldsymbol{q}_{pc}$ zur Markierung s_{test}.

Die Prüfung erfolgt über eine gewisse Strecke entlang des neuen Pfades, solange Kollisionstests zur Verfügung stehen (Algorithmus 3.5 Zeile 2). Diese Strecke ist in den in Abbildung 3.4 dargestellten Fällen durch die grüne Markierung von $\boldsymbol{q}_{pc}$ zur Wegmarke s_{test} hervorgehoben. Ein Test der Kanten muss mindestens über die Strecke erfolgen, die der Roboter bis zum Ende des nächsten Planungszyklus verfährt, denn sonst kann dieser Pfad nicht zur Ausführung angewiesen werden, da nicht bekannt ist, ob er mit einem Hindernis kollidiert. In Abbildung 3.4 ist diese Mindestlänge durch die Wegmarke s_{min} dargestellt. Alle Kanten, die als kollisionsbehaftet sind, werden invalidiert

In Zeile 4 des Algorithmus werden zwei Fälle betrachet, von denen hier zunächst der erstere erläutert werden soll. Falls der Pfad auf der Strecke bis s_{min} kollisionsbehaftet ist (Fälle Abb. 3.4 oben rechts und links), muss der Roboter angehalten werden (Zeile 5), da sonst eine Ausweichplanung mit Berücksichtigung der detektierten Kollision nicht möglich

wäre, da der Punkt der Kollision in diesem Fall vor dem Startpunkt der nächsten Planung liegen würde, also der Roboter die Stelle der Kollision erreicht, bevor die nächste Planung beendet ist. Anschließend wird der Pfad P_{local} gelöscht (Zeile 6), da in der nächsten Planung vom aktuellen Pfad keine Teile verwendet werden können. An der Stelle $\boldsymbol{q}_{pc}$ wird dann noch ein Knoten $v_{new} = v(\boldsymbol{q}_{pc})$ eingefügt (Zeile 7), sodass die Planung im nächsten Zyklus von dort aus starten kann.

```
(1)  testPath(P_local)
(2)      test and invalidate edges along path from q_pc to s_max
(3)      e_pc = edge that contains q_pc
(4)      if  (P_local is colliding on path from q_pc to s_min) or (e_pc is colliding)
(5)          stopRobot()
(6)          clear P_local
(7)          v_new = v(q_pc)
(8)          if e_pc is colliding
(9)              invalidate  e(v_new, v_ps)
```

Algorithmus 3.5: Details der testPath(P_{local})-Funktion

Falls eine Kollision auf der Kante auftritt, die $\boldsymbol{q}_{pc}$ enthält (zweite Bedingung in Zeile 4 und Abbildung 3.5), wird der Roboter auch gestoppt, der Pfad gelöscht und ein Knoten $v_{new} = v(\boldsymbol{q}_{pc})$ eingefügt. Wenn dieser Punkt mit seinen Nachbarn verbunden wird, muss dann jedoch eine eventuell existierende Kante von v_{new} nach v_{ps} ohne weiteren Test invalidiert werden, da sonst eine unnötige Zusatzschleife entsteht (Zeilen 8 und 9). Denn im nächsten Planungsschritt würde der Pfad die Kante von v_{new} nach v_{ps} eventuell enthalten, die dann wiederum zu invalidieren wäre, wenn sich das unbekannte Objekt nicht bewegt hat.

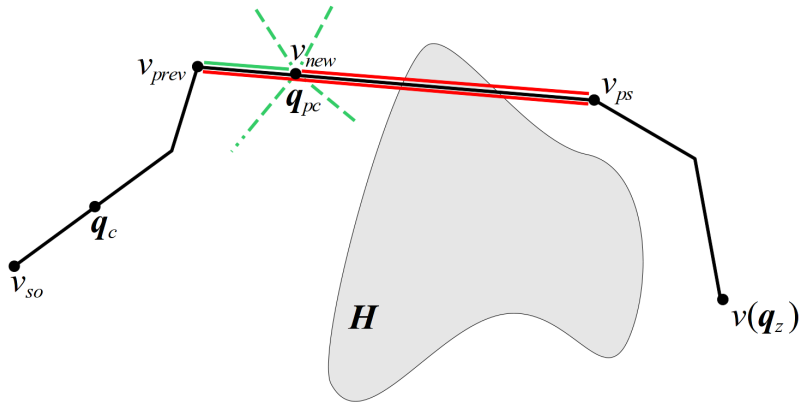


Abbildung 3.5: Kollision mit einem unbekannte Objekt auf der Kante von v_{prev} nach v_{ps}. Diese wird invalidiert (lange rote Markierung). Der Roboter muss gestoppt werden, da diese Kante nicht weiter verfolgt werden kann. Ein neuer Punkt wird an der Stelle $v_{new} = v(\boldsymbol{q}_{pc})$ eingefügt. Die Kante von v_{new} nach v_{ps} muss ohne weiteren Test ebenfalls invalidiert werden, da sie notwendigerweise ebenfalls kollidiert.

Echtzeitfähigkeit durch Kollisionstestbeschränkung

Die Latenz des Systems soll für eine sinnvolle Reaktions auf dynamische Objekte möglichst gering sein. In der Praxis hat sich eine maximale Zykluszeit von 100 ms bewährt, sodass mit einer Latenz von einem Zyklus auf äußere Ereignisse reagiert werden kann und bei einer gegebenen maximalen Objektgeschwindigkeit noch eine sinnvolle Verfahrgeschwindigkeit des Roboters erreicht wird. Der Pfadplaner soll wie oben dargestellt einmal pro Systemzyklus die Bahn an Veränderungen der Umwelt anpassen. Daher muss er innerhalb einer Laufzeit terminieren, die eine Einhaltung der vorgegebenen Latenz erlaubt. Abzüglich der Vorverarbeitungsschritte für den bildbasierten Kollisionstest bleiben daher etwa 50 - 70 ms pro Zyklus an Rechenzeit. Für einen Kollisionstest mit nicht zu vernachlässigenden Kosten (Kapitel 2.6) und einem Graphen realistischer Größe (ca. 4000 Knoten und 65000 Kanten) sind damit nicht alle Knoten des Graphen testbar und insbesondere nicht die Kanten, deren Kollisionstest im Durchschnitt deutlich länger läuft, da er mehrere einzelne Kollisionstests umfasst.

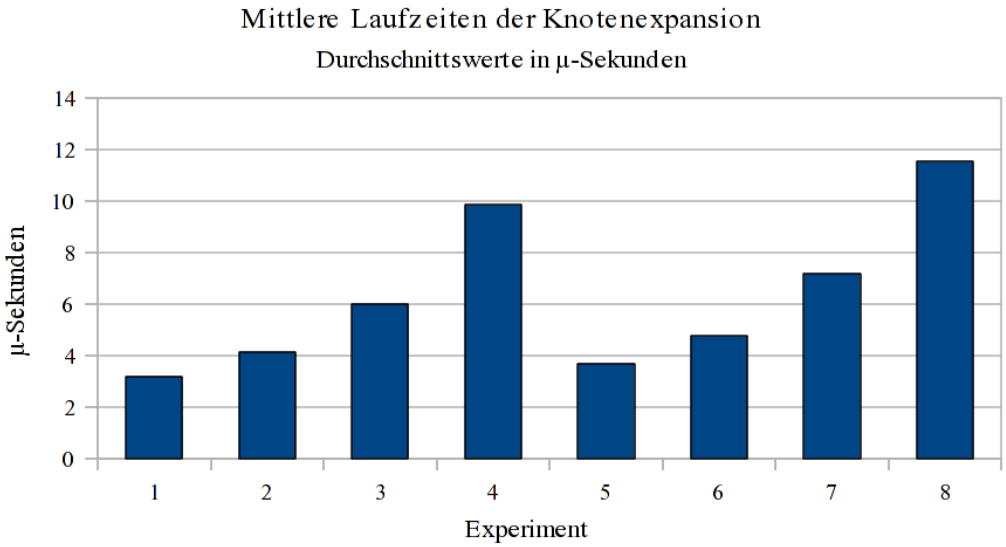


Abbildung 3.6: Durchschnittliche Laufzeit eines Expansionsschrittes des A (Algorithmus 3.1 Zeilen 8 - 14) in Abhängigkeit von der durchschnittlichen Anzahl Kanten pro Knoten: Experimente 1 - 4: Graph mit 4000 Knoten auf einem regulärem Gitter und durchschnittlich 11,76 , 15,81 , 23,99 bzw. 40,57 Kanten pro Knoten. Experimente 5-8: Graph mit 4000 Knoten und randomisierter Knotenplatzierung mit 12,38 , 16,76 , 25,32 bzw. 42,16 Kanten pro Knoten im Schnitt.*

Betrachtet man lediglich den Laufzeitanteil der reinen Graphensuche ohne Kollisionstests mittels des A*-Algorithmus, liegt die Laufzeit im Bereich von wenigen Millisekunden, selbst wenn der ganze Graph erfasst wird. Dies ist dadurch garantiert, dass der A*-Algorithums für $w \leq 0.5$ bei einer zulässigen Heuristik $h()$ jeden Knoten des Graphen nur einmal expandiert, und daher die Obergrenze für die Laufzeit t_{search} der Graphensuche bei N Knoten bei $t_{search} = N \cdot t_{exp}$ liegt, mit t_{exp} als mittlerer Laufzeit des Expansionsschrittes (für entsprechende Messwerte siehe Abbildung 3.6). Bei einem Graph

mit 4000 Knoten und ca. 16 Kanten pro Knoten ergibt sich aus den gemessenen Daten eine maximale Laufzeit von 16 ms. In der Praxis liegen die Werte deutlich darunter, da nur ein Teil der Knoten durch die Suche erfasst wird.

Da der Test aller Kanten und Knoten um ein Vielfaches teurer ist als die Ausführung der Graphensuche selbst, müssen die benötigten Kollisionstests minimiert werden. Zu diesem Zweck orientiert sich der hier eingesetzte Planungsalgorithmus am Konzept des Lazy-Collision-Checking (LCC). Bei LCC wird eine Kante/ein Knoten zunächst als kollisionsfrei betrachtet, bis sich die Notwendigkeit zum Testen der Kante/des Knotens ergibt und sich dann der Status ändern kann. Somit werden Kollisionstests zunächst vollständig vermieden.

Die Notwendigkeit zum Testen der Kanten liegt prinzipiell betrachtet allein vor der eigentlichen Ausführung, also nach Abschluss der Graphensuche, um eine drohende Kollision zu detektieren und eine Umplanung im nächsten Planungszyklus auszulösen. Die Sicherheit wird zwar durch die Geschwindigkeitsregelung gewährleistet, aber das darüber hinaus gehende Ziel der Bahnplanung ist es ja, einer drohenden Kollision durch Ausweichen zu begegnen und daher ist es sinnvoll, Kollisionen entlang der Bahn zu detektieren. Hier ist es nicht ausreichend, die Knoten zu testen, da die Kanten mit höherer Wahrscheinlichkeit kollisionsbehaftet sind.

Der Test eines ganzen Pfades vor der Ausführung ist für lange Pfade jedoch sehr aufwendig und kann unter den hier gegebenen Rahmenbedingungen nicht geleistet werden, ohne die typischerweise vorgesehene Laufzeit zu überschreiten. In dynamischen Arbeitszellen ist zudem eine zeitlich weitgehende Vorausschau entlang des Pfades nicht sinnvoll, da detektierte Kollisionen durch Bewegungen der unbekannten Objekte wieder zu Freiraum werden können, bis der Roboter die Position erreicht (der umgekehrte Fall gilt entsprechend). Aus diesen Gründen wird die Berechnung des Kollisionstests hier auf einen Teil des berechneten Pfades beschränkt ausgehend von der projizierten Roboterposition $\boldsymbol{q}_{pc}$ wie oben beschrieben. Die Anzahl der hierfür benötigten Kollisionstests ist limitiert auf *MAXCOLL*. Diese Anzahl sollte mindestens ausreichen, um Kollisionen zu detektieren, die entlang des Pfades soweit in der Zukunft liegen, dass eine Kollision erkannt wird, die im nächsten Planungszyklus zum Stillstand des Roboters führen würde. In Kombination mit der Graphensuche ergibt sich damit eine durch den Benutzer (nahezu) frei wählbare obere Grenze für Ausführungsdauer des Bahnplaners: $t_{PP} = MAXCOLL \cdot t_{coll} + t_{search}$ mit t_{coll} als Kollisionstestdauer und t_{search} als maximaler Laufzeit der Graphensuche, welche unabhängig ist von der Anzahl der Kollisionstests.

Je weiter der Pfad vor Ausführung auf Kollisionen überprüft wird, desto früher kann eine drohende Kollision festgestellt werden. Dieses grundlegende Konzept konzentriert die Kollisionsberechnungen jedoch in jedem Systemzyklus auf die Kanten des durch den Planer

gefundenen Pfades. Dies ist aus verschiedenen Gründen nicht wünschenswert wie im Folgenden diskutiert wird.

Kollisionstests in den Knoten

Grundsätzlich können die limitierten Kollisionstests sowohl auf die Kanten als auch auf die Knoten des Planungsgraphen verteilt werden. Die Kollisionstests entlang einer Kante erfordern eine hohe Anzahl von Kollisionstests an interpolierten, einzelnen Konfigurationen. Dadurch ergibt sich eine hohe Konzentration von teuren Berechnungen in einem kleinen Ausschnitt des Konfigurationsraumes. Geht man von einer gewissen Größe der transformierten Objekte $\boldsymbol{H}_l$ im Konfigurationsraum aus, ist eine breitere Verteilung der Kollisionstest sinnvoller, da die räumliche Ausdehnung der Objekte mit einer geringeren Anzahl an Testpunkten erfasst werden kann (siehe Beispiel in Abbildung 3.7). Der Test von Knoten auf Kollision kann schon während der Graphensuche erfolgen und diese gleichzeitig modifizieren, wie unten beschrieben. Wenn ein Knoten kollidiert, werden alle zugehörigen Kanten invalidiert.

Ein alleiniger Test von Knoten andererseits erfasst jedoch keine Kollisionen, die nur auf den Kanten auftreten, ohne dass die zugehörigen Knoten betroffen wären. Dies tritt vor allem bei langen Kanten auf. Der Test der Kanten vor der Ausführung des Pfades hat daher Berechtigung. Die hierfür zur Verfügung gestellten Kollisionstests müssen mit den für die Knoten verwendeten Kollisionstests abgewogen werden.

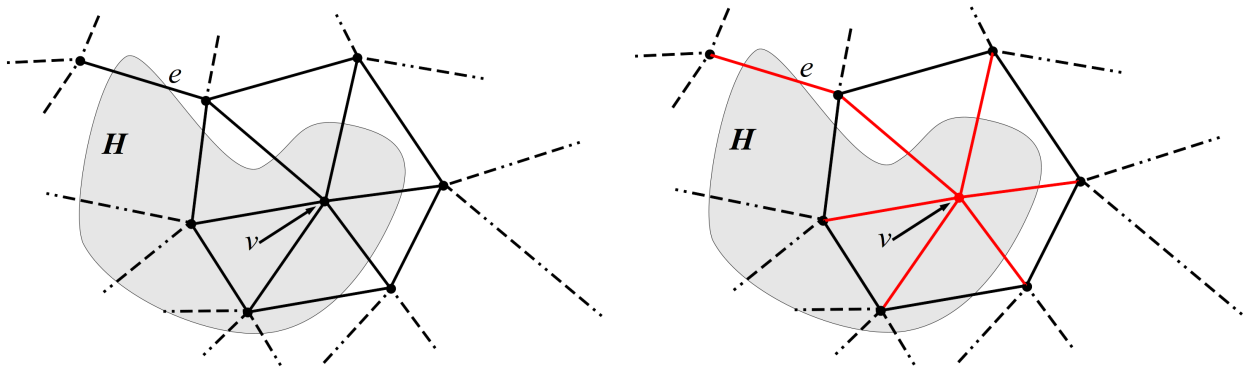


Abbildung 3.7: Kanten- und Knotentests. ***Links****: Statischer Graph mit zu testender Kante e und Knoten v. Alle Kanten und Knoten sind validiert (nicht kollisionsbehaftet). Ein dynamisches unbekanntes Objekt* ***H*** *nimmt den grauen Bereich ein.* ***Rechts****: Kante e ist kollisionsbehaftet und wird invalidiert. Knoten v ist ebenfalls kollisionsbehaftet und invalidiert sowie alle zugehörigen Kanten am Knoten v. Der Test des Knotens v erfasst hier mit einem Kollisionstest viele der Kanten, die zum Objekt gehören. Der Test der Kante e erfasst nur einen kleinen Teil des Objekts.*

Verteilung der Kollisionstests auf Graphsuche und Pfadtest

Wenn die limitierte Anzahl an Kollisionstests ausreicht, um entlang des berechneten Pfades zeitlich „weit“ in die Zukunft schauen zu können, ist es sinnvoll, für den Test des Pfades eine geringere Anzahl an Kollisionstests zur Verfügung zu stellen und die

Verfügbaren innerhalb der Graphensuche zu verwenden. Die Verteilung der Kollisionstests kann z.B. abhängig von der Robotergeschwindigkeit gestaltet werden. Die maximale Anzahl an Kollisionstests *MAXCOLL* wird also aufgeteilt in die Kollsionstests entlang der Kanten des Pfades vor Ausführung $MAXCOLL_{path}$ und in Kollisionstests während der Planung $MAXCOLL_{plan}$. Die maximale Ausführungsdauer ergibt sich somit zu:

$$t_{PP} = (MAXCOLL_{path} + MAXCOLL_{plan}) \cdot t_{coll} + t_{search} \tag{3.4}$$

Die Ausführungsdauer eines Kollisionstests ist zwar durch die Auflösung nach oben begrenzt, schwankt in der Realität jedoch stark bedingt durch die Pose des Roboters und der Anzahl der sichtbaren Objektpixel (siehe hierzu Abbildung 2.32). Daher ist es für die Praxis sinnvoll, mehr Kollisionstests vorzugeben, als im schlimmsten Fall machbar wären, und basierend auf einer Zeitmessung nach jedem Kollisionstest zu entscheiden, ob weitere Tests möglich sind. Dies gilt sowohl für $MAXCOLL_{plan}$ als auch für $MAXCOLL_{path}$.

Mit der Berechnung der Kollisionsinformation eines Knotens werden im Falle einer Kollision alle Kanten invalidiert, die mit diesem Knoten assoziiert sind (Abbildung 3.7), da diese Kanten schon kollidieren, wenn einer ihrer Endpunkte (oder beide) kollidiert. Der Aufwand ist hierbei erheblich geringer als er für das Testen all dieser Kanten wäre. Die verfügbaren Kollisionstests für die Graphensuche reichen jedoch im Allgemeinen nicht aus, um alle Knoten zu testen, die durch die Suche expandiert werden, von daher ist die Auswahl der zu testenden Knoten zu diskutieren.

Platzierung der limitierten Kollisionstests während der Graphensuche

Eine sehr naheliegende Einschränkung ergibt sich zunächst, wenn man Kollisionstests nur auf den Knoten durchführt, die für die aktuelle Graphensuche relevant sind (also alle Knoten, die durch die Graphensuche erfasst werden, dies sind zum Ende der Planung alle Knoten in ***OPEN*** und ***CLOSED***). Kollisionstests auf anderen Knoten des Graphen können offensichtlich keine Änderung am resultierenden Pfad erzeugen und sind daher nutzlos. Diese Einschränkung reicht jedoch als alleiniges Auswahlkriterium im Allgemeinen nicht aus, da je nach Hindernissituation eine große Anzahl von Knoten expandiert wird.

Zum einen ist es sinnvoll, in Gegenwart dynamischer unbekannter Objekte $\boldsymbol{H}_l$ denjenigen Knoten zu testen, der noch nicht getestet wurde und dessen minimaler Pfad im Graph zum Startknoten der Graphensuche am kürzesten ist. Dies ist sinnvoll, weil dessen Information zeitlich gesehen für die Bewegung des Roboters früher relevant wird, als die weiter entfernter Knoten. Hierin steckt die Annahme, dass zunächst für alle Pfade eine konstante Robotergeschwindigkeit gilt. Da die Distanzen zu den unbekannten Objekten entlang der Pfade hier unbekannt sind (siehe Kapitel 4 für Alternativen), ist dies eine sinnvolle Annahme. Die Distanz vom Startknoten der Suche wird durch die Pfadkosten $g()$ des A*-Algorithmus gemessen.

Weiterhin ist es sinnvoll, unter den Knoten kürzesten Weges diejenigen auszuwählen, die mit hoher Wahrscheinlichkeit Teil des resultierenden Pfades sind, sodass die Berechnung der Kollisionsinformation an dieser Stelle einen großen Einfluss auf den resultierenden Pfad hat. Dies sind alle Knoten, deren Weg zum Ziel kurz ist. Dieser Weg wird durch die Heuristik $h()$ des A*-Algorithmus geschätzt. Eine Kombination beider Kriterien spiegelt sich in der Kostenfunktion der A*-Suche wieder. Der günstigste Knoten aus ***OPEN*** ist somit die den obigen Kriterien entsprechende Wahl für den nächsten Kollisionstest (*Top-of-OPEN-Prinzip*).

Um die Distanzberechnungen weiter zu streuen, kann zusätzlich die Durchführung des Kollisionstests zufällig erfolgen. Weitere heuristische Methoden sind denkbar.

Persistenz von Kollisionstestinformationen

Die Kollisionstestinformationen, die in den Kanten und Knoten des dynamischen Graphen kodiert sind, werden von Systemzyklus zu Systemzyklus beibehalten, bis eine Revalidierung ausgelöst wird (Kapitel 3.4). In Arbeitszellen, die häufig lediglich statische unbekannte Objekte enthalten, können auf diese Art sehr viele Kollisionstests eingespart werden, denn für statische unbekannte Objekte ist die Kollisionstestinformation weiter gültig. Die resultierenden Pfade der Planung werden durch die in diesem Fall stetig zunehmende Umweltinformation immer weiter an den idealen Pfadverlauf approximiert, der sich bei vollständiger Umweltinformation ergibt. Zu diesem Zweck wird auf den Knoten des Graphen eine Markierung eingeführt, die die Berechnungsinformation trägt und drei Zustände umfasst: UNKNOWN für Knoten, für die keine Kollisionsinformation vorliegt, CALCED_FREE für kollisionsfreie Knoten und CALCED_COLLISION für kollisionsbehaftete Knoten. Initial besitzen alle Knoten die Markierung UNKNOWN. Während der Graphensuche werden lediglich Knoten auf Kollision getestet, die mit UNKNOWN markiert sind.

Die Kollisionsinformation auf den Kanten wird in zwei Zuständen gespeichert, die Kanten sind entweder validiert oder invalidiert. Invalidierte Kanten werden nicht in die Graphensuche miteinbezogen (Algorithmus 3.6, Zeile 16) und werden daher nicht mehrfach berechnet, bis sie wieder revalidiert wurden und darauf erneut in der Graphensuche erfasst werden. Validierte Kanten können durch den Test des Pfades vor der Ausführung potentiell ein weiteres mal getestet werden. Die wiederholte Überprüfung auszuführender Kanten ist damit begründet, dass die auszuführenden Kanten ausgezeichnet und von besonderer Bedeutung sind im Vergleich zu den anderen Kanten des Netzwerks und die Erkennung von Kollisionen vor der Ausführung für die Wahrung der Verfügbarkeit des Systems wichtig ist, um tatsächliche Kollisionen entlang des ausgeführten Pfades zu vermeiden. Hierfür soll daher immer die neueste Umweltinformation verwendet werden. Daher werden diese

Kollisionstests auch durchgeführt, wenn die Kanten in vorhergehenden Systemzyklen als nicht kollisionsbehaftet bestimmt wurden.

Graphensuche

Zur Umsetzung der oben aufgeführten Konzepte muss die Graphensuche mittels A*-Algorithmus geeignet angepasst werden. Die modifizierte A*-Graphensuche ist in Algorithmus 3.6 dargestellt. Die Unterschiede zu A* sind hervorgehoben.

Die maximale Anzahl der während der Suche zu verwendenden Kollisionstests wird durch $MAXCOLL_{plan}$ vorgegeben. Die verbrauchten Kollisionstests werden durch die Variable *collTestCount* mitgezählt. Der kostengünstigste Knoten aus ***OPEN*** wird in Zeile 8 daraufhin überprüft, ob eine Berechnung der Kollisionsinformation schon stattgefunden hat.

```
(1)  searchGraph(v_s, v_z)
(2)      collTestCount = 0
(3)      CLOSED = {v_s}
(4)      OPEN = {v_s} // OPEN is a set, ordered by costs of node
(5)      while OPEN is not empty
(6)          v = first(OPEN)
(7)          pop(OPEN)
(8)          if (state(v) is UNKNOWN) and (collTestCount <= MAXCOLL_plan)
(9)              testCollisionAndMarkVertex(v)
(10)             increment collTestCount
(11)             if v is colliding
(12)                 invalidate all associated edges
(13)                 continue
(14)         if v == v_g exit(CLOSED)
(15)         forall v_i in Succ(v)
(16)             if e({v, v_i}) is invalid continue
(17)             if v_i is not in CLOSED
(18)                 Insert(v_i, OPEN)
(19)                 Insert(v_i→v, CLOSED)
(20)             elseif  g(v) + k(v, v_i) < g(v_i)
(21)                 Redirect(v_i→v, CLOSED)
(22)                 Reopen(v_i, OPEN)
(23) return FAILURE
```

Algorithmus 3.6: Modifizierte A-Graphensuche mit Kollisionstests auf den Knoten*

Falls für den Knoten bisher noch kein Kollisionstest berechnet wurde und noch Kollisionstests zur Verfügung stehen, wird die Berechnung in Zeile (9) veranlasst und der Knoten anschließend mit einem der beiden Zustände markiert, abhängig von dem Ausgang des Kollisionstests (CALCED_FREE oder CALCED_COLLISION, s.o.). Falls eine Kollision vorliegt, müssen alle mit dem Knoten assoziierten Kanten invalidiert werden (Zeilen 11 – 13). Im Falle einer Kollision kann dieser Knoten nicht weiter durch den A* expandiert werden (alle Kanten sind invalidiert) und daher wird mit dem nächsten Punkt aus ***OPEN*** fortgefahren. Der Expansionschritt (Zeile 15 – 22) überspringt alle invaliden Kanten (Zeile 16). Die Funktion $e(\{v, v_i\})$ liefert die Kante, die über die beiden Endknoten eindeutig bestimmt ist (per Konstruktion sollen doppelte Kanten ausgeschlossen sein).

3.4 Kantenrevalidierung

Für das Ziel der Verfügbarkeit und der Effizienz des Robotersystems ist es notwendig, dass die durch die dynamischen unbekannten Objekte $\boldsymbol{H}_l$ invalidierten Kanten und Knoten nicht unnötig dauerhaft belegt sind. Dies würde primär dazu führen, dass die Effizienz durch unnötige Umwege reduziert würde und im schlimmsten Fall die Verfügbarkeit durch unerreichbare Zielknoten nicht mehr gegeben wäre, bis durch Hinzufügen von Knoten ein neuer Weg gefunden würde.

In diesem Kapitel werden Methoden vorgestellt, die zur Revalidierung invalidierter Kanten bzw. markierter Knoten verwendet werden können. Neben den Aspekten des Rechen- und Speicheraufwands werden verschiedene Kriterien für die Beurteilung der Leistungsfähigkeit für den gegebenen Anwendungsbereich eingesetzt.

Terminologie

Gegeben sei ein Graph $\boldsymbol{G}$ mit einer Menge von Knoten $\boldsymbol{V}$ und Kanten $\boldsymbol{E}$. Die Menge der berechneten Knoten $\boldsymbol{V}_i$ mit den Markierungen CALCED_FREE bzw. CALCED_COLLISION ist eine Teilmenge von $\boldsymbol{V}$ und die Menge der invalidierten Kanten $\boldsymbol{E}_i$ eine Teilmenge von $\boldsymbol{E}$.

Aufgabe der Revalidierungsmethode ist es nun, diejenigen Knoten in $\boldsymbol{V}_i$ und Kanten in $\boldsymbol{E}_i$ zu bestimmen, die zurückgesetzt werden, das heißt also in die Menge der unberechneten Knoten $\boldsymbol{V}\backslash\boldsymbol{V}_i$ mit Markierung UNKNOWN und validen Kanten $\boldsymbol{E}\backslash\boldsymbol{E}_i$ verschoben werden. Diese Bestimmung findet einmal pro Zyklus statt, bevor die Graphensuche gestartet wird (siehe Algorithmus 3.3 Zeile 2).

Grundsätzlich wäre es möglich, alle Knoten in $\boldsymbol{V}_i$ und Kanten in $\boldsymbol{E}_i$ zu Beginn jedes Zyklus erneut mithilfe des Kollisionstests zu überprüfen. Dies würde jedoch einen erheblichen Rechenaufwand erzeugen und ab einer gewissen Größe von $\boldsymbol{V}_i$ und $\boldsymbol{E}_i$ nicht mehr echtzeitfähig sein. Bei typischen Hindernissituationen und sinnvollen Graphgrößen

werden solche Größenordnungen in der Praxis schnell erreicht. Diese Strategien der Revalidierung durch erneute Kollisionstests ist also nicht anwendbar und wird daher im Folgenden nicht betrachtet.

Tabelle 3.2 listet eine Reihe von heuristischen Revalidierungsstrategien auf, weitere sind denkbar. Im Folgenden wird aus Gründen der Übersichtlichkeit lediglich von der Revalidierung von Kanten gesprochen, die Strategien werden für die berechneten Knoten jedoch analog angewandt.

Strategie	Vorteile	Nachteile
Keine Revalidierung	Kein Rechen- und Speicheraufwand	Ineffiziente Struktur des dynamischen Graphen in dynamischen Umgebungen
Ziel-Revalidierung	Sehr niedriger Rechen- und Speicheraufwand	Ineffiziente Struktur des dynamischen Graphen in dynamischen Umgebungen
Auszeit-Revalidierung	Niedriger Rechen- und Speicheraufwand, Adaption einer Objektgeschwindigkeit	Lokale Sackgassen auch bei statischen unbekannten Objekten
Sensorbasierte Revalidierung	Variable Adaption an die Arbeitsraumdynamik	Moderator Rechen- und Speicheraufwand

Tabelle 3.2: Vergleich von Revalidierungsstrategien

Die Strategie „**Keine Revalidierung**“ dient lediglich zu Vergleichszwecken und ist äquivalent zum Löschen von Kanten aus dem statischen Graphen. In dynamischen Arbeitsräumen führt diese Strategie zu ineffizienten, unnötigen Ausweichbewegungen, da Kanten und Knoten als kollisionsbehaftet markiert bleiben, selbst wenn sich das entsprechende Objekt von der Position entfernt, an der es die detektierte Kollision verursacht hat. Dadurch kann es häufig notwendig werden, neue Knoten einzufügen, da der Zielknoten nicht mehr erreichbar ist. Auch wird die Kollisionstestinformation bezüglich statischer bekannter Objekte, welche im statischen Graphen kodiert ist, bei dieser Strategie quasi verworfen und nicht wieder verwendet. Allerdings führt diese Strategie dazu, dass Bereiche, in denen sich Hindernisse auf halten, prinzipiell vermieden werden. Dies kann von Vorteil sein, wenn sich Hindernisse häufig in bestimmten Bereichen aufhalten und genügend Spielraum zur Planung bleibt, sodass der Nachteil des reduzierten Graphen nicht so stark ins Gewicht fällt.

Die Strategie „**Ziel-Revalidierung**“ revalidiert alle Kanten beim Erreichen des aktuellen Ziels der Planung. Damit gibt es im Vergleich zur Minimal-Strategie „Keine Revalidierung“ keinen zusätzlichen Speicheraufwand, da beim Erreichen des Ziels über alle Kanten iteriert wird und somit für die Auswahl der zu revalidierenden Kanten keine extra Datenstruktur benötigt wird. Durch die Iteration entsteht ein gewisser, jedoch sehr geringer Rechenaufwand. In dynamischen Umgebungen jedoch verspielt diese Strategie mögliche Revalidierungen von Kanten aufgrund von Objektbewegungen und erzeugt damit

ineffiziente Pfade. Durch die Revalidierung am Ende jeder Planung wird zu diesen Zeitpunkten eine Anpassung an dynamische Räume möglich und somit die Ineffizienz dieser Methode reduziert. In Umgebungen mit statischen unbekannten Objekten hat diese Vorgehensweise jedoch den Nachteil, dass sich Kollisionstests auf denselben Kanten und Knoten wiederholen für die wiederholte Planung einer identischen Start/Ziel-Knoten-Kombination, wie dies in industriellen Umgebungen typisch ist.

Eine bessere Adaption an dynamische Umgebungen hat die Strategie „**Auszeit-Revalidierung**“ zum Ziel. Hierbei werden Kanten nach einer fest vorgegebenen Zeitdauer t_{reval} in Sekunden ab dem Zeitpunkt ihrer Invalidierung revalidiert. Hierzu wird eine Datenstruktur in Form einer Schlange benötigt, die zusätzlich zu Kantenreferenzen den jeweiligen Invalidierungszeitpunkt enthält. Der Speicheraufwand ist daher begrenzt auf ((maximale Anzahl an invalidierbaren Kanten)/sec * t_{reval}). Der Rechenaufwand ist hierzu proportional, da er im wesentlichen aus der Verwaltung der Datenstruktur besteht. Die Länge von t_{reval} kann aus der erwarteten durchschnittlichen Bewegungsgeschwindigkeit der unbekannten Objekte $\boldsymbol{H}_l$ berechnet werden, die dann auch nicht breit streuen sollte. Ein zu kurzes t_{reval} wird in statischen Umgebungen mit großen unbekannten Objekten ein zyklisches Verhalten erzeugen, das einer lokalen Sackgasse entspricht. Dieses Verhalten entsteht, wenn durch Revalidierung von Kanten nach t_{reval} wieder Pfade zum Ziel ermöglicht werden, die zu früheren Zeitpunkten durch den Pfadplaner ausgewählt wurden. Diese Pfade werden durch die starre Revalidierungsstrategie wieder reaktiviert, obwohl sie aufgrund der statischen Hindernissituation nicht gültig sind, was dann auch bei einem erneuten Kantentest vor Ausführung festgestellt wird. So entsteht eine Schleife, die wie bei lokalen Planern in einer Sackgasse darauf beruht, dass das Wissen über die Umwelt begrenzt ist, in diesem Fall auf die maximale Größe der Schlange abhängig von t_{reval}. Diese Problematik lässt sich durch Erhöhung von t_{reval} reduzieren, führt jedoch zu einer Angleichung dieser Strategie an die „Ziel-Revalidierung“ und „Keine Revalidierung“ mit den genannten Problemen.

Um die Nachteile der vorigen Strategien bezüglich Arbeitsräumen mit stark wechselnder Dynamik zu verbessern, wird bei der Strategie „**Sensorbasierte Revalidierung**“ die Dynamik des Arbeitsraumes approximativ gemessen. Dies kann konservativ erfolgen, indem die Dynamik höher eingeschätzt wird, als sie ist, und in Konsequenz mehr Kanten revalidiert werden, als aufgrund der Bewegungen der Arbeitsraumhindernisse notwendig wäre. Wenn die völlige Abwesenheit von unbekannten Objekten im Arbeitsraum detektiert werden kann, ist eine automatische Anpassung von entsprechenden Parametern möglich, indem die verbleibende Restmengen an Kanten und Knoten in $\boldsymbol{V}_i$ und $\boldsymbol{E}_i$ betrachtet werden und die Parameter so angepasst werden, dass die Restmengen im Durchschnitt der leeren Menge entsprechen, wenn der Arbeitsraum leer ist.

Durch die vereinfachte Anforderung der approximativen Messung ergibt sich die

Möglichkeit Schätzverfahren einzusetzen, die lediglich moderaten Rechen- und Speicheraufwand erfordern. Obwohl diese Verfahren im Vergleich zu tabellenbasierten Verfahren die Anpassung von Knoten und Kanten suboptimal vornehmen, sind die Ergebnisse für reale Umgebungen überzeugend. Speziell wird so das Problem zyklischen Verhaltens bei statischen unbekannten Objekten umgangen. Für dynamische unbekannte Objekte werden die Kanten des Graphen schnell wieder revalidiert und bieten daher die Möglichkeit der effizienten Umplanung. Für Umgebungen mit einer ständigen Mischung statischer und dynamischer unbekannter Objekte kann sich jedoch weiterhin zyklisches Verhalten ähnlich der „Auszeit-Revalidierung“ ergeben, falls ein statisches unbekanntes Objekt $\boldsymbol{H}_l$ alleinig für die Bewegungsplanung zum Ziel relevant ist und damit die Invalidierung von Kanten und Knoten verursacht, ein dynamisches $\boldsymbol{H}_l$ jedoch die Revalidierung von Kanten und Knoten verursacht. In diesen Fällen kann die Dynamik des Raumes künstlich niedriger eingeschätzt werden über eine Anpassung entsprechender Parameter. Dieses Problem ist von geringer Relevanz in gemischt-dynamischen Umgebungen, in denen dynamische $\boldsymbol{H}_l$ größer als Statische sind, und daher seltener oder keine Fallen am statischen Objekt entstehen. Dies ist in der Praxis typisch, z.B. bei einem Arbeiter, der Wartungsarbeiten bei laufendem Prozess durchführt und dazu einen im Verhältnis zu ihm kleineren Werkzeugkoffer mitführt, der dann z.B. als statisches unbekanntes Objekt auf einem Tisch platziert wird. Im Folgenden wird ein einfaches Schätzverfahren beschrieben, welches für das im Rahmen dieser Arbeit realisierte System verwendet wird.

Dynamikschätzung

Die Dynamikschätzung kann direkt auf den Differenzbildern geschehen und verwendet dazu den Zeitverlauf und somit die Differenz der Differenzbilder. Über einen Schwellwert wird das Rauschen kompensiert. Da der Roboter maskiert wird, wird eine Roboterbewegung nicht als Objektbewegung erfasst.

Um starke Schwankungen der hier verwendeten Objektpixelmengen durch in den Verdeckungen rekonstruierte Objektpixel zu vermeiden, werden nur diejenigen Objektpixel betrachtet, die aktuell zu sehen sind, welche also nicht in den jeweiligen $\boldsymbol{U}_c(g)$ liegen. Dies geschieht vor allen Verarbeitungsschritten auf den Szenenbildern (siehe Kapitel 2.3).

Die Funktion *detectObjectDynamics*(I_{cur}) aus Algorithmus 3.7 nimmt das aktuelle Szenenbild I_{cur} als Aufrufparameter entgegen. Mit diesem Bild wird zunächst ein Filterbild I_f aktualisiert (Zeile 3). Dieses Filterbild besteht aus einem Zähler pro Pixel. Durch die Funktion *updateFilteredImage*(I_{cur}, I_f) wird der Zähler eines Pixels aus I_f inkrementiert, wenn beim entsprechenden Pixel aus I_{cur} der Pixel die Markierung OBJECT trägt und dekrementiert, wenn der Pixel die Markierung FREE trägt. Die Zählung ist in beide

Richtungen begrenzt.

```
(1)  detectObjectDynamics(I_cur)
(2)      subCount = 0
(3)      updateFilteredImage(I_cur, I_f)
(4)      foreach pixel p in I_cur
(5)          if(p is visible)
(6)              if(filtered(p, I_f) is FREE and ref(p, I_r) is OBJECT) increase subCount
(7)      updateReferenceImage(I_f, I_r)
(8)      return subCount
```

Algorithmus 3.7: Heuristische Bestimmung der Objektdynamik

Alsdann wird für jeden Pixel p des gefilterten Bildes I_f bestimmt, wie und ob er sich im Vergleich zu dem korrespondierenden Pixel eines Referenzbildes I_r verändert hat (Zeilen 4 – 6), dies unter der Prämisse, dass das Pixel nicht verdeckt ist (Zeile 5). Die Funktion *filtered*(p, I_f) liefert über eine Hysterese-Funktion den Wert OBJECT oder FREE für jedes Pixel des gefilterten Bildes basierend auf den Zählerständen und dem aktuellen Zustand. Die Hysterese realisiert im Zusammenspiel mit den Zählern eine zeitliche Integration der Belegungsinformationen, durch die das Rauschen verringert wird. Die Funktion *ref*(p, I_r) liefert den Wert des Pixels p im Referenzbild. Die Bedingung in Zeile 6 beschreibt Objektpixel, die zu Freiraum werden. Diese zeigen an, dass Kanten revalidiert werden sollten, da Objekte sich aus ihren bisherigen Positionen entfernt haben. Daher wird die Variable *subCount* inkrementiert, die die Anzahl dieser Art von Pixeln sammelt. Abschließend (Zeile 7) wird mit den aktuellen Werten, die durch *filtered*(p, I_f) geliefert werden, das Referenzbild aktualisiert in der Funktion *updateReferenceImage*(I_f, I_r).

Der Rückgabewert *subCount* (Zeile 8) kann dann über einen Proportionalitätsfaktor in die Anzahl der maximal zu revalidierenden Kanten/Knoten umgerechnet werden. Dabei können auch noch die entsprechenden Werte der anderen Kameras fusioniert werden. Die Auswahl der zu revalidierenden Kanten und Knoten erfolgt dann über eine FIFO-Queue, in welche Referenzen auf die Kanten und Knoten bei deren Invalidierung eingestellt werden. Auf diese Weise werden Kanten bzw. Knoten revalidiert, die zeitlich am längsten invalidiert sind. Der Proportionalfaktor kann per Messung von Hand oder zur Laufzeit automatisch bestimmt werden. Die automatische Bestimmung ist wie oben schon beschrieben möglich, indem die Abwesenheit von unbekannten Objekten im Arbeitsraum detektiert wird und der Proportionalfaktor so geregelt wird, dass in diesen Fällen die Restmenge an invalidierten Kanten leer wird.

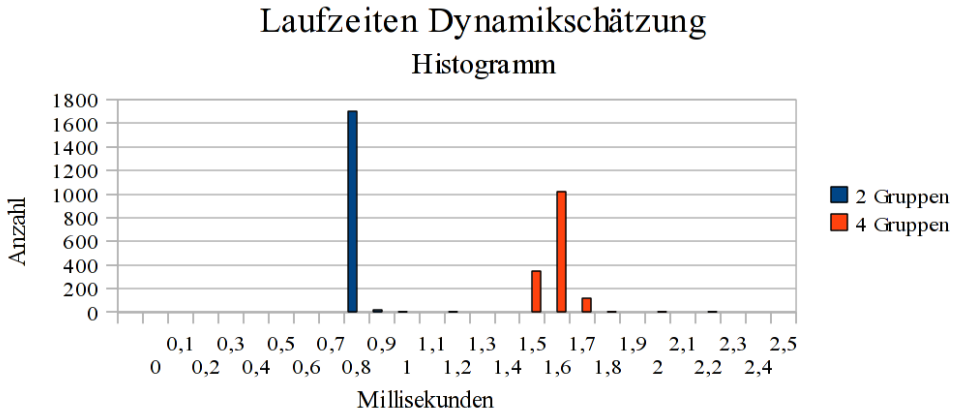


Abbildung 3.8: Laufzeithistogramme Dynamikschätzung. ***Histogramm in Blau****: Laufzeiten für eine Konfigurations von zwei Kameragruppen a vier Kameras mit einer Auflösung von 80 auf 60 Pixel, durchschnittliche Laufzeit 829 µSec.* ***Histogramm in Rot****: Vier Kameragruppen a vier Kameras, durchschnittliche Laufzeit 1641 µSec. Die Laufzeit der Dynamikschätzung ist linear abhängig von der Anzahl der Pixel mit geringer Streuung.*

Zwei Laufzeithistogramme der Dynamikschätzung für eine Kamerakonfiguration in zwei und vier Gruppen zu je vier Kameras mit einer Auflösung von 80x60 Pixeln sind in Abbildung 3.8 dargestellt. Die Kamerabilder wurden zu Vergleichszwecken simuliert mit einem Objekt, welches sich mit unterschiedlichen Geschwindigkeiten im Arbeitsraum bewegt. Die Laufzeiten der Dynamikschätzung weisen eine geringe Streuung auf, da sie immer über alle Pixel iterieren.

Experimente zu den Auswirkungen der Kantenrevalidierungsstrategien auf das Verhalten des Planers in verschiedenen Szenarien finden sich in Kapitel 5.3.

3.5 Objekttransport

In diesem Kapitel wird die Einbindung von bekannten Objekten $\boldsymbol{Z}_k$ in die Planung dargestellt, die aufgrund des Prozesses vom Roboter transportiert werden müssen.

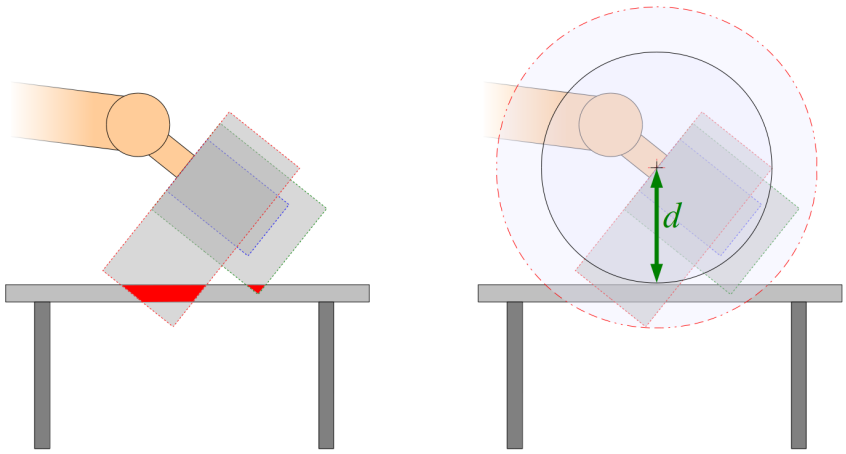


Abbildung 3.9: Zwei Varianten der Behandlung transportierter Objekte. ***Links****: Offline-Kollisionstests.* ***Rechts****: Approximation durch umhüllende Kugel, zentriert im Greifermittelpunkt. In Grün eingetragen ist der kürzeste Abstand d vom Greifermittelpunkt zum nächsten bekannten Objekt.*

Die transportierten Objekte beeinflussen über die Änderung der Roboterform den statischen und dynamischen Kollisionstest. Während die Beeinflussung des bildbasierten Kollisionstests durch die für jeden Test notwendige Neuberechnung des gesamten Robotermodells keine Änderung der Algorithmen erfordert, ist durch die Beeinflussung des Kollisionstests für die bekannten Objekte der statische Graph betroffen. Im Folgenden werden verschiedene Methoden dargestellt, um mit dieser Problemstellung umzugehen.

Offline-Berechnung/Kantenmarkierung

Ist die Geometrie aller transportierten Objekte im Voraus bekannt, die im Rahmen des Prozesses transportiert werden sollen, so kann diese Information in die Konstruktion des statischen Graphen einfließen.

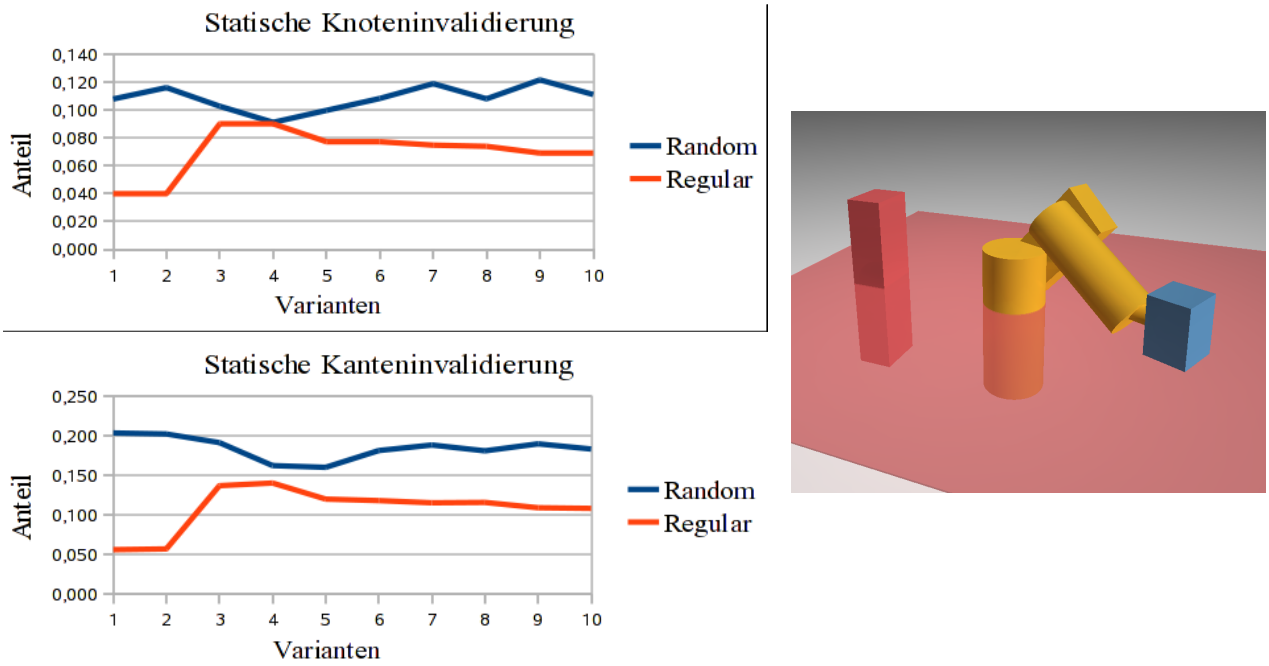


Abbildung 3.10: Anteile der Objektmarkierungen in den Kanten und Knoten an der Gesamtanzahl von Knoten und Kanten im Graphen für unterschiedliche Graphgrößen und Verbindungsdichten. ***Rechts****: Demozelle mit statischen bekannten Objekten (in (transparentem) Rot, Bearbeitungsstation und Tischebene); Roboter mit dem größten transportierten Objekt im Greifer im Rahmen dieses Experiments, eine Kiste in Blau mit den Abmessungen 300x400x280 mm.* ***Links oben****: Anteil makierter Knoten.* ***Links unten****: Anteil markierter Kanten. Die Varianten ergeben sich aus der Kombination von 500, 1000, 2000, 4000 und 5000 Knoten mit jeweils 4 oder 16 Kanten pro Knoten. Variante 1 ist die Kombination (500,4), Variante 2 ist die Kombination (500, 16), Variante 3 ist die Kombination (1000,4) usw. bis zur Variante 10 mit Kombination (5000,16). Die Knotenplatzierung ist uniform zufällig („Random“) oder auf einem festen Gitter („Regular“).*

Zu diesem Zweck kann für jede Kante und jeden Knoten eine Markierung berechnet werden, die pro Objekt festlegt, ob diese Kante bei Transport des jeweiligen Objekts kollisionsbehaftet ist oder nicht. Dies wird mithilfe des Kollisionstests für die $\boldsymbol{Z}_k$ offline für den gesamten Graphen berechnet, indem alle Objekte auf jeder Kante und jedem Knoten durchgespielt werden (Abbildung 3.9 links). Da lediglich Kanten und Knoten betroffen sind, deren Verlauf in der Nähe von statischen bekannten Objekten liegt, ist bei uniformen

Sampling-Strategien (siehe Kapitel 3.6) nur ein kleiner Teil der Kanten markiert (siehe Abbildung 3.10 für experimentelle Ergebnisse) und somit der Speicheraufwand gering.

Die Anteile der markierten Kanten und Knoten schwanken stark für Graphen mit wenigen Knoten (Varianten 1-4) insbesondere bei Platzierung auf einem regulären Gitter, da hier die spezifische Wahl der Gitterunterteilung starken Einfluß darauf hat, ob das transportierte Objekt an vielen Knoten und Kanten der Tischebene nah ist. Für eine hohe Anzahl von Knoten sollten sich die Anteile bei gegebener Raumgeometrie stabilisieren und für beide Samplingstrategien annähern, da durch den Kollisionstest die bekannten Objekte des Arbeitsraums für den statischen Graphen quasi um den maximalen Radius des transportierten Objekts erweitert werden und der Anteil dieses Zusatzvolumens am Gesamt-Freiraum-Volumen konstant bleibt.

Diese Methode ist vollständig informiert und dennoch insbesondere eine Ersparnis zu tabellenbasierten Planern (siehe Stand der Forschung in Kapitel 3.2), die pro Objekt eine zusätzliche Tabelle brauchen, die alle Kanten und Knoten des Graphen erfasst und alle Voxel speichern muss, die das Objekt berührt, wenn es durch den Roboter entlang einer Konfigurationsraumkante geführt wird („Swept-Volume“).

Die Überprüfung einer Kante/eines Knotens kann einfach in den Expansionsschritt des A*-Planers integriert werden und ist mit konstantem, sehr geringem Aufwand pro Kante/Knoten verbunden, die Laufzeit eines Expansionsschrittes erhöht sich unwesentlich im einstelligen Prozentbereich.

Verwendung eines Distanz-Offsets im Bahnplaner

Eine stark vereinfachte Version der Offline-Berechnung berechnet für jede Kante offline die minimale Distanz zu den statischen bekannten Objekten vom Greifpunkt aus. Für jedes gegriffene Objekt ist weiterhin der Radius der Kugel bekannt, die das Objekt umschließt, mit dem Mittelpunkt der Kugel im Greifpunkt (Abbildung 3.9 rechts). Dieser Radius wird online mit der Minimaldistanz der jeweiligen Kante verglichen und somit approximativ bestimmt, ob diese Kante mit diesem Objekt im Greifer fahrbar ist oder nicht. Somit müssen die Arten der Objekte nicht in die Kante kodiert werden und es können auch neue, bisher unbekannte Objekte transportiert werden. Dieser Test ist ideal für Objekte, die exakt kreisförmig um den Greifpunkt ausgedehnt sind, suboptimal für sehr längliche oder asymmetrisch gegriffene Objekte. Die Rechenanforderungen zur Laufzeit sind hierbei ähnlich gering anzusetzen, wie für die Methode der Objektmarkierungen, da lediglich Zahlenwerte verglichen werden müssen.

Online-Berechnung im Bahnplaner

Hierbei wird der Kollisionstest von transportierten Objekten mit statischen bekannten Objekten während der Planung durchgeführt. Diese Methode ist recht langsam, da jedoch

lediglich die gegriffenen Objekte selbst auf Kollision getestet werden müssen, ist sie schneller als der Test des gesamten Roboters inklusive Objekt. Die Kollisionsfreiheit der Robotergeometrie ist schon im statischen Graphen kodiert. Die Laufzeit dieser Variante ist stark abhängig von der Komplexität des verwendeten Modells von Objekt und Umgebung.

Online-Berechnung nach dem Bahnplaner

In Anlehnung an die Konzept aus dem oben dargestellten Online-Bahnplaner kann der statische Kollisionstest von gegriffenen Objekten auch nach der Planung auf dem resultierenden Pfad erfolgen. Die kollidierenden Kanten können dann invalidiert werden. Dies ermöglicht die Reduktion von Kollisionstests im Vergleich zur Online-Berechnung, führt jedoch zu wiederholten Ausführungen des Bahnplaners und verursacht unnötige Berechnungen des sensorbasierten Kollisionstests für Knoten, die aufgrund des tranportierten Objekts gar nicht Teil der Planung sein können. Somit ist diese Methode noch ungünstiger als die Überprüfung während der Planung.

3.6 Graphkonstruktion – Sampling-Strategien

Die Erzeugung von neuen Knoten für den Planungsgraphen findet beim hier dargestellten Planungskonzept zu verschiedenen Zeitpunkten und mit unterschiedlichen Zielsetzungen statt: während der Konstruktion des statischen Graphen und bedingt durch die Hindernissituation zur Laufzeit. Für die Erzeugung von Knoten gibt es in der Literatur eine Reihe von bekannten Methoden (siehe [Geraerts03] oder [Burns07] für einen Überblick). Im Folgenden wird mit $\boldsymbol{C}_{free}$ die Untermenge des Konfigurationsraumes bezeichnet, die die kollisionsfreien Roboterpositionen bezüglich der statischen bekannten Objekte bezeichnet: $\forall\, \boldsymbol{q} \in \boldsymbol{C}_{free} : \neg Coll_S(\boldsymbol{q})$. Die Komplementärmenge sei $\boldsymbol{C}_{coll}$, für diese gilt $\forall\, \boldsymbol{q} \in \boldsymbol{C}_{coll} : Coll_S(\boldsymbol{q})$.

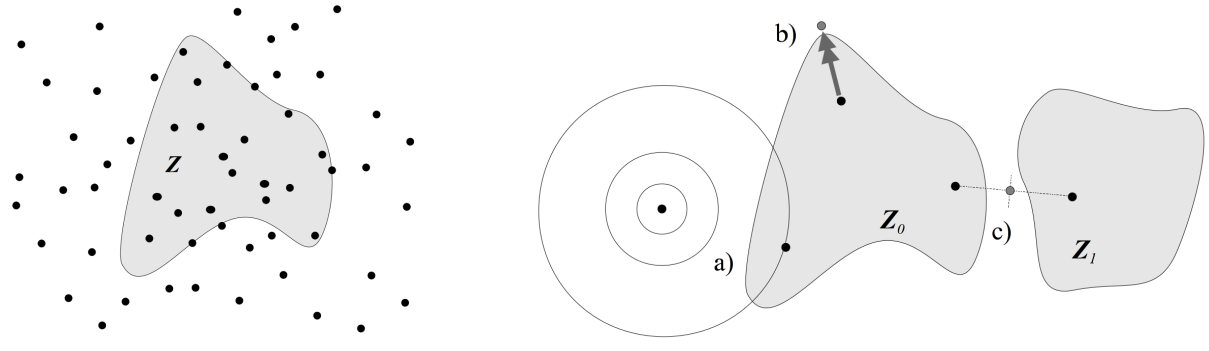


*Abbildung 3.11: Schematische Visualisierung verschiedener Sampling-Strategien. **Links**: Uniformes Sampling mit zufällig platzierten Knoten. Ein bekanntes Objekt **Z** nimmt im Konfigurationsraum den grauen Bereich ein. **Rechts**: Eine Auswahl von nicht-uniforme Sampling-Strategien, a) Ausgehend von einem zufällig gesetzten Knoten wird über eine multivariate Gauss-Verteilung ein neuer Knoten gewählt. Wenn ein Knoten in $\boldsymbol{C}_{free}$ und der andere in $\boldsymbol{C}_{coll}$ liegt, wird der Knoten in $\boldsymbol{C}_{free}$ zum Graphen hinzugefügt. b) Ein Knoten in $\boldsymbol{C}_{coll}$ wird hinzugefügt, nachdem er bis in $\boldsymbol{C}_{free}$ verschoben wird. c) Brückentest: Wenn zwei Knoten beide in $\boldsymbol{C}_{coll}$ liegen, wird ein mittlerer Knoten konstruiert und hinzugefügt, falls er in $\boldsymbol{C}_{free}$ liegt.*

Grundsätzlich können zwei Formen des Sampling unterschieden werden: Uniformes Sampling von Konfigurationen für neue Knoten ist unabhängig von der räumlichen Verteilung bzw. Aufteilung von $\boldsymbol{C}_{free}/\boldsymbol{C}_{coll}$. Nicht-uniformes Sampling berücksichtigt diese Verteilung und orientiert sich an den Grenzflächen dieser Mengen (siehe Abbildung 3.11 rechts). Dies basiert auf der Beobachtung, das problematische, enge Passagen in Konfigurationsräumen so mit größerer Wahrscheinlichkeit erfasst werden und die Lösung des Planungsproblems mit einer geringeren Anzahl von Samples zu finden ist.

Die Konstruktion des statischen Graphen hat einerseits das Ziel die durch das Roboterprogramm gegebene Aufgabenstellung zu erfüllen und andererseits den Freiraum dicht zu erfassen, um Pfadalternativen bei dynamischen unbekannten Objekten anzubieten. Da die Aufgabenstellung im Allgemeinen Pick-und-Place-Aufgaben umfasst, die an der Grenze des Freiraums im Konfigurationsraum liegen, wäre hier nicht-uniformes Sampling angebracht, um diese effizient zu lösen. Der Freiraum sollte andererseits möglichst uniform abgetastet werden, da die Position bzw. die Verteilung dynamischer unbekannter Objekte im Allgemeinen nicht im Voraus bekannt ist. Zur Lösung dieser beiden Vorgaben ist ein hybride Sampling-Strategie geeignet, die uniform abtastet und lokal in der Nähe der Pick-und-Place-Positionen die Abtastdichte mittels nicht-uniformer Strategien erhöht, wenn diese Positionen aus dem Roboterprogramm extrahiert werden können. Für die Experimente wurde der Focus auf die Abdeckung des Freiraums gelegt, um die dynamischen unbekannten Objekte gut zu erfassen. Daher wurden uniforme Strategien verwendet.

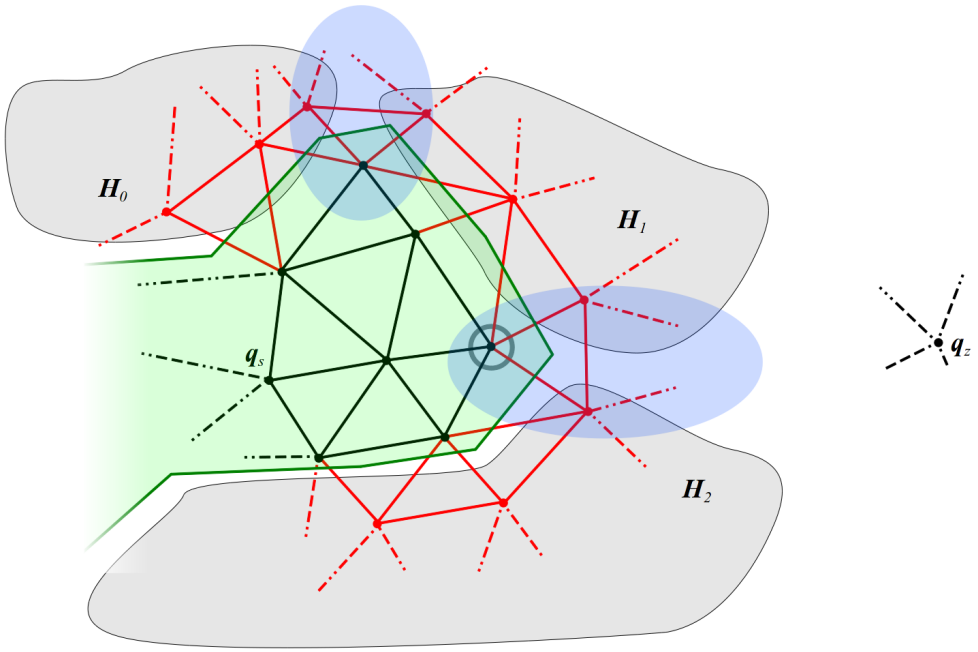


Abbildung 3.12: Query-based Sampling: Planung von Knoten $\boldsymbol{q}_s$ nach $\boldsymbol{q}_z$. Rot markierte Knoten und Kanten sind getestet und kollisionsbehaftet. Der transparente, grüne Bereich umfasst alle Knoten in G_{free}. Die tatsächlichen Konfigurationsraumhindernisse sind als hellgraue Regionen dargestellt. Passagen ergeben sich in den blau markierten Bereichen. Das Sampling würde in dieser Situation in der Nähe des grau umrandeten Knotens im blauen Bereich zwischen $\boldsymbol{H}_1$ und $\boldsymbol{H}_2$ beginnen.

Zur Laufzeit kann es auch notwendig sein, Knoten einzufügen, beispielsweise wenn kein Pfad gefunden wurde, eine Kante aufgeteilt werden musste (siehe Abbildung 3.5) oder Start- und Zielknoten erzeugt werden müssen. Während die letzteren beiden Fälle keine Sampling-Strategie erfordern, da die Konfigurationen bekannt sind, eröffnet der erste Fall neue Aspekte für die Platzierung der Knoten. Da zur Laufzeit über die Kollisionstests $Coll_S(...)$ und $Coll_D(...)$ alle Informationen sowohl bezüglich der bekannten als auch unbekannten Objekte bekannt sind, ist hier nicht-uniformes Sampling angebracht, da nicht alle möglichen Verteilungen von unbekannten Objekten bedacht werden müssen und das Auffinden der engen Passagen zwischen den Konfigurationsraumhindernissen relevant ist.

Bezeichne im Folgenden G_{free} alle Knoten aus G, die in $\boldsymbol{C}_{free}$ liegen und G_{coll} alle Knoten in $\boldsymbol{C}_{coll}$ (beides bezogen auf $Coll_S(...)$ und $Coll_D(...)$). Hierbei sind G_{free} und $\boldsymbol{C}_{free}$ nicht notwendigerweise zusammenhängend. Durch den Planungsvorgang wird der Teil von G_{free} erfasst, in dem $\boldsymbol{q}_s$ liegt, sei dies als $G_{free}(\boldsymbol{q}_s)$ bezeichnet. Diese Erfassung ist möglicherweise nicht völlig korrekt, da Knoten und Kanten durch Objektbewegungen falsche Informationen tragen können wenn sie über Systemzyklen hinweg beibehalten werden, im Folgenden soll jedoch von diesem Fall abstrahiert werden und korrekte Informationen angenommen werden.

Die Grenze von $G_{free}(\boldsymbol{q}_s)$ kann repräsentiert werden als Menge $\boldsymbol{P}_{free}$ aller Paare von Knoten, von denen der eine als kollisionsbehaftet (CALCED_COLLISION) und der andere als nicht kollisionsbehaftet (CALCED_FREE) markiert ist. Diese Paare bilden eine gute Basis für nicht-uniforme Sampling-Strategien wie z.B. den Brückentest. Wenn man annimmt, dass die engen Passagen entlang der Grenze gleichverteilt sind, ist es irrelevant für die Wahrscheinlichkeit des Findens einer Passage, in welcher Reihenfolge die Knotenpaare ausgewählt werden. Unter dieser Annahme kann für die Auswahl-Reihenfolge der Knotenpaare ein Kriterium herangezogen werden, dass als *Query-based Sampling* bezeichnet werden soll (siehe hierzu auch Abbildung 3.12).

Query-based Sampling

Für jedes Knotenpaar aus $\boldsymbol{P}_{free}$ sind für den jeweiligen Knoten in $G_{free}(\boldsymbol{q}_s)$ durch den Pfadplaner die Kosten $f(v)$ berechnet worden. Diese Kosten können als Sortierkriterium für die $\boldsymbol{P}_{free}$ verwendet werden. Auf dieser Sortierung können nun Auswahlstrategien (s.u.) definiert werden, die es wahrscheinlicher machen, dass von der Menge an existierenden Passagen diejenige gewählt wird, die die günstigste Verbindung in Richtung des aktuellen Planungszieles ermöglicht. Die Anzahl an gesampleten Knoten bis zum Finden dieser Verbindung soll minimiert werden. Auswahlstrategien auf der sortierten Liste sind beispielsweise Round-Robin beginnend mit dem günstigsten Paar oder Auswahl nach Wahrscheinlichkeitsverteilung mit Betonung der günstigen Paare.

Wenn p die Wahrscheinlichkeit ist, an einer Stelle eine Passage zu finden (p sei für alle Stellen gleich, es wird also eine Gleichverteilung der Passagen angenommen), dann ist die Wahrscheinlichkeit, beim k-ten Test einen Weg zu finden $(1 - p)^k \cdot p$. Diese ist über k monoton fallend und daher sollten per Round-Robin die ersten Punkte gewählt werden, um hier mit höherer Wahrscheinlichkeit den ersten Treffer zu finden und somit gemittelt die günstigsten Kosten für die gefundenen Passagen zu erreichen. Dies lässt sich durch eine darüber liegende Verteilung entsprechend weiter auf die „günstigen" Passagen konzentrieren. Der Vorteil des Round-Robin-Verfahrens liegt allerdings darin, dass man im generellen Auffinden einer Passage genauso gut ist wie eine gleichverteilte, zufällige Auswahl der Punkte in $\boldsymbol{P}_{free}$, da die generelle Wahrscheinlichkeit des Findens einer Passage für alle gewählten Punke als gleich wahrscheinlich angenommen ist.

3.7 Zusammenfassung

In diesem Kapitel wurde ein Planungskonzept präsentiert, welches eine echtzeitfähige, globale Bahnplanung in hochdimensionalen Konfigurationsräumen unter Berücksichtigung dynamischer Hindernisse ermöglicht. Die globale Planung wird durch Verwendung einer graphbasierten Suche im Konfigurationsraum erreicht. Die Rechenzeit ist flexibel an die verfügbare Zykluszeit (hier 100 ms) adaptierbar. Hierzu wird der größte Aufwandsanteil bei Bahnplanern – die Berechnung des Kollisionstests – deutlich reduziert. Kollisionstests mit statischen bekannten Objekten werden weitgehend durch Offline-Konstruktion eines statischen Graphen im Konfigurationsraum vermieden, welcher kollisionsfrei bezüglich statischer bekannter Objekte ist. Kollisionsberechnungen mit a priori unbekannten, sensorisch erfassten Objekten werden mittels des bildbasierten Kollisionstests (Kapitel 2) berechnet und während eines Planungzyklus in ihrer Anzahl begrenzt. Hierzu wird eine Abwandlung des Lazy-Collision-Checking verwendet, welches zunächst von einer Kollisionsfreiheit nicht getesteter Knoten und Kanten des Graphen ausgeht. Während der Graphensuche werden ausgewählte Knoten des Graphen auf Kollisionen getestet, die für die Planung relevant sind. Die Ausführung der so berechneten Bahn und der Planung einer neuen, angepassten Bahn erfolgen in jedem Systemzyklus parallel. Vor Ausführung einer Bahn werden die unmittelbar auszuführenden Kanten des Graphen auf Kollisionen überprüft, wobei die Anzahl der Kollisionstests begrenzt ist. Durch zyklenübergreifende Persistenz der Kollisionsinformationen auf den Knoten und Kanten des Graphen erweitert sich die Umweltinformation kontinuierlich. Dabei wird die Dynamik unbekannter Objekte bestimmt und für eine konservative Anpassung der Kollisionsraum-Informationen (Zurücksetzen kollisionsbehafteter Knoten und Kanten) verwendet. Im Falle von statischen unbekannten Objekten liegen so nach einigen Systemzyklen vollständige Konfigurationsraum-Informationen vor. Bei dynamischen unbekannten Objekten werden die Kollisionstests durch das Planungskonzept lokal konzentriert, um dem nächsten

Hindernis adäquat zu begegnen. Der Transport bekannter Objekte ist mit geringem Zusatzaufwand durch Kodierung der entsprechenden Kollisionstestinformationen im statischen Graphen realisierbar. Experimentelle Ergebnisse werden in Kapitel 5 dargestellt.

4 Zeitoptimierte Bahnplanung in dynamischen Arbeitszellen

Im diesem Kapitel wird aufbauend auf der wegoptimierten Bahnplanung zunächst der Nutzen einer Zeitoptimierung des Pfades motiviert (Kapitel 4.1), um anschließend die sich aus der Forderung nach Zeitoptimierung und den Umgebungsbedingungen ergebende Aufgabenstellung zu konkretisieren (Kapitel 4.2). Im Stand der Forschung (Kapitel 4.3) werden die bisher bekannten Ansätze im Rahmen der Aufgabenstellung kritisch beleuchtet. In Kapitel 4.4 wird das Gesamtkonzept in der Übersicht dargestellt und anschließend in den Kapiteln 4.5 bis 4.10 im Detail erläutert. Experimentelle Ergebnisse für verschiedene Konfigurationsraumszenarios sind in Kapitel 5 zusammengefasst und mit den Ergebnissen des wegoptimierenden Planers verglichen. Relevante Teile dieses Kapitels wurden in [Gecks09] veröffentlicht.

4.1 Motivation

Im Bereich der Mensch-Roboter-Koexistenz ist eine große Distanz zwischen Mensch und Roboter wünschenswert, um die Ergonomie und Unfallsicherheit zu erhöhen. Die Steigerung der Ergonomie ergibt sich dabei aus dem Aspekt, dass ein schwerer, starker Roboterarm mit zuweilen scharfkantigen transportierten Objekten ein für den menschlichen Kollegen psychisch ungünstiges Verhalten hat, wenn sich der Roboterarm mit hoher Geschwindigkeit in dessen Nähe bewegt. Dies tritt besonders dann erheblich zutage, wenn der Arbeiter mit einer eigenständigen Aufgabe betraut ist, die seine volle Aufmerksamkeit erfordert, wodurch eine Beobachtung des Roboters, bzw. Einschätzung seiner Bewegungen nicht oder nur eingeschränkt möglich ist. Die daraus folgende Notwendigkeit des Vertrauens in das Robotersystem wird durch eine distanzbasierte Geschwindigkeitsregelung unterstützt, welche die Geschwindigkeit des Robotersarms bei Annäherung an Hindernisse reduziert. Die hier entwickelte Planung soll dies unterstützen, indem Bahnen mit größerem Hindernisabstand gewählt werden, in Abwägung mit der sich daraus oft ergebenden Länge der Bahn, um so insgesamt die Ausführungzeit der Bahn zu optimieren, was auch der wirtschaftlichen Effizienz zugute kommt.

Eine distanzbasierte Geschwindigkeitsregelung wird auch durch die Dynamik der Umgebung erforderlich. Im Kontext von unbekannten, sensorisch erfassten Objekten mit nicht vorhersagbaren Bewegungen muss der Roboter in der Lage sein, eine Kollision zu vermeiden unter Berücksichtigung der eigenen und der Objektgeschwindigkeit, d.h. er muss innerhalb einer gewissen Zeitschranke zum Stehen kommen können. Das bedingt in Kombination mit einer vorgegebenen Latenz des Steuerungssystems und der begrenzten Entschleunigungsfähigkeit des Roboterarms eine distanzbasierte Geschwindigkeitsregelung (siehe auch [Kuhn06]).

Aus der distanzbasierten Geschwindigkeitsregelung ergibt sich wie im Folgenden dargestellt eine Forderung nach zeitoptimierten Bahnen für die Planung der Bewegung des Roboterarms. Vor dem Hintergrund der Wirtschaftlichkeit betrachtet, ist eine möglichst minimale Verlängerung der üblichen, ungestörten Zykluszeit für den jeweiligen Prozess wünschenswert, um die erwarteten Gewinne aus der Mensch-Roboter-Koexistenz/Kooperation nicht durch Verluste aufgrund zu großer Zykluszeiten zu reduzieren. Daher soll die Planung zeitlich kurze Bahnen bevorzugen. In Kombination mit den oben aufgeführten Gründen für die distanzbasierte Geschwindigkeitsregelung sind somit im Allgemeinen Bahnen mit größerer Distanz zu unbekannten Objekten (und damit hoher Geschwindigkeit) zu planen, wobei die räumliche Bahnlänge mit der Geschwindigkeit entlang der Bahn balanciert werden muss, um die Ausführungsdauer der Bahn zu optimieren.

Ein weiterer Vorteil zeitoptimierter Bahnen ist der durch die im Allgemeinen größere Distanz glattere Verlauf der Bahn, da sich ein aufgeblähtes Objekt einer Kugel annähert und der Bahnplaner die Bahn an die Kontur des expandierten Objekts approximiert, statt der eigentlichen Kontur des Objekts zu folgen, wie es für rein wegoptimierende Planer typisch ist.

Zeitoptimierte Bahnplanung in Verbindung mit der distanzbasierten Geschwindigkeitsregelung versprechen somit eine höhere Ergonomie, glattere Bahnverläufe und wirtschaftliche Zykluszeiten. In den folgenden Kapiteln soll dargestellt werden, wie dies online echtzeitfähig realisierbar ist.

4.2 Aufgabenstellung

Diese Aufgabenstellung entspricht in weiten Teilen der Aufgabenstellung in Kapitel 3.1. Es werden hier daher nur kurz die gemeinsamen Aufgabenstellungen wiederholt und die Unterschiede hervorgehoben.

Gegeben sei eine Startkonfiguration $\boldsymbol{q}_s$ und Zielkonfiguration $\boldsymbol{q}_z$ für einen Roboter. Weiterhin gegeben sind zwei Kollisionstests, die für eine Konfiguration $\boldsymbol{q}$ oder eine Linearbahn im Konfigurationsraum von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ eine Aussage über die Kollisionsfreiheit liefern. Einer der beiden Kollisionstests liefert eine Kollisionsaussage für die statischen, bekannten Objekte (true, wenn eine Kollision vorliegt):

$$Coll_S(\boldsymbol{q}) \in \{\text{true, false}\} \qquad Coll_S(\boldsymbol{q}_a, \boldsymbol{q}_b) \in \{\text{true, false}\} \tag{4.1}$$

Der andere Kollisionstest liefert eine *minimale Distanz* entlang einer Linearbahn bzw. an einer Konfiguration zwischen Roboter und den unbekannten Objekten $\boldsymbol{H}_l$: (siehe hierzu auch Kapitel 2):

$$Coll_D(\boldsymbol{q}) \in \mathbb{R}_0^+ \qquad Coll_D(\boldsymbol{q}_a, \boldsymbol{q}_b) \in \mathbb{R}_0^+ \tag{4.2}$$

Gegeben ist weiterhin eine Geschwindigkeitsregelung aufbauend auf der aktuellen minimalen Objektdistanz, wie sie $Coll_D(\boldsymbol{q})$ liefert. Diese Regelung erreicht aus Sicherheitsgründen im Allgemeinen die Geschwindigkeit 0, bevor die Distanz 0 erreicht wird (siehe auch [Kuhn06]).

Gesucht ist eine durch den Roboter *verfahrene* Bahn $\boldsymbol{q}(t)$, mit t aus $[0, t_{max}]$, $\boldsymbol{q}(0) = \boldsymbol{q}_s$ und $\boldsymbol{q}(t_{max}) = \boldsymbol{q}_z$, für die gilt:

$$\left(\left(\neg Coll_S(\boldsymbol{q}(t))\right) \wedge \left(Coll_D(\boldsymbol{q}(t)) \geq d_{min}\right)\right) \quad \forall\, t \in [0, t_{max}] \tag{4.3}$$

d_{min} ist dabei die Distanz, bei deren Unterschreitung die Geschwindigkeit 0 erreicht wird. Auch hier seien wie bei wegoptimierende Planer Kollisionen ausgeschlossen, die durch Bewegungen der unbekannten Objekte $\boldsymbol{H}_l$ verursacht werden (siehe Kapitel 3.1).

Die folgenden Anforderungen (**Ax**) und Bedingungen (**Bx**) aus Kapitel 3.1 werden übernommen:

- A1 Echtzeitfähigkeit
- A3 Hohe Anzahl an Freiheitsgraden
- A4 Parallelität von Ausführung und Planung
- A5 Vollständigkeit des Planers
- A6 Effiziente(r) Einsatz der Kollisionstests bzw. Erfassung der Umwelt
- A7 Sicherheit
- A8 Anwendungsdomäne
- B1 Implizite Konfigurationsraumdarstellung
- B2 Rigidität des Roboters und der transportierten Objekte
- B3 Roboterdynamik
- B4 Umweltmodell für statische Objekte

Eine Änderung ergibt sich für die Anforderung A2, die von der Weg-Optimierung zur Zeitoptimierung gewandelt wird:

A2 Zeit-Optimierung

Die Ausführungszeit einer Roboterbahn vom Start zum Ziel ist durch die aktuelle Hindernissituation und die Distanz-Geschwindigkeitsfunktion gegeben. Die Bahnplanung soll eine Minimierung dieser Zeitdauer anstreben. Über die zukünftige Bewegung der Objekte $\boldsymbol{H}_I$ werden keine Aussagen gemacht, die Ausführungzeit ist daher auf einer während der Planung als statisch angenommenen Umwelt approximativ zu minimieren, wie dies auch bei der wegoptimierenden Planung der Fall ist.

4.3 Stand der Forschung

Im diesem Kapitel wird der Stand der Forschung dargestellt und im Hinblick auf die Aufgabenstellung bewertet. Außer explizit zeitoptimierenden Ansätzen sind auch Ansätze aufgeführt, für wegoptimierte Pfade die einen möglichst maximalen Hindernisdistanz realisieren, welche eingeschränkt zur Erfüllung der Aufgabenstellung eingesetzt werden könnten.

Zeitoptimierung in der mobilen Robotik

Ansätze in der mobilen Robotik schlagen Vorteile aus der Planung im Arbeitsraum des Roboters, wodurch der Kollisionstest sehr günstig wird oder eine einfache Transformation des Arbeitsraums in den Konfigurationsraum möglich wird. Diese vereinfachenden Grundbedingungen sind für die hier betrachtete Aufgabenstellung nicht gegeben (bezüglich

A3, A8 und B1), dennoch sollen im Folgenden einige Ansätze dargestellt werden, die unter dem Stichwort Zeitoptimierung zu finden sind, um die unterschiedlichen Ansätze noch einmal hervorzuheben.

In [Philippsen05] wird für einen mobilen Roboter in einem 2-dimensionalen Arbeitsraum ein risikominimierender Pfad geplant. Das Risiko wird dabei aus der Distanz, der Geschwindigkeit und der Richtung der Objekte für eine gleichmäßige Aufteilung (Gitter) des Arbeitsraumes für jeden Gitterpunkt berechnet. Eine sogenannte *Navigation Function* (Wellenausbreitung, [Koditschek90]) findet den risikominimalen Weg zum Ziel. In die Bahnplanung fließt durch die Risiko-bestimmte Robotergeschwindigkeit somit die Hindernisdistanz ein und es wird die Ausführungsdauer optimiert.

Verschiedene Ansätze der Bahnplanung im Bereich der mobilen Robotik vereinen zwar Geschwindigkeits- und Bahnplanung (Velocity Obstacle [Fiorini96], Global Dynamic Window [Brock00]) und optimieren zum Teil die Ausführungszeit unter diesen Kriterien, jedoch dient hier die Geschwindigkeitsregulierung lediglich zur Kollisionsvermeidung und bezieht dazu im Allgemeinen die Bewegungsvektoren von Roboter und Objekten mit ein. Dadurch lassen sich diese Algorithmen hier nicht anwenden, da keine distanzbasierte Geschwindigkeitsregelung stattfindet.

Im Bereich der Service-Roboter wird der Ergonomie des Menschen Rechnung getragen, indem die Distanz zwischen Mensch und Roboter sowie weitere Kriterien in die Bahnplanung einfließen. In [Sisbot05] werden für einen als punktförmig abstrahierten mobilen Roboter auf einem zweidimensionalen Gitter die Kostenfunktion von Personen und Gegenständen überlagert, um dann mittels A*-Planung den Pfad minimaler Kosten zu finden. Hierbei wird jedoch lediglich die Distanz zum Menschen in die Planung mit einbezogen (A8) und eine direkte, distanzabhängige Geschwindigkeitsregulierung findet nicht statt. Weiterhin gibt es für Service-Roboter verhaltensbasierte Ansätze (siehe z.B. [Althaus04]), die die Hindernisdistanz zur Planung ihrer Trajektorie und Geschindigkeits-regelung verwenden. Hierbei sind jedoch keine Ziele, sondern ein Verhalten vorgegeben, sodass eine Zeitoptimierung des Pfades nicht explizit berechnet wird und auch nicht erreicht wird.

Zeitoptimierung in hochdimensionalen Konfigurationsräumen

Zur Optimierung bestimmter Kriterien (z.B. Ausführungzeit) unter Randbedingungen eignen sich prinzipiell Methoden aus der Trajektorienplanung [Choset05, Kapitel 11]. Hierbei wird eine globale, lokal differenzierbare Kostenfunktion für den Roboterpfad angegeben und weitere Randbedingungen wie z.B. die Objekte im Arbeitsraum oder die maximalen Drehmomente der angesteuerten Motoren. Aufgrund der Komplexität ist im Allgemeinen keine analytische Lösung des Optimierungsproblems möglich, weshalb ein

initial gegebener Pfad mittels geeigneter numerischer Verfahren optimiert wird, bis das (lokale) Optimum der Kostenfunktion erreicht ist. Dabei wird in der Regel nur ein lokales Optimum gefunden, weshalb diese Methoden beispielsweise mit randomisierenden Suchmethoden kombiniert werden, um gute Startwerte für die Optimierung zu finden. Die Kostenfunktion kann so definiert werden, dass sie der Ausführungszeit der Bahn unter Berücksichtigung der Hindernisdistanzen entspricht und damit der hier gegebenen Aufgabenstellung entspricht. Die Minimierung der Kostenfunktion ist allerdings sehr aufwendig und daher nur für einfache Umgebungen echtzeitfähig (A1 nicht erfüllt).

Eine generelle Klasse von Algorithmen zur Optimierung vorgegebener Kriterien bilden die Genetischen Algorithmen, die in [Vannoy04] eingesetzt werden, um in diesem Fall distanzoptimiert zu planen. Eine Zeitoptimierung ist durch die Definition einer entsprechenden Kostenfunktion für die Gensequenzen erreichbar. Der genetische Code jedes Individuums besteht dabei aus den Knotenpunkten des jeweiligen Gesamtpfades vom Start zum Ziel. Der dargestellte Planer ist sehr schnell (5ms Planungszyklus) und anytime-fähig ab der ersten Lösung, diese Zahlen sind jedoch durch die kleine Population (20 Individuen) und die kurzen Chromosomlängen zu relativieren (Echtzeitfähigkeit A1 ist in Frage gestellt). Zudem ist bei steigender Objektzahl das Risiko nicht auszuschließen, dass kein Pfad gefunden wird, da immer der gesamte genetische Code eines Individuums in der Kostenfunktion betrachtet wird und das Risiko von Kollisionen mit der Länge steigt. Dadurch steigt die Anzahl der Generationen bis zum ersten verwendbaren Pfad an. Die Kosten für einen Kollisionstest und die notwendige Anzahl an Kollisionstests sind nicht angegeben. Bei dem dargestellten Algorithmus ist es nicht möglich, das statische Umweltmodell (Bedingung B4) auszunutzen, da jeder genetische Code randomisiert erzeugt wird und die Verbindungsstruktur eines statisch berechneten Graphen so nicht ausgenutzt werden kann. Durch die zufälligen Pfadsegmente werden die zur Verfügung stehenden Kollisionstests nicht effizient eingesetzt, sondern zufällig im Planungsraum verteilt (A6).

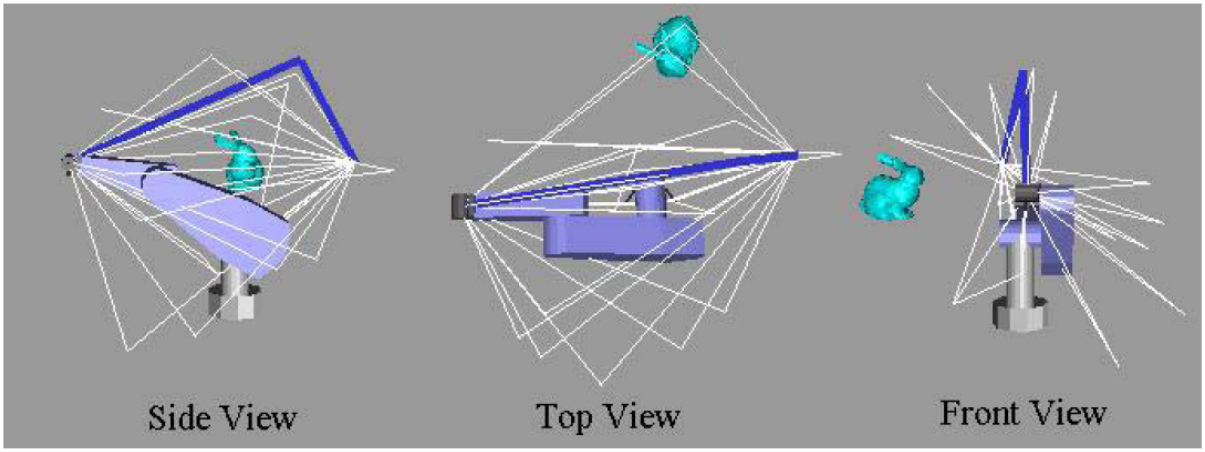


Abbildung 4.1: Aus [Vannoy04]: Beispiel-Population der einen Pfad representierenden Individuen im genetischen Planungsalgorithmus.

Kürzeste Wege mit Maximierung der Hindernisdistanz

Im folgenden werden einige Planer dargestellt, die kürzeste Wege mit maximierter Hindernisdistanz finden. Diese Pfade sind im Allgemeinen suboptimal, da Umwege gefahren werden, sodass trotz höherer Verfahrgeschwindigkeit insgesamt eine längere Zeitdauer für das Erreichen des Ziels benötigt wird als bei explizit zeitoptimierenden Verfahren.

Eine Familie von Algorithmen verwendet geometrische Informationen des Konfigurationsraumes, um die Punkte maximaler Distanz von allen Hindernissen zu bestimmen (Kapitel 5 und 6 in [Choset05]). Dabei ist es grundlegend für die Effizienz der Algorithmen, dass der Raum vollständig bekannt ist und möglichst einfach geometrisch beschrieben werden kann, weshalb diese Art von Algorithmen häufig im mobilen Bereich Anwendung findet, da dort im Allgemeinen im Arbeitsraum geplant wird. Der kollisionsfreie Bereich des Planungsraums wird durch geometrische Primitive komplett erfasst oder approximiert und über die topologische Beziehungen der Primitive und deren Traversierungskosten kann einen Pfad vom Start zum Ziel bestimmt werden. Zur Vergrößerung der Distanz kann dann innerhalb eines Primitivs leicht die Bahn maximaler Distanz zu dem Rand des Primitivs für die Bahn des Roboters verwendet werden. An den Übergängen der Primitive müssen die Bahnen kollisionsfrei ineinander übergeführt werden. Die Technik der Zerlegung in geometrische Primitive kann auch mit Potentialfeld-Techniken verknüpft werden, indem für jedes geometrische Primitiv ein Potentialfeld berechnet wird, das durch seine Funktion die Hindernisdistanz vergrößert, auf dynamische Hindernisse reagieren kann und durch die Ausrichtung an der Primitivtopologie das Erreichen des Zieles ermöglicht [Lindemann05, Rohrmüller08]. Der für diese Algorithmen notwendige vollständig bekannte, zweidimensionale Konfigurationsraum ist hier nicht gegeben (B1 und A3).

In [Yang06] wird ein potentialfeldbasierter Planer für die Planung der Verbindungen zwischen den Knoten eines globalen Graphen verwendet, um echtzeitfähig auf die Bewegung von Hindernissen reagieren zu können Die Knoten sind im Arbeitsraum an den Hindernisobjekten orientiert und stellen hier die Endeffektorposition dar. Die gegenseitige Verbindungsmöglichkeit der Knoten untereinander wird abgeschätzt durch Überprüfung, ob die Koordinatensysteme aller Glieder des Roboters eine Sichtverbindung haben. Aus den gültigen Verbindungen wird dann ein Graph aufgebaut. Bei Bewegungen der Objekte werden auch die zugeordneten Punkte bewegt und die Verbindungsmöglichkeiten aktualisiert. Der Planer ist durch den potentialfeldbasierten Ansatz nicht vollständig und verhindert durch die Verschiebung der Bahnpunkte eine Ausnutzung von statischen Kollisionsinformationen in einem statischen Graphen. Die Ausführungszeit oder der Weg werden durch den Planer nicht explizit optimiert. Ein Modell des Arbeitsraums ist

erforderlich, welches hier nicht zur Verfügung steht.

Für die Klasse der PRM (Probabilistic Roadmap)-Planer gibt es Erweiterungen, die für die geplante Bahn eine hohe Distanz zu den Hindernissen erzeugen. Bei der Erzeugung des Graphen können zu diesem Zweck Sampling-Punkte bevorzugt werden, die auf der sogenannte *Mittenachse* (*medial axis*) liegen, bei denen die Distanz zu den zwei nächsten Hindernissen gleich groß ist [Holleman00]. Eine andere Möglichkeit besteht in der Verschiebung des durch einen PRM-Planer erzeugten Pfades auf die genannte Mittenachse in einem Nachbearbeitungsschritt [Geraerts04] (Abbildung 4.2 links). Überflüssige Seitenzweige, die zweimal durchlaufen werden, werden dabei gelöscht. Beide Ansätze verwenden geometrische Informationen über den Planungsraum, bzw. eine große Anzahl von Kollisionstests und haben daher dieselben Nachteile wie die im vorigen Abschnitt genannten Algorithmen. Auch ist eine Zeitoptimierung des geplanten Pfades nur durch die Bestimmung des kürzesten Pfades im Planungsgraphen möglich, welche von dem optimalen Pfad stark abweichen kann. Zudem ist jeder erstelle Graph auf eine spezifische Hindernissituation festgelegt und kann nicht für mehrere Anfragen verwendet werden, wenn die Umgebung dynamisch ist.

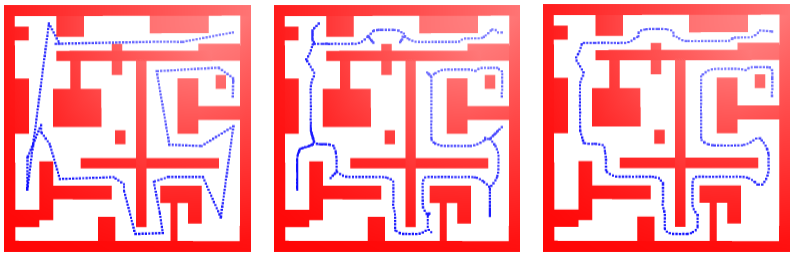
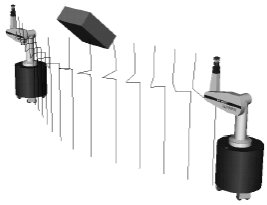
a)  b) 

Abbildung 4.2: a) Distanzoptimierte Bahnplanung aus [Geraerts04] mit (von links nach rechts) Planung eines zufälligen Pfades, Retraktion auf die Mittenachsen und Löschung überflüssiger Seitenzweige, b) „Elastic Strips" Simulationsbeispiel aus [Brock99]

Ein Verknüpfung von globaler Bahnplanung und lokaler Distanzoptimierung bietet auch die „Elastic Strips-Methode" aus [Brock99 (Abbildung 4.2b), Quinlan93]: Ein global errechneter Pfad wird als eine Folge von Gummibändern simuliert und basierend auf den lokal detektierten Hindernissen mittels Potentialfeldmethoden durch die berechneten Kräfte deformiert, bis zu einer maximalen Verformung. In diesem Fall oder im Falle einer nicht verhinderbaren Kollision wird der Pfad global neu geplant. Der nicht näher spezifizierte globale Planer hat Laufzeiten im Sekundenbereich und genügt daher nicht der Anforderung A1. Die Verformung des Pfades kann zu stark suboptimalen Pfaden führen (Anforderung A2 verletzt) und erfordert einige Heuristiken, die jeweils eine Entscheidung zur Neuplanung auslösen. Durch die Abweichung von dem ursprünglich geplanten Pfad müssen Kollisionen auch mit bekannten statischen Hindernissen zur Laufzeit in jedem Schritt neu berechnet werden (Bedingung B4 nicht nutzbar).

Schlussfolgerungen

Die vorgestellten Methoden realisieren eine Zeitoptimierung oder Distanzmaximierung des geplanten Pfades, haben jedoch einige Nachteile, die eine Anwendung unter den hier gegebenen Vorgaben verhindern. Die geometrische möglichst exakte und einfache Rekonstruktion des kompletten Konfigurationsraums ist in Echtzeit für Roboter mit vielen Freiheitsgraden nicht realisierbar oder nur mit eingeschränkter Auflösung oder Approximation der realen Situation. Ansätze, die durch die Verwendung lediglich lokaler Informationen Echtzeitfähigkeit erreichen, sind unvollständig, da sie sich in lokalen Minima verfangen können und daher das Ziel nicht erreichen, trotz Existenz eines Pfades. Für andere Verfahren wiederum ist die Existenz eines sehr günstigen Kollisionstests Voraussetzung für ihre Echtzeitfähigkeit. Viele Verfahren realisieren keine Zeitoptimierung sondern maximieren lediglich Hindernisdistanzen auf ihrer Bahn zum Ziel, was zu suboptimalen Lösungen führt. Gesucht ist also ein Bahnplanungsverfahren, das die Aufgabenstellung erfüllt und die gegebenen Anforderungen und Bedingungen berücksichtigt. In den folgenden Kapiteln wird das Konzept und die Realisierung für einen entsprechenden Bahnplaner dargestellt.

4.4 Konzept

Der in Kapitel 3 vorgestellte Algorithmus zur wegoptimierten Planung in dynamischen Arbeitsräumen bildet in Teilen die Basis für das im folgenden dargestellte Planerkonzept, da er schon alle Anforderungen bis auf die hier geänderte Anforderung A2 (Zeitoptimierung) erfüllt und die gegebenen Bedingungen verwendet. Durch die Verwendung von Distanzberechnungen statt einfacher Kollisionstests ergeben sich jedoch in einigen Aspekten Veränderungen, wie beispielsweise der Platzierung der Distanzberechnungen während der Graphensuche (Kapitel 4.8) und Adaption an Arbeitsraumdynamik (Kapitel 4.10). In den Kapiteln 4.5 - 4.7 werden die Details der im Rahmen des Konzepts eingeführten Distanzschätzverfahren erläutert und die daraus resultierenden Effekte im Verhalten des Planers in Kapitel 4.9.

Im Folgenden wird die reine Graphensuche beschrieben, da sich das umschließende Algorithmusgerüst für die parallele Ausführung und Planung, welches in Kapitel 3 vorgestellt wurde, nicht verändert. Dazu gehören beispielsweise auch die Kantentests vor der Ausführung eines Pfades und die Revalidierungsmechanismen für invalidierte Kanten. Geändert wird lediglich Algorithmus 3.4 in Zeile (2):

```
(1)  plan(v_s, v_z)
(2)      P = searchGraphTimeOptim(v_s, v_z)
(3)      if P is not empty
(4)          testPath(P)
(5)      else
(6)          if(not Coll_D(v_z)) sampleVertex()
(7)      return P
```

Algorithmus 4.1: Veränderung der plan(v_s, v_z)-Funktion aus Algorithmus 3.4. Es wird lediglich die Graphensuche ausgetauscht. Die Funktion searchGraphTimeOptim(v_s, v_z) berechnet einen zeitoptimierten Pfad (siehe Algorithmus 4.12).

Adaption der Kostenfunktion der Graphensuche

Zur Optimierung der Ausführungszeit des Pfades muss diese in die Kostenfunktion der Graphensuche einfließen. Dies wird erreicht, indem sowohl in die Kosten der Kanten als auch die heuristische Schätzung der Kosten zum Ziel die Distanz zu den unbekannten Objekten in die Zeit umgerechnet wird, die für die Bewegung des Roboters entlang dieser Strecken im Konfigurationsraum benötigt wird. Dazu wird neben den Positionsparametern der Kante der Verlauf der Geschwindigkeit entlang der Kante benötigt. In [Kuhn06] wird dargestellt, wie die Geschwindigkeit des Roboters in Abhängigkeit von der sensorbasierten Distanz geregelt wird. Diese Geschwindigkeitsregelung wird auch in diesem System eingesetzt und bildet daher die Basis für die Kantenkosten, die sich aus der Dauer der Roboterbewegung $t_R(\boldsymbol{q}_a, \boldsymbol{q}_b)$ entlang der Kante von von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ ergeben.

Aus einem Distanzverlauf $d(\boldsymbol{q}(u))$ entlang einer linearen Verbindung $\boldsymbol{q}(u)$ im Konfigurationsraum von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$

$$d(\boldsymbol{q}(u)) = d(\boldsymbol{q}_a + u \cdot (\boldsymbol{q}_b - \boldsymbol{q}_a)) \quad , u \in [0,1] \tag{4.4}$$

ergibt sich über die entsprechend vorgegebene Distanz-Geschwindigkeitsfunktion $speed(d)$ ein Geschwindigkeitsverlauf $speed(d(\boldsymbol{q}(u)))$. Zur Berechnung der Kantenkosten im Konfigurationsraum muss dieser Geschwindigkeitsverlauf in den Konfigurationsraum abgebildet werden $speed_c(speed(d(\boldsymbol{q}(u))))$. Dabei bezieht sich $speed_c$ auf die Geschwindigkeit des Roboters im Konfigurationsraum. Um die Echtzeitfähigkeit des Algorithmus zu gewährleisten, sind effizient berechenbare Approximationen für diese Abbildung zu wählen (siehe hierzu Kapitel 4.6.2). Im Folgenden wird $speed_c(speed(d(\boldsymbol{q}(u))))$ kürzer als $speed_c(d(\boldsymbol{q}(u)))$ geschrieben.

Unterteilt man $\boldsymbol{q}(u)$ in n Teilstücke, so kann man die Fahrtzeit t_R approximieren mit:

$$\tilde{t}_R(\boldsymbol{q}_a, \boldsymbol{q}_b) = \sum_{i=0}^{n-1} \left(\frac{1}{speed_c(d_{avg}(\boldsymbol{q}(\frac{i}{n}), \boldsymbol{q}(\frac{i+1}{n})))} \left\| \boldsymbol{q}(\frac{i+1}{n}) - \boldsymbol{q}(\frac{i}{n}) \right\| \right) \tag{4.5}$$

mit $d_{avg}(\boldsymbol{q}(u_i), \boldsymbol{q}(u_{i+1}))$ als Durchschnittsdistanz für das Teilstück von $\boldsymbol{q}(u_i)$ nach $\boldsymbol{q}(u_{i+1})$. Für die Grenzwertentwicklung $n \to \infty$ und eine stetige Funktion $\boldsymbol{q}(u)$ entspricht dies dem Kurvenintegral über $\boldsymbol{q}(u)$:

$$t_R(\boldsymbol{q}_a, \boldsymbol{q}_b) = \int_0^1 \left(\frac{1}{speed_c(d(\boldsymbol{q}(u)))} \right) \left\| \frac{\delta \boldsymbol{q}(u)}{\delta u} \right\| du \tag{4.6}$$

Für lineare Bewegungen lässt sich die Streckenlänge vor das Integral ziehen:

$$t_R(\boldsymbol{q}_a, \boldsymbol{q}_b) = \left\| \boldsymbol{q}_b - \boldsymbol{q}_a \right\| \int_0^1 \left(\frac{1}{speed_c(d(\boldsymbol{q}(u)))} \right) du \tag{4.7}$$

Diese verallgemeinerte Beschreibung ist genau genommen auch eine Approximation der Realität, denn sie vernachlässigt die Dynamik des Roboters und geht von einer unendlich hohen Beschleunigung aus (B3). Dies ist jedoch für die Praxis eine sinnvolle Annahme, da die PTP-Bewegungen überschliffen werden und daher der Anteil der Beschleunigungsphasen minimiert wird.

Kollisionstestkosten und Echtzeitfähigkeit durch Schätzung

Die exakte Berechnung der Kosten mittels der Ausführungsdauer einer Kante setzt voraus, dass ein Kollisionstest existiert, der Abstände entlang der Kanten liefern kann und dies wegen Anforderung A1 in Echtzeit (weitgehend) unabhängig von der Anzahl und Länge der getesteten Kanten. Die Laufzeiten des hier eingesetzten Kollisionstests (siehe Kapitel 2) sind jedoch schon für Tests einzelner Konfigurationen erheblich (siehe Abbildung 2.32) und liegen für die hier verwendeten, moderaten Ortsauflösungen bei typischen Laufzeiten von etwa 1,3 Millisekunden.

Aufgrund dieser Bedingungen (siehe auch B1) ist es notwendig, die Anzahl der Kollisionstests zu beschränken und die unbekannten Distanzen in den Knoten und Kanten des Graphen aus den Berechneten zu schätzen, denn die Rechendauer $t_{colltest}$ für den Kollisionstest ist deutlich höher als die Rechendauer t_{est} für eine Schätzung sowohl der Kanten (e) als auch der Knoten (v):

$$\left(t_{colltest}(e) \gg t_{est}(e) \right) \quad \wedge \quad \left(t_{colltest}(v) \gg t_{est}(v) \right) \tag{4.8}$$

Aus Experimenten bestimmt sich für die weiter unten beschriebene Distanzschätzung einer Konfiguration aus der Distanzinformation an einer anderen Konfiguration eine durchschnittliche Laufzeit von 0.310446 * 10^{-6} Sekunden. Somit ergibt sich im Schnitt ein

Faktor von mehr als 2500 zwischen der Laufzeit der Schätzung und der durchschnittlichen Laufzeit der Distanzberechnung (siehe Daten aus Abbildung 2.32). Dieses Verhältnis fällt für die Kantenschätzung noch wesentlich gravierender aus und hängt von Kantenlänge und Diskretisierung ab. Weil Kantentest also deutlich teurer sind und schon ein Kantentest vor der Ausführung des Pfades steht (Algorithmus 4.1, Zeile 4), werden während der Graphensuche nur Knotentests vorgenommen und die Distanzen entlang der Kanten vollständig geschätzt, beziehungsweise lediglich in Spezialfällen berechnet (siehe hierzu auch die Argumentation zur Platzierung der Kollisionstests in den Knoten des Graphen in Kapitel 3).

Ist die maximale Anzahl an Kollisionstests aufgeteilt in $MAXCOLL_{path}$ für den Test des Pfades vor der Ausführung und $MAXCOLL_{plan}$ für die Distanzberechnungen während der Graphensuche, so lässt sich für die Graphensuche eine Abschätzung der Laufzeit vornehmen. Mit den Bedingungen aus Formel 4.8 lässt sich die Laufzeit t_{pp} der Graphensuche abschätzen als:

$$t_{pp} = MAXCOLL_{plan} \cdot (t_{colltest} + c_{update} \cdot t_{est}) + t_{search} \tag{4.9}$$

mit t_{search} als Laufzeit des Suchanteils ohne Distanzberechnungen auf dem Graphen. Der Faktor c_{update} ergibt sich durch die notwendige Aktualisierung der Knoten und Kantenschätzungen (siehe Kapitel 4.5) bei Neuberechnung einer Distanz. Wie noch gezeigt wird, ist c_{update} von der Komplexität $O(|V|M)$ mit der Knotenmenge V und durchschnittlich M Kanten pro Knoten. Die Aktualisierung hat daher für einen gegebenen statischen Graphen eine feste maximale Laufzeit. Da auch hier wie beim wegoptimierenden Planer die Laufzeiten des Kollisionstests zwar begrenzt sind, jedoch stark schwanken können und dies ebenso durch die jeweilige Situation bedingt für die Knotenaktualisierung gilt, ist es alternativ sinnvoll, die maximale Anzahl an Kollisionstests zu erhöhen und nach jedem Kollisionstest aufgrund der verbleibenden Restzeit zu entscheiden, ob ein weiterer Kollisionstest möglich ist.

Detailierte Kapitelübersicht

In den folgenden Kapiteln wird zunächst dargestellt, wie die Distanzinformationen in den Knoten und Kanten aus den vorhandenen Informationen geschätzt werden können (Kapitel 4.5 und 4.6). Darauf aufbauend wird die Einbindung der Schätzung in die Graphensuche (Kapitel 4.8) inklusive adaptierter Platzierung der Distanzberechnungen dargestellt und anschließend die Wirkungen unterschiedlicher Schätzstrategien für Kanten, Knoten und die Zielheuristik (Kapitel 4.9). Kapitel 4.10 widmet sich dem Aspekt der Arbeitraumdynamik und dessen Behandlung. Experimentelle Ergebnisse des gesamten Planers werden gesondert in Kapitel 5 dargestellt.

4.5 Knotenschätzung

Im folgenden Kapitel wird ein Rahmenwerk für die Schätzung von Distanzen auf einem Graphen aus wenigen vorhanden Distanzinformationen erarbeitet. Dabei wird zunächst die prinzipielle Schätzung aus einer Distanzinformation (Kapitel 4.5.1) erläutert, danach die Schätzung aus mehreren Distanzinformationen (Kapitel 4.5.2) und abschließend die konsistente Aktualisierung des Graphen nach einer Distanzberechnung (Kapitel 4.5.3).

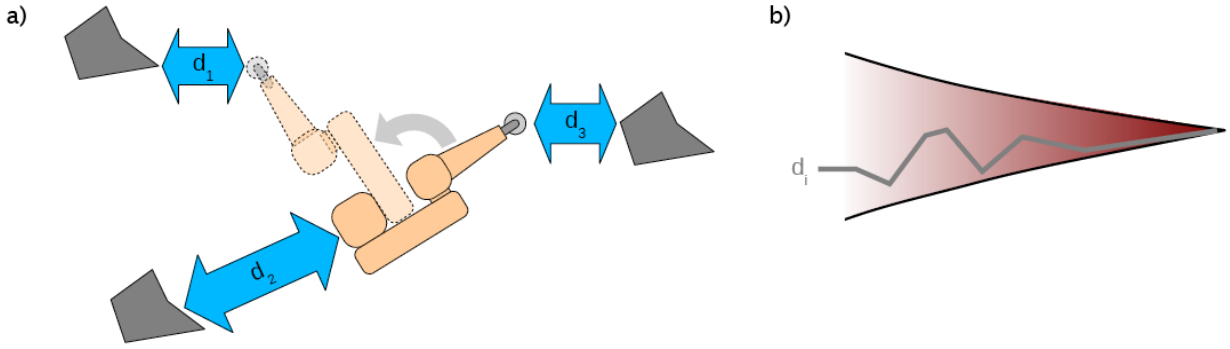


Abbildung 4.3: a) Auswahl von Distanzverläufen für eine Roboterbewegung (grauer Pfeil): d_1 nimmt monoton zu, d_2 bleibt im wesentlichen konstant und d_3 nimmt monoton ab b) Prinzipielles Distanzverlaufsspektrum für eine Distanz d_i mit eingezeichnetem konkreten Beispielverlauf für d_i.

Das Ziel der Knotenschätzung ist eine recheneffiziente Schätzung der Distanz eines Knotens zum nächsten unbekannten Objekt aus der vorhandenen Distanzinformationen in den Knoten des Planungsgraphen. Dabei besteht die besondere Aufgabe bedingt durch Anforderung A3 darin, für eine Roboterkinematik mit einer großen Anzahl an Freiheitsgraden für eine gegebene lineare Bewegung entlang einer Kante des Graphen im Konfigurationsraum eine Veränderung der Objektdistanz zu schätzen, ohne die tatsächliche Hindernissituation zu kennen, denn deren Berücksichtigung würde die Verwendung der Distanzberechnung implizieren, die jedoch durch die Schätzung gerade vermieden werden soll (Bedingung B1). Aufgrund der unbekannten Hindernissituation gibt es ein großes Spektrum an möglichen Verläufen der Minimaldistanz für eine gegebene Bewegung des Roboterarms von einem Knoten des Graphen zum nächsten entlang einer Kante, wie in Abbildung 4.3 beispielhaft skizziert wird. Eine Schätzung der Distanz soll daher die Bandbreite dieser Möglichkeiten abschätzen. Dies soll zur Gewährleistung einer sicheren Planungsgrundlage konservativ geschehen für eine angenommene „Schlimmster Fall"-Situation (ähnlich den Distanzen d_1 und d_3 in Abbildung 4.3).

4.5.1 Distanzintervallschätzung

Gesucht ist die Schätzung der Distanzen zu unbekannten Objekten eines Knotens $\boldsymbol{q}_b$ aus den berechneten Distanzen eines Knotens $\boldsymbol{q}_a$. Ausgangspunkt der Schätzung ist eine Menge von n berechneten Distanzen d_i mit $i = 1,...,n$ zu jeweils einem zugeordneten

Testvolumen v_i des Roboterarms (siehe Kapitel 2.4) in $\boldsymbol{q}_a$. Von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ gibt es eine Gerade $\boldsymbol{q}(t)$ (für t von 0 bis 1) im Konfigurationsraum mit $\boldsymbol{q}_a = \boldsymbol{q}(0)$ und $\boldsymbol{q}_b = \boldsymbol{q}(1)$. Gesucht ist die jeweilige Unter- und Obergrenze (Distanzintervall) für die Distanzen zu den Hindernissen $\boldsymbol{H}_l$ bei der Bewegunge des Roboters von der Konfiguration $\boldsymbol{q}_a$ zur Konfiguration $\boldsymbol{q}_b$.

Dazu betrachtet man die Anfragevolumen v_i, für welche im Kollisionstest jeweils die Distanz berechnet wurde. Ganz allgemein existiert für jedes v_i eine Funktion, die seine Bewegung abhängig von einer gegebenen Bahn im Konfigurationsraum beschreibt, sodass für jeden Punkt $\boldsymbol{p}(0)$ aus dem Volumen v_i gilt:

$$\boldsymbol{p}(\boldsymbol{q}(t)) = f_p(q(t)) = \boldsymbol{T}(\boldsymbol{q}(t)) \cdot \boldsymbol{p}(0) \quad t \in [0,1], \boldsymbol{p}(0) \in v_i \tag{4.10}$$

In der Formel 4.10 stellt $\boldsymbol{T}(\boldsymbol{q}(t))$ eine homogene Transformation dar, die eindeutig aus der zum jeweiligen Zeitpunkt gegebenen Konfiguration bestimmt werden kann. Die Anfragevolumina beschränken sich daher auf starre Körper, deren Form sich während der Bewegung nicht verändert (Bedingung B2).

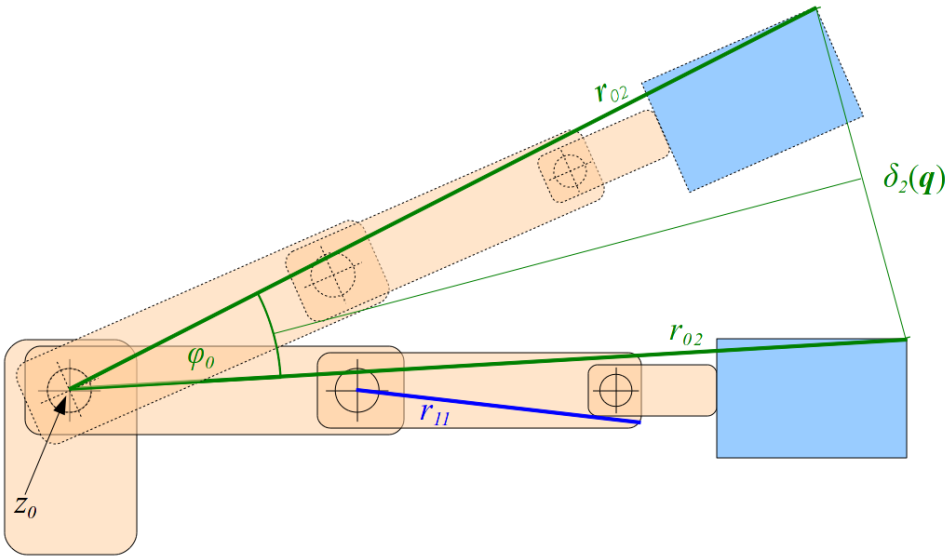


Abbildung 4.4: Beispielberechnung von „Schlimmster Fall"-Bewegungsdistanzen für einen Knickarmroboter mit drei Gelenkachsen z_0 bis z_2 und drei Gliedern ohne die Basis (Glied 0 verbindet Achse 0 und 1, Glied 1 ist mit Glied 0 über Achse 1 verbunden und Glied 2 ist mit Glied 1 über Achse 2 verbunden). Mit Glied 2 starr verbunden ist ein transportiertes Objekt in Blau. Eine Bewegung in z_0 um den Winkel φ_0 ist durch die gestrichelte Kopie des ausgestreckten Arms (dies ist der schlimmste Fall) dargestellt. Beispielhaft ist eine gesuchte Distanz $\delta_2(\boldsymbol{q})$ eingetragen, die in diesem Szenario lediglich von φ_0 und r_{02} abhängt, da die anderen Gelenke nicht bewegt werden.

Da die Hindernissituation für die Zeitdauer der Planung als konstant angenommen wird (siehe Anforderung A2), ist für die Schätzung der maximalen Distanzänderungen δ_i in positive und negative Richtung allein die Bewegung des Roboters entscheidend und damit die jedes Punkts der Oberfläche des Roboters, die eine Teilmenge der Anfragevolumina v_i ist. Der Betrag der Distanzänderungen δ_i kann dann bestimmt werden durch:

$$\delta_i = \max_{p,t} \left(\left\| f_p(\boldsymbol{q}(t)) - f_p(\boldsymbol{q}(0)) \right\| \right), \quad p \in v_i \tag{4.11}$$

Zur recheneffizienten Abschätzung der maximal zurückgelegten Distanzen für jeden Punkt auf der Roboteroberfläche gibt es aus der Literatur bekannte Ansätze, die für eine vorgegebene Bewegungsdistanz eines Punktes eine minimal notwendige Roboterbewegung berechnen [Henrich98], dort als MAXMOVE-Funktion bezeichnet. Gesucht ist hier die Umkehrung (MAXMOVE^{-1}), die für einen beliebigen Differenzvektor $\boldsymbol{q} = \boldsymbol{q}_b - \boldsymbol{q}_a$ im Konfigurationsraum die maximale Bewegungsdistanz δ für einen beliebigen Punkt auf der Oberfläche des Roboters liefert: δ = MAXMOVE^{-1}($\boldsymbol{q}$). Diese Berechnung wird im Folgenden kurz dargestellt.

Für diese Approximation wird ein Knickarmroboter betrachtet (serielle Kinematik mit rein rotatorischen Gelenken, siehe auch Abbildung 4.4). Um die Komplexität der exakten Berechnung der Bewegungsdistanz eines beliebigen Punktes auf der Oberfläche zu vermeiden, die von der Startkonfiguration $\boldsymbol{q}_a$ der Bewegung abhängt, wird als schlimmster Fall ein ausgestreckter Arm angenommen.

Sei als einfaches Beispiel die Situation aus Abbildung 4.4 gegeben. Hier wird der ausgestreckte Arm um die Rotationsachse z_0 um den Winkel φ_0 gedreht (im Folgenden bezeichnen die φ_i, $i \in N$ die Komponenten von $\boldsymbol{q}$). Alle anderen Gelenke seien unbewegt. Der Radius r_{02} ist das Maximum der Distanz aller Punkte der Robotergeometrie von der Gelenkachse z_0 inklusive Glied 2 und dem transportierten Objekt. Diese Position dieser Punkte wird nur durch φ_0 parametrisiert, wenn wie in diesem Beispiel keine anderen Gelenke bewegt werden. Dann ergibt sich die maximale Distanz $\delta_2(\boldsymbol{q})$, die ein Punkt der Robotergeometrie sich von seiner Ursprungsposition entfernt, in diesem Beispiel zu:

$$\delta_2(\boldsymbol{q}) = r_{02} \cdot f(|\varphi_0|) \tag{4.12}$$

mit

$$f(\varphi) = \begin{Bmatrix} 2 \cdot \sin\left(\frac{\varphi}{2}\right) & , \text{falls } \varphi \leq \pi \\ 2 & , \text{falls } \varphi > \pi \end{Bmatrix}, \varphi \in \mathbb{R}_0^+, f(\varphi) \in [0,2] \tag{4.13}$$

Für einen Punkt auf dem Roboterkörper sind jedoch im Allgemeinen mehrere Gelenkbewegungen um φ_i in den Gelenkachsen z_i relevant. Mit den Radien r_{ij} vom Rotationszentrum z_i zum maximal entfernten Punkt auf dem Glied j, für den die Bewegungsdistanz δ_j berechnet werden soll, ergibt sich somit eine Summe über alle Gelenke, die mit ihrer Bewegung die Punkte des entsprechenden Gelenks beeinflussen:

$$\delta_j(\boldsymbol{q}) = MAXMOVE_j^{-1}(\boldsymbol{q}) = \sum_{i \leq j} r_{ij} \cdot f(|\varphi_i|) \tag{4.14}$$

Die n abgeschätzten Distanzintervalle an der Konfiguration $\boldsymbol{q}_b$ ergeben sich damit als

Vektor $\boldsymbol{I}$ von Intervallen aus den n berechneten Distanzen d_i in Konfiguration $\boldsymbol{q}_a$ und den durch die Bewegung entlang der Gerade von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ entstehenden Distanzen δ_i sowohl in negative als auch positive Richtung, da sich der Roboter durch die Bewegung den unbekannten Objekten sowohl annähern als auch von ihnen entfernen kann:

$$\left(\left[d_0-\delta_0(\boldsymbol{q}), d_0+\delta_0(\boldsymbol{q})\right] \ , \ldots , \ \left[d_n-\delta_n(\boldsymbol{q}), d_n+\delta_n(\boldsymbol{q})\right]\right) \tag{4.15}$$

Dieses Konzept lässt sich verallgemeinern, sodass in jedem Knoten des Graphen die Distanzen generell als Intervalle $[l, u]$ mit Untergrenze l und Obergrenze u gespeichert werden, wobei für berechnete Distanzen die Ober- und Untergrenze gleich ist. Jeder Knoten besitzt somit einen Vektor von Distanzintervallen $\boldsymbol{I}$. Dann lässt sich ohne Fallunterscheidung eine Schätzungsfunktion *EST*(...) definieren, die Distanzintervalle sowohl aus bereits geschätzten Distanzintervallen als auch aus berechneten Distanzen bestimmen kann:

$$EST(\boldsymbol{I}, \boldsymbol{q}) = \left(\left[l_0(\boldsymbol{I})-\delta_0(\boldsymbol{q}), u_0(\boldsymbol{I})+\delta_0(\boldsymbol{q})\right], \ldots, \left[l_n(\boldsymbol{I})-\delta_n(\boldsymbol{q}), u_n(\boldsymbol{I})+\delta_n(\boldsymbol{q})\right]\right) \tag{4.16}$$

Die Funktion $l_i(\boldsymbol{I})$ liefert die Untergrenze des i-ten Intervalls des Vektors $\boldsymbol{I}$, $u_i(\boldsymbol{I})$ liefert analog die Obergrenze. Die Funktion *EST*(...) ist wie oben auch für ein einzelnes Intervall I definiert.

4.5.2 Intervallfusion

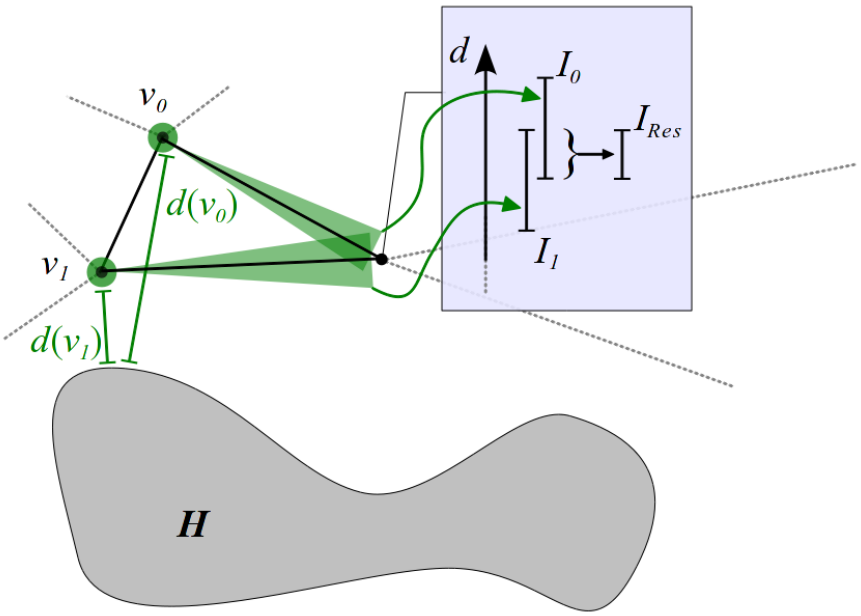


*Abbildung 4.5: Intervallfusion am Beispiel zweier Schätzungen aus Nachbarknoten v_0, v_1 mit den jeweils berechneten Distanzen $d(v_0)$, $d(v_1)$ zum unbekannten Objekt **H**. Das resultierende Intervall I_{Res} entsteht aus der Überlappung der geschätzten Intervalle I_0, I_1.*

Die Distanzschätzung für einen Knoten des Planungsgraphen wird deutlich verbessert, wenn nicht nur die Informationen **eines** Nachbarknotens zur Berechnung der Schätzung

verwendet wird, sondern die aller benachbarter Knoten im Graph. Gesucht ist also eine Funktion *FUSE*(...), die eine Menge von *k* Vektoren von jeweils *n* Distanzintervallen zu einem Vektor von *n* Distanzintervallen fusioniert, der die verbesserte Schätzung repräsentiert:

$$\left(\left[l_{0,}u_0\right],\ldots,\left[l_i,u_i\right],\ldots,\left[l_n,u_n\right]\right)=FUSE\left(\ldots,\left(\left[l_0,u_0\right],\ldots\right)_j,\ldots\right),\quad j\in 1...\mathrm{k} \tag{4.17}$$

Jedes Intervall an Position *i* in einem Intervallvektor wird unabhängig von den anderen Positionen aus den entsprechenden Intervallen der Eingaben fusioniert. Zur Fusion der Eingabeintervalle an einer bestimmten Position *i* wird im folgenden eine Operation $DF(I_0, I_1)$ (**D**istance Interval **F**usion) zur Fusion zweier Intervalle I_0, I_1 vorgestellt (siehe hierzu auch Abbildung 4.5).

Grundlage der Fusion zweier Schätzungsintervalle bildet dabei eine Definition des „Informationsgewinns“ für die Fusion. Dabei erzeugt die Fusionsmethode den größen Informationsgewinn, die das kleinste zulässige Intervall erzeugt. Die Definition der Zulässigkeit fußt dabei auf der Konservativität der MAXMOVE^{-1}-Schätzung der Roboterbewegung, die die Grundlage für die einzelnen Intervallschätzungen bildet. Das resultierende Intervall muss die maximal mögliche Roboterbewegung basierend auf den gegebenen Informationen enthalten. Da jede aus den Informationen der Nachbarn erzeugte Intervallschätzung gültig ist und die maximal mögliche Roboterbewegung jeweils korrekt beschreibt, kann das Zielintervall wie folgt sicher fusioniert werden (siehe auch Abbildung 4.5):

$$I_{new}=\left[l_{new},u_{new}\right]=DF\left(I_{0,}I_1\right)=DF\left(\left[l_{0,}u_0\right],\left[l_{1,}u_1\right]\right)=\left[max\left(l_{0,}l_1\right),min\left(u_{0,}u_1\right)\right] \tag{4.18}$$

Die Untergrenze wird auf das Maximum der aus den Nachbarn geschätzten Untergrenzen gesetzt, da dies die minimale Distanz darstellt, die in diesem Knoten aus Sicht mindestens eines Nachbarknotens konservativ erreicht wird und damit nicht unterschritten werden kann. Für die Obergrenze ergibt sich die analoge Argumentation. Für die Betrachtung in den folgenden Abschnitten wird die Menge der Intervalle/berechneten Distanzen pro Knoten auf ein globales Intervall für den gesamten Roboterarm o.B.d.A. reduziert.

Die Operation *DF* kann, da sie als Ergebnis wieder ein Intervall erzeugt, beliebig auf den möglichen Paaren von Intervallen und Ergebnissen von Fusionen ausgeführt werden. Sie wird dasselbe Ergebnis liefern, wenn alle Intervalle mindestens einmal berücksichtigt werden, insbesondere da gilt:

- Idempotenz: $I_0 = DF(I_0, I_0)$
- Symmetrie: $DF(I_0, I_1) = DF(I_1, I_0)$
- Assoziativität: $DF(I_0, DF(I_1, I_2)) = DF(DF(I_0, I_1), I_2)$

- Distributivität: $DF(DF(I_0, I_1), I_2) = DF(DF(I_0, I_2), DF(I_1, I_2))$

Eine einfache Möglichkeit der Realisierung der Funktion *FUSE* ist bespielsweise die sequentielle Abarbeitung einer Menge von Intervallen $I_0,...,I_n$: $Serialize(I_0,...,I_n) := DF(I_0, Serialize(I_1,...,I_n))$ mit $Serialize(I) := I$. Auch baumartige Zusammenfassungen sind zur Steigerung der Effizienz vorstellbar.

Der Operator *DF* ist auch auf Intervallvektoren definierbar:

$$DF\left(\boldsymbol{I}_a, \boldsymbol{I}_b\right) = \left(DF\left(I_0(\boldsymbol{I}_a), I_0(\boldsymbol{I}_b)\right), \ldots, DF\left(I_n(\boldsymbol{I}_a), I_n(\boldsymbol{I}_b)\right)\right) \tag{4.19}$$

Die Funktion $I_i(\boldsymbol{I})$ liefert das *i*-te Intervall aus $\boldsymbol{I}$. Oben aufgeführte Eigenschaften gelten auch hier.

4.5.3 Indirekte Distanzpropagierung

Dieser Abschnitt befasst sich mit der Verteilung (Propagierung) einer geänderten Distanzinformation am Knoten *v* an seine Nachbarschaft, woraus die Bezeichnung „indirekt“ (über die Nachbarschaft) resultiert. Diese Information kann aus einer Berechnung oder wiederum einer Schätzung stammen. Im ersten Teil dieses Abschnitts werden Grundbegriffe eingeführt, die im Weiteren zur Diskussion der Algorithmen verwendet werden, welche im zweiten und dritten Teil dargestellt werden. Im abschließenden Teil werden die vorgestellten Algorithmen und mögliche Optimierungen experimentell verglichen und diskutiert. Im darauffolgenden Abschnitt (4.5.4) wird die direkte Distanzpropagierung präsentiert, die im Vergleich zu der hier dargestellten Methode die Distanzen in geschätzten Knoten auf einer direkten, gedachten Verbindung berechnet.

Begriffsdefinitionen

Definition 4.1: *Ein Intervall I ist **intervallkonsistent**, wenn das Prädikat IK(I) := ($l \leq u$) wahr ist.*

Nach Anwendung des Operators *DF*(...) kann das resultierende Intervall prinzipiell inkonsistent sein. Dies kann auf ungültige Distanzmessungen zurückgeführt werden, die beispielsweise durch einen fehlerhaften Kollisionstest oder durch Objektbewegungen entstehen. Letzteres tritt auf, wenn Distanzmessungen von Zyklus zu Zyklus beibehalten werden. Dieser Fall kann auf verschiedene Arten behandelt werden, welche in Kapitel 4.10 dargestellt werden. $IK(\boldsymbol{I})$ ist analog für Intervallvektoren definiert.

Definition 4.2: *Ein Knoten v_0 ist **schätzungkonsistent** bezüglich eines Knotens v_1, wenn das Prädikat $SK(v_0, v_1)$ wahr ist:*

$$SK(v_0, v_1) := IK(\boldsymbol{I}(v_0)) \wedge \left(DF\left(EST\left(\boldsymbol{I}(v_1), \boldsymbol{q}(v_0) - \boldsymbol{q}(v_1)\right), \boldsymbol{I}(v_0)\right) = \boldsymbol{I}(v_0)\right)$$

Wobei $\boldsymbol{I}(v)$ den Intervallvektor eines Knoten v liefert und $\boldsymbol{q}(v)$ die zugehörige Roboterkonfiguration.

Dies bedeutet, dass der Intervallvektor des Knotens v_0 durch die Fusion mit der Schätzung ausgehend von v_1 nicht verändert wird, d.h. dass die Schätzung von v_1 aus keinen Informationsgewinn erzeugen kann.

Definition 4.3: *Ein Knoten v ist* ***nachbarkonsistent****, wenn das Prädikat NK(v) wahr ist:*

$$NK(v) := \left(\forall\, v_i \in neighbours(v) : SK(v, v_i) \right)$$

Die Funktion *neighbours*(*v*) liefert die durch Kanten verbundenen Nachbarn des Knoten *v*. Diese Knoten werden im Folgenden auch als *Nachbarknoten* bezeichnet. Nachbarkonsistente Knoten werden im Folgenden auch kurz als *konsistent* bezeichnet.

Konsistenzinvariante. Die Nachbarkonsistenz gilt für alle Knoten des Graphen. Sie gilt, bis eine Änderung beispielsweise durch Distanzberechnungen auftritt. Nach einer entsprechenden Anpassung des Graphen muss die Invariante wiederum erfüllt sein. Diese hier *direkte bzw. indirekte Distanzpropagierung* genannte Anpassung wird im Folgenden beschrieben. Initial sind alle Knoten des Graphen auf das Intervall $[-\infty, \infty]$ gesetzt und daher nachbarkonsistent.

Indirekte Distanzpropagierung durch Nachbarschaftsbeziehungen

Bei jeder Änderung eines Intervallvektors in einem Knoten (z.B. durch Distanzberechnung) muss diese durch den Graphen propagiert werden und alle betroffenen Knoten angepasst werden, um die Nachbarkonsistenz aufrecht zu erhalten. Diese Anpassung betrifft im Allgemeinen nicht alle Knoten des Graphen. Ausgehend vom geänderten Punkt wird die Änderung rekursiv zu den Nachbarknoten propagiert, solange wie eine Änderung auftritt. Da die Änderungen unidirektional sind, wie im Folgenden hergeleitet wird, fällt die Propagierung nicht auf den auslösenden Punkt zurück, terminiert und kann effizient durchgeführt werden.

Ausgangspunkt für diese Betrachtung ist die Konsistenzinvariante des Graphen. Wird nun die Distanzinformation eines Knoten v_c aus den Distanzinformationen aller Nachbarn berechnet über den Operator *DF*(...), so wird qua Konstruktion von *DF*(...) in v_c ein nachbarkonsistenter Intervallvektor erzeugt. Ist dies nicht der Fall, liegt ein Fehler vor, der detailliert in Kapitel 4.10 besprochen wird. Änderungen durch Distanzberechnung in v_c müssen ebenfalls auf ihre Nachbarkonsistenz überprüft werden, denn die Berechnung muss nach Definition von MAXMOVE in einer als statisch betrachteten Umwelt innerhalb des aus den Nachbarknoten geschätzten Intervallvektors liegen, also nachbarkonsistent sein. Tritt hierbei eine Inkonsistenz auf, ist dies ebenso ein Fehlerfall.

Nach der Änderung des Intervallvektors von v_c durch Schätzung oder Berechnung ist dieser Knoten zwar nachbarkonsistent, jedoch seine Nachbarknoten potentiell nicht mehr. Das heißt also, dass es in diesem Fall für einen veränderten Knoten v_c keinen oder mehrere

Nachbarknoten v_i gibt, die nicht mehr nachbarkonsistent sind. Diese Inkonsistenz kann nur durch den veränderten Knoten verursacht werden, da die Knoten nach Vorbedingung vor der Änderung nachbarkonsistent zu allen ihren Nachbarknoten waren.

Unidirektionalität der Änderung

Wenn nun aufgrund von Inkonsistenzen eine Anpassung von Intervallgrenzen notwendig wird, so ist diese ausgehend vom veränderten Knoten v_c im Graphen unidirektional, wie im Folgenden gezeigt wird. Eine notwendige Anpassung kann prinzipiell sowohl die obere als auch die untere Intervallgrenze in einem inkonsistenten Knoten betreffen. Da der Fall für die untere Grenze analog ist, wird hier o.B.d.A. der Fall betrachtet, dass sich die *j*-te obere Grenze $u_j(v_i)$ des Nachbarknotens v_i ändern muss, dieser also inkonsistent ist und somit gilt:

$$u_j(v_c)+\delta_j\left|\boldsymbol{q}(v_i)-\boldsymbol{q}(v_c)\right|<u_j(v_i) \tag{4.20}$$

Diese Ungleichung muss so gelten, da die Knoten vor der Änderung konsistent waren und durch die Änderung das Intervall von v_c nur kleiner geworden sein kann. Ausgehend von dieser Situation wird hier nun durch einen Widerspruchsbeweis gezeigt, dass die Inkonsistenz in den Nachbarknoten nicht auf den verursachenden Knoten v_c zurückfällt. Nach Anpassung des Nachbarknoten wird die Ungleichung 4.20 zur Gleichung (siehe Definition von *DF*(...)). Wenn nun der Knoten v_c durch diese Anpassung bezüglich v_i nicht mehr nachbarkonsistent wäre, so müsste gelten:

$$u_j(v_i)+\delta_j\left|\boldsymbol{q}(v_i)-\boldsymbol{q}(v_c)\right|<u_j(v_c) \tag{4.21}$$

Die Summe von (4.20) und (4.21) ergibt jedoch für nicht-negative und symmetrische δ ein Widerspruch. Somit erhält jegliche Anpassung der Schätzintervalle der Nachbarknoten die Nachbarkonsistenz des die Veränderung verursachenden Knoten, die initiale Nachbarkonsistenz vorausgesetzt.

Diese Betrachtung gilt auch für alle Folgeknoten im Graphen, für die eine Anpassung notwendig ist. Wie sich leicht zeigen lässt, können so auch keine endlosen Anpassungsschleifen entstehen.

Kürzeste Wege für die Anpassung

Für Knoten, die nicht zu den Nachbarknoten von v_c gehören, gilt, dass die betragsmäßig maximale Anpassung einer Grenze nur auf dem kürzesten Weg (in δ gemessen) erfolgen kann. Das heißt, es gibt eine Sequenz von Zwischenknoten („hops“), sodass für einen beliebigen Knoten *v* die folgende Summe minimal wird:

$$u_j(v_c)+\sum_i \delta_j\left|\boldsymbol{q}(v_{i+1})-\boldsymbol{q}(v_i)\right|,\ \text{mit } i\in 0...\mathrm{N},\ v_0=v_c \text{ und } v_{N+1}=v \tag{4.22}$$

```
(1)  init(v)
(2)      CLOSED_du = OPEN_du = {}; push(v, CLOSED_du)
(3)      forall v_i in neighbours(v)
(4)          insert(v_i, OPEN_du) with key(v_i) = δ(q(v_i) – q(v)) and pred(v_i) = v
(5)
(6)  insertOPEN(v, v_pred)
(7)      if ( untouched(v) or ( (key(v_pred) + δ(q(v) – q(v_pred))) < key(v) )
(8)          if (v in OPEN_du) remove(v, OPEN_du)
(9)          insert(v, OPEN_du) with
(10)             key(v) = key(v_pred) + δ(q(v) – q(v_pred)) and pred(v) = v_pred
(11)
(12) estimateDist(v_from, v_to)
(13)     if not SK(v_to, v_from)
(14)         if ( state(v_to) equals CALC ) raise estimation-error
(15)         else
(16)             I(v_to) = DF(EST(I(v_from), q(v_to) − q(v_from)), I(v_to))
(17)             if (not IK(v_to)) raise estimation-error
(18)         return true
(19)     else return false
(20)
(21) updateDistanceEstimations(v_c, COLLECTEDNODES)
(22)     push(v_c, COLLECTEDNODES)
(23)     init(v_c)
(24)     while (OPEN_du not empty)
(25)         v_curr = pop(OPEN_du)
(26)         push(v_curr, CLOSED_du)
(27)         push(v_curr, COLLECTEDNODES)
(28)         if (estimateDist(pred(v_curr), v_curr))
(29)             forall v_i in neighbours(v_curr)
(30)                 if (v_i not in CLOSED_du)
(31)                     insertOPEN(v_i, v_curr)
```

*Algorithmus 4.2: Indirekte Distanzpropagierung nach Änderung von Knoten v_c. Einstiegsfunktion ist updateDistanceEstimations(...). In der Menge **COLLECTEDNODES** werden die veränderten Knoten gesammelt.*

Jede andere Folge kann nicht zu einem besser informierten Intervall führen, da sie die *j*-te obere Grenze im Knoten *v* nicht weiter senken kann, denn sonst wäre eben diese Folge die Minimale. Die einzige Ursache der Anpassung einer Intervallgrenze eines beliebigen Knotens *v* kann auch nur der geänderte Knoten v_c sein, da sonst zuvor keine Nachbarkonsistenz vorgelegen haben kann.

Aus diesen Bedingungen lässt sich ein optimaler Aktualisierungsalgorithmus formulieren, der jeden Knoten, der aktualisiert werden muss, nur einmal aktualisiert. Er entspricht einem Dijkstra-Algorithmus auf dem Graphen, dessen Kantenkosten sich über die Bewegungsdistanz $\delta(e)$ für die jeweilige Kante *e* bestimmen:

$$\delta(e)=\delta\left|\boldsymbol{q}(v_0)-\boldsymbol{q}(v_1)\right| \text{ mit } e=(v_0,v_1) \tag{4.23}$$

Der Algorithmus bezieht sich dabei auf *eindimensionale Intervallvektoren* und ist bei mehrdimensionalen Intervallvektoren für jede Dimension zu wiederholen.

Zur Vereinfachung des Algorithmus (hier und in den folgenden Kapiteln) und zur Behandlung von Spezialfällen werden für die Distanzinformationen der Knoten Zustände eingeführt, die die in Kapitel 3.3 beschriebenen ersetzen. Der Zustand eines Knotens wird von der Funktion *state*(*v*) geliefert und kann folgende Werte annehmen: CALC für berechnete Knoten dessen Intervallgrenzen gleich sind und ESTIM für geschätzte Knoten mit ungleichen Intervallgrenzen.

Der Einsprungpunkt in den Algorithmus (4.2) bildet die Funktion *updateDistanceEstimations*(v_c, ***COLLECTEDNODES***), die für den Knoten v_c aufgerufen wird, dessen Intervall geändert wurde. Die Liste ***COLLECTEDNODES*** enthält abschließend alle Knoten, die von der Änderung betroffen sind, darunter auch solche, bei denen sich nur der Nachbarknoten verändert hat, da somit die Kantenkosten verändert wurden und sich damit die Pfadkosten verändert haben können. Dies wird für die Integration der Schätzung in die Graphensuche benötigt (siehe Kapitel 4.8). Vorbedingung des Algorithmus ist, dass alle Knoten nachbarkonsistent sind bis auf die, die Nachbarn zu dem verursachenden Knoten v_c sind. Der Knoten v_c selbst ist aufgrund ersterer Vorbedingung ebenso nachbarkonsistent. Nachbedingung des Algorithmus ist, dass alle Knoten nachbarkonsistent sind.

In der Funktion *init*(v_c) werden zunächst alle Nachbarn von v_c in $\boldsymbol{OPEN}_{du}$ eingefügt und v_c selbst in $\boldsymbol{CLOSED}_{du}$, da dieser bereits nachbarkonsistent ist und aufgrund der oben beschriebenen Unidirektionalität nicht mehr verändert werden kann. Die sortierten Mengen werden hier als $\boldsymbol{OPEN}_{du}$ und $\boldsymbol{CLOSED}_{du}$ bezeichnet (du = „distance update“), um Verwechslung mit den Mengen ***OPEN*** und ***CLOSED*** aus dem A*-Algorithmus zu vermeiden.

In Zeile 28 wird die Schätzung jedes aus $\boldsymbol{OPEN}_{du}$ entnommenen Knotens über die

Funktion *estimateDist*(v_{from}, v_{to}) angepasst mittels seines Vorgängers *pred*(v_{curr}) im Aktualisierungsbaum, der durch den Dijkstra-Algorithums aufgebaut wird. Diese Anpassung resultiert in das minimale Schätzintervall für diese Knoten, da über den Vorgänger der kürzeste Weg vom verursachenden Knoten v zum aktuellen Knoten v_{curr} läuft, was durch die Konstruktion der Vorgänger garantiert wird (Zeilen 4 und 11).

Die Aktualisierung des Knotens geschieht über die Funktion *estimateDist*(v_{from}, v_{to}). Diese überprüft, ob die Schätzung aktualisiert werden muss (Zeile 13), aktualisiert diese (Zeile 16) und gibt zurück, ob eine Aktualisierung statt gefunden hat. In Zeile 14 enthält die Funktion eine zusätzliche Konsistenzbedingung: Berechnete Distanzen werden nicht angepasst, da ihr Intervall schon minimal ist und innerhalb der Schätzungen der Nachbarn liegen muss. Sollte sich dies dennoch aus der Schätzung ergeben, muss ein Fehlerzustand erzeugt werden (Behandlung siehe Kapitel 4.10). Zusätzlich wird in Zeile 17 überprüft, ob der Knoten nach der Schätzung noch intervallkonsistent ist.

Falls v_{curr} aktualisiert wurde, werden alle Nachbarn, die nicht in $\boldsymbol{CLOSED}_{du}$ sind, durch die Funktion *insertOPEN*(...) bearbeitet (Zeilen 29 – 31 und 6 - 10). Die Funktion *insertOPEN* fügt den Knoten neu in $\boldsymbol{OPEN}_{du}$ ein, falls er nicht schon in $\boldsymbol{OPEN}_{du}$ ist oder löscht diesen Knoten und fügt ihn wieder neu ein falls der neu berechnete *key*(v_i) kleiner ist. Nicht aktualisierte Knoten werden nicht weiter expandiert, da sie ihre Nachbarn nicht verändern können, da diese zu ihnen schon nachbarkonsistent waren (Vorbedingung des Algorithmus).

Komplexität

Potentiell wird jeder Knoten des Graphen einmal in Zeile 25 aus $\boldsymbol{OPEN}_{du}$ entnommen und es werden dann ab Zeile 29 potentiell alle Nachbarknoten untersucht. Für eine Anzahl von N Knoten im Graphen mit jeweils maximal M Nachbarn ergibt sich somit eine Komplexität von O(NM). Dies ist zwar nur linear und begrenzt, bei großen Graphen jedoch potentiell sehr teuer und muss bei jeder Distanzberechnung durchgeführt werden. Für jeden geänderten Knoten müssen zudem die Kosten für alle zugehörigen Kanten neu berechnet werden (siehe folgendes Kapitel). Daher sind die im folgenden dargestellten Optimierungen sinnvoll. Ihr Effekt auf die Laufzeit wird im Abschluss dieses Kapitels in einem Experiment dargestellt.

Optimierungen

Im Folgenden werden zwei Optimierungen dargestellt, die in simulierten und realen Umgebungen einen deutlichen Effekt gezeigt haben und sich daher als Standard-Optimierung für jeden Anwendungsfall empfehlen.

Eine Optimierungsmöglichkeit ist die *Intervalllimitierung*. Dabei wird der Umstand ausgenutzt, dass die Maximalgeschwindigkeit V_{max} des Roboters bei einer bestimmten

Maximaldistanz d_{max} mit $speed(d_{max}) = V_{max}$ erreicht wird, bzw. der Stillstand des Roboters $V_{min} = 0$ bei einer bestimmten Minimaldistanz d_{min} mit $speed(d_{min}) = V_{min}$. Daraus ergibt sich eine sinnvolle Beschränkung des Schätzungsintervalls, die von der ursprünglichen Vorgabe $[-\infty, \infty]$ abweicht und stattdessen $[d_{min}, d_{max}]$ verwendet. Durch diese Begrenzung wird die Propagierung in ihrer Reichweite begrenzt, da in einer gewissen δ-Distanz zum aktualisierten Knoten v_c die Schätzung dieses maximale Intervall überschreitet. Mit dieser Begrenzung wird die Expansion spätestens dann abgebrochen, wenn gilt:

$$\left(\left(u(v_c)+\sum_i \delta(v_i, v_{i+1})\right) > d_{max}\right) \wedge \left(\left(l(v_c)-\sum_i \delta(v_i, v_{i+1})\right) < d_{min}\right) \quad \text{mit } i \in \mathbb{N}, v_{i+1} \in neighbours(v_i), v_0 = v_c \tag{4.24}$$

Diese Begrenzung könnte zu Inkonsistenzen mit tatsächlich berechneten Distanzen führen, da reale Distanzen über und unter der entsprechenden Grenze liegen können. Daher sind auch die berechneten Distanzen zu begrenzen, bevor sie in einem Knoten eingetragen werden, um Fehlerbehandlungen zu vermeiden, die teuer sind. Die tatsächliche Einsparung von Laufzeit durch die Intervalllimitierung ist stark von der der Größe des limitierenden Intervalls abhängig. Ist dieses so weit gefasst, dass potentiell alle Knoten des Graphen aktualisiert werden müssen, ist die Optimierung wirkungslos. Die Intervalllimitierung hat Einfluss auf die Kantenkosten(siehe Folgekapitel).

Eine weitere Optimierung durch Begrenzung der Aktualisierungstiefe referriert auf die in Kapitel 4.8 dargestellte Graphensuche zur Bahnplanung und wird im Folgenden als *Suchmengenlimitierung* bezeichnet. Die Graphensuche untersucht im Allgemeinen nur einen kleineren Teil der Knoten des Graphen während ihrer Ausführung. Daher ist es sinnvoll, die Aktualisierung der Distanzinformation der Knoten zunächst nur auf die *Suchmenge* zu beschränken. Dies sind alle Knoten, die durch die Graphensuche berührt wurden. Während des Expansionschritts der Graphensuche werden die noch nicht berührten Knoten durch ihre Nachbarknoten aus der Suchmenge aktualisiert (Intervallschätzung) und gegebenenfalls eine Propagierung der Distanzinformation innerhalb der Suchmenge durchgeführt.

```
(1) adjustExpandedNode(v, COLLECTEDNODES)
(2)     if(state(v) is CALC)
(3)         if(not NK(v)) raise estimation-error
(4)     else
(5)         makeNeighbourConsistent(v)
(6)     updateDistanceEstimations(v, COLLECTEDNODES)
```

Algorithmus 4.3: Suchmengenoptimierung in der indirekten Distanzpropagierung. Alle Unteraufrufe von Funktionen betrachten lediglich Knoten der Suchmenge.

Die Suchmenge kann durch eine Markierung der Knoten während der Graphensuche kenntlich gemacht werden. Für die Funktion *updateDistanceEstimations*(*v*, ***COLLECTEDNODES***) wirken die Markierungen dann wie ein Filter für die Knoten des Graphen. Dies begrenzt die aktualisierte Menge an Knoten sehr effektiv, erfordert jedoch einen Aufruf der Distanzaktualisierung in jedem Expansionschritt der Graphensuche, soweit dieser einen Knoten zur Suchmenge hinzufügt. Die Funktion *adjustExpandedNode*(...) (Algorithmus 4.3) wird auf jedem Knoten ausgeführt, der zur Suchmenge hinzugefügt wird (siehe auch die Darstellung der Graphensuche in Kapitel 4.8, Algorithmus 4.14 Zeile 8).

In der Funktion *adjustExpandedNode*(…) muss zunächst unterschieden werden, ob ein Knoten mit berechneter Distanz oder geschätzter Distanz zur Suchmenge hinzugefügt werden soll. Wird ein berechneter Knoten hinzugefügt, so muss eine Nachbarkonsistenz zu den Nachbarn in der Suchmenge überprüft werden (Zeile 3) und im Fehlerfall eine Behandlung erfolgen. Im Falle eines geschätzten Knotens wird der Knoten aus den nachbarkonsistenten Nachbarknoten in der Suchmenge geschätzt (Zeile 5). Im Abschluss werden alle benachbarten Punkte in der Suchmenge aktualisiert. Dabei ist die Funktion *updateDistanceEstimations*(...) so modifiziert, dass auch sie nur die Knoten der Suchmenge betrachtet.

Die Suchmengenlimitierung zeigt in Experimenten eine sehr effektive Reduzierung der Planungszeit, die Effizienz dieser Optimierung ist jedoch stark von der Anzahl der expandierten Knoten abhängig, d.h. also der aktuellen Hindernissituation. Für die unoptimierte Variante können allerdings auch recht hohe Laufzeitschwankungen aufgrund der aktuellen Hindernissituation auftreten.

In Abbildung 4.6 sind die Ergebnisse eines vergleichenden Testlaufs in einer 2D-Simulationsumgebung (siehe auch Kapitel 5.1) für die verschiedenen Optimierungs-kombinationen dargestellt, jeweils im Verhältnis zur nicht-optimierten Variante und im Vergleich untereinander. Als Experimentszenario wurde ein bewegtes Objekt gewählt, um in jedem Planungszyklus die Neuberechnung der Distanzen zu erzwingen und somit die Distanzpropagierungen auszulösen. An den Ergebnissen lässt sich erkennen, dass sowohl die Intervalllimitierung als auch die Suchmengenlimitierung ähnlich gute Ergebnisse produzieren mit jeweils 65 bis 70 % an mittlerer Laufzeitreduktion, was in der Kombination beider Optimierungen noch ein wenig gesteigert werden kann. Die Ergebnisse für die Intervalllimitierung sind jedoch nicht parameterfrei und sind daher umso effektiver, je niedriger die Grenzen angesetzt sind und damit je weniger Knoten des Graphen von der Aktualisierung betroffen sind. In realen Anwendungen kommt es daher darauf an, wie sich das Verhältnis von effektiver Arbeitsraumgröße des Roboters zu der maximal messbaren Distanz zu unbekannten Objekten gestaltet.

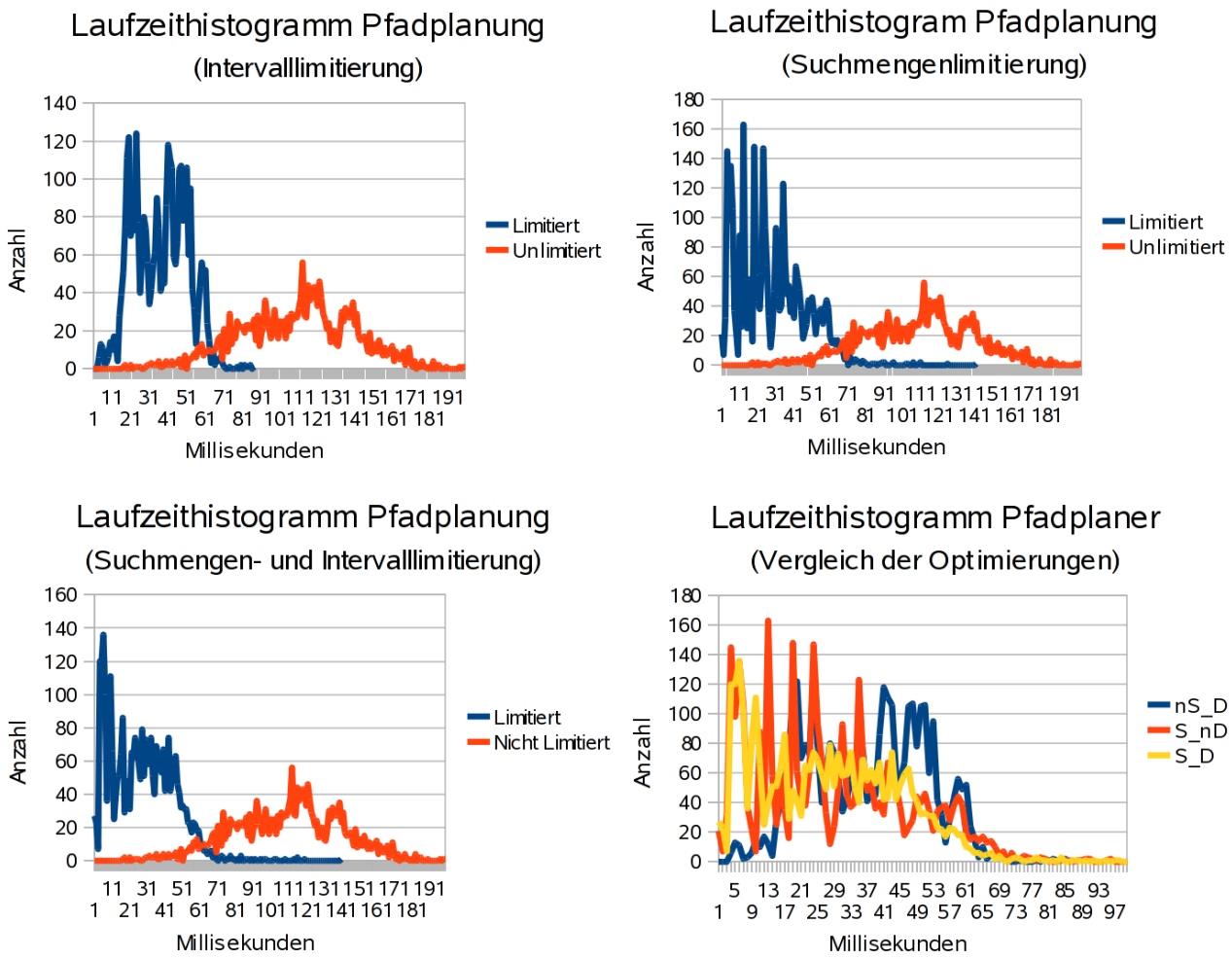


Abbildung 4.6: Laufzeithistogramme für die Bahnplanung mit den oben genannten Optimierungsmethoden in der indirekten Distanzpropagierung. Es wurden jeweils 20 Durchläufe eines Experimentszenarios mit bewegtem Objekt berechnet (Graph mit 4000 Knoten und 20 Kanten je Knoten). Gemessen wurde die Zeitdauer eines Planungszyklus ohne die Kosten für den Kollisionstest, diese würden für alle Diagramme zu einer konstanten Verschiebung um ca 10 ms führen. Links oben: Optimierungsmethode Intervalllimitierung, durchschnittliche Laufzeit: 38,8 ms, zum Vergleich ist das Laufzeithistogramm der unoptimierten Distanzpropagierung dargestellt (mit 120,9 ms durchschnittlicher Laufzeit), dies gilt auch für die folgenden beiden Diagramme, Rechts oben: Optimierungsmethode Suchmengenlimitierung, durchschnittliche Laufzeit 30,7 ms, Links unten: Kombination der Optimierungsmethoden Suchmengenlimitierung und Intervalllimitierung, durchschnittliche Laufzeit 28,8 ms, Rechts unten: Direkter Vergleich der Optimierungsmethoden (Legende: nS_D = Intervalllimitierung, S_nD = Suchmengenlimitierung, S_D = Suchmengen- und Intervalllimitierung).

Indirekte Distanzpropagierung für mehrdimensionale Distanzvektoren

Der oben beschriebene Algorithmus zur indirekten Distanzpropagierung muss (auch mit den gegebenen Optimierungen) für mehrdimensionale δ mehrfach angewendet werden. Eine vereinfachte Version der indirekten Distanzpropagierung wurde als Alternative implementiert, welche über die Bedingung der Intervalländerung definiert ist, und dabei die Änderung aller Dimensionen des Distanzvektors gleichzeitig betrachtet (Algorithmus 4.4).

```
(1)  updateDistanceEstimations(v, COLLECTEDNODES)
(2)      push(v, COLLECTEDNODES)
(3)      QUEUE = {}
(4)      push(v, QUEUE)
(5)      while(QUEUE not empty)
(6)          v_curr = pop(QUEUE)
(7)          forall v_i in neighbours(v_curr)
(8)              if(estimateDist(v_curr, v_i)) push(v_i, QUEUE)
(9)              push(v_i, COLLECTEDNODES)
```

Algorithmus 4.4:Indirekte Distanzpropagierung nach dem Warteschlangen-Prinzip basierend auf der Änderung von Distanzschätzungen.

Solange Änderungen der Intervalle der Nachbarknoten auftreten (Zeile (8)) werden hierbei Knoten in eine FIFO-Warteschlange (***QUEUE***) eingestellt. Die Funktion *estimateDist*(...) entspricht der Funktion aus Algorithmus 4.2, ausser das die Vektor-Varianten des Operators *DF*(…) und der Funktion *EST*(…) verwendet werden. Die Funktion terminiert, wenn die Warteschlange geleert ist, was durch die Monotonie der Intervallschätzung garantiert ist.

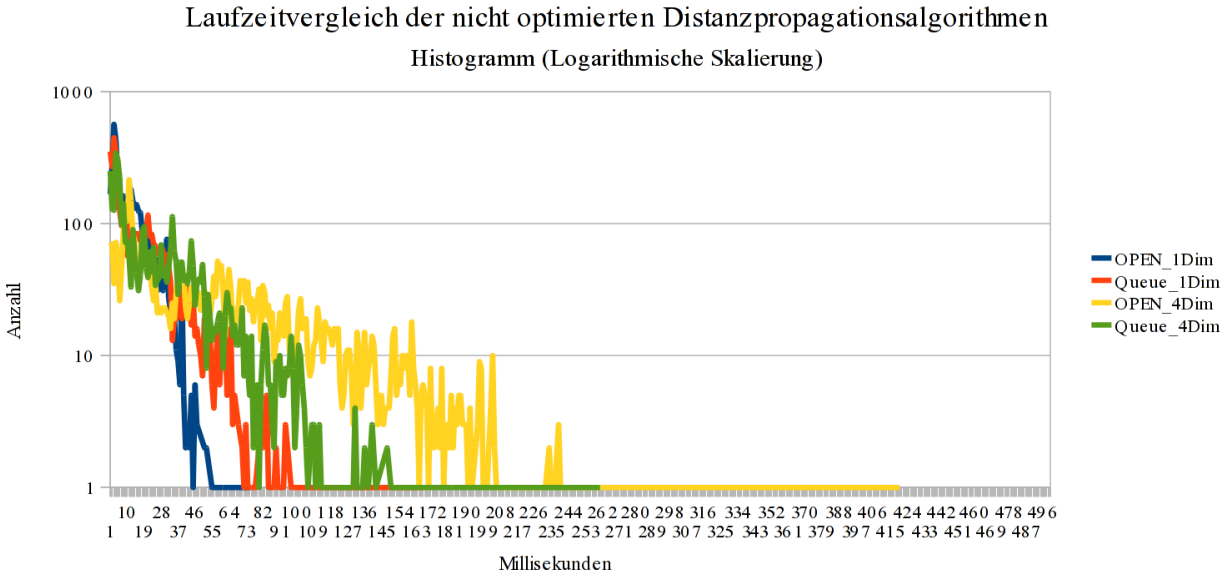


*Abbildung 4.7: Laufzeitvergleich der Nachbarschaftspropagierung mittels **OPEN**$_{du}$-Liste (Algorithmus 4.2) und mittels Warteschlange (Algorithmus 4.4). Dargestellt sind vier Histogramme der Laufzeiten einer Propagierung (ohne die oben dargestellten Optimierungen, für ein Netzwerk mit 4000 Knoten und ca 40. Kanten/pro Knoten). Basis sind jeweils etwa 4000 Messwerte. Die Skalierung der y-Achse des Histogramms ist logarithmisch, um die Unterschiede klarer darzustellen. In der Legende ist die Propagierung mittels **OPEN**$_{du}$-Liste als „OPEN_xDim" vermerkt, mit x = 1 oder 4 für ein- und vierdimensionale Distanzvektoren in den Knoten des Graphen. Für die Propagation mittels Warteschlange gilt die Kodierung mit Queue_xDim analog.*

Die Warteschlange ist unsortiert, da kein Sortierkriterium existiert. Dadurch kann ein

Knoten mehrfach aktualisiert werden, da nicht notwendigerweise der (unbekannt) optimale Nachbarknoten zuerst expandiert wird. Daher kann ein sehr hoher Berechnungsaufwand entstehen, bzw. dieser ist stark von der lokalen Graphtopologie abhängig.

In Abbildung 4.7 ist ein Experiment zum Vergleich der Propagierung mittels $\boldsymbol{OPEN}_{du}$-Liste (Algorithmus 4.2) und mittels Warteschlange (Algorithmus 4.4) dargestellt. Bei der Warteschlangen-Propagierung treten im Fall eindimensionaler Distanzvektoren häufiger hohe Laufzeiten auf, während sich dieses Bild im vierdimensionaler Distanzvektoren umkehrt. Die Experimente wurden ohne die oben dargestellten Optimierungen auf einem großen Netz mit einer hohen Anzahl von Kanten pro Knoten durchgeführt, um das unterschiedliche Verhalten der Algorithmen klar darzustellen und sind daher nicht für die Verwendung in realen Umgebungen gedacht.

4.5.4 Direkte Distanzpropagierung

Die im vorigen Kapitel dargestellte Methode zur Distanzschätzung in den Knoten beruht auf der Nachbarschaft eines Knotens, die durch die Verbindungsstruktur des Graphen definiert ist, und wird als indirekte Distanzpropagierung bezeichnet. Die im Folgenden vorgestellte Schätzmethode verwendet hingegen alle berechneten Knoten in der Umgebung des zu schätzenden Knotens unabhängig davon, ob ein Pfad im Graphen existiert und wird als *direkte Distanzpropagierung* bezeichnet. Bei dieser Methode wird aus den vorhandenen berechneten Distanzinformationen in einer Untermenge aller Knoten für den jeweiligen zu schätzenden Knoten v_s das minimale Schätzintervall erreicht. Dazu wird von allen berechneten Knoten v_i eine Schätzung über eine gedachte, direkte Verbindung von v_i nach v_s berechnet. Das daraus resultierende Schätzintervall ist minimal im Vergleich zu der Schätzung über jeden anderen Weg von v_i nach v_s im Graphen, da für die Berechnung von einzelnen MAXMOVE^{-1}-Distanzen die Dreiecksungleichung gilt. Das minimale Intervall für einen zu schätzenden Knoten lässt sich daher bestimmen über die Menge aller bekannten Distanzinformationen, mithin die Menge aller berechneten Knoten $\boldsymbol{M}_{calc}$ (hierzu gehören alle Knoten mit Status CALC):

$$\begin{pmatrix} l(v_s) \\ u(v_s) \end{pmatrix} = \begin{pmatrix} \max\limits_{v_i \in \boldsymbol{M}_{calc}} \left(l(v_i) - \delta(v_i, v_s) \right) \\ \min\limits_{v_i \in \boldsymbol{M}_{calc}} \left(u(v_i) + \delta(v_i, v_s) \right) \end{pmatrix} \tag{4.25}$$

Die Formel 4.25 ist lediglich für eindimensionale Distanzinformationen dargestellt, lässt sich jedoch leicht auf mehrdimensionale erweitern. Jede neue Distanzberechnung kann aufgrund dieses Zusammenhangs das Intervall jedes geschätzten Knotens beeinflussen. Eine Distanzberechnung hat deshalb zur Folge, dass alle geschätzten Knoten angepasst werden müssen. Ein trivialer Algorithmus hätte dabei nach Gleichung 4.25 einen Aufwand von O(NM) für einen Graphen mit N Knoten und $|\boldsymbol{M}_{calc}| = M$. Hierbei werden jedoch eine Menge

von überflüssigen Berechnungen ausgeführt, da sich lediglich ein Knoten geändert hat und nicht alle Knoten aus $\boldsymbol{M}_{calc}$ betrachtet werden müssen. Dies lässt sich auf O(N) reduzieren, wenn die Berechnung umgedreht wird und ausgehend von einem neu berechneten Knoten v_c alle geschätzten Knoten mittels der direkten Schätzung und Intervallfusion (Gleichung 4.18) aktualisiert werden, wie dies in Algorithmus 4.5 dargestellt ist.

```
(1) updateDistanceEstimations(v, COLLECTEDNODES)
(2)     forall estimated nodes v_i of the graph G not in M_calc
(3)         if(estimateDist(v, v_i))
(4)             insert v_i and all neighbours of v_i uniquely in COLLECTEDNODES
```

Algorithmus 4.5: Direkte Distanzpropagierung

Nach Aktualisierung aller Intervalle erfüllt die direkte Distanzpropagierung die in Kapitel 4.5.3 eingeführte Nachbarkonsistenz. Dies lässt sich über einen Widerspruchsbeweis herleiten. Im Folgenden wird o.B.d.A. nur die obere Grenze $u(v)$ der Schätzintervalle betrachtet, um Fallunterscheidungen einzusparen.

Sei Knoten v nicht nachbarkonsistent, d.h. es gibt mindestens einen Nachbarknoten v_n, so dass gilt:

$$u(v_n)+\delta(v_n,v)<u(v) \tag{4.26}$$

Seien sowohl der Knoten v_n als auch v geschätzt (andere Fälle können direkt über die Definition von Gleichung 4.25 behandelt werden und über die Minimalität der Schätzung zu einem Widerspruch geführt werden). Sei für die minimale Schätzung des Knotens v_n der Knoten v_0 verantwortlich, gelte also:

$$\left(u(v_n)\right)=\left(\min_i\left(u(v_i)+\delta(v_i,v_n)\right)\right)=u(v_0)+\delta(v_{0,}v_n) \tag{4.27}$$

Über die Dreiecksungleichung gilt weiterhin:

$$u(v_0)+\delta(v_{0,}v_n)+\delta(v_n,v)\geq u(v_0)+\delta(v_{0,}v) \tag{4.28}$$

Damit gilt mit 4.26:

$$u(v)>u(v_0)+\delta(v_{0,}v_n)+\delta(v_n,v)\geq u(v_0)+\delta(v_{0,}v) \tag{4.29}$$

was der Minimalität von $u(v)$ nach Gleichung 4.25 widerspricht.

Die Vorteile der direkten Distanzpropagierung liegen in der Optimalität der Schätzung für eine gegebene Menge an Distanzinformationen und der Tatsache, dass die Schätzung unabhängig von der durch die bekannten Objekte $\boldsymbol{Z}_k$ erzeugten Geometrie/Topologie des Graphen berechnet wird. Dieser Zusammenhang ist in Abbildung 4.8 dargestellt. Die Knoten v_0 und v_1 sind vom Knoten v_c aus im Rahmen der indirekten Distanzpropagierung

nur über weite Wege erreichbar und werden damit nicht optimal geschätzt, da über die Länge des Weges die Breite des Schätzintervalls determiniert wird. Durch die direkte Distanzpropagierung jedoch werden die Knoten v_0 und v_1 über gedachte Verbindungen von v_c aus geschätzt, ohne zu berücksichtigen, dass diese direkten Verbindungen wegen des Objekts ***Z*** kollisionsbehaftet wären.

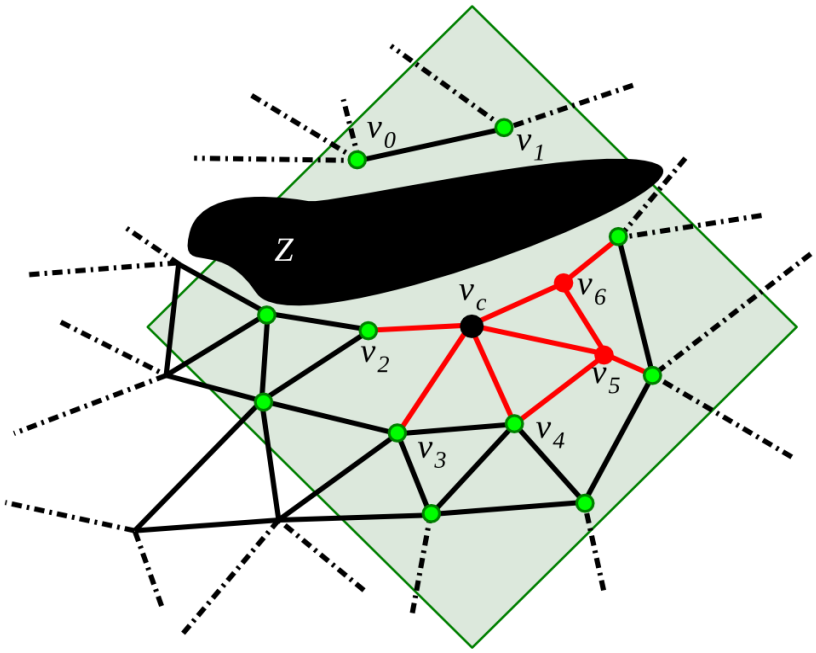


*Abbildung 4.8: Schematische Darstellung der Unterschiede in der Knotenschätzung mittels direkter Distanzpropagierung und indirekter Distanzpropagierung. Für den Knoten v_c wird eine Distanz zu einem unbekannten Objekt berechnet (**nicht** zum ebenfalls dargestellten bekannten Objekt **Z**), die nun im Graphen propagiert werden muss. Bei einer Limitierung des Schätzintervalls werden abhängig von der konkreten Distanz in v_c alle Knoten im dadurch bestimmten Rhomboid (im Bild transparent grün) aktualisiert, ohne Limitierung alle Knoten des gesamten Graphen. Die ebenfalls dargestellte indirekte Distanzpropagierung umfasst die rot gefärbten Teile des Graphen und aktualisiert die Knoten v_5 und v_6, wobei eine Schätzung für die Knoten v_2 bis v_6 berechnet wird.*

Die Nachteile der direkten Distanzpropagierung ergeben sich gerade aus der fehlenden Information über die Distanzen in der Nachbarschaft der Knoten. Für die Propagierung müssen immer alle Knoten des Graphen geschätzt werden (oder alle Knoten innerhalb eines bestimmten Rhomboids, wenn die Optimierung Intervalllimitierung verwendet wird), da deren aktuelles Schätzintervall potentiell angepasst werden muss. Weiterhin kann das Schätzintervall eines Knotens prinzipiell auch dann durch direkte Propagierung reduziert werden, wenn die Intervalle aller Nachbarknoten nicht reduziert wurden, somit ist aus der Nachbarschaft des Knotens keine Bedingungen für einen Abbruch der Propagierung herleitbar. Generell gilt zwar, dass für jedes Paar Knoten mit berechnenten Distanzen eine komplizierte Grenzfläche berechnet werden könnte, die den Aktualisierungsbereich des einen Knotens vom anderen trennt. Da diese Flächen jedoch von den konkreten Distanzen in den Knoten abhängen und zudem für jedes Paar von Knoten berechnet werden müsste ist dies zur Laufzeit ein rechnerisch sehr aufwendiges Verfahren und daher nicht in Echtzeit

anwendbar.

Zwar sind die Optimierungen der Intervalllimitierung und Suchmengenlimitierung hier wie für die indirekte Distanzpropagierung anwendbar, dies verändert jedoch nicht den oben beschriebenen Nachteil. Eine zusätzliche, auf Sortierung nach der MAXMOVE^{-1}-Distanz basierende Optimierung wird im folgenden ebenso dargestellt (neben Intervalllimitierung und Suchmengenlimitierung), kompensiert jedoch für hohe Knotendichten die Laufzeitnachteile ebenfalls nicht ausreichend.

Optimierungen

Die Optimierung durch **Intervalllimitierung** erzeugt für einen berechneten Knoten v_c abhängig von dessen Distanz eine Submenge von Knoten, die im relevanten Bereich liegen, d.h. für die das geschätzte Intervall mit mindestens einer Grenze innerhalb der vorgegebenen Intervallgrenzen liegt. Dies erzeugt einen Rhomboid im Konfigurationsraum (siehe Abbildung 4.8) und bietet die Möglichkeit zur Optimierung durch Vorberechnung und Sortierung der Knoten innerhalb des Rhomboids. Diese ist jedoch wie bei der indirekten Distanzpropagierung für jede Dimension der MAXMOVE^{-1}-Distanzen einzeln zu betrachten. Insbesondere bei dieser Optimierung ist es wichtig, die Berechnung der Aktualisierung im Gegensatz zu Gleichung 4.25 umzukehren (siehe Algorithmus 4.5), da nicht mehr alle Knoten des Graphen aktualisiert werden müssen, die außerhalb des Rhomboids liegen.

Die durch diese Optimierung definierte Submenge an Knoten im Rhomboid kann zur Laufzeit nicht sinnvoll über alle Knoten berechnet werden, da auf diese Weise natürlich der Vorteil dieser Optimierung gerade wieder entfällt. Daher muss pro Knoten die maximal relevante Submenge an Knoten des Graphen konservativ berechnet und abgespeichert werden. Dies erzeugt einen Speicheraufwand von $O(N^2)$ bei N Knoten im Graph, was jedoch in der Praxis durch die relativ kleine assoziierte Konstante vertretbar ist. Die Submenge an Knoten bestimmt sich konservativ aus allen Knoten, deren MAXMOVE^{-1}-Distanz zum betrachteten Knoten v_c unterhalb der Differenz der Intervallgrenzen ($|d_{max} - d_{min}|$) liegt, da eine berechnete Distanz in v_c innerhalb des Intervalls von d_{min} bis d_{max} liegen muss, und daher insbesondere für alle Knoten v_i der Submenge für jede Distanz $d(v_c)$ gelten muss:

$$\left(d(v_c)+\delta\left(v_c,v_i\right)\leq d(V_{max})\right)\vee\left(d(v_c)-\delta\left(v_c,v_i\right)\geq d(V_{min})\right) \tag{4.30}$$

Die resultierende Menge von Knoten kann nach den δ-Distanzen sortiert werden, sodass sich zur Laufzeit eine Optimierungsmöglichkeit durch frühen Abbruch ergibt, wie in Algorithmus 4.6 in Zeile 3 dargestellt:

(1) **updateDistanceEstimations**(v_c, ***COLLECTEDNODES***)
(2) **forall** nodes v_i of the *updateNodes* of node v_c
(3) **if**(($d(v_c) + \delta(v_c, v_i) > d_{max}$) **and** ($d(v_c) - \delta(v_c, v_i) < d_{min}$)) **return**
(4) **if**(*estimateDist*(v_c, v_i))
(5) *insert* v_i and all neighbours of v_i uniquely in ***COLLECTEDNODES***

Algorithmus 4.6: Direkte Distanzpropagierung mit Intervalllimitierung über die dadurch konstruierte Submenge (updateNodes) aus dem durch die Limitierung definierten Rhomboid

Die Anzahl der besuchten Knoten in der Submenge *updateNodes* hängt bei diesem Algorithmus ausser von der durchschnittlichen Knotendichte stark von der Größe des notwendigen δ ab (welches die Abbruchbedingung in Zeile 3 definiert) und von der Dimensionalität d des Konfigurationsraums: $O((\delta)^d)$. Dies ist jedoch im mittleren Fall deutlich günstiger im schlechtesten, da pro Dimension durchschnittlich nur bis zur Hälfte der maximalen Distanz iteriert werden muss, sich also eine Ersparnis von $(1/2)^d$ ergibt. Von Schwankungen der Knotendichte wird hier abstrahiert.

Die experimentellen Ergebnisse (Abbildung 4.9) zeigen, dass die Optimierung durch Intervalllimitierung für die direkte Distanzpropagierung sehr wichtig ist, da ohne Limitierung die auftretenden Laufzeiten deutlich zu lang sind, um für die in der Praxis eingesetzten Graphgrößen echtzeitfähig planen zu können.

(1) **adjustExpandedNode**(v, ***COLLECTEDNODES***)
(2) **if**(*state*(v) is CALC)
(3) **if**(**not** *neighbourConsistent*(v)) **raise** estimation-error
(4) **else**
(5) **forall** nodes v_i in (*updateNodes* of node n **and** $\boldsymbol{M}_{visited}$)
(6) **if**(*state*(v_i) is ESTIM) **continue**
(7) **if**(($u(v) - \delta(v, v_i) < d_{min}$) and ($l(v) + \delta(v, v_i) > d_{max}$)) **breakloop**
(8) *estimateDist*(v_i, v)
(9) push all neighbours of v to ***COLLECTEDNODES***
(10) *updateDistanceEstimations*(v, ***COLLECTEDNODES***)

Algorithmus 4.7: Suchmengenlimitierung für die direkte Distanzpropagierung

Für die Optimierung durch **Suchmengenlimitierung** (Algorithmus 4.7) gelten auch hier die in Kapitel 4.5.3 dargestellten Vor- und Nachteile. Zusätzlich wirkt sich jedoch insbesondere aus, dass für jede Schätzung eines in die Suchmenge aufgenommenen Knotens eine im Allgemeinen deutlich größere Anzahl an Knoten betrachtet werden muss, als dies für die indirekte Distanzpropagierung der Fall ist und daher für die gesamte Planung von

Start bis Ziel ein insgesamt hoher Aufwand entsteht, was durch die Experimente bestätigt wird (s.u.).

Die in Algorithmus 4.7 dargestellte Funktion *adjustExpandedNode*(*v*, ***COLLECTEDNODES***) entspricht in den Zeilen 1-4 und 10 der Variante der indirekten Distanzpropagierung (Algorithmus 4.3). Auch hier arbeiten alle Funktionen auf einer gefilterten Variante des Graphen, die nur die Knoten der Suchmenge ($\boldsymbol{M}_{visited}$) enthält. Die Zeilen 5 bis 8 aktualisieren den gegebenen Knoten durch die Knoten der Menge *updateNodes* (geschnitten mit $\boldsymbol{M}_{visited}$), die nicht selbst geschätzt sind (Zeile 6). Diese Iteration kann abgebrochen werden, sobald von den sortierten Knoten aus *updateNodes* keine Änderung des aktuellen Distanzintervalls mehr zu erwarten ist (Zeile 7). Die Knotenmenge *updateNodes* umfasst alle Knoten des Graphen, wenn die Intervalllimitierung nicht gesetzt ist. Im Anschluss an die Änderung des Knotens werden alle Nachbarn für die Kostenaktualisierung gespeichert (Zeile 9).

Experimentelle Ergebnisse

Für den experimentellen Vergleich wurde die Begrenzung der Laufzeit des Planers aufgehoben um die keine Verzerrung der Ergebnisse zu erzeugen und einen Vergleich der Methoden zu ermöglichen. Die Begrenzung erfolgt in einer realen Umgebung entweder über eine feste Anzahl an Kollisionstests erfolgen oder über eine variable Anzahl, welche sich an der verbrauchten Laufzeit ausrichtet (siehe Kapitel 4.4, Abschnitt „Kollisionstestkosten und Echtzeitfähigkeit durch Schätzung“).

Die Ergebnisse der 2D-Simulationen (siehe Abbildung 4.9) zeigen unter anderem, dass die Laufzeit des Planers für eine hohe Knotendichte lediglich für die Intervalllimitierung und die unoptimierte Variante mit der indirekten Distanzpropagierung vergleichbare Ergebnisse erreicht. Die Suchmengenlimitierung hingegen erzeugt durch die ständig notwendige Aktualisierung der geschätzten Distanzen aus der Suchmenge bei Expansion einen hohen Rechenaufwand, womit sich diese Art der „Optimierung“ bei der direkten Distanzpropagierung verbietet. Mit steigender Knotenanzahl wird direkte Propagierung relative Nachteile erfahren, da dann die Knotendichte steigt. Für eine steigende Kantenanzahl pro Knoten ergibt sich der umgekehrte Effekt, da hier bei der Nachbarschaftspropagierung eine Erhöhung des Aufwands entsteht, bei der direkten Propagierung jedoch nicht. Daher muss die Wahl der Distanzpropagierung von den angesprochenen Graphparametern abhängig gemacht werden.

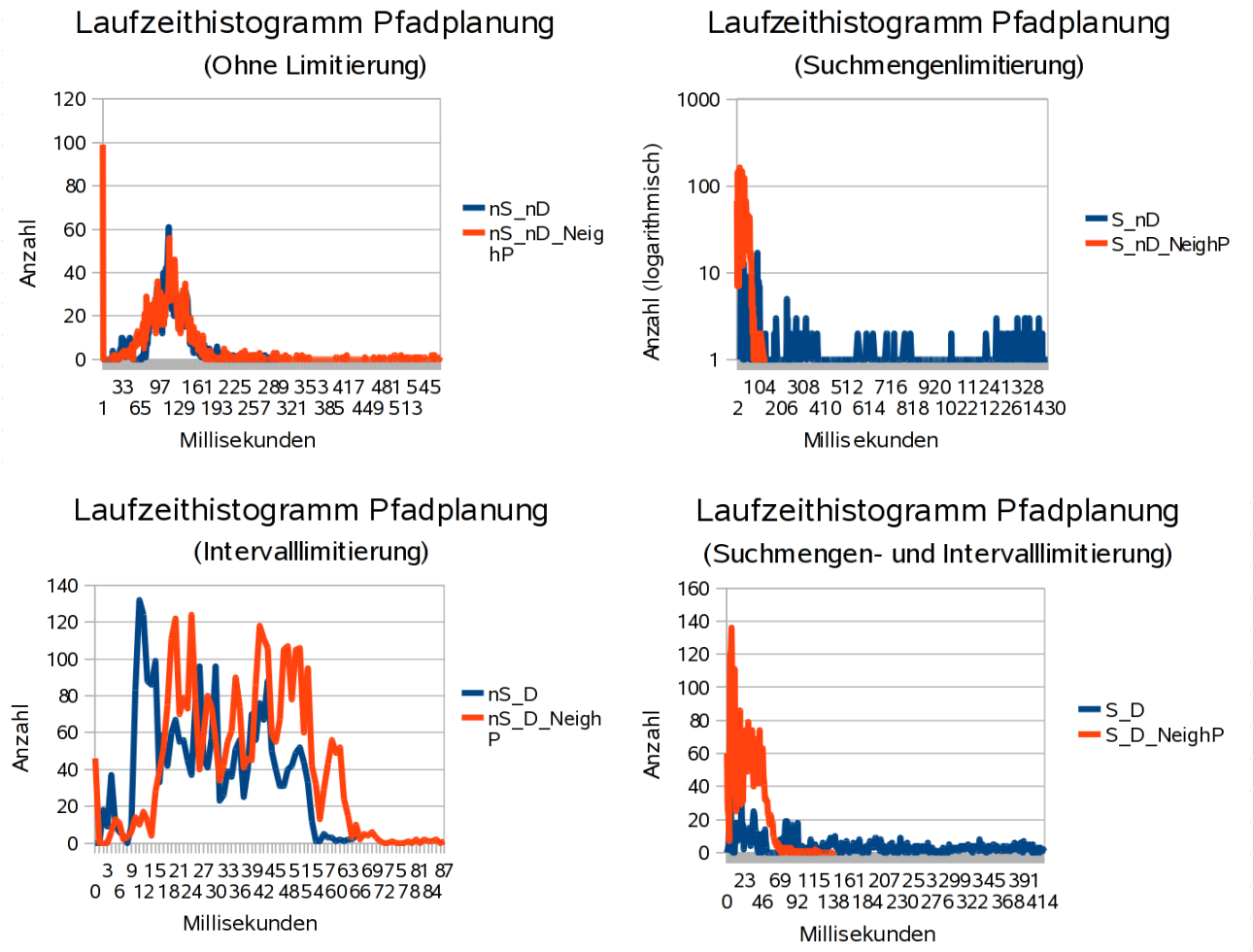


Abbildung 4.9: Laufzeithistogramme für die Planung mit direkter Distanzpropagierung (im Vergleich mit der indirekten Distanzpropagierung). Es wurden jeweils 20 Durchläufe eines Experimentszenarios mit bewegtem Objekt berechnet (Graph mit 4000 Knoten und 20 Kanten je Knoten). Gemessen wurde die Zeitdauer eines Planungszyklus (ohne die Kosten für den Kollisionstest, diese würden für alle Diagramme zu einer konstanten Verschiebung um ca 10 ms führen). Die Legende aller Diagramme kodiert die Einstellungen des Experiments mit z.B. nS_nD = keine Suchmengenlimitierung und keine Intervalllimitierung, S_nD = Suchmengenlimitierung und keine Intervalllimitierung, usw., das angehängte _NeighP referenziert die Nachbarschaftspropagierung. ***Links oben****: Ohne Optimierung, durchschnittliche Laufzeit: 116,4ms (zum Vergleich indirekte Distanzpropagierung mit 120,9 ms durchschnittlicher Laufzeit),* ***Rechts oben****: Optimierungsmethode Suchmengenlimitierung, durchschnittliche Laufzeit 458,2 ms (zu 30,7 ms),* ***Links unten****: Intervalllimitierung, durchschnittliche Laufzeit 29,9 ms (zu 38,8 ms),* ***Rechts unten****: Kombination der Methoden, durchschnittliche Laufzeit 151,1 ms (zu 28,8 ms).*

4.6 Kantenschätzung

Die Berechnung des Distanzverlaufs entlang eines Geradenstücks im Konfigurationsraum würde je nach Länge einen erheblichen Berechnungsaufwand durch die hohe Anzahl an Messungen an diskreten Punkten auf der Kante erfordern. Zudem müsste selbst dann der Distanzverlauf zwischen den diskreten Punkten geschätzt werden. Daher werden hier Distanzen nur in den Knoten des Graphen berechnet beziehungsweise geschätzt und der Verlauf der Distanzen entlang einer Kante wird immer abgeschätzt. Basis für die Schätzungen bildet eine Funktion, die aus dem Distanzintervall eines Knotens einen Wert auswählt, hier beispielsweise beschränkt auf Minimum, Maximum und Durchschnitt, also beispielsweise $select(v) := select_{min}(v) := l(v)$. Diese Funktion muss während der Planung gleich bleiben, denn sonst müssten alle Kantenkosten neu berechnet werden. Auf diesen gewählten Distanzwerten aufbauend werden in Abschnitt 4.6.1 einige Abschätzungsvarianten dargestellt und in Abschnitt 4.6.2 die daraus resultierende effiziente Berechnung der Kantenkosten. Die Auswirkungen der Abschätzungsvarianten werden in Kapitel 4.9 dargestellt und experimentelle Ergebnisse in Kapitel 5.

4.6.1 Distanzverlaufsspektrum

In diesem Kapitel werden die realen Distanzverläufe vereinfacht und durch Modellbildung analytisch handhabbar gemacht. Gegeben sei ein Geradenstück $\boldsymbol{q}(\lambda) = (1-\lambda)\,\boldsymbol{q}_a + \lambda \boldsymbol{q}_b$ im Konfigurationsraum von $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ mit der Laufvariable $\lambda \in [0,1]$ zwei Distanzen $d_a = select(v(\boldsymbol{q}_a))$ und $d_b = select(v(\boldsymbol{q}_b))$ in den Punkten $\boldsymbol{q}_a$ und $\boldsymbol{q}_b$. Gesucht ist der Distanzverlauf $d(\boldsymbol{q}(\lambda))$ entlang des Geradenstücks.

Im hier vorgestellten Modell wird zunächst das Geradenstück $\boldsymbol{q}(\lambda)$ in L diskrete Schritte zerlegt mit einem festen $\Delta\lambda = \lambda/L$ (Abbildung 4.10). Daraus ergibt sich über das konservativ abgeschätzte Modell des Roboters eine maximale Veränderung der Distanz pro Schritt um $\Delta d = \pm MAXMOVE^{-1}(\boldsymbol{q}(\Delta\lambda)) = \pm\delta(\boldsymbol{q}(\Delta\lambda))$ mit positivem und negativem Vorzeichen in jedem diskreten Schritt. Δd und $\Delta\lambda$ erzeugen ein Gitter, mit dem durch entsprechend hohe Auflösung die beiden Punkte $(0,\ d_a)$ und $(1,\ d_b)$ beliebig genau angenähert werden können.

Bedingt durch das diskrete Modell ergeben sich nun beginnend in Punkt $(0,\ d_a)$ eine endliche Anzahl von möglichen Kurvenverläufen. Eine weitere Einschränkung für das Modell und damit die möglichen Distanzverläufe wird erreicht, indem für jede Schrittweite $\Delta\lambda$ die Distanz nur um genau $\pm\Delta d$ geändert werden kann, also um die Schritte $(\Delta\lambda, \pm\Delta d)$ auf dem diskreten Gitter. Durch Verkleinerung der Gitterkonstanten Δd und $\Delta\lambda$ kann ein kontinuierlicher Distanzverlauf beliebig angenähert werden. Ein Kurvenverlauf lässt sich dadurch in einer abgekürzten Notation als eine Sequenz aus L +1- und −1-Schritten repräsentieren. Für diese Kurvenverläufe lassen sich zwei Invarianten feststellen:

- Die Gesamtanzahl L der Schritte ist konstant. Konkret ist die Summe aus +1- und −1-Schritten äquivalent zu L: $(\#{+}1) + (\#{-}1){=}L$
- Da der Endpunkt $(1, d_b)$ erreicht werden muss, muss die Differenz der Schritte folgende Bedingung erfüllen: $(\#{+}1) - (\#{-}1){=}(d_a - d_b / \Delta d)$

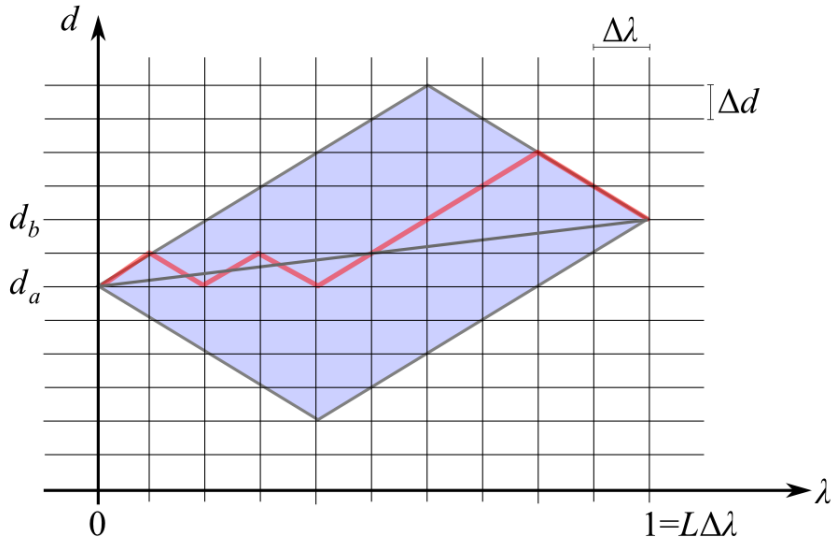


Abbildung 4.10: Diskretisiertes Distanzverlaufsspektrum entlang einer linearen Verbindung im Konfigurationsraum von Konfiguration $\boldsymbol{q}_a$ nach $\boldsymbol{q}_b$ mit bekannten Distanzen d_a, d_b. Jeder diskrete Schritt $\Delta\lambda$ entlang der λ-Achse ist assoziiert mit zwei möglichen Veränderungen der Distanz um $\pm\Delta d$ (+1- und −1-Schritt). Die endliche Menge aller Spektren ergibt sich in diesem Modell durch die Permutation einer konstanten Zahl von +1- und −1-Schritten.

Aus diesen beiden Invarianten folgt, dass für alle Varianten von Distanzverläufen die Anzahl der +1- und −1-Schritte jeweils konstant sind und jede Sequenz lediglich eine Permutation jeder anderen möglichen Sequenz darstellt. Die Grenzen dieses Distanzverlaufsspektrums (siehe Abbildung 4.10) resultieren damit in zwei Kurvenverläufen, die den Best-Case und den schlimmsten Fall der Distanzschätzung darstellen, im Folgenden als *optimistische* und *pessimistische Kantenschätzung* bezeichnet. Eine weitere Abschätzung entsteht natürlich aus diesem Modell und wird im folgenden Abschnitt ausführlicher erläutert und hergeleitet. Sie wird als *durchschnittliche* oder *Kantenschätzung* bezeichnet.

Durchschnittliche Kantenschätzung

Für die durchschnittliche Kantenschätzung ist es notwendig, alle möglichen Kurvenverläufe innerhalb des Distanzspektrums zu betrachten. Jeder dieser Verläufe ist ohne weiteres als gleich wahrscheinlich zu betrachten. Ein Verlauf wird durch Sequenz von L +1- und −1-Schritten repräsentiert.

$$S=\{s_0 \ldots s_L\}, \quad s_i \in \{-1, 1\} \tag{4.31}$$

Werden n als Anzahl der +1-Schritte und m als Anzahl der −1-Schritte so gewählt,

dass die beiden Invarianten erfüllt sind, ergibt sich die Gesamtanzahl aller möglichen Verläufe V_{all} zu:

$$V_{all} = \binom{L}{n} = \frac{L!}{n!(L-n)!} = \frac{L!}{n!m!} \tag{4.32}$$

Für die Berechnung des statistisch optimalen Verlaufs betrachten wir nun den Erwartungswert $E_{dist}(i)$ der Distanzänderung pro Schritt von (i, d_i) nach $(i + 1, d_{i+1})$:

$$E_{dist}(i) = p_i^+ \cdot (\Delta d) \; + \; p_i^- \cdot (-\Delta d) \tag{4.33}$$

mit p_i^+ als Wahrscheinlichkeit für einen +1-Schritt an der i-ten Stelle und p_i^- entsprechend. Die Wahrscheinlichkeit p_i^+ entspricht dem Verhältnis der Anzahl aller Verläufe $V_1(i)$, die an der Stelle i einen +1-Schritt beinhalten zu der Anzahl aller möglichen Verläufe V_{all}. $V_1(i)$ ergibt sich für $L >= n >= 1$ zu:

$$V_1(i) = \frac{(L-1)!}{(n-1)! \cdot m!}, \tag{4.34}$$

da über die verbleibenden +1-Schritte über die verbleibenden $L - 1$ Stellen verteilt werden, wiederum ohne Zurücklegen und ohne Beachtung der Reihenfolge. Man sieht, dass diese Anzahl schon unabhängig von der Stelle i ist und mit

$$p_i^+ = \frac{V_1(i)}{V_{all}} = \frac{\frac{(L-1)!}{(n-1)! \cdot m!}}{\frac{(L)!}{(n)! \cdot m!}} = \frac{(L-1)! \cdot (n)!}{(L)! \cdot (n-1)!} = \frac{n}{L} \tag{4.35}$$

ist auch die Wahrscheinlichkeit p_i^+ von i unabhängig und daher für alle i gleich. Damit ergibt sich komplementär $p_i^- = m/L$. Der Erwartungswert $E_{dist}(i)$ ergibt sich damit für alle i zu

$$E_{dist} = \left(\frac{n}{L}\right) \cdot (\Delta d) \; + \; \left(\frac{m}{L}\right) \cdot (-\Delta d) = \frac{(n-m)}{L} \cdot \Delta d \tag{4.36}$$

Ausgehend von einem beliebigem Punkt innerhalb des Distanzspektrums gibt der Vektor $(1, ((n-m)/L)\Delta d)$ den Differenzvektor zum durchschnittlich erwarteten nächsten Punkt im Kurvenverlauf an. Wenn wir das Koordinatensystem zur einfacheren Darstellung um $(0, -d_a)$ verschieben und vom Ursprung ausgehend L Differenzvektoren aufsummieren, so ergibt sich:

$$(0,0) + L \cdot \left(\Delta\lambda, \frac{(n-m)}{L} \cdot \Delta d\right) = (1, (n-m) \cdot \Delta d) = (1, d_b - d_a) \tag{4.37}$$

was dem Endpunkt aller Distanzverläufe im verschobenen Koordinatensystem entspricht. Die Aneinanderreihung der erwarteten Differenzvektoren ergibt somit einen (nicht im diskreten Koordinatensystems darstellbaren) Distanzverlauf, der einer linearen

Verbindung der beiden Endpunkte entspricht. Der *durchschnittliche* Distanzverlauf ist daher die Gerade von (0, d_a) nach (1, d_b).

4.6.2 Berechnung der Kantenkosten

Das oben dargestellte Modell für die Distanzverläufe entlang einer Kante ist linear für den optimistischen, pessimistischen und durchschnittlichen Schätzer und ermöglicht damit eine einfache Berechnung der Kantenkosten unter der Voraussetzung, dass die Distanz-Geschwindigkeitsfunktion *speed*(d) ebenfalls stückweise linear ist. Für invalidierte Kanten sind die Kantenkosten implizit unendlich, für die anderen Kanten werden die Kosten wie im Folgenden dargestellt berechnet.

Das Gittermodell für die Kantendistanzen wird hierzu ins Kontinuierliche entwickelt, indem $\Delta\lambda$ gegen 0 wandert. Die Steigung der Geraden, also $\Delta d/\Delta\lambda$, strebt dann gegen einen Grenzwert für jede Dimension j eines gegebenen Distanzvektors $\boldsymbol{d}$:

$$\lim_{\Delta\lambda\to 0}\frac{\Delta d}{\Delta\lambda}(j)=\lim_{\Delta\lambda\to 0}\frac{\delta_j(\boldsymbol{q}(\Delta\lambda))}{\Delta\lambda}=\lim_{\Delta\lambda\to 0}\frac{\sum_{i\leq j} r_{ij}\cdot 2\cdot\sin\left(q_i(\Delta\lambda)\,c\,0.5\right)}{\Delta\lambda}\approx\sum_{i\leq j} r_{ij}\cdot q_i \qquad (4.38)$$

Der letzte Schritt in 4.38 verwendet eine Linearisierung $\sin(|\varphi|)\leq|\varphi|$ des Sinus für kleine Winkel und δ_j ist definiert wie in Kapitel 4.5.1. Wie in Abbildung 4.11 dargestellt, können auch für mehrdimensionale Distanzvektoren Schätzungen berechnet werden, hier beispielhaft für zweidimensionale Vektoren. Bei mehrdimensionalen, geschätzten Distanzverläufen entlang der Kante muss in jedem Punkt der Kante die minimale Distanz zur Geschwindigkeitsregelung verwendet werden, daher ergibt sich als resultierender Distanzverlauf jeweils die rote Linie als untere Begrenzung. Diese Untergrenze lässt sich mit sehr geringem Aufwand aus den linearen Stücken der Distanzverläufe bestimmen. Sie ist dann stückweise linear und kann für jedes Stück über einen Proportionalitätsfaktor in die Verfahrdauer für dieses Stück übersetzt werden, womit über Summierung dann die gesamte Verfahrdauer für die Kante resultiert, die den Kosten für die Kante entspricht.

Die berechneten, stückweise linearen Geschwindigkeiten werden noch durch die Geschwindigkeitsgrenzen $V_{min} + \varepsilon$ und V_{max} beschränkt mit einer kleinen positiven Konstante ε. Dies dient dazu, eine Blockade des Ziels aufgrund unendlicher geschätzter Kantenkosten zu vermeiden. Details hierzu finden sich in Kapitel 4.9.

Trotz dieser geringen Kosten kann die multidimensionale Variante der Distanzverläufe durch den häufigen Aufruf dieser Funktion während der Graphensuche einen erheblichen Aufwand erzeugen, der mit den Zugewinnen von etwa 15 bis 20% in den berechneten Distanzen in der Praxis abgewogen werden muss (siehe hierzu Kapitel 2.6). Die Berechnung der Kantenkosten hat in der eindimensionalen Variante einen Anteil von 20-50% an der Gesamtlaufzeit des Planers. Die Komplexität der Berechnung ist mindestens linear in der

Anzahl der Dimensionen. In dieser Arbeit werden aufgrund dieser Abwägung nur eindimensionale Verläufe für die Experimentreihen verwendet.

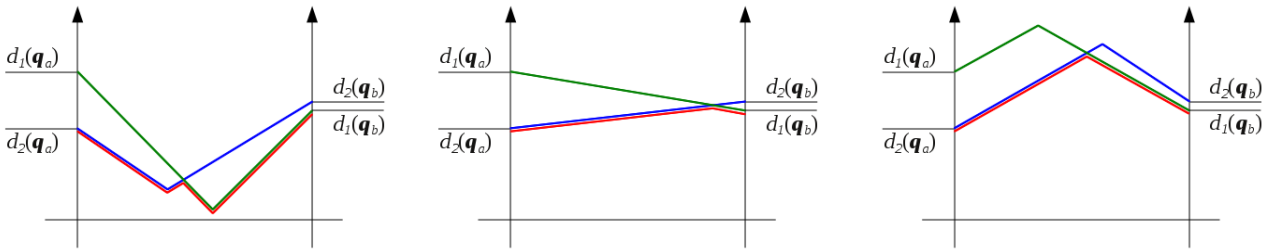


*Abbildung 4.11: Distanzverläufe für mehrdimensionale Distanzvektoren **d**. Und Kantenschätzer in den Varianten pessimistisch (links), Durchschnitt (Mitte) und optimistisch (rechts). Entscheidend für die Geschwindigkeitsregelung ist immer die kleinste Distanz, in den Schätzerverläufen ist diese rot eingezeichnet.*

Ausblick

Abschätzung durch k-Nachbarschaft (Abbildung 4.12). Die Distanzen entlang einer Kante können noch genauer geschätzt werden, wenn die Kante in *n* diskrete Schritte zerlegt wird und für jede Zwischenkonfiguration von *k* nächsten Knoten eine Distanzschätzung durchgeführt wird, über gedachte direkte Verbindungen (ähnlich der direkten Distanzpropagierung, siehe Kapitel 4.5.4). Nach Durchführung der Schätzung kann dann nach gewählter Schätzvariante für die Kante das Minimum, Maximum oder der Mittelwert des resultierenden Intervalls verwendet werden. Die Komplexität dieser Methode ist O(*nk*) und kann daher je nach Wahl der Werte einen erheblichen Aufwand erzeugen, der sehr sorgfältig mit dem erwarteten Gewinn abgewogen werden muss.

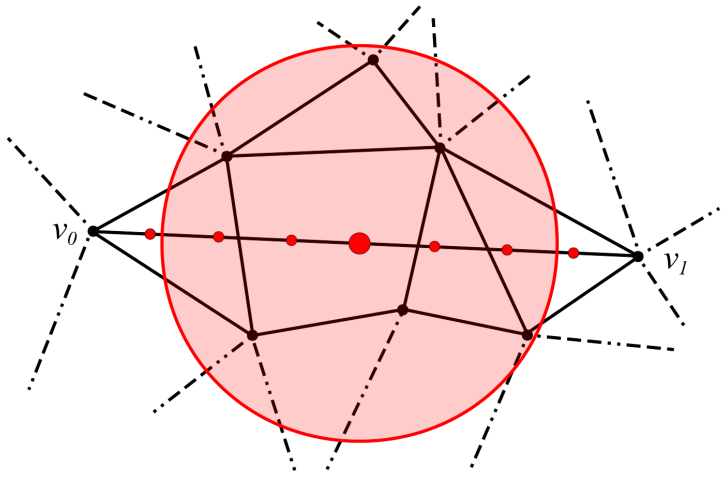


Abbildung 4.12: Schätzung der Kantendistanzen aus den k nächsten Nachbarknoten für die Kante von v_0 bis v_1. Dargestellt ist die Unterteilung dieser Kante in diskrete Schritte (rote Punkte). Der aktuell geschätzte Zwischenschritt ist etwas größer dargestellt und der Radius der erfassten Nachbarknoten in transparentem Rot angedeutet.

Verbesserung der Abschätzung durch Distanzberechnung. Für lange Kanten weitet das Spektrum der möglichen Distanzverläufe stark auf. Um die Unsicherheit entlang dieser

Kanten zu reduzieren können n Distanzberechnungen auf langen Kanten eingesetzt werden. Hierbei ist jedoch zu beachten, dass der Aufwand für Distanzberechnungen massiv höher ist als eine Schätzung der Kante und dass diese Tests dem Pfadplaner an anderer Stelle fehlen, da die Anzahl der Berechnungen begrenzt wird (siehe Kapitel 4.4 und 4.8).

4.7 Zielheuristik

Die Zielheuristik $h(v)$ schätzt die verbleibenden Kosten bis zum Zielknoten v_z am Knoten v (siehe hierzu A*-Algorithmus 3.1 auf Seite 94). Sie ist *zulässig*, wenn sie immer unter den tatsächlichen Kosten bleibt. Dann erzeugt die Graphensuche den kürzesten Pfad durch den Graphen, wenn die heuristische Gewichtung $w \leq 0{,}5$ gewählt wird. Eine triviale Heuristik nimmt für die Strecke bis zum Zielknoten daher die maximale Verfahrgeschwindigkeit V_{max} an. Diese verwendet jedoch keinerlei verfügbare Distanzinformationen und ist daher *schlecht informiert*.

Eine besser informierte Heuristik ergibt sich, wenn man die Distanzinformationen an v und v_z verwendet und mit den gewählten Distanzinformationen am aktuellen Knoten $select(v)$ und am Zielknoten $select(v_z)$ eine optimistische Kantenschätzung berechnet, welche in die Kosten $c_{optim}(v, v_z)$ resultiert (zur Funktion $select(v)$ siehe Kapitel 4.6).

Wenn die Heuristik zulässig im Sinne des A* sein soll, kann dies nur eine optimistische Kantenschätzung sein, da die Kosten dieser virtuellen Kante damit minimal sind und somit auch bei einem sehr positiven Verlauf der tatsächlichen Distanzen bis zum Ziel kleiner gleich den tatsächlichen Kosten sind. Diese Heuristik wird im Folgenden als *optimistische Heuristik* bezeichnet.

Die Distanzintervallvektoren beider Knoten $\boldsymbol{I}(v)$ und $\boldsymbol{I}(v_z)$ können im Laufe der Graphensuche durch Berechnung oder Schätzung verändert werden (Kapitel 4.5.1). Dadurch ändern sich eventuell auch die durch die optimistische Heuristik geschätzten Kosten. Daher stellt sich die Frage einer (unter Umständen kostenintensiven) Anpassung der heuristischen Kosten. Wenn die Kosten **nicht** angepasst werden, hängt die Zulässigkeit der berechneten Kosten von der gewählten $select(v)$-Funktion ab.

Wenn die Obergrenze der Distanzintervalle $u_i(v)$ gewählt wird, also $select_i(v) = select_{i,max}(v) = u_i(v)$, so muss bei Veränderungen keine Anpassung erfolgen, um die Zulässigkeit der Heuristik zu erhalten. Denn die oberen Intervallgrenzen eines Knotens können durch den Operator $DF(...)$ bedingt nur kleiner werden, wenn sie neu geschätzt werden. Sollte die obere Grenze wider Erwarten nach oben angepasst werden müssen (z.B. auch durch Berechnung), so ist dies ein Fehlerfall, der in Kapitel 4.10 besprochen wird. Wenn die Distanzen kleiner werden, steigen die heuristischen Kosten bei einer Neuberechnung. Wenn die heuristischen Kosten in diesem Fall nicht angepasst werden, bleiben sie daher zulässig, sind allerdings schlechter informiert.

Jede andere Wahl der *select*(v)-Funktion, also z.B. die Wahl der unteren Intervallgrenzen $select_i(v) = select_{i,min}(v) = l_i(v)$, kann dazu führen, dass die gewählten Distanzen an den Knoten v und v_z größer werden. Dann müsste eine Neuberechnung erfolgen, denn die heuristischen Kosten würden in diesem Fall sinken und damit wären die heuristischen Kosten nicht zulässig, wenn sie beibehalten würden.

Unabhängig von der Betrachtung der Zulässigkeit werden im Folgenden Behandlungsmöglichkeiten im Fall von Änderungen erörtert. Es gibt bezüglich der Änderungen von $\boldsymbol{I}(v)$ und $\boldsymbol{I}(v_z)$ zwei prinzipielle Möglichkeiten: ignorieren oder behandeln. Daraus ergeben sich vier Kombinationsmöglichkeiten, die im Folgenden kurz aufgeführt sind mit den jeweiligen Möglichkeiten der Implementierung. Grundsätzlich ist von den unten aufgeführten Fällen nur Variante 4 zulässig. Die anderen Lösungen produzieren suboptimale Pfade.

- **Variante 1**: Die Änderungen von $\boldsymbol{I}(v)$ **ignorieren** und $\boldsymbol{I}(v_z)$ **ignorieren**: Es werden für die Verbindung zum Ziel immer die erstmalig berechneten Kosten angenommen und unabhängig von Änderungen beibehalten. Die heuristischen Kosten werden so pro Knoten nur einmal berechnet. Die Kosten für die Berechnung gehen daher gegen Null, je mehr Knoten berechnet sind.
- **Variante 2**: Die Änderungen von $\boldsymbol{I}(v)$ **behandeln** und $\boldsymbol{I}(v_z)$ **ignorieren**: Jeder Knoten v, dessen Distanzintervallvektor sich ändert, berechnet $h(v)$ neu mit seinem veränderten Distanzintervallen und wird wieder neu in ***OPEN*** eingefügt. Für neu einzufügende Knoten müssen auch die Pfadkosten $g(v)$ neu berechnet werden, sodass der zusätzliche Rechenaufwand für die Neuberechnung der Heuristik kaum ins Gewicht fällt. Der Distanzintervallvektor $\boldsymbol{I}(v_z)$ wird als konstant angenommen auf den Zustand zu Beginn der Graphensuche. Diese Daten müssen daher global als Kopie für die Neuberechnung der Kosten bekannt sein, falls sich $\boldsymbol{I}(v_z)$ ändert.
- **Variante 3**: Die Änderungen von $\boldsymbol{I}(v)$ **ignorieren** und $\boldsymbol{I}(v_z)$ **behandeln**: Wenn die Distanzinformationen von v_z sich ändern werden die heuristischen Kosten aller Knoten neu berechnet, die Teil von ***OPEN*** sind, bzw. neu eingefügt werden, da die Änderung der heuristischen Kosten auch alle betrifft und für die Sortierung in ***OPEN*** relevant ist. Da die Änderungen von $\boldsymbol{I}(v)$ ignoriert werden, ist eine Kopie der Distanzintervalle in jedem Knoten vorzuhalten, mit entsprechendem Speicheraufwand.
- **Variante 4**: Die Änderungen von $\boldsymbol{I}(v)$ **behandeln** und $\boldsymbol{I}(v_z)$ **behandeln**: Dies ist die Kombination der Methoden aus den Varianten 2 und 3 ohne den Speicheraufwand für die zu kopierenden Distanzintervalle. Der Rechenaufwand ist hier jedoch im Vergleich zu den anderen Varianten maximal.

Die heuristischen Kosten werden über Planungszyklen hinweg beibehalten, bis das Ziel der Planung sich ändert und werden der jeweiligen Variante entsprechend angepasst. Dabei werden nur die jeweiligen Knoten angepasst, die sich in ***OPEN*** befinden, bzw. durch Expansion in ***OPEN*** eingefügt werden, da der Wert nur für die Sortierung der Knoten in ***OPEN*** relevant ist. Knoten aus ***CLOSED*** müssen nicht wieder eingefügt werden, da bei einer zulässigen Heuristiken immer der jeweils kürzeste Pfad zu einem Knoten resultieren muss.

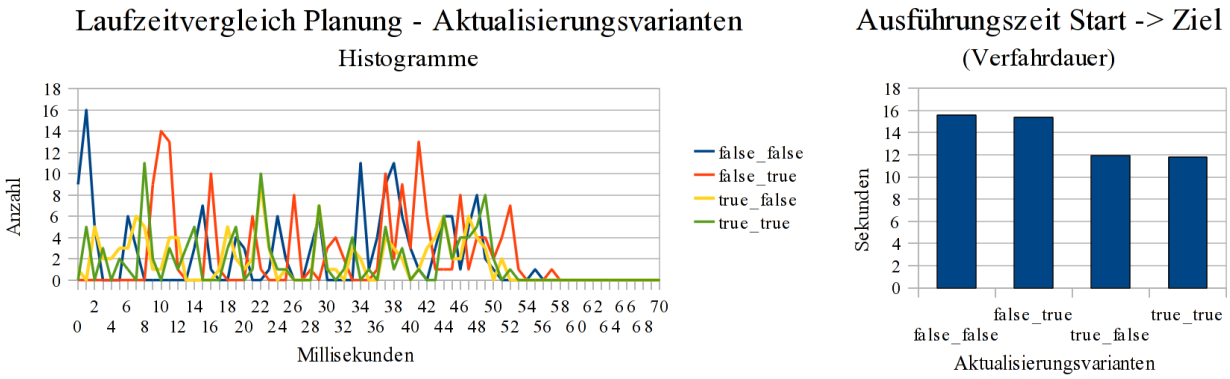


*Abbildung 4.13: Kenngrößen für die Varianten der Anpassung an **I**(v) und **I**(v_z). Die Variantenbezeichnung false_false bespielsweise bezeichnet die Variante 1 (**I**(v) **ignorieren** und **I**(v_z) **ignorieren**) true_false die Variante 2 (**I**(v) **behandeln** und **I**(v_z) **ignorieren**), usw. Das Experiment wurde in einer Simulationsumgebung wiederholt durchgeführt mit einem Graphen von 4000 Knoten und durchschnittlich 20 Kanten pro Knoten. Simuliert wurde ein bewegtes Objekt, was häufige Anpassung der Distanzen in den Knoten erfordert. **Links**: Histogramm der Laufzeiten eines Pfadplaner-Aufrufs in jedem Zyklus, **rechts**: die gesamte Ausführungszeit vom Start der Roboterbewegung bis zum Ziel.*

Da die Veränderung des Distanzintervalls im Zielknoten weniger häufig auftreten und kumuliert geringer ausfallen im Vergleich zu den Änderungen der Distanzen durch Berechnung in den anderen Knoten des Graphen, ist die globale Anpassung der Zielheuristiken bei Änderungen in $\boldsymbol{I}(v_z)$ weniger relevant für das Planerverhalten im Vergleich zu der Änderung der lokalen Distanzinformationen $\boldsymbol{I}(v)$.

Dies zeigen auch die in Abbildung 4.13 dargestellten Experimente. Während für die Laufzeiten des Planers pro Zyklus kaum ein Unterschied festgestellt werden kann für die verschiedenen Varianten, so ist die Beeinflussung der verfahrenen Bahn deutlich durch die verlängerten Ausführungsdauern zu erkennen, wenn keine Behandlung von $\boldsymbol{I}(v)$ vorgenommen wird und die Heuristik somit unzulässig ist. Es ist daher sinnvoll, immer die Variante 4 zu wählen bei geringfügig erhöhtem Gesamtaufwand für die Planung.

4.8 Graphensuche

In diesem Kapitel wird die Realisierung der Graphensuche thematisiert sowie die Einbindung der bildbasierten Distanzberechnung (Kapitel 2.4) und Schätzungsalgorithmen (Kapitel 4.5 und 4.6). Durch die Distanzberechnungen während der Suche ändern sich die

Kosten der Kanten des Graphen während der Suche und auch für den Teil des Graphen, der durch die Menge ***CLOSED*** repräsentiert wird. Dies wird durch spezielle, vom originalen A* abweichende Suchalgorithmen adressiert.

Generell ist für die Graphensuche der Standard-A*-Algorithmus zur effizienten Bestimmung des kostengünstigsten Weges geeignet. Durch die Änderung der Distanzintervallvektoren in den Knoten, verursacht durch die Berechnung und Schätzung der Distanzen während der Suche, werden jedoch die Kantenkosten modifiziert, sodass in diesen Fällen eine Aktualisierung der Knoten notwendig ist, die bereits durch die Graphensuche erfasst wurden und damit in ***OPEN*** oder ***CLOSED*** sind. Dafür bieten sich die aus der Literatur bekannten Modifikationen des A* an, die veränderte Kantenkosten effizient berücksichtigen und ***OPEN*** entsprechend modifizieren. Dazu gehört beispielsweise LPA (Lifelong Planning A-Star) aus [Koenig04], welcher im Folgenden kurz dargestellt und für die hier vorliegende Aufgabenstellung modifiziert wird. v_s bezeichnet den Startknoten, v_z den Zielknoten der Planung.

LPA-Adaption

Der LPA-Algorithmus (Lifelong Planning A-Star) realisiert eine Graphensuche wie der A*-Algorithmus (siehe Algorithmus 3.1). Das Ziel des Algorithmus ist es bei Veränderung von Kantengewichten während der Suche oder danach eine effiziente Anpassung des existierenden Suchbaumes durchzuführen, um auf die Änderung ohne vollständige Neuplanung reagieren zu können.

Wenn sich ein Kantengewicht ändert, muss an den beiden assozierten Endknoten der Kante festgestellt werden, ob aus einer Veränderung eines Kantengewichts eine notwendige Veränderung des Suchbaumes resultiert. Dazu werden pro Knoten v zwei Werte abgespeichert: Zum einen sind dies die für diesen Knoten berechneten Kosten vom Start der Suche im Wert $g(v)$ – dieser Wert ist äquivalent zu $g(v)$ im A* - zum anderen wird eine Berechnung der Wegkosten basierend auf den Kosten in den Nachbarknoten im Wert $rhs(v)$ mitgeführt, die sogenannte *Ein-Nachbar-Kostenschätzung*. Diese bestimmt die Kosten von v aus den Wegkosten der Nachbarn und den Kosten der verbindenden Kanten:

$$rhs(v) = \min_{v_i \in neighbours(v)} \left(g(v_i) + c(v_i, v) \right) \tag{4.39}$$

Die Funktion $neighbours(v)$ ist in 4.39 und im Folgenden so modifiziert, dass nur die validen Kanten des Knotens v verfolgt werden, also nur Nachbarknoten v_i geliefert werden, für die die Kante $e(\{v, v_i\})$ valide ist.

Grundlegend für die Funktion des LPA ist die Definition der Kostenkonsistenz, die auf den beiden Werten $g(v)$ und $rhs(v)$ fußt: Ein Knoten ist *konsistent*, wenn $g(v) = rhs(v)$, *überkonsistent*, wenn $g(v) > rhs(v)$ und *unterkonsistent*, wenn $g(v) < rhs(v)$. In den nicht

konsistenten Fällen muss eine Neuberechnung des Suchbaumes erfolgen. Zu diesem Zweck werden inkonsistente Knoten in ***OPEN*** eingefügt, um adäquat behandelt zu werden. Zu Beginn der Suche sind alle Knoten des Graphen konsistent ($g(v) = \infty$ und $rhs(v) = \infty$), bis auf den Startknoten, welcher in ***OPEN*** eingefügt wird (siehe den folgenden Algorithmus 4.8).

```
(1)  initGraphForLPA(G, v_s) // v_s ist Startknoten der Suche
(2)      forall nodes v in G
(3)          g(v) = rhs(v) = ∞
(4)      rhs(v_s) = 0
(5)      insert v_s into OPEN with key(v_s)
```

Algorithmus 4.8: Initialisierung des Graphen für den LPA

Der Sortierschlüssel $key(v)$ für die Knoten in ***OPEN*** (Algorithmus 4.9) ist so definiert, dass eine korrekte Anpassung des Suchgraphen garantiert werden kann, wenn die inkonsistenten Knoten aus der aufsteigend sortierten Menge ***OPEN*** entnommen werden. Die Sortierung erfolgt lexikographisch.

```
(1)  key(v)
(2)      base = min(g(v), rhs(v))
(3)      return [base + h(v, v_z); base]
```

Algorithmus 4.9: Schlüsselberechnung aus den Teilkosten eines Knotens v

Der erste Teil dieses Schlüssels realisiert einen der Standard-A*-Kostenfunktion ähnlichen Wert, wobei die g(v)-Werte durch das Minimum von $g(v)$ und $rhs(v)$ ersetzt werden (die temporäre Variable *base*). Falls Knoten in diesem Teil des Schlüssels gleiche Kosten enthalten wird weiter nach *base* sortiert, um eine weitere Differenzierung zu ermöglichen und die Knoten mit den günstigsten Wegkosten vorzuziehen.

Die $g(v)$-Werte aller Knoten sind initial auf ∞ gesetzt, bevor sie durch den Algorithmus bearbeitet werden. Die $rhs(v)$-Werte werden beim Einfügen in ***OPEN*** gesetzt. Zu diesem Zweck gibt es eine *updateState*(v)-Funktion, die bei jeder Expansion auf den Nachbarn des aktuellen Knotens aufgerufen wird (siehe auch Funktion *computeShortestPath*(), Algorithmus 4.11).

Im Gegensatz zum Original aus [Koenig04] sind hier die Zeilen 2 und 3 vertauscht, da die Modifikation des Schlüssels in Zeile 3 durch Berechnung von $rhs(v)$ nach Formel 4.39 zur Aktualisierung von ***OPEN*** führen müsste und zusätzlichen Aufwand erfordern würde. Diese Operation ist jedoch mit der obigen Reihenfolge unnötig und ändert nichts an dem Ergebnis der Funktion. Durch die Überprüfung in Zeile 4 werden nur Knoten zu ***OPEN***

hinzugefügt, die nicht konsistent sind. Da initial alle $g(v)$-Werte auf ∞ gesetzt sind, sind unberührte Knoten nach der Berechnung immer überkonsistent und werden somit zu ***OPEN*** hinzugefügt.

(1) **updateState**(v)
(2) **if**(v in ***OPEN***) remove v from ***OPEN***
(3) **if**($v \neq v_s$) $rhs(v) = \min_{v_i \in neighbours(v)} \left(g(v_i) + c(v_i, v) \right)$ // v_s ist Startknoten der Suche
(4) **if**($g(v) \neq rhs(v)$) insert v into ***OPEN*** with $key(v)$

Algorithmus 4.10: Aktualisierung der Teilkosten rhs(v) eines Knotens v

Die Funktion zur Suche des kürzesten Pfades im Rahmen des LPA ist in Algorithmus 4.11 aufgeführt. Durch die Bedingung in Zeile 2 wird die Graphensuche solange fortgesetzt, wie der Schlüssel des aktuellen Knotens kleiner als der Schlüssel des Ziels oder das Ziel inkonsistent ist. Nach De Morgan: Sobald das Ziel konsistent ist und günstiger als der günstigste Knoten in ***OPEN*** ist, wird die Suche abgebrochen. In Zeile 3 und 4 wird der günstigste Knoten aus ***OPEN*** entnommen und dann auf Über- oder Unterkonsistenz geprüft (Zeilen 5 und 8). Der konsistente Fall kann nicht vorkommen, da solche Knoten nicht in ***OPEN*** enthalten sind.

(1) **computeShortestPath**()
(2) **while**(($key(top(\textbf{OPEN})) < key(v_z)$) **or** ($g(v_z) \neq rhs(v_z)$)) //v_z: Ziel der Suche
(3) $v = top(\textbf{OPEN})$
(4) $pop(\textbf{OPEN})$
(5) **if**($g(v) > rhs(v)$)
(6) $g(v) = rhs(v)$
(7) **forall** v_i in $neighbours(v)$ $updateState(v_i)$
(8) **else**
(9) $g(v) = \infty$
(10) **forall** v_i in $\{neighbours(v) \cup \{v\}\}$ $updateState(v_i)$

Algorithmus 4.11: Suche des kürzesten Pfades mittels LPA

Im überkonsistenten Fall wird der Knoten in einen konsistenten Zustand überführt (Zeile 6) und alle Nachbarn mit den jetzt günstigeren Wegkosten des aktuellen Knotens v aktualisiert (Zeile 7), wodurch auch die Nachbarn potentiell überkonsistent werden. Im unterkonsistenten Fall wird der Knoten in den überkonsistenten Fall überführt (Zeile 9, wenn $rhs(v) = \infty$, dann ist der Knoten konsistent) und anschließend alle Nachbarn und der Knoten selbst aktualisiert (Zeile 10). Alle Nachbarn werden durch dieses Vorgehen

potentiell unterkonsistent.

In der klassischen Hauptfunktion des LPA-Algorithmus (Algorithmus 4.12) wird der Graph zunächst initialisiert (Zeilen 2-4) und dann wiederholt c*omputeShortestPath*() aufgerufen, um zwischen den Aufrufen auf sich ändernde Kantenkosten zu warten (z.B. durch entstehende Blockaden im Weg). Während des Aufrufs von c*omputeShortestPath*() werden die Kantenkosten als statisch angenommen. Im einem Lauf ohne Änderung der Kantenkosten expandiert der LPA die Knoten in derselben Reihenfolge wie ein Standard-A*-Algorithmus (siehe hierzu [Koenig04]).

```
(1)  searchGraphTimeOptim(v_s, v_z)
(2)      OPEN={}
(3)      initGraphForLPA(G,v_s)
(4)      forever
(5)          computeShortestPath()
(6)          wait for changes in edge costs
(7)          forall changed edges (u, v)
(8)              updateState(u)
(9)              updateState(v)
```

Algorithmus 4.12: Hauptfunktion der Graphensuche nach LPA mit Initialisierung von Startknoten v_s und Zielknoten v_g. Zwischen den Aufrufen der Funktion computeShortestPath() wartet der Algorithmus auf Änderungen der Kantenkosten.

Diese Aufteilung muss hier verändert werden, um schon während der Planung zum Ziel Modifikationen der Netzkosten zuzulassen (Algorithmus 4.13). Dazu werden die Anweisungen aus *computeShortestPath*() und *searchGraphTimeOptim(...)* neu aufgeteilt in die Funktionen *expandNode*(...) und *searchGraphTimeOptim(...)*. Weiterhin werden die Funktionen erweitert um eine Menge ***COLLECTEDNODES***, die die Knoten enthält, zwischen denen sich die Kantenkosten geändert haben. Im Folgenden wird nur auf die Änderungen eingegangen.

Die *adjustDistance*(*v*)-Funktion (Aufruf in Zeile 17) berechnet die Distanz des Knotens *v*, falls nicht bereits alle Distanzberechnungen aufgebraucht wurden oder dieser Knoten bereits berechnet wurde und aktualisiert dann gegebenenfalls alle Nachbarn von *v* mittels (in-)direkter Distanzpropagierung (siehe Kapitel 4.5.3 und 4.5.4). Falls sich durch die Distanzaktualisierung die Kosten der Kanten verändert haben, werden alle betroffenen Knoten in ***COLLECTEDNODES*** gesammelt.

```
(1)  expandNode(v, COLLECTEDNODES)
(2)      if(g(v) > rhs(v))
(3)          g(v) = rhs(v)
(4)          forall v_i in neighbours(v) insert v_i in COLLECTEDNODES
(5)      else
(6)          g(v) = ∞
(7)          forall v_i in {neighbours(v) ∪ {v}} insert v_i in COLLECTEDNODES
(8)
(9)  searchGraphTimeOptim(v_s, v_z)
(10)     g(v_z) = rhs(v_z) = ∞; g(v_s)=∞ ; rhs(v_s)=0
(11)     OPEN={}
(12)     insert v_s into OPEN with key(v_s)
(13)     while( ( key(top(OPEN)) < key(v_z) ) or ( (g(v_z) ≠ rhs(v_z)) ) )
(14)         v = top(OPEN)
(15)         pop(OPEN)
(16)         COLLECTEDNODES = {}
(17)         adjustDistance(v, COLLECTEDNODES)
(18)         expandNode(v, COLLECTEDNODES)
(19)         forall v_i in COLLECTEDNODES that have been touched  updateState(v_i)
```

Algorithmus 4.13: Modifikation von LPA

Die Expansion des obersten Knotens aus ***OPEN*** ist in die Funktion *expandNode*(...) (Aufruf in Zeile 18) verlagert zur Verbesserung der Übersichtlichkeit und Austauschbarkeit für Optimierungen. Da auf allen Knoten der Liste ***COLLECTEDNODES*** die Funktion *updateState*(*v*) aufgerufen wird, werden alle die Knoten, für die dies im originalen Algorithmus direkt aufgerufen wird, in *expandNode*(...) in diese Liste eingefügt.

Abschließend (Zeile 19) müssen die Kosten aller Knoten in ***COLLECTEDNODES*** über Aufruf von *updateState*(*v*) aktualisiert werden und die Knoten bei Inkonsistenz wieder in ***OPEN*** eingefügt werden.

Diese Umsortierung des Quellcodes verändert nicht den grundsätzlichen Ablauf des Algorithmus, da die Unterbrechungen zur Suche nach veränderten Kantenkosten lediglich feingranularer gesetzt werden (nach jedem Expansionsschritt, statt nach dem Erreichen des Ziels in vielen Expansionsschritten) und die Überprüfung der Abbruchbedingung unverändert bleibt. Die Sortierung von ***OPEN*** garantiert dabei, dass veränderterte Kosten der Knoten in der richtigen Reihenfolge bearbeitet werden. Insofern muss das Ergebnis übereinstimmen mit dem, das der originale LPA erzeugen würde, wenn die Kantenkosten

durch Berechnungen verändert würden zwischen den Aufrufen von *computeShortestPath*().

Komplexität

Die Aktualisierungen und das damit verbundene Neueinfügen in die ***OPEN***-List betrifft maximal die durch die Schätzung geänderten Knoten, und dies ist an die Anzahl der Knoten gebunden, in denen eine Distanz berechnet wird. Maximal sind jeweils alle Knoten des Graphen betroffen ausser den schon berechneten, daraus ergibt sich mit N als Knotenzahl, M als durchschnittliche Anzahl der Nachbarn pro Knoten und D als Anzahl der Distanzberechnungen eine Komplexität von $O(NM^2D)$, da die Aktualisierung garantiert, das alle zu aktualisierenden Knoten maximal zweimal auf die OPEN-Liste geschoben werden. Der Faktor M^2 ergibt sich aus der Funktionsweise des Algorithmus. Jede Expansion erfasst M Nachbarknoten und in der *updateState*(v)-Funktion werden für jeden dieser Knoten ebenfalls M Nachbarn betrachtet.

Wenn die Kantenkosten des Graphen stabil bleiben, da sie nicht mehr aktualisiert werden, verhält sich der LPA wie ein gewöhnlicher A*-Algorithmus und expandiert jeden Knoten bis zum Erreichen des Ziels einmal und hat damit eine Komplexität von $O(NM^2)$. Somit ist die Gesamtkomplexität für die Graphensuche $O((ND + N)M^2)$ linear in N, jedoch bleibt gewählte Anzahl an Distanzberechnungen der bestimmende Faktor, allein wegen der damit verbundenen Konstanten. Über diese Anzahl lässt sich somit das Laufzeitverhalten an eine maximale Grenze in der gegebenen Auflösung anpassen. Eine Optimierung des Expansionsschrittes ist im Anhang (Kapitel 8.2) dargestellt.

Integration der Suchmengenoptimierung für die Distanzpropagierung

```
(1)  expandNode(v, COLLECTEDNODES)
(2)      if(g(v) > rhs(v))
(3)          g(v) = rhs(v)
(4)      else
(5)          g(v) = ∞
(6)          insert v uniquely in COLLECTEDNODES
(7)      forall v_i in neighbours(v)
(8)          if(v_i not in set of touched points) adjustExpandedNode(v,
COLLECTEDNODES)
(9)          insert v_i uniquely in COLLECTEDNODES
```

Algorithmus 4.14: Adaption des Expansionsschritts für die Suchmengenoptimierung der Distanzaktualisierung

Prinzipiell ist die Suchmengenoptimierung für die Distanzpropagierung ein reiner

Filter, der alle nicht besuchten Knoten des Graphen ausblendet. Dadurch ergeben sich jedoch Inkonsistenzen im ausgeblendeten Teil des Graphen, die bei Expansion eines Knotens in den ausgeblendeten Teil berücksichtigt werden müssen. Die *expandNode*()-Funktion muss daher entsprechend angepasst werden (Algorithmus 4.14).

Die Sammlung der Nachbarknoten in ***COLLECTEDNODES*** wird hier heraus gelöst, um Code-Duplikationen zu reduzieren. In den Zeilen 7 - 9 wird dann die eigentliche Expansion durchgeführt und dabei in Zeile 8 ein Knoten, der noch nicht Teil der Suchmenge war, bezüglich seiner Distanzinformationen aktualisiert. Dies erfolgt in der Funktion *adjustExpandedNode*(…) (siehe Kapitel 4.5.3 und 4.5.4).

Effizienter Einsatz der limitierten Distanzberechnungen

Bereits in Kapitel 3.3 wurde der effiziente Einsatz der Kollisionstests für den wegoptimierenden Planer erörtert. Dort wird vorgeschlagen, die Distanzberechnungen in den Knoten zu platzieren, die aus der OPEN-Liste expandiert werden (Top-of-OPEN-Prinzip). Grundsätzlich gelten die dabei dargestellten Überlegungen auch für den zeitoptimierenden Planer. Ein einfacher Kollisionstest liefert jedoch nur eine binäre Aussage auf der Distanz. Der zeitoptimierende Planer hat hier die volle Distanzinformation zur Verfügung und kann dadurch die Platzierung der Distanzberechnungen in den Knoten an weiteren Kriterien ausrichten.

Die im Folgenden dargestellten Platzierungsvarianten haben ein wesentliches, übergeordnetes Ziel: den Nutzen der einzelnen Distanzberechnung zu erhöhen. Dazu können primär unnütze Distanzberechnungen eingespart werden oder die Distanzberechnung weiter gestreut werden, da nahe beieinanderliegende Distanzberechnungen nur geringen Informationsgewinn bezüglich der Umwelt bringen.

Grundsätzlich muss für die beschriebenen Varianten beachtet werden, dass eine breitere Streuung der Distanzberechnung den prinzipiellen Nachteil hat, dass die maximale Anzahl an Distanzberechnungen potentiell nicht erreicht wird, wenn das Ziel erreicht wird. Der resultierende Pfad beinhaltet dann eventuell geschätzte Knoten. Die Planung wird nichtsdestotrotz abgebrochen, obwohl noch Distanzberechnungen übrig sind. Mit diesen könnte man die Umweltinformation erhöhen und den Pfad verbessern. Für die Behandlung dieses Falles sind verschiedene Heuristiken denkbar, die hier jedoch nicht weiter untersucht werden (Neustart der Planung mit den verbleibenden Distanzberechnungen, etc.).

Distanzberechnungen können eingespart werden, wenn sie offensichtlich keinen Vorteil bringen können. Dies ist beispielsweise der Fall, wenn das geschätzte Distanzintervall eines Knotens unterhalb d_{min} liegt. Eine Messung könnte in diesem Fall nicht verhindern, dass der Knoten als kollidierend betrachtet würde, da sie innerhalb der Schätzung liegen muss (MAXMOVE^{-1}) und daher immer unter d_{min} liegen würde. Somit

kann keine Änderung der Kosten der assoziierten Kanten errreicht werden (sie sind alle unendlich).

Wenn das Distanzintervall d_{max} überschreitet, muss genauer untersucht werden, ob die Kosten der assoziierten Kanten des Knotens eine Änderung erfahren würden, denn dies ist vom aktuellen Kanten- und Knotenschätzer abhängig. Wenn diese Berechnung bestimmt, dass die Schätzung des Distanzverlaufs entlang der Kante $d(\boldsymbol{q})$ immer oberhalb von d_{max} liegt für die Wahl der Untergrenze l der Schätzung in den jeweiligen Knoten, dann kann keine Änderung der Kantenkosten eintreten und der Knoten ohne Distanzberechnung verbleiben. Die Berechnung kann sich lohnen, wenn viel Freiraum für den Roboter existiert und somit die Abstände groß werden.

Für die folgenden Betrachtungen wird die Definition der *planungsrelevanten Knotenmenge* benötigt: Wenn alle Knoten berechnet sind, kann der Planer darauf für eine gegebene Kantenschätzung einen zeitoptimierten Pfad planen. Zur Erzeugung dieses Pfades ist typischerweise lediglich ein Teil der Knoten des Graphen zu berechnen. Wenn man mit einen Standard-A*-Planer die Graphensuche mit einer unbegrenzten Menge an Distanzberechnungen durchführen lässt, so kann man aus den expandierten Knoten eine Menge konstruieren, die hier als planungsrelevante Knotenmenge $\boldsymbol{V}_p$ bezeichnet wird.

Die Platzierung von Distanzberechnungen in $\boldsymbol{V}_p$ soll dazu dienen, die Unsicherheit bezüglich der Umwelt in dieser Menge zu reduzieren. Die Unsicherheit ist dabei definiert über die Summe der Beträge aller Distanzintervalle in dieser Menge:

$$uncertainty(\boldsymbol{V}_p) = \sum_{v \in \boldsymbol{V}_p} |u(v) - l(v)| \tag{4.40}$$

Da die Menge nicht im Voraus bekannt ist und von der aktuellen, unbekannten Hindernissituation im Konfigurationsraum abhängt, werden im Folgenden Heuristiken dargestellt, die dazu beitragen sollen, die Unsicherheit nach Gleichung 4.40 zu reduzieren.

Eine Auswahl der Distanzberechnung der Knoten nach dem Top-of-OPEN-Prinzip, wie in Kapitel 3.3 dargestellt, ist für nah beieinander liegende Knoten nicht sinnvoll, wenn die Distanz zum Ziel hoch ist, da auf diese Weise $uncertainty(\boldsymbol{V}_p)$ nur geringfügig reduziert wird. Eine Methode, um dies anzugehen ist es, eine Distanzberechnung erst dann auf einem Knoten durchzuführen, wenn der Betrag des Distanzintervalls eine Grenze überschreitet, also falls gilt: $|u(v) - l(v)| > \tau$. Diese Methode wird als *Intervallbetrag-Selektor* bezeichnet.

Für dynamische Arbeitsräume kann diese Methode erweitert werden, um den Aspekt berücksichtigen, dass Distanzberechnungen, die weit entfernt von der aktuellen Roboterposition durchgeführt werden, durch die Bewegungen der Objekte schon eine prinzipielle Unsicherheit über die Zeit enthalten. Dies heißt, dass für jede Distanzberechnung in einem Knoten, der erst nach einer Zeitdauer t_{diff} vom aktuellen

Zeitpunkt an potentiell erreicht werden wird, das berechnete Distanzintervall um eine mögliche Objektbewegung in t_{diff} nach oben und unten erweitert gedacht werden muss. Dadurch haben Distanzberechnungen eine immer geringere Reduktion der Unsicherheit zur Folge, je weiter sie von der aktuellen Roboterposition entfernt sind. Um mehr Distanzberechnung in der Nähe der aktuellen Roboterposition zu platzieren, kann der Schwellwert am Knoten v für den Intervallbetrag daher t_{diff} approximierend abhängig von den Pfadkosten vom Startknoten v_s zum Knoten v gewählt werden: $|u(v) - l(v)| > \tau(g(v))$ und damit zur sogenannten *Kostenorientierten Intervallbetrag-Selektor-Methode*. Als Nachteil dieser Methode werden eventuell zuwenige Distanzberechnungen platziert, wenn Start- und Zielknoten nah aneinander liegen, da dann die entsprechenden Intervallschwellen nicht überschritten werden, bevor das Ziel entlang der geschätzten Knoten erreicht wird. Für diesen Aspekt kann auch noch die Distanz von Start und Ziel in die Betrachtung einfließen: $|u(v) - l(v)| > \tau(g(v), h(v))$. Neben diesen grundsätzlichen Betrachtungen werden im Folgenden noch weitere Heuristiken dargestellt.

Um eine breitere Streuung der Distanzberechnungen zu erreichen, sind weitere Strategien der Platzierung von Distanzberechnungen denkbar. Für jede Knoten-Expansion kann z.B. zufällig die Wahl zwischen Schätzung und Berechnung getroffen werden (*Zufalls-Selektor*).

Eine weitere Möglichkeit besteht darin, nur Distanzberechnungen zu setzen, wenn für das Schätzintervall gilt $l(v) < d_{min} < u(v)$, also wenn der Knoten möglicherweise kollidiert (für das oben beschriebene Vermeiden von Distanzberechnungen gilt im Unterschied $l(v) < u(v) < d_{min}$).

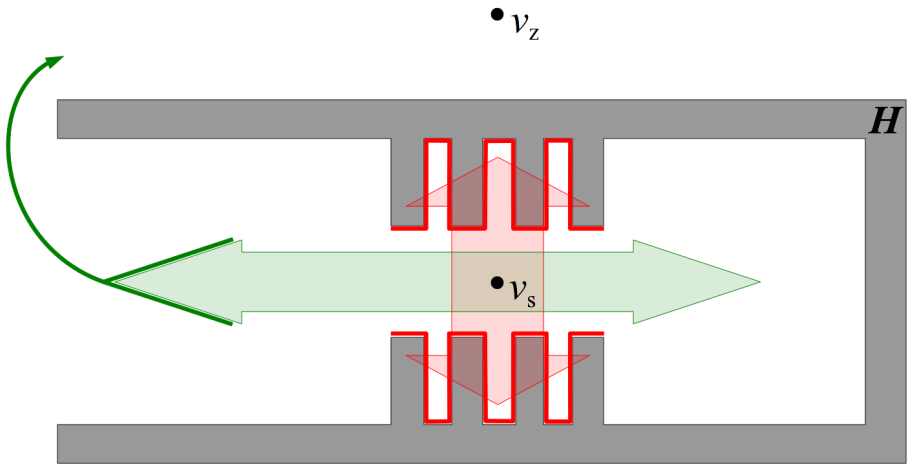


Abbildung 4.14: Nachteile der Platzierung von Distanzberechnungen in der Nähe des unbekannten Objekts ***H***

Diese als *Kollisions-Selektor* bezeichnete Methode platziert jedoch durch ihre Funktionsweise viele Distanzberechnungen in der Nähe von unbekannten Objekten $\boldsymbol{H}_l$. Wichtig für die Planung ist es jedoch, die Knoten des Freiraums zu finden, die die Zeitdauer

der Roboterbewegung optimieren. Dies sind im Allgemeinen nicht Knoten in der Nähe der unbekannten Objekte.Ein Beispiel zu dieser Problematik ist in Abbildung 4.14 dargestellt. Wenn Distanzberechnungen in der Nähe von Objekten platziert werden, verlieren sich diese im gegebenen Beispiel in den Ausbuchtungen des Objekts (der entsprechende Rand ist rot hervorgehoben). Die Platzierung von Distanzberechnungen nach Top-of-OPEN-Prinzip würde in der Tendenz die grün markierte Richtung wählen und daher früher den Ausweg aus den Sackgassen finden. Dies geschieht, da Knoten mit geringer Distanz zu unbekannten Objekten besonders hohe Kosten haben und daher selten expandiert werden, womit dort nach dem Top-of-OPEN-Prinzip keine weiteren Distanzberechnungen stattfinden. Die Kollisions-Selektor-Methode zur Platzierung von Distanzberechnungen ist daher nicht zu empfehlen.

In Kapitel 5.4 werden zu den hier präsentierten Platzierungsvarianten durchgeführte Experimentreihen dargestellt und die Auswirkungen der Varianten verglichen.

4.9 Auswirkungen der Knoten- und Kantenschätzer

Im folgenden werden die Auswirkungen der Wahl der Schätzer für Knoten und Kanten auf das Verhalten des Planers untersucht, wobei für die jeweiligen Effekte nicht immer Knoten- und Kantenschätzung gleichzeitig relevant sind.

Pessimistische Schätzung und Pfadexistenz

Die pessimistische Schätzung von Distanzen in den Knoten wird durch $select_i(v) = l_i(v)$ repräsentiert und für die Kanten durch den schlechtest möglichen Verlauf der Distanzen (siehe Kapitel 4.6). Diese Schätzer erreichen für viele Hindernissituationen die minimale Distanz d_{min}, bevor das Ziel erreicht wird. Denn wäre dies nie der Fall, wäre eine Bahnplanung quasi unnötig, da nie eine Kollision auftreten könnte.

Die geschätzten Distanzen, welche unter d_{min} liegen, werden als Kollision bewertet. Diese Bewertung führt dazu, dass kein Pfad gefunden wird, wenn die Kanten in Zielnähe alle als kollidierend (invalidiert) geschätzt werden (siehe Abbildung 4.15). Bei der Berechnung der Kantenkosten werden daher im realisierten System in den Fällen, in denen d_{min} **durch Schätzung** unterschritten würde, die Kosten der Kante mit einer Geschwindigkeit $V_{min} + \varepsilon$ mit einem kleinen $\varepsilon > 0$ berechnet. Dies ist weiterhin darauf beschränkt, dass die optimistische Knotenschätzung über dieser Grenze liegt, da diese Schätzung die realistische Breite des Distanzspektrums nach oben begrenzt. Wenn diese Schätzung die Grenze ebenfalls unterschreitet, müssten auch reale Distanzen diese Grenze unterschreiten. Die Schätzung der Kanten und Knoten selbst wird durch diese Anpassung nicht verändert, um keine Inkonsistenzen zu erzeugen. Lediglich die Berechnung der Kantenkosten wird modifiziert.

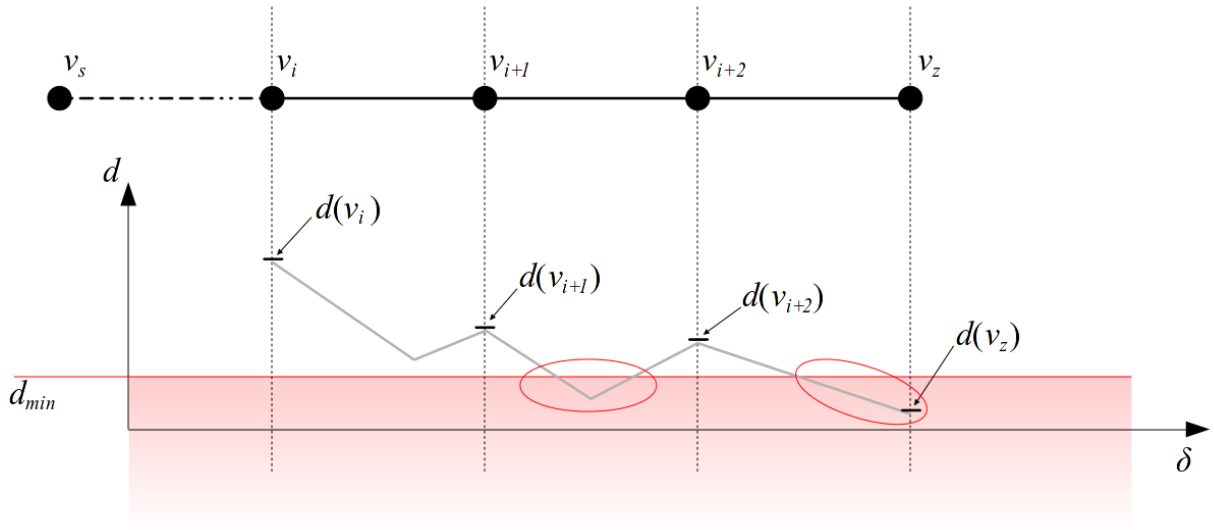


Abbildung 4.15: Problemstellung "unendliche Kantenkosten" durch Schätzung. Oberhalb des Koordinatensystems ist schematisch ein Pfad vom Startknoten v_s bis v_z dargestellt. Zwischen v_s und v_i liegen weitere, nicht dargestellte Punkte (durch die gestrichelte Linie symbolisiert), v_i bis v_{i+2} repräsentieren die letzten drei Knoten vor dem Zielknoten v_z auf diesem Pfad. Im Koordinatensystem über die Länge δ sind die Distanzen in den Knoten aufgetragen. Die Distanzen in den Knoten v_i bis v_{i+2} seien berechnet und in Knoten v_z aus v_{i+2} minimal geschätzt $d(v_z) = select_{min}(I(v_z))$. Die grauen Linien repräsentieren die pessimistische Schätzung der Distanzverläufe entlang der jeweiligen Kante. Rot eingekreist sind die problematischen Bereiche, in denen d_{min} durch die Schätzung unterschritten wird.

Diese Adaption erzeugt kein Sicherheitsproblem, da kollisionsbehaftete, invalidierte Kanten gar nicht in der Planung berücksichtigt werden und bei Knoten, die durch Berechnung als kollisionsbehaftet erkannt werden, alle zugehörigen Kanten invalidiert werden (siehe Kapitel 3.3).

Zielheuristik

Falls für die Knotenschätzung die pessimistische Variante gewählt wird, so ist die Zielheuristik (Kapitel 4.7) im Allgemeinen sehr schlecht informiert, d.h. sie schätzt die verbleibenden Wegkosten zum Ziel viel zu günstig im Verhältnis zum Knotenschätzer entlang dieses Weges. Dies ist vergleichbar mit einer geringen Gewichtung der Zielheuristik und führt dazu, dass die Graphensuche sich zu einer Breitensuche entwickelt, die durch ihre Menge an expandierten Knoten teuer wird, wie in Abbildung 4.16 dargestellt. Für die Experimente mit pessimistischem Knotenschätzer (P8-P32) nimmt die Anzahl explorierter Knoten im Verhältnis zum optimistischen Knotenschätzer stark zu (etwa das dreifache in diesem Experimentszenario, Abbildung 4.16 rechts). Die Laufzeit des Planers und der Schätzung (in Abbildung 4.16 links dargestellt) nehmen ebenso zu, da die erfassten Punkte zunehmen und die Schätzung somit für mehr Punkte berechnet werden muss. Bei den Experimenten wurden die Optimierungen Suchmengen- und Intervalllimitierung angewandt.

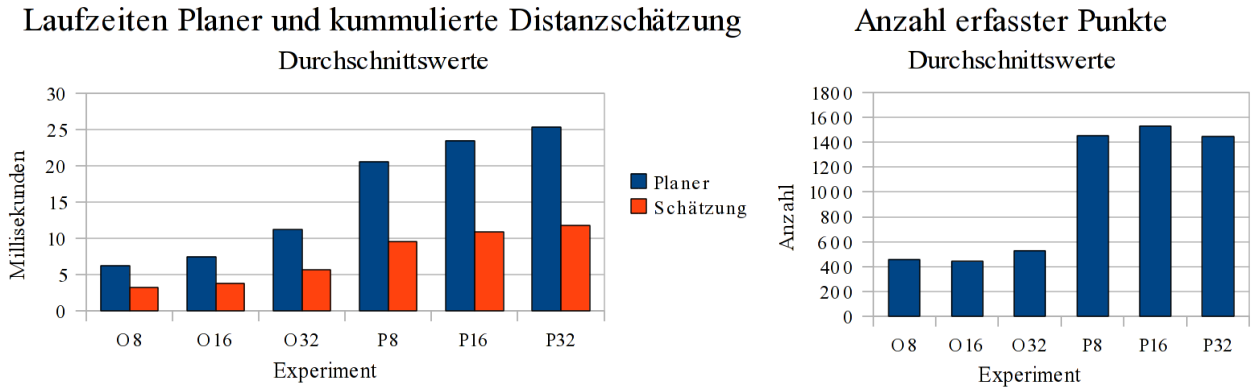


Abbildung 4.16: Auswirkungen der optimistischen und pessimistischen Knotenschätzung während der Planung bei Verwendung einer optimistischen Zielheuristik. 2D-Simulations-Experiment (siehe auch Kapitel 5) mit bewegtem Hindernis mit einem Graph von 4000 Knoten und durchschnittlich 20 Kanten pro Knoten. Die Experimentbezeichnung besteht aus einer Identifikation für den verwendeten Knotenschätzer O = Optimistisch bzw. P = Pessimistisch und einer Identifikation für die Anzahl (8-32) verwendeter Distanzberechnungen.

Pfadselektion

Durch die Wahl des Knotenschätzers wird nicht nur das Suchverhalten des Algorithmus beeinflusst wie oben dargestellt, sondern auch der Verlauf des resultierenden Pfades beeinflusst.

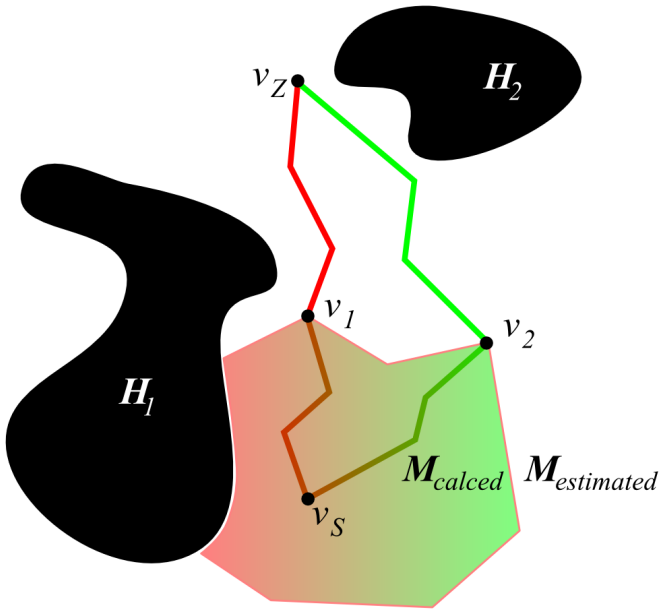


Abbildung 4.17: Pfadverläufe nach Wahl der Knotenschätzung für eine Konfigurationsraumbelegung durch die unbekannten Objekte $\boldsymbol{H}_1$ und $\boldsymbol{H}_2$. Es sind zwei Pfade dargestellt vom Startknoten v_s zum Zielknoten v_z. Die Pfade verlaufen in einem zum Teil durch Distanzberechnungen definierten Menge von Knoten $\boldsymbol{M}_{calced}$(welche in diesem Beispiel zusammenhängend ist) und einer Menge von Knoten, die geschätzt sind ($\boldsymbol{M}_{estimated}$). Der Pfad über v_1 resultiert aus einer pessimistischen Knotenschätzung für die Knoten in $\boldsymbol{M}_{estimated}$, der Pfad über v_2 resultiert aus einer optimistischen Knotenschätzung.

Grundlage der folgenden Betrachtung ist die Unterscheidung zwischen der Menge an Knoten des Graphen, die bereits durch Distanzberechnungen abgedeckt sind (in Abbildung 4.17 mit M_{calced} bezeichnet) und der verbleibenden Menge der geschätzten Knoten ($M_{estimated}$).

Seien zur Vereinfachung der Betrachtung im Folgenden die ersten Teilstücke eines Pfades immer berechnet, da Distanzberechnungen nach dem Top-of-OPEN-Prinzip platziert werden und daher zusammenhängend um die Startposition herum berechnet werden, bis alle verfügbaren Distanzberechnungen verbraucht sind. „Zusammenhängend“ ist in diesem Fall über die Knotennachbarschaft im Graphen definiert.

Betrachtet man nun die Gesamtkosten C der Pfade, so können diese aufgeteilt werden in Kosten K, die in der Menge der berechneten Knoten anfallen und Kosten S, die im Bereich der geschätzten Knoten anfallen:

$$C_{strategy}(v_s, v_i, v_z) = K(v_s, v_i) + S_{strategy}(v_i, v_z) \tag{4.41}$$

Das Subscript *strategy* referenziert den gewählten Knotenschätzer und kann Werte aus {*optimistic*, *pessimistic*} annehmen. Der Kantenschätzer sei für beide Fälle gleich, z.B. der Durchschnittsschätzer. Zur Vereinfachung wird im Folgenden als Approximation für die Kosten $S_{pessimistic}$ und $S_{optimistic}$ verwendet:

$$S_{pessimistic}(v_i, v_z) = \frac{1}{V_{min} + \epsilon} \cdot L(v_i, v_z) \text{ und } S_{optimistic}(v_i, v_z) = \frac{1}{V_{max}} \cdot L(v_i, v_z) \tag{4.42}$$

mit $L(v_i, v_z)$ als Summe über die geschätzten Verfahrdistanzen $\delta(...)$:

$$L(v_i, v_z) = \sum_{j=0}^{N-1} \delta(v_j, v_{j+1}), \text{ mit } v_0 = v_i \text{ und } v_N = v_z \tag{4.43}$$

In Worten: der Roboter verfährt mit maximaler Geschwindigkeit auf den Pfaden in $\boldsymbol{M}_{estimated}$ für den optimistischen Knotenschätzer und er verfährt mit sehr geringer Geschwindigkeit für den pessimistischen Knotenschätzer.

Wenn sich der Planer unter Verwendung des optimistischen Schätzers für den Weg über v_2 statt über v_1 entscheidet, dann gilt:

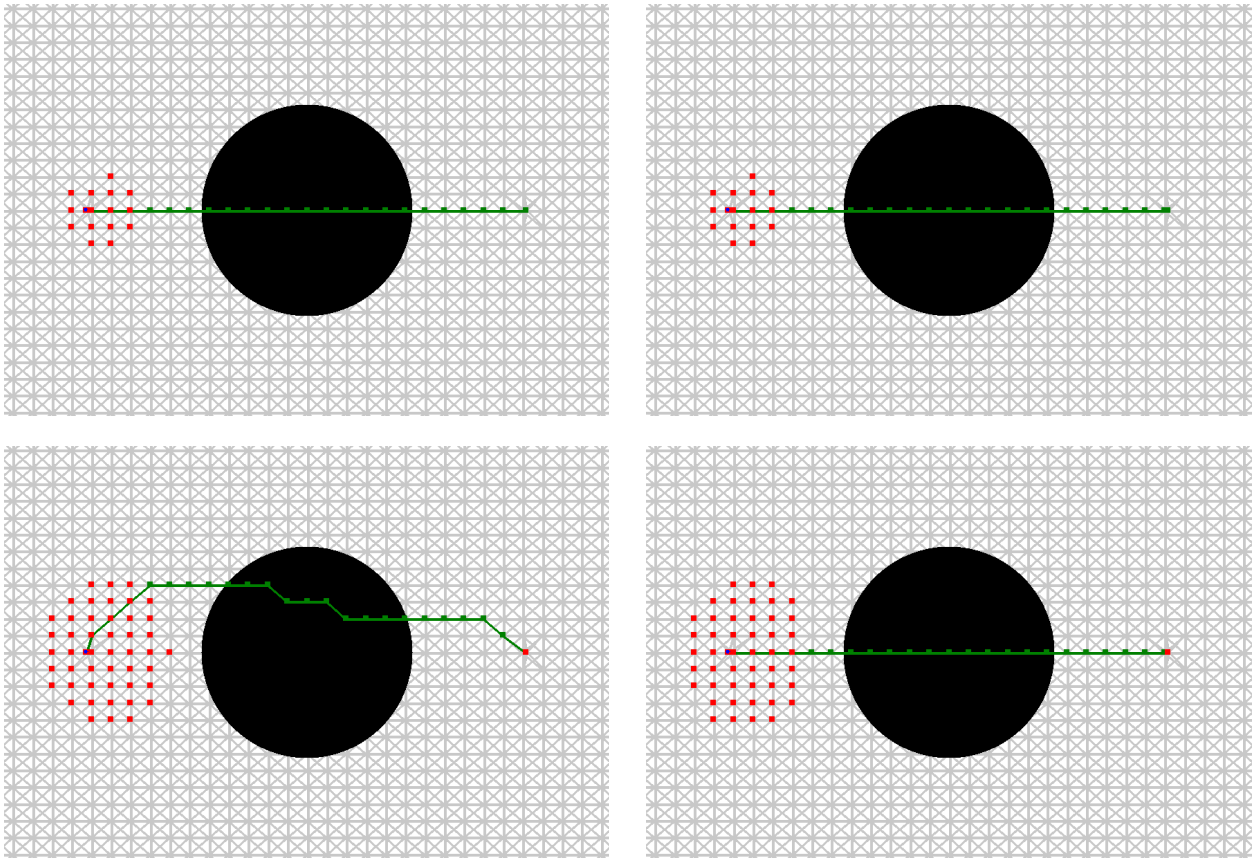
$$\begin{aligned} C_{optimistic}(v_s, v_{1,} v_z) &> C_{optimistic}(v_s, v_{2,} v_z) \\ K(v_s, v_1) + \frac{1}{V_{max}} \cdot L(v_{1,} v_z) &> K(v_s, v_2) + \frac{1}{V_{max}} \cdot L(v_{2,} v_z) \\ K(v_s, v_1) - K(v_s, v_2) &> \frac{1}{V_{max}} \cdot (L(v_{2,} v_z) - L(v_{1,} v_z)) \end{aligned} \tag{4.44}$$

Wie man sieht, ist der Einfluss der Kostendifferenz in $\boldsymbol{M}_{calced}$ deutlich gewichtiger für die Entscheidung für einen der beiden Wege als die Streckenlänge in $\boldsymbol{M}_{estimated}$, da der Vorfaktor $1/V_{max}$ klein ist. Daher werden auch längere Wege in $\boldsymbol{M}_{estimated}$ durch den optimistischen Schätzer gewählt, wenn dadurch Wege mit günstigeren Kosten in $\boldsymbol{M}_{calced}$ gewählt werden können. Dies sind häufig Wege, die innerhalb von $\boldsymbol{M}_{calced}$ zunächst eine größere Distanz zur Folge haben, wie in Abbildung 4.17 schematisch dargestellt der Weg über v_2.

Für den pessimistischen Schätzer gelten die Ungleichungen aus 4.44 analog. Hier ist der Faktor vor der dem Anteil des Weges in $\boldsymbol{M}_{estimated}$ mit $1/(V_{min} + \varepsilon)$ groß und daher wählt der Planer unter Verwendung dieses Schätzers vergleichsweise kurze Wege in $\boldsymbol{M}_{estimated}$, auch wenn dadurch höhere Kosten in $\boldsymbol{M}_{calced}$ anfallen.

In Abbildung 4.18 ist eine Testsequenz für ein Konfigurationsraumszenario mit konvexem Objekt in der Simulationsumgebung dargestellt. Für diesen Testlauf wurden wurden die Schätzer-Parameter in Richtung der Approximationen aus Gleichung 4.42 angepasst, um die Effekte deutlicher hervorzubringen. In der linken Spalte ist das Verhalten des Planers unter Verwendung des optimistischen Schätzers zu sehen, rechts das Verhalten unter Verwendung des pessimistischen Schätzers.

In der Praxis ist die Ausprägung dieses Effekts noch von weiteren Parametern wie Anzahl der Distanzberechnungen, Hindernissituation und -dynamik, Aufbau des Graphen und Robotergeschwindigkeit abhängig. Es ist daher möglich, dass die Wahl des Knotenschätzers auf das tatsächliche Verhalten einen geringen Einfluss hat.



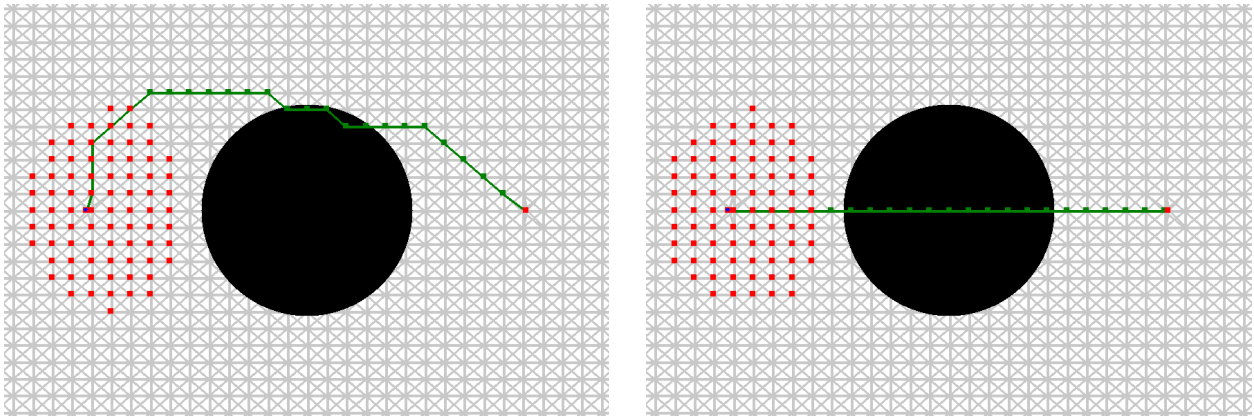


Abbildung 4.18: Ausschnitte aus einer Planungssequenz für ein Konfigurationsraumszenario mit konvexem Objekt, welches den hier untersuchten Effekt deutlich verursacht.. Dargestellt sind die Zyklen 0, 2 und 4 (von oben nach unten) und jeweils links das Verhalten des Planers unter Verwendung des optimistischen Knotenschätzers und rechts unter Verwendung des Pessimistischen. Der Graph ist nicht randomisiert und enthält nur Verbindungen der Knoten zu ihrer 8er-Nachbarschaft,, um weitere Effekte auszuschließen, die den dargestellten Effekt verzerren würden. In Grün dargestellt der geplante Pfad und in rot Knoten, in denen eine Distanz berechnet wurde.

Kantenselektion

Die Varianten der Kantenschätzer haben unter Voraussetzung der gleichen Hindernissituation deutlich unterschiedliche Kosten für die Kanten zufolge. Neben diesem (offensichtlichen) grundsätzlichen Ergebnis gibt es weitere Unterschiede und Effekte, die von der Kantenlänge abhängen, wie im Folgenden erläutert. Abbildung 4.19 zeigt schematisch diesen Zusammenhang für eine pessimistische Kantenschätzung.

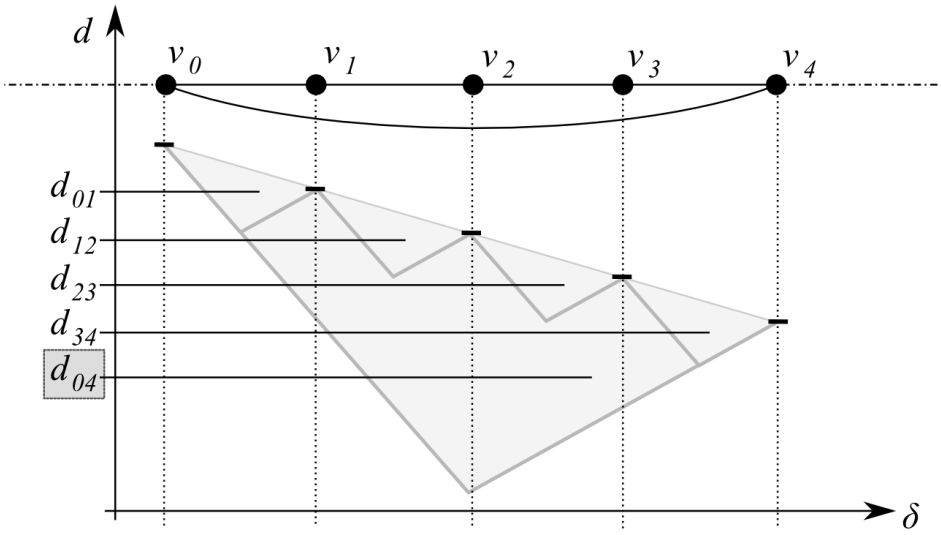


Abbildung 4.19: Pessimistische Distanzschätzung für lange Kanten im Vergleich mit kurzen Kanten. Im oberen Teil des Koordinatensystems ein Ausschnitt aus dem Graphen G mit den Knoten v_0 bis v_4 und den jeweiligen verbindenden Kanten. Die Distanzen aller Knoten sind berechnet und mittels kurzer schwarzer Balken markiert. Zwischen diesen Distanzen wird der Distanzverlauf entlang der Kanten pessimistisch geschätzt. Die Durchschnittswerte der Schätzung von Knoten v_i nach v_j sind als d_{ij} eingetragen.

Eine Auswahl an Knoten v_0 bis v_4 und deren verbindende Kanten sind im oberen Teil des Koordinatensystems eingezeichnet (vgl. auch Abbildung 4.15). Es existiert auch eine

Kante, die v_0 mit v_4 verbindet. In jedem Knoten sind die Distanzen berechnet und zwischen den Knoten wird die Distanz entlang der Kanten pessimistisch geschätzt. Dies ist in der Abbildung durch die grauen Linien schematisch repräsentiert. Im Koordinatensystem schematisch eingetragen sind auch die Durchschnittsdistanzen d_{ij} entlang der Kante von v_i nach v_j, sie berechnen sich aus dem Mittelwert der Distanzen der pessimistischen Schätzung der Distanzen entlang der Kante. Klar zu erkennen ist, dass für die lange Kante von v_0 nach v_4 die mittlere Distanz d_{04} für die Schätzung deutlich niedriger liegen muss, als für die Sequenz der kurzen Kanten, wo sich die mittlere Distanz für die Sequenz für den Weg von v_0 nach v_4 aus dem Mittelwert der einzelnen Distanzen berechnet: $¼(d_{01} + d_{12} + d_{23} + d_{34})$. Durch die damit im Schnitt niedrigere Geschwindigkeit entlang der Kante von v_0 nach v_4 im Vergleich zu den kurzen Kanten zwischen den Knoten werden durch den pessimistischen Kantenschätzer also lange Kanten durch hohe Kosten „bestraft“ und kurze für die Planung bevorzugt. Für den optimistischen Schätzer ergibt sich die umgekehrte Argumentation, dieser begünstigt lange Kanten und bestraft kurze Kanten. Der pessimistische Schätzer wird daher die Distanzmessungen auf benachbarte Knoten in der Nähe konzentrieren, während der optimistische Schätzer die langen Kanten bevorzugt und somit je nach Aufbau des Graphen die Distanzmessungen im Konfigurationsraum breiter streut.

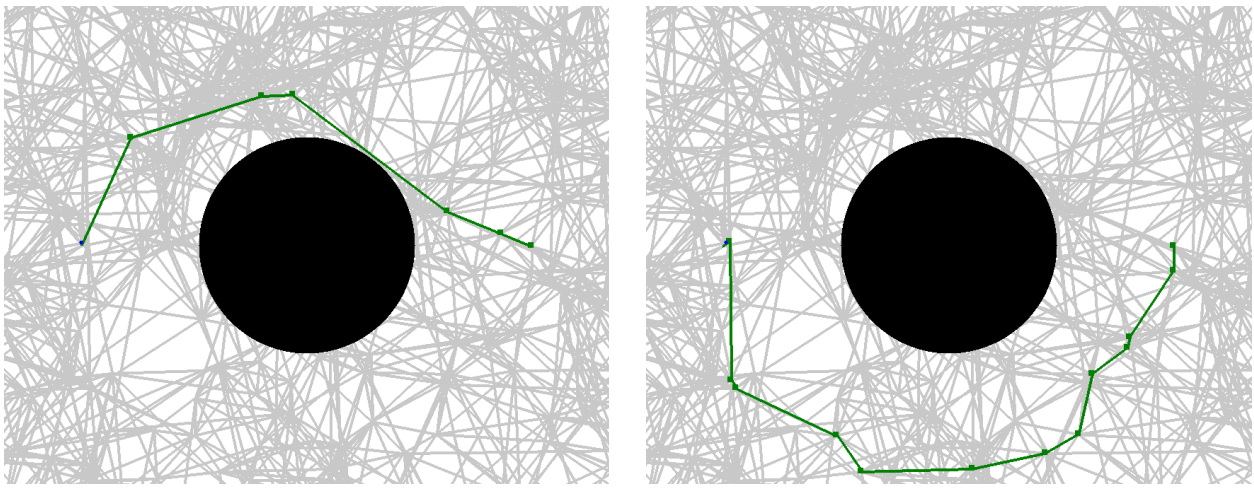


*Abbildung 4.20: Experiment zur Darstellung der Auswirkungen der Kantenschätzer. **Links** der optimistische Kantenschätzer, **rechts** der pessimistische Kantenschätzer. Der Graph ist grau, das unbekannte Objekt schwarz und der Pfad dunkelgrün dargestellt. Die Knoten im Pfad sind durch kleine grüne Punkte markiert. In beiden Fällen wird ein Pfad auf einem Graph mit randomisiertem Sampling geplant (500 Knoten, 12 Kanten pro Knoten). Zur besseren Vergleichbarkeit wurden beide Pfade auf vollständig berechneten Knoten geplant, um Effekte durch den unterschiedlichen Planungsverlauf über mehrere Systemzyklen auszuschließen. Der optimistische Kantenschätzer (links) erzeugt einen Pfad, der aus längeren Kanten besteht und nahe an das unbekannte Objekt heran reicht. Der pessimistische Kantenschätzer erzeugt einen Pfad, der aus kurzen Teilstücken aufgebaut ist, die einen größere Distanz zum unbekannten Objekt aufweisen.*

Da die Kantenkosten in der Nähe von unbekannten Objekten durch den pessimistischen Schätzer steigen, wählt dieser auch Pfade, die einen größeren Abstand vom unbekannten Objekt halten, als dies umgekehrt für den optimistischen Schätzer gilt. Die

entsprechenden Ergebnisse eines Simulationsexperiments sind in Abbildung 4.20 dargestellt.

Durch das Zusammenwirken der Grenzen V_{min}, V_{max} (bzw. d_{min}, d_{max}) der Distanz-geschwindigkeitsfunktion *speed*(d) (siehe Kapitel 4.4) mit den Varianten des Kantenschätzers ergeben sich weiterhin Effekte auf die Auswahl der Kanten. Wie in Abbildung 4.21 für zwei Fälle dargestellt, werden die geschätzten Distanzverläufe entlang einer Kante durch die nichtlineare (oder nur stückweise lineare) Funktion *speed*(d) zu den eingezeichneten gestrichelten Distanzverläufen deformiert, da die gestrichelten Verläufe dieselben Kosten für die Kante erzeugen, wie die Durchgezogenen.

Für das linke Beispiel bedeutet dies, dass eine Kante, deren Distanz größer ist als der jeweilige Abstand der Beispielknoten $d(v_i)$ und $d(v_j)$, bei Verwendung des optimistischen Kantenschätzers keine Veränderung der Kosten erfährt, bei dem pessimistischen Kantenschätzer aber sich noch in den Kosten verbessern kann, wenn sich die Abstände $d(v_i)$ und $d(v_j)$ vergrößern. Daher wird der optimistische Kantenschätzer Kanten, deren Endknoten Distanzen oberhalb von d_{max} aufweisen nur nach ihrer Länge differenzieren und keinen Vorteil mehr aus den höheren Distanzen ziehen können. Der pessimistische Kantenschätzer jedoch wird auch in diesen Fällen noch einen Kostenunterschied feststellen können und daher im Vergleich Kanten wählen, die weiter vom unbekannten Objekt entfernt sind.

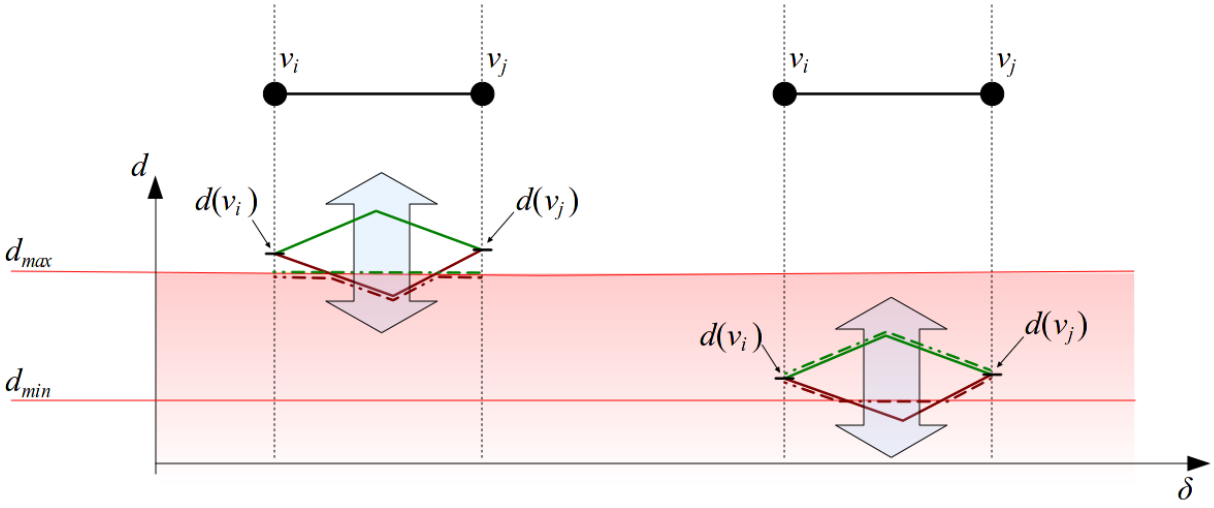


Abbildung 4.21: Zwei Beispiele für die Effekte der Kantenschätzer bei unterschiedlichen Distanzen in den Knoten v_i und v_j. Beide sind über eine Kante verbunden. Jeweils eingezeichnet ist die optimistische (in durchgängigem Grün) und die pessimistische (in durchgängigem Dunkelrot) Kantenschätzung. Die blauen, transparenten Doppelpfeile sollen jeweils die Betrachtung einer Translation der Knotendistanzen und somit Veränderungen der Schätzungen andeuten. ***Links:*** *Annäherung an die obere Grenze d_{max},* ***rechts:*** *Annäherung an die untere Grenze d_{min}.*

Ein ähnliches Bild ergibt sich bei Betrachtung der unteren Grenze d_{min}. Hier wird der pessimistische Kantenschätzer bei der Kostenberechnung durch die Verwendung der

Geschwindigkeit $V_{min} + \varepsilon$ begrenzt (siehe oben), und daher fallen die Änderungen der Kantenkosten im Vergleich zum optimistischen Kantenschätzer geringer aus für eine Kante mit größeren/kleineren Distanzen $d(v_i)$ und $d(v_j)$.

Diskussion

Die optimistische Knotenschätzung bietet ein günstiges Laufzeitverhalten und wählt Wege mit vergrößerter Distanz, weicht also unbekannten Objekten mehr aus, als dies pessimistische Schätzer tun. Dies kann beispielsweise mit der pessimistischen Kantenschätzung verbunden werden, da diese auch gegebenenfalls Kanten mit größerer Distanz wählt. Die pessimistische Knotenschätzung ist in Zielnähe günstiger, wenn selbst bei der daraus folgenden Abwandlung zur Breitensuche aufgrund der Zielnähe voraussichtlich nur wenige Kanten expandiert werden und führt zu einer stärkeren Zielorientierung des Planers. Das Umschalten zur Laufzeit zwischen beiden Schätzervarianten ist möglich, erfordert jedoch eine Neuberechnung aller Kantenkosten.

4.10 Arbeitsraumdynamik

Berechnete Distanzen in den Knoten und die daraus geschätzten Distanzen in den Knoten und Kanten werden so lange wie möglich beibehalten (vergleiche die Kantenrevalidierung für den wegoptimierenden Planer, Kapitel 3.4). Dies ist insbesondere in den Phasen auch hilfreich, in denen der Arbeitsraum eine geringe Dynamik oder gar lediglich statische unbekannte Objekte aufweist. Durch die in diesem Fall stetig zunehmende Information bezüglich der Hindernissituation können auch schwierige Planungssituationen bewältigt werden. Im Vergleich zum wegoptimierenden Planer (Kapitel 3) ist es sogar von Vorteil, die Distanzinformationen länger zu konservieren als die Belegungsinformationen für eine Kante. Die korrekten Belegungsinformationen für eine Kante können relevant für das Erreichen des Ziels als solches sein, korrekte Distanzinformationen beeinflussen lediglich die Optimalität des Pfades, wodurch fehlerhafte Distanzinformation keine erheblichen Auswirkungen haben, fehlerhafte Kollisionsinformationen in den Kanten hingegen schon.

Bewegungen der Objekte im Arbeitsraum führen im Allgemeinen dazu, dass alle berechneten (und daher indirekt auch die geschätzten) Distanzen aus vorhergehenden Systemzyklen die reale Situation lediglich annähern. Daher führen Objektbewegungen in diesem Fall zu globalen Änderungen der Distanzinformationen, während die Kollisions-informationen pro Kante nur lokal verändert werden.

Fehlerhafte Distanzinformationen zeigen sich insbesondere durch Knoten, für die keine Nachbarkonsistenz erreicht werden kann. Dies tritt beispielsweise bei einer Distanzberechnung in einem Knoten auf oder bei der Distanzpropagierung. Im Folgenden

werden zunächst Methoden beschrieben, um durch Modifikation der Distanzinformationen in den Knoten an die Bewegungen der Objekte im Arbeitsraum die Distanzinkonsistenzen zu vermeiden. Dies ist am Vorbild der Revalidierungsmechanismen (Kapitel 3.4) orientiert. Anschließend wird dargestellt, wie Distanzinkonsistenzen behandelt werden können, wenn sie auftreten. Beides fügt sich in das Rahmenwerk der Distanzintervalle ein, da die bestehenden Intervalle des Planungsgraphen modifiziert werden. Generell verwendet der zeitoptimierende Planer die Kantenrevalidierungsmechanismen des wegoptimierenden Planers und ersetzt die Knotenrevalidierung durch die im Folgenden beschriebenen Anpassungen.

Adaption der Kantenrevalidierungsmechanismen

Die in Kapitel 3 dargestellten Revalidierungsmechanismen für Belegungsinformationen von Kanten können zum Teil auch für die Distanzinformationen geeignet angepasst werden. Die Namensgebung ist angelehnt an die Bezeichner in Kapitel 3.4, um Zuordnungen zu erleichtern.

Die Strategien „Keine Revalidierung“ oder „Ziel-Revalidierung“ sind hier nicht sinnvoll adaptierbar, da ihre Adaption bedeuten würde, dass für größere Zeiträume keine Anpassung der Distanzinformationen in den Knoten geschehen würde. Bei Bewegungen der Objekte im Arbeitsraum würden dann während der Planung sehr viele Distanzinkonsistenzen in den Knoten entstehen, die aufwendig behandelt werden müssten (s.u.). Diese Strategien werden daher hier nicht adaptiert.

Die Adaption der Strategie „Auszeit-Revalidierung“ impliziert eine ständige, gleichförmige Dynamik der Objekte des Arbeitsraumes. Die Distanzintervalle jedes Knotens werden hierbei mit einem festen δ_{move} pro Zeitintervall erweitert: $[l - \delta_{move}, u + \delta_{move}]$. Diese Strategie kann in vielen Anwendungsbereichen für eine sinnvoll gewählte Konstante ausreichend sein; sie ist jedoch ähnlich zur Revalidierungsstrategie für phasenweise statische Hindernissituationen nicht geeignet, da sie Distanzinformationen modifiziert, die sich in diesen Situationen nicht ändern.

Interessant für den zeitoptimierenden Planer ist daher insbesondere die Adaption der Strategie „Sensorbasierte Revalidierung“. Hierbei wird der Indikator aus Kapitel 3.4 für die Bewegung der Objekte im Arbeitsraum eingesetzt. Basierend auf diesem konservativen Indikator kann ein angepasstes δ_{move} berechnet werden, mit dem alle Distanzintervalle expandiert werden: $[l - \delta_{move}, u + \delta_{move}]$.

Durch die Modifikation aller Distanzintervalle mit einem festen δ_{move} wird die Nachbarkonsistenz der Knoten aufrecht erhalten, was sich leicht zeigen lässt. Durch die Nicht-Linearität der Geschwindigkeitsfunktion, die bei V_{min} und V_{max} begrenzt ist, müssen jedoch für alle veränderten Knoten die Kantenkosten neu berechnet werden. Wenn dies

während der Planung geschieht, müssen die Knoten bei geänderten Kosten wieder neu in ***OPEN*** eingefügt werden (*reopening*). Dies kann recht teuer sein und daher sind die Modifikationen sinnvoll im Vorfeld der Planung, d.h. mit leerer ***OPEN***-Liste durchzuführen, wie dies auch bei Anwendung der Revalidierungsmechanismen geschieht (siehe Algorithmus 3.3).

Distanzinkonsistenzen

Distanzänderungen durch Arbeitsraumdynamik, die durch obige Methoden nicht abgefangen werden, äußern sich dadurch, dass Knoten während der Distanzpropagierung nicht mehr konsistent sind. Dies tritt bei Schätzung eines Knotens zutage, der dann nach der Schätzung nicht mehr intervallkonsistent ist. Eine andere Möglichkeit ist, dass ein Knoten, für den eine neue Distanzberechnung durchgeführt wurde, nicht nachbarkonsistent ist.

In diesem Fall gibt es mehrere Ansätze, um diesen Fehlerzustand zu beheben. Es ist möglich, die Planung bei Auftreten eines solchen Fehlers abzubrechen, alle Distanzintervalle zurückzusetzen und dann auf diesem neu initialisierten Graph die Planung im nächsten Zyklus neu zu beginnen. Dies entspricht einer Art Ausnahmebehandlung.

Alternativ können alle Distanzintervalle im Sinne der obigen Methoden erweitert werden, um die Nachbarkonsistenz der Knoten zu garantieren. Sei $[l_f, u_f]$ das nicht intervallkonsistente Distanzintervall in Knoten v_f, so dass gilt: $l_f > u_f$. Um dieses Intervall wieder zur Konsistenz zurückzuführen und gleichzeitig nicht die Nachbarkonsistenz aller anderen Knoten zu beeinflussen, werden alle Intervalle $[l_i, u_i]$ einschließlich des inkonsistenten Intervalls wie folgt korrigiert:

$$\left[l_i, u_i\right]' = \left[l_i - \frac{\left(l_f - u_f\right)}{2}, u_i + \frac{\left(l_f - u_f\right)}{2}\right] \tag{4.45}$$

Auch diese Korrektur ist generell sehr kostenintensiv, da sie alle Kantenkosten verändert. Ein Neustart der Planung und Löschen der ***OPEN***-Liste ist daher in diesem Fall sinnvoll. Dies ist jedoch dem Zurücksetzen der Distanzintervalle (s.o.) vorzuziehen, da die Intervalle durch diese Modifikation im Normalfall enger gefasst sind und somit mehr Informationen über den Arbeitsraum verbleiben.

Eine weitere Alternative ergibt sich durch die Beobachtung, dass die Ursache für Intervallinkonsistenzen die neuen Distanzberechnungen eines Systemzyklus sind und die Inkonsistenzen schon und zunächst nur bei den direkten Nachbarknoten des Knotens zutage treten, der durch Distanzberechnung die Ursache für diese Inkonsistenz bildet, da zuvor alle Knoten des Graphen nachbarkonsistent gewesen sind (was impliziert, dass die Optimierung „Suchmengenlimitierung“ nicht verwendet werden darf).

Da die neu berechnete, inkonsistente Distanzinformation durch Distanzpropagierung

verteilt werden muss, ergibt sich im Rahmen der indirekten Distanzpropagierung mittels $\boldsymbol{OPEN}_{du}$-Liste (Algorithmus 4.2, Seite 143) die Möglichkeit, die Nachbarkonsistenz (und damit auch die Intervallkonsistenz) zu erzwingen. Dies geschieht während der Propagierung in Zeile 17, indem die Operation $DF(I_{from}, I_{to})$ mit einer angepassten Operation $DF'(I_{from}, I_{to})$ ersetzt wird. I_{from} ist hierbei das aus dem Vorgängerknoten v_{from} geschätzte Intervall entlang der Kante und I_{to} das zu aktualisierende Intervall des Knotens v_{to}. Die Operation $DF'(I_{from}, I_{to})$ ist nicht mehr symmetrisch definiert, sodass die Reihenfolge der Argumente relevant ist:

$$DF'(I_{from}, I_{to}) = \begin{cases} [l_{to} = l_{from}, u_{to} = l_{from}] & \text{,falls } u_{to} < l_{from} \\ [l_{to} = u_{from}, u_{to} = u_{from}] & \text{,falls } l_{to} > u_{from} \\ DF(I_{from}, I_{to}) & \text{, sonst} \end{cases} \tag{4.46}$$

Durch diese angepasste Operation ist die Schätzungskonsistenz zwischen v_{from} und v_{to} wieder hergestellt. Dies geschieht im Unterschied zu der vorherigen Version allerdings auch mit Intervallen I_{to}, die aus Distanzberechnungen stammen, für die also gilt $l(I_{to}) = u(I_{to})$. Weil auch berechnete Distanzintervalle angepasst werden, breitet sich diese Aktualisierung unter Umständen recht weit im Graphen aus. Bei korrekter Funktion des Kollisionstests kann dies jedoch nicht die im aktuellen Zyklus berechneten Knoten betreffen, da diese konsistent zueinander sein müssen, also auch zu dem Knoten, der die Inkonsistenz auslöst.

Die Warteschlangen-Propagierung kann hierbei allerdings nicht verwendet werden, da in diesem Algorithmus keine definierte Richtung der Aktualisierung vorgegeben ist. Die Propagierung kann daher auf den ursprünglichen Knoten zurückfallen, da die Intervallinkonsistenz hier symmetrisch ist, d.h. sowohl Nachbarknoten als auch der modifizierte Knoten zueinander inkonsistent sind (dies muss nicht für alle Nachbarknoten gelten).

4.11 Zusammenfassung

In diesem Kapitel wurde dargestellt, wie das Konzept des echtzeitfähigen, wegoptimierenden Planers (Kapitel 3) so modifiziert werden kann, dass die Berechnung von zeitoptimierten Pfaden echtzeitfähig möglich ist unter Berücksichtigung dynamischer Hindernisse.

Ausgangspunkt der Betrachtungen ist die zur Gewährleistung der Sicherheit notwendige Geschwindigkeitsregelung des Roboterarms basierend auf der Distanz zum nächsten sensorisch erfassten unbekannten Objekt, wie z.B. dem Menschen. Die Distanz in jedem Knoten und entlang jeder Kante eines Pfades im statischen Graphen kann dann verwendet werden, um für diesen Pfad über die Distanz-Geschwindigkeitsfunktion eine Zeitdauer zu ermitteln, die für die Ausführung des Pfades benötigt wird. Über die Anpassung der Kostenfunktion und der Zielheuristik der Graphensuche zur Berücksichtigung der Zeitdauer kann ein zeitoptimierender Pfad auf dem Graphen gefunden

werden, welcher die Pfadlänge und den Hindernisabstand ausbalanciert.

Zur Erreichung der Echtzeitfähigkeit werden die (teuren) Distanzberechnungen auch hier für eine Planung innerhalb eines Systemzyklus limitiert, um die Echtzeitfähigkeit zu erreichen. Dies wird begleitet von einem Konzept zur Schätzung unbekannter Distanzen in Knoten und Kanten, welches auf den berechneten Distanzen in den Knoten basiert. Die Distanzberechnungen werden während der Planung durch den Graphensuchalgorithmus planungsrelevant auf den Knoten verteilt, hierzu werden verschiedene Varianten diskutiert und implementiert. Basierend auf der Schätzung werden effiziente Algorithmen entwickelt, die bei einer Berechnung eines Knotens die Distanzinformationen im Graphen verteilen. Diese hierzu entwickelten unterschiedlichen Varianten wurden in ihrer Auswirkung auf die Graphensuche untersucht (Kapitel 4.9). Die Graphensuche ist eine adaptierte und optimierte Form des Lifelong-Planning-A-Star (LPA, [Koenig04]), die benötigt wird, da durch die Schätzung auch Kantenkosten von Knoten modifiziert werden, welche schon durch die Graphensuche erfasst wurden.

Auch die Distanzinformationen werden zyklenübergreifend beibehalten und gegebenfalls an die Dynamik der unbekannten Objekte angepasst. So wird über einige Systemszyklen hinweg eine stetige Erweiterung der Kollisionsrauminformationen erreicht. Im Vergleich zur rein wegoptimierenden Planung erreicht der zeitoptimierende Planer deutliche Reduzierungen für die Gesamtdauer der Ausführung eines Pfades (siehe experimentelle Ergebnisse in Kapitel 5) und ist daher vor allem im Bereich der Mensch-Roboter-Kooperation/Koexistenz vorteilhaft einsetzbar.

5 Experimentelle Ergebnisse für die weg- und zeitoptimierende Bahnplanung

In diesem Kapitel werden Experimente dargestellt, die die Planung und Ausführung für verschiedene Konfigurations- und Arbeitsraumsituationen betrachten. Dazu wurden sowohl Simulationsexperimente (Kapitel 5.1) durchgeführt, als auch Realwelt-Experimente (Kapitel 5.2). Spezielle Aspekte wie Revalidierungsstrategien und Platzierung der Distanzberechnungen werden in den Kapiteln 5.3 und 5.4 beleuchtet.

5.1 Simulationsexperimente

Die Untersuchungen des weg- und zeitoptimierenden Bahnplaners finden in einem simulierten 2D-Konfigurationsraum statt. Die Simulation garantiert die Vergleichbarkeit und Wiederholbarkeit der Experimente mit verschiedenen Parametern. In einem 2D-Konfigurationsraum lassen sich einfach spezielle Konfigurationsraumszenarien konstruieren und testen, um die Eigenschaften des Planers heraus zu arbeiten. Im Allgemeinen sind Konfigurationsräume aus Teilen der hier untersuchten Konfigurationsraumszenarien zusammengesetzt.

Die Simulation wurde für alle Konfigurationsraumszenarien sowohl mit dem weg- als auch mit dem zeitoptimierenden Planer durchgeführt. Der verwendete Graph umfasst 4000 Knoten mit uniform randomisiertem Sampling und etwa 20 Kanten pro Knoten. Für den zeitoptimierenden Planer wurden die optimistische Knotenschätzung und die Durchschnittswert-Kantenschätzung verwendet.

Die Bahnplanung wurde dann zu jedem Zyklus zweimal durchgeführt, zunächst mit der begrenzten Anzahl an Kollisionstests bzw. Distanzberechnungen, anschließend mit unbegrenzter Anzahl. Die Distanz-/Kollisions-Informationen des zweiten, unbegrenzten Planers (Idealer Planer) wurden zu Beginn des jeweils nächsten Zyklus wieder aus dem Graphen entfernt, um den Ablauf des begrenzten Planers nicht zu beeinflussen. Für beide Varianten wurden im Anschluss an die jeweilige Planung die verbleibenden Kosten bis zum Ziel mit berechneten Distanzen/Kollisionen entlang der jeweiligen Bahn ermittelt. Die Kosten für den begrenzten und den idealen Planer sind in den entsprechenden Diagrammen gemeinsam dargestellt.

Konfigurationsraum mit statischen Hindernissen

Für die Darstellung des Verhaltens im einem statischen Konfigurationsraum wurden Konfigurationsraumszenarien (auch „Benchmark“) gewählt, welche hier mit SIMPLE und DETOUR bezeichnet werden. Das Szenario SIMPLE enthält ein konvexes Hindernis. Im Szenario DETOUR (s.u.) gibt es eine wegoptimale Verbindung durch eine enge Ansammlung mehrerer konvexer Objekte und einen zeitoptimierten Weg, welcher diese Hindernisse weiträumig vermeidet.

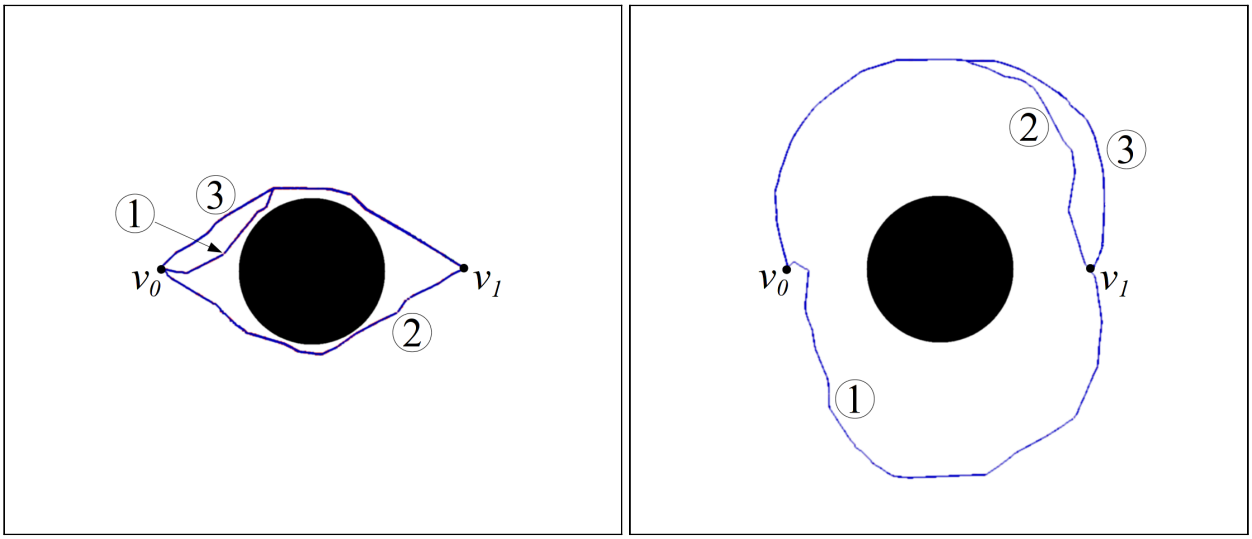


*Abbildung 5.1: Konfigurationsraumszenario SIMPLE mit statischem konvexem Objekt. Dargestellt sind jeweils mehrere Bewegungen (blaue Linien, mit den Markierungen 1 bis 3 versehen) des Roboters zwischen den Knoten v_0 und v_1, mit Planung durch den wegoptimierenden Planer (**links**) und den zeitoptimierenden Planer (**rechts**).*

In Abbildung 5.1 sind die durch den Roboter tatsächlich verfahrenen Bahnen mit dem weg- und zeitoptimierenden Planer für das Konfigurationsraumszenario SIMPLE dargestellt. Der zugrunde-liegende statische Graph ist ausgeblendet, da bei 4000 Knoten und 20 Kanten pro Knoten die Darstellung überfrachtet wäre. Der Konfigurationsraum und der Graph umfassen jeweils das gesamte Rechteck, dies gilt auch für alle im Folgenden dargestellten Experimente. Die Planung wurde zyklisch von Knoten v_0 zu v_1 und wieder zurück ausgeführt, wenn der jeweils andere Knoten erreicht war. Hierbei ergeben sich sichtbare Unterschiede zwischen den einzelnen verfahrenen Bahnen, da die Informationen über die Umgebung während der Ausführung zunehmen und daher andere Bahnen gewählt werden. Auffallend ist vor allem für die erste Bahn die zu Beginn direkte Ausrichtung auf das Ziel, was in diesem Fall auf das Hindernis zuführt und daher für die erste Bahn eine Kostennachteil erzeugt. Dies spiegelt sich auch in der Ausführungdauer (Abb. 5.2) wieder, wobei der Nachteil für den wegoptimierenden Planer größer ausfällt, als für den zeitoptimierenden (hier ist sie messbar, aber minimal).

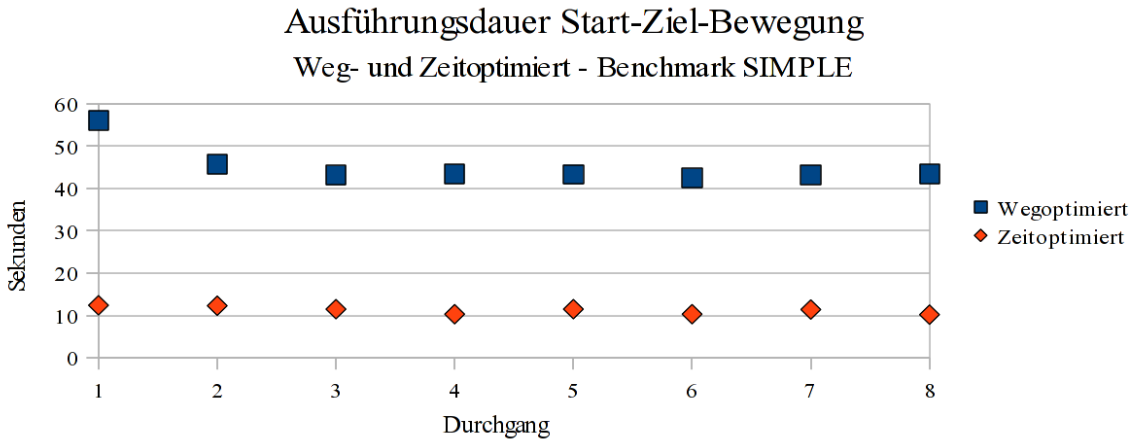


Abbildung 5.2: Ausführungszeiten in Sekunden im Konfigurationsraumszenario SIMPLE für Start-Ziel-Bewegungen zwischen den Knoten v_0 und v_1 für den weg- und zeitoptimierten Planer. Dargestellt sind acht Durchgänge (vier Schleifen mit jeweils Hin- und Rückweg).

Dies liegt darin begründet, dass der zeitoptimierende Planer mehr Informationen der Umwelt erfasst und daher schneller und besser auf sie reagieren kann. In einfachen Szenarien wie diesem kann somit schnell die ideale Bahn gefunden werden.

In den folgenden Diagrammen sind die oben beschriebenen Kosten für den jeweiligen (idealen) Planer und die jeweils verwendete Anzahl an Kollisionstest aufgetragen. Die Diagramme für den weg- und zeitoptimierenden Planer unterscheiden sich in der Skalierung der Zeitachse, da jeweils in etwa ein Hin- und Rückweg dargestellt wird und die Ausführungsdauern für die Planer abweichen (siehe auch Abbildung 5.3). Wie in den Diagrammen zu erkennen können beide Planer schnell die idealen Kosten bezüglich vollständiger Information erreichen. Der wegoptimierende Planer schöpft dabei jedoch die verfügbaren Kollisionsberechnungen nicht vollständig aus, da er kurze, direkte Wege zum Ziel wählt und nur auf diesen die Knoten testet, solange keine Kollision auftritt. Eine Kollision auf einer Kante wird erst kurz vor der Ausführung der Kante „entdeckt“ und führt dann zu zusätzlich notwendiger Umplanung und damit einem erneuten Anstieg der verwendeten Kollisionstests (etwa ab Sekunde 2,3). Der zeitoptimierende Planer hingegen betrachtet aufgrund der Größe des Objekts einen erheblich größeren Anteil des Konfigurationsraums, um die zeitoptimierende Bahn zu finden, welche auch ausladender um das Objekt herum führt (siehe Abbildung 5.1). Der zeitoptimierende Planer verwendet daher im Allgemeinen eine größere Anzahl an Distanzberechnungen, bevor für die folgenden Planungszyklen in statischen Konfigurationsräumen keine weiteren Tests mehr benötigt werden und auch hier die Anzahl verwendeter Distanzberechnungen auf Null fällt. Der erstmalige Rückweg (ab Sekunde 56 bzw. 13) erzeugt eine neue Verteilung der Kollisionstests, da sich das Ziel der Planung verändert hat und die Exploration durch den A* bzw. LPA in Rückrichtung eine andere ist. Da Start und Ziel einer Planung sich von Durchgang zu Durchgang wechseln sowie die verfügbaren Umweltinformationen sich

ändern, werden auch zu späteren Zeitpunkten weitere Distanzberechnungen bzw. Kollisionstests durch den Planer eingesetzt.

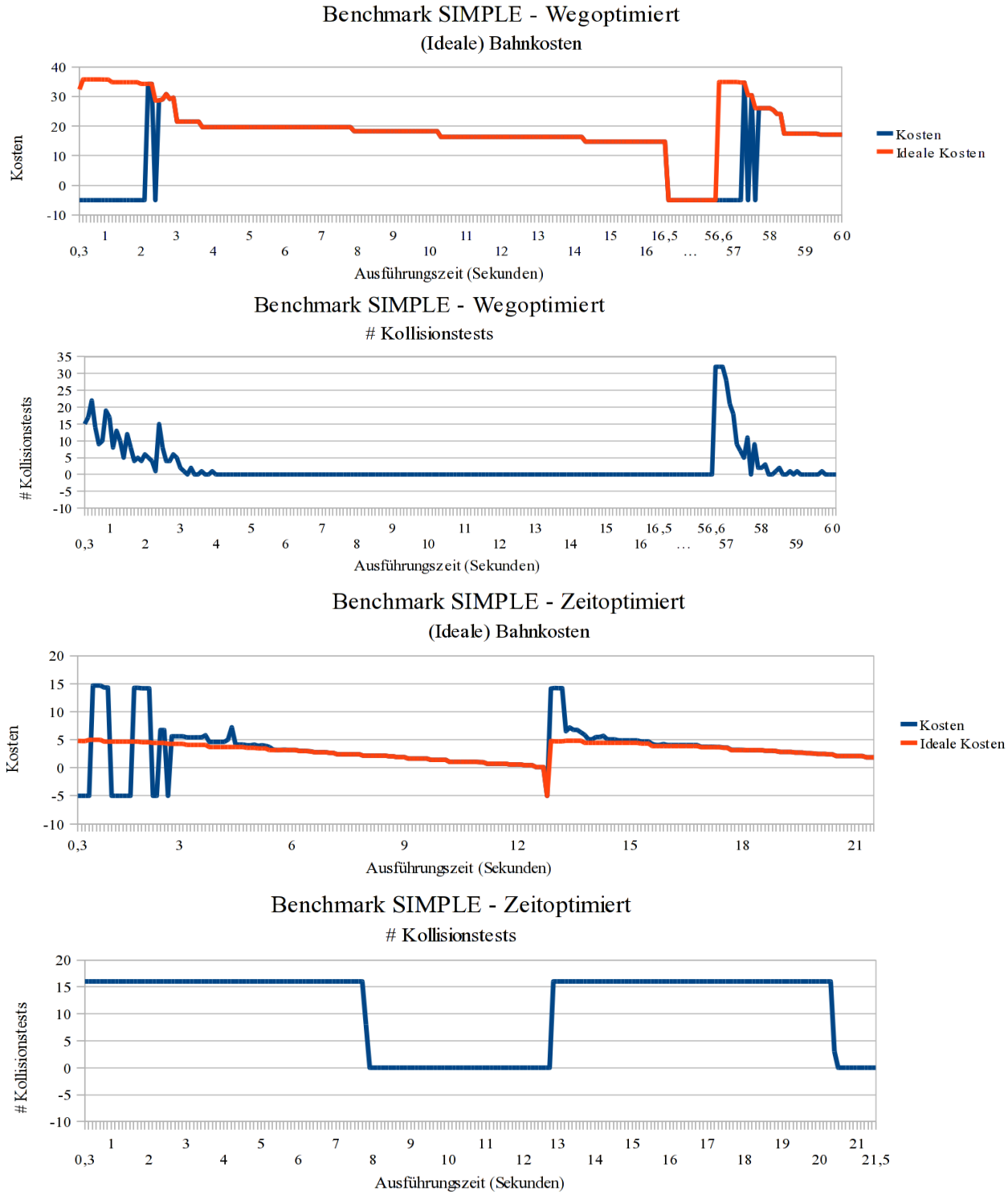


Abbildung 5.3: Kosten und verwendete Kollisionstests für den wegoptimierenden und zeitoptimierenden Planer für jeweils knapp zwei Durchgänge. Die Kosten sind jeweils für den begrenzten Planer berechnet („Kosten“) und für den unbegrenzten („Ideale Kosten“) vom jeweils aktuellen Punkt des Roboters bis zum Ziel. Wenn der Pfad des begrenzten Planers kollidiert wurden keine Kosten berechnet und die Kosten pauschal auf -5 gesetzt, um dies im Diagram gut erkennen zu können. Beim Übergang zu einem neuen Ziel werden beide Kosten auf -5 festgesetzt, um den Übergang zu markieren. Die Diagramme sind nicht exakt ausgerichtet.

Die verfahrene Bahn wechselt beim zweiten Durchgang in den Beispielen auf die andere Seite des Hindernisses (jeweils Bahn 2 in Abbildung 5.1 links und rechts), da während der Planung auf Bahn 1 dieser Teil des Konfigurationsraumes nicht so umfassend erfasst wurde, wie der Teil, der für die aktuelle Planung relevant ist. Aufgrund der optimistischen Grundannahmen (Kollisionsfreiheit für ungetestete Knoten) wird für die Folgeplanung die Bahn 2 gewählt, bis auch hier die vollständigen Informationen in den Knoten vorliegen.

Das Konfigurationsraumszenario DETOUR (Abbildung 5.4) steht hier prototypisch für komplexe Szenarien, bei denen der optimale Weg erst bei (nahezu) vollständiger Erfassung der gesamten Distanzinformationen in den Knoten berechnet werden kann. In solchen Situationen kann es passieren, dass der Planer zu Beginn suboptimale Wege wählt. Die Erfassung aller Knoten eines 4000-Knoten-Graphen dauert bei 16 getesteten Knoten pro Zyklus in diesem Experiment 250 Zyklen, also 25 Sekunden, wenn pro Zyklus alle Distanzberechnungen verbraucht werden. In diesem Beispiel ist die komplette Erfassung des Konfigurationsraumes daher nach etwa zwei Start-Ziel-Bewegungen erreicht, was sich auch in der Stabilität der Kostenverläufe in Diagram 5.5 zeigt. Für den wegoptimierende Planer ist aufgrund der fehlenden Distanzinformationen der Weg durch die engen Bereiche immer die beste Lösung. Wenn das Hindernis die Bahn in mehr Wendungen zwingen würde, würde auch der wegoptimierende Planer eine äußere Bahn wählen.

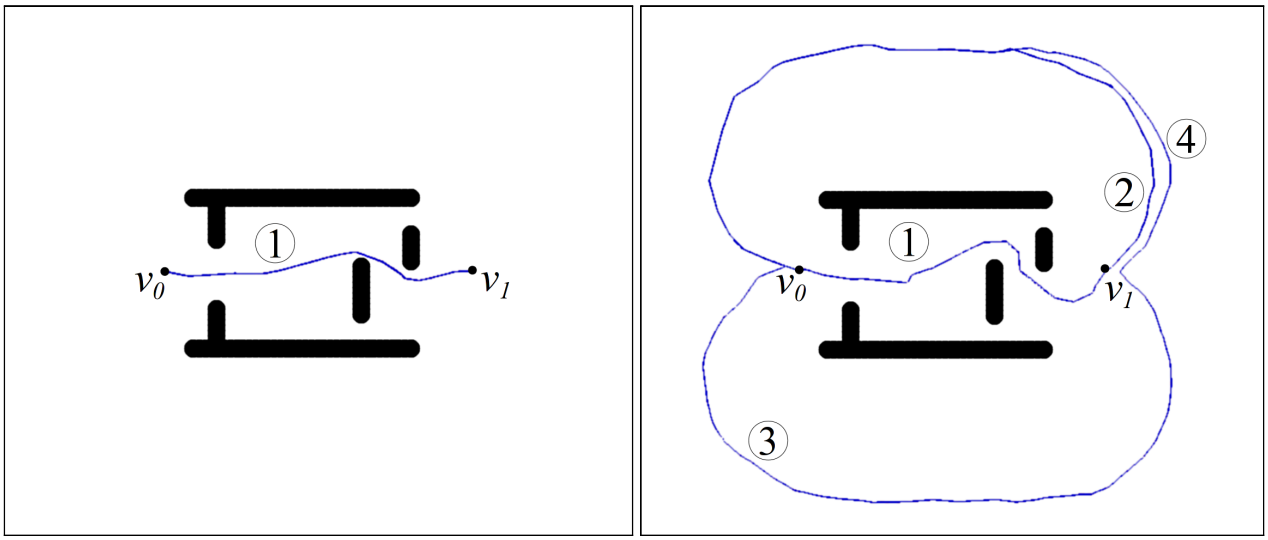


*Abbildung 5.4: Konfigurationsraumszenario DETOUR mit Planung durch den wegoptimierenden Planer (**links**) und den zeitoptimierenden Planer (**rechts**). Für den zeitoptimierenden Planer sind jeweils mehrere Start-Ziel-Bewegungen dargestellt (Markierungen 1-4), für den wegoptimierenden Planer sind die Bewegungen identisch für alle acht Durchgänge. Die Bahnen schließen aus simulationsmeßtechnischen Gründen nicht immer an die dargestellten Knoten an, da die Abtastung der verfahrenen Bahn hier nicht ausreichend war (die Abtastung fand einmal pro Zyklus statt).*

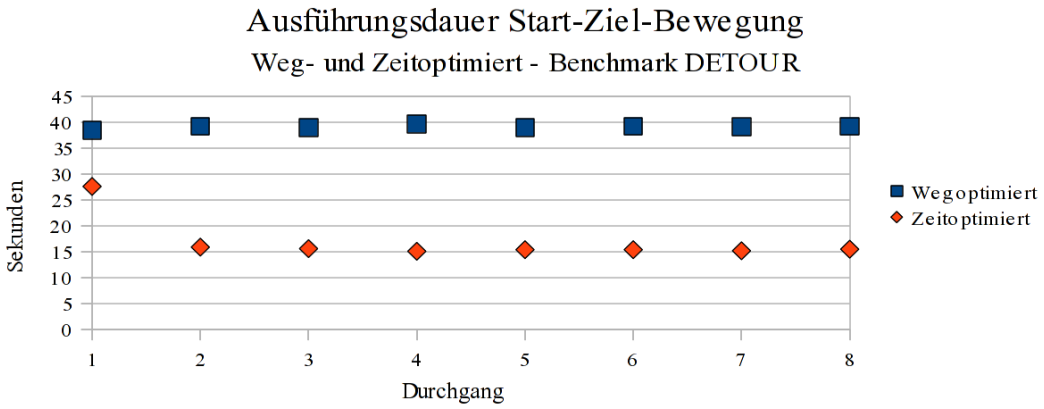
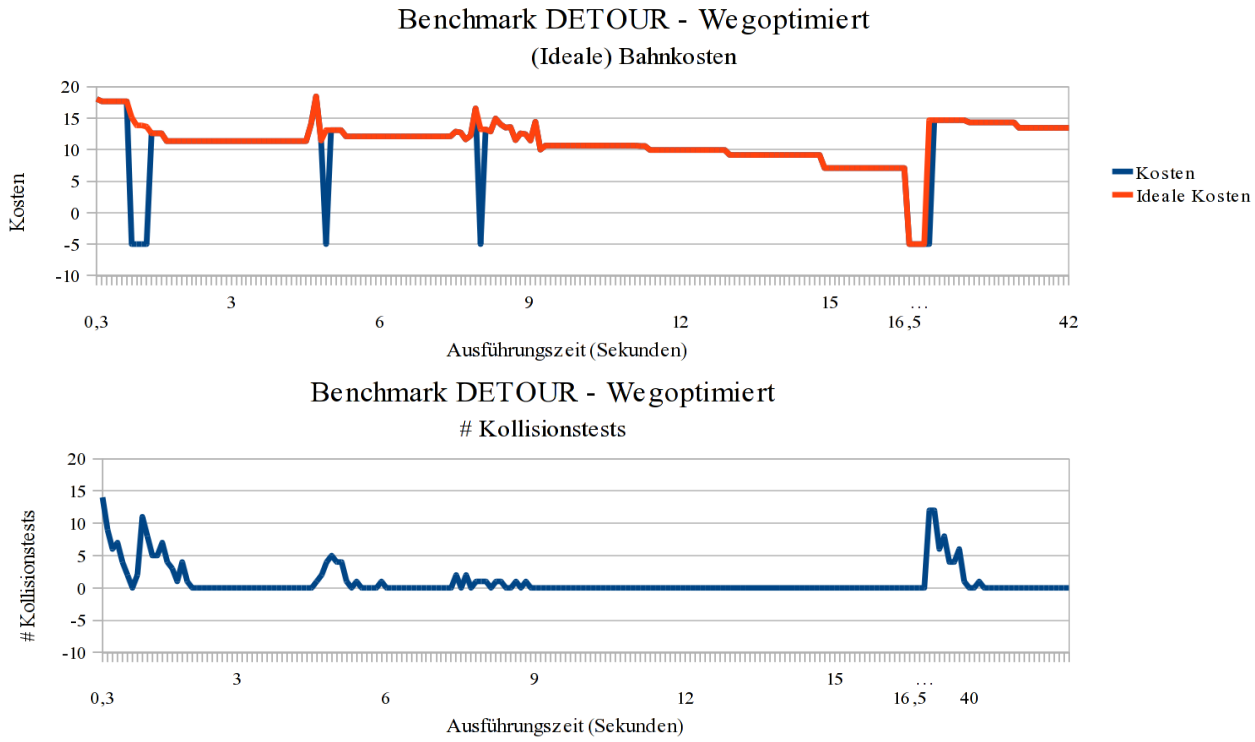


Abbildung 5.5: Ausführungszeiten in Sekunden im Benchmark DETOUR für Start-Ziel-Bewegungen zwischen den Knoten v_0 und v_1 für den weg- und zeitoptimierten Planer. Dargestellt sind acht Durchgänge (vier Schleifen mit jeweils Hin- und Rückweg).

Die in Abbildung 5.6 dargestellt Kosten für diesen Benchmark zeigen auch hier wieder, dass die idealen Kosten durch den wegoptimierenden Planer schneller erreicht werden (wenngleich diese aufgrund der Zielstellung der Optimierung auch höher liegen im Vergleich zum zeitoptimierenden Planer) und die Anzahl verwendeter Kollisionstests auch in diesem Beispiel nur dann wieder ansteigt, wenn Kollisionen entlang des Weges auf Kanten auftreten, wie oben beschrieben.



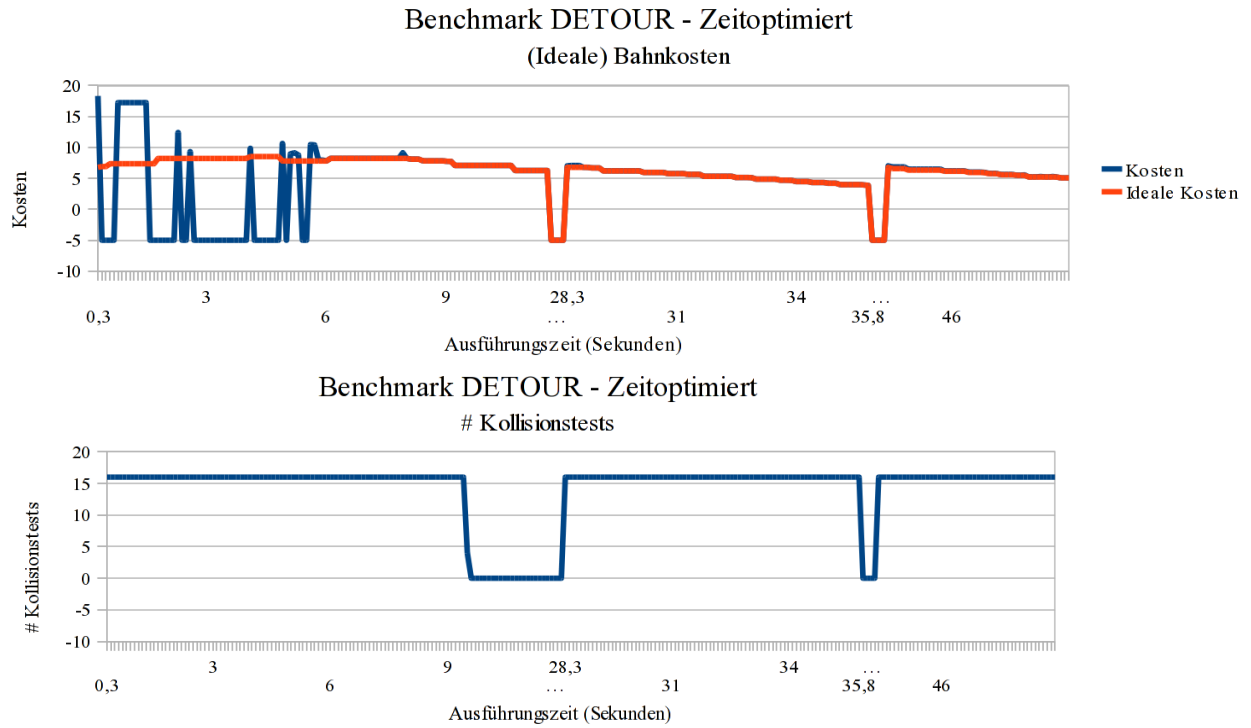


Abbildung 5.6: Kosten und verwendete Kollisionstests für den weg- und zeitoptimierenden Planer. Für den wegoptimierenden Planer ist etwas mehr als ein Durchgang dargestellt, für den zeitoptimierenden etwas mehr als zwei.

Abschließend kann festgehalten werden, dass eine statische Hindernissituation schnell durch den Planer erfasst werden kann, um den idealen Pfad zu bestimmen. Hierbei ist jedoch aufgrund der parallelen Ausführung und Planung damit zu rechnen, dass der Planer zunächst einen suboptimalen Weg findet bis die erfassten Informationen ausreichend sind. Da dies im Allgemeinen zu Beginn einer Start-Ziel-Bewegung geschieht, könnte hier ein verlängerter Planungszyklus Abhilfe schaffen, muss jedoch mit der Anforderung nach Reaktivität in dynamischen Räumen abgewogen werden.

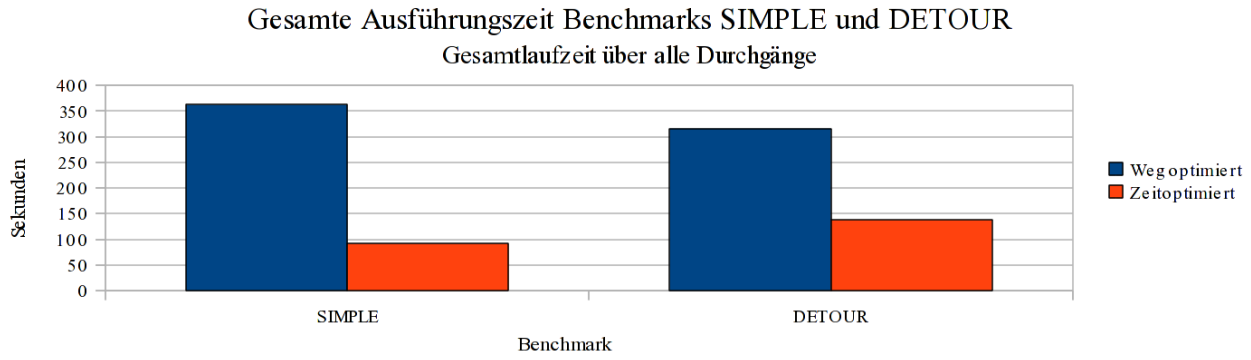


Abbildung 5.7: Gesamtlaufzeiten der Benchmarks SIMPLE und DETOUR für jeweils alle acht Durchgänge. Die Verbesserung der Laufzeiten des zeitoptimierenden Planers zum wegoptimierenden Planer liegen für SIMPLE bei 75 % und für DETOUR bei 56 %.

Konfigurationsraum mit dynamischen Hindernissen

Für dynamische Hindernisse im Konfigurationsraum gibt es außer ihrer geometrischen Komplexität noch weitere Parameter wie die Trajektorie (inkl. Geschwindigkeiten) und mögliche Verformung der Objekte. Hierbei können sehr komplexe Situationen entstehen, welche die Möglichkeit beinhalten, dass der Planer nie einen Weg zum Ziel findet. Ein einfaches Beispiel hierfür ist die bewegte Sackgasse (Benchmark „MOVING TRAP“, siehe Abbildung 5.8).

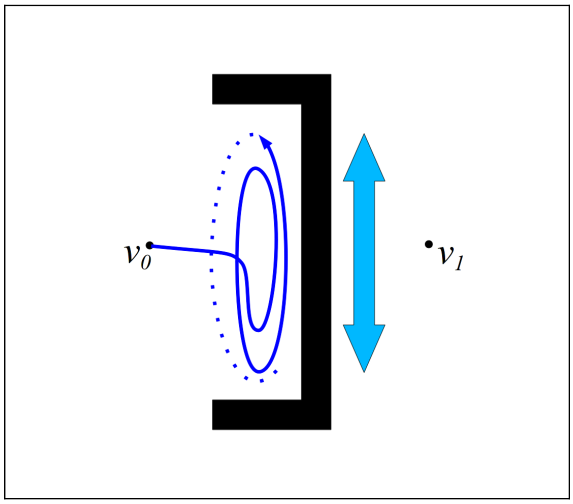


Abbildung 5.8: Pathologisches Konfigurationsraumszenario mit Sackgasse („MOVING TRAP“).

Auch ein idealer Planer mit der Information über den gesamten Arbeitsraum wird hier ein zyklisches Verhalten erzeugen, da mal die eine, mal die andere Richtung die günstigsten Bahn zum Ziel liefert. Als Lösungsansatz wäre eine (hier nicht betrachtete) Bewegungsvorhersage möglich und daraufhin die Planung im Zeit-Zustands-Raum („State-Time-Space“) oder die Annahme der dauerhaften Blockierung der Bewegungsbereiche des Objekts wie dies z.B. implizit durch die Revalidierungs-Strategien „Keine Revalidierung“ oder „Ziel-Revalidierung“ erreicht wird. Beim ersteren Ansatz ist Vorraussetzung, dass sich ein unbekanntes Objekt vorhersagbar verhält (was hier nicht angenommen wird), bei zweiten Ansatz kann es geschehen, dass keine Bahn zum Ziel existiert, wenn das Objekt durch seine Bewegung große Teile des Konfigurationsraums belegt. Diese Szenarien sind daher hoch problematisch, aber in der Praxis weniger relevant, da diese Szenarien einen nicht kooperativen Menschen beschreiben, der versucht, durch ständiges „In-den-Weg-Stellen“ den Roboter „in eine Ecke zu drängen“, bzw. vom Erreichen des Zieles abzuhalten. Eine adäquate Antwort auf ein solches Verhaltens ist jedoch prinzipiell nicht möglich und daher werden diese Szenarien im Folgenden nicht weiter betrachtet.

Im Benchmark MOVING OBJECT wurde ein bewegtes Objekt bzw. mehrere Objekte mit unterschiedlichen Bewegungsgeschwindigkeiten (ca. 4 bzw. 8% der *maximalen* Robotergeschwindigkeit, d.h. der Roboter kann dem Objekt schnell ausweichen) simuliert

und die Planer mit einer unterschiedlichen Anzahl an verfügbaren Kollisionstests ausgestattet. In Abbildung 5.9 sind für eine dieser Varianten die resultierenden Bahnen für acht Durchgänge dargestellt und die Bewegung eines Objekts gekennzeichnet, in Abbildung 5.11 analog für mehrere Objekte.

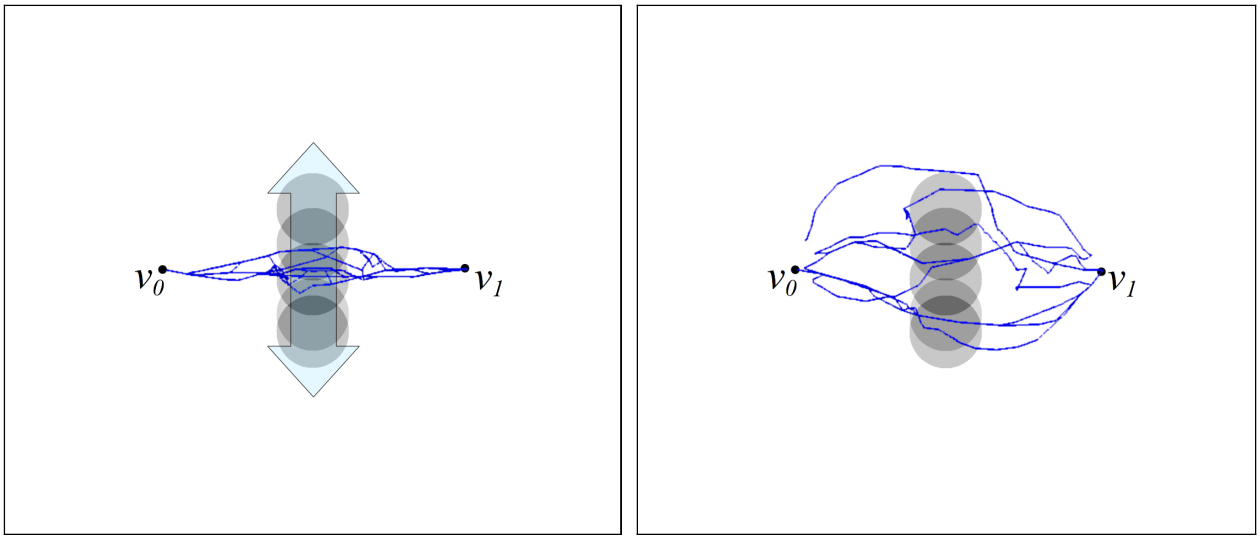


Abbildung 5.9: Konfigurationsraum mit dynamischem Objekt. ***Links****: Bahnen des wegoptimierenden Bahnplaners von v_0 nach v_1 für acht Durchgänge, Bewegung des Objekts angedeutet durch überlagerte Positionen und Doppelpfeilmarkierung.* ***Rechts****: Bahnen des zeitoptimierenden Planers. Beide Experimente wurden mit 32 Kollisionstests/Distanzberechnungen durchgeführt.*

In den Experimenten zeigen sich bei dynamischen Objekten Vorteile für die zeitoptimierende Bahnplanung, da hier der Bahnplaner bewegten Objekten weiträumiger ausweicht mit dementsprechend höheren Bewegungsgeschwindigkeiten. In Abbildung 5.9 ist dies gut zu erkennen. Die Bahnen des wegoptimierenden Planers führen hingegen so nah an das Hindernis, dass ein häufiger Stillstand aufgrund der Unterschreitung der Mindestdistanz durch die Bewegung des Objekts verursacht werden kann, was wiederum eine stark verlängerte Bewegungsdauer zur Folge haben kann, wenn die Blockierung durch das Objekt länger anhält (siehe hierzu auch die Diagramme in Abbildung 5.10). Dies macht sich insbesondere bei Objekten bemerkbar, die sich langsam bewegen, was bei den Gesamtlaufzeiten der Ausführung in den Diagrammen in Abbildung 5.12 zu erkennen ist: die Variante 4 mit einem langsamen Objekt führt zu erheblich verlängerten Ausführungszeiten beim wegoptimierenden Planer. Es können jedoch auch beim zeitoptimierenden Planer Blockaden auftreten (Abbildung 5.10 unten), wenn sich das Objekt so schnell bewegt, dass es den Roboter „einfängt“ oder wenn durch den Planer lange Kanten ausgewählt werden, deren jeweilige Endpunkte eine große Distanz zum Objekt aufweisen, während dies für den Verlauf der Kante nicht gilt. Vor allem bei optimistischer Kantenschätzung werden lange Kanten ausgewählt (siehe hierzu auch Kapitel 4.9 „Kantenselektion“). Hier kann alternativ der pessimistische Kantenschätzer gewählt werden oder unter Umständen Techniken, wie sie in Kapitel 4.6 im Ausblick beschrieben werden.

Je schneller sich ein Objekt bewegt, desto geringer werden die Unterschiede zwischen weg- und zeitoptimierender Planung, da die Ausführungsdauer mehr von den durch das Objekt verursachten Kollisionen bestimmt wird, als von den adäquaten Reaktionen des Planers.

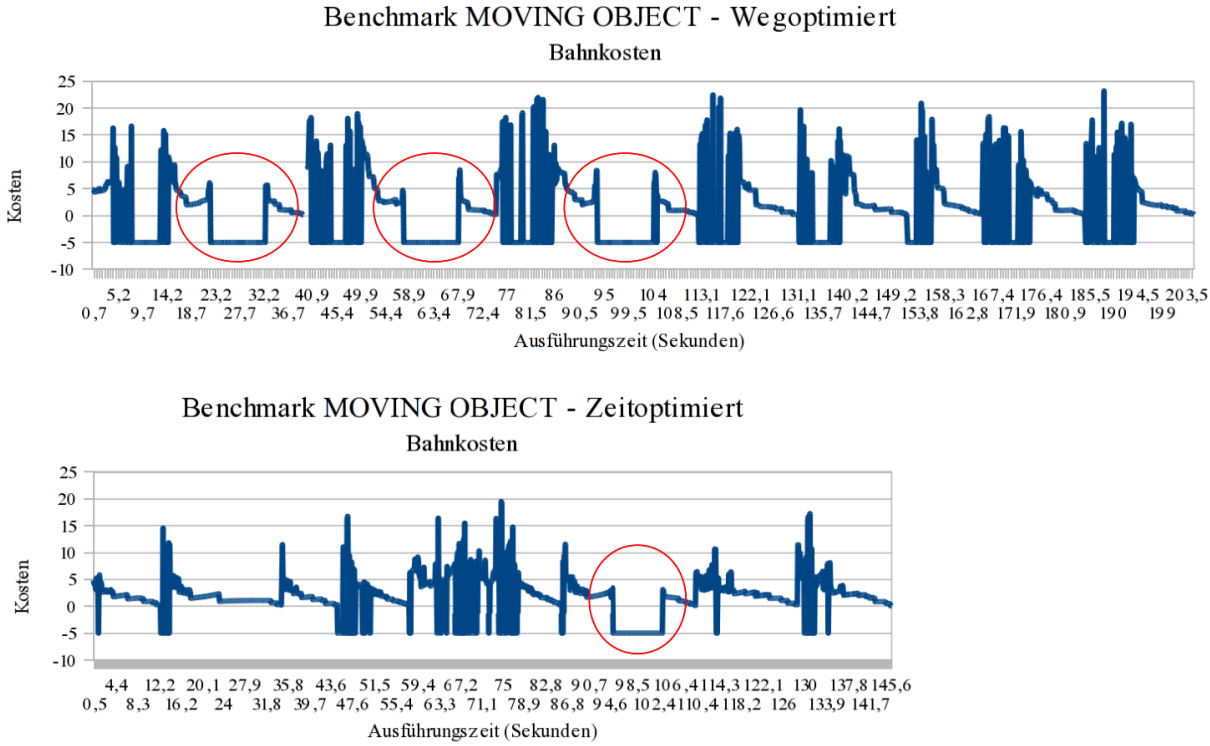


Abbildung 5.10: Bahnkosten Benchmark MOVING OBJECT mit einem Objekt. ***Oben****: Bahnkosten des wegoptimierenden Bahnplaners von v_0 nach v_1 für acht Durchgänge. Kollidierende Pfade werden mit Kosten „-5“ gekennzeichnet. Im Diagram rot eingekreist sind Blockierungen des Planers durch Kontakt mit dem bewegten Objekt, welche durch die Kollision der Bahn auch mit -5 gekennzeichnet ist.* ***Unten****: Kosten des zeitoptimierenden Planers. Hier ist eine geringere Anzahl von Blockaden zu beobachten. Die Diagramme sind ausgerichtet.*

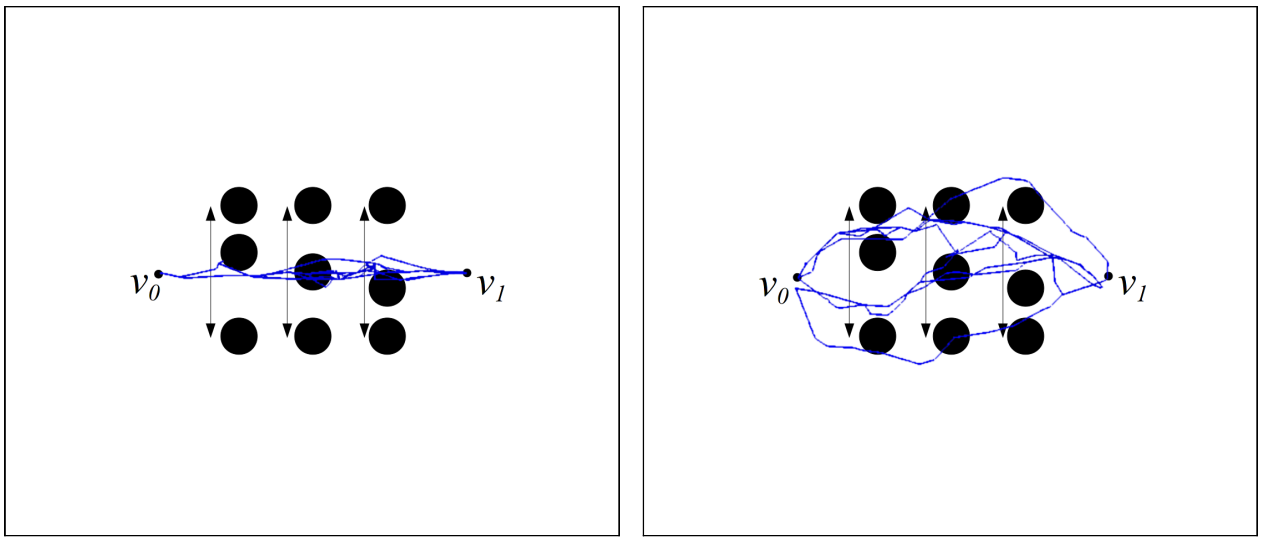


Abbildung 5.11: Konfigurationsraum mit drei dynamischen Objekten, welche horizontal angeordnet sind und sich vertikal bewegen, angedeutet durch die Doppelpfeile und jeweils drei mögliche Positionen. ***Links****: Bahnen des wegoptimierenden Bahnplaners von v_0 nach v_1 für acht Durchgänge.* ***Rechts****: Bahnen des zeitoptimierenden Planers. Beide Experimente wurden mit 32 Kollisionstests/Distanzberechnungen pro Zyklus durchgeführt.*

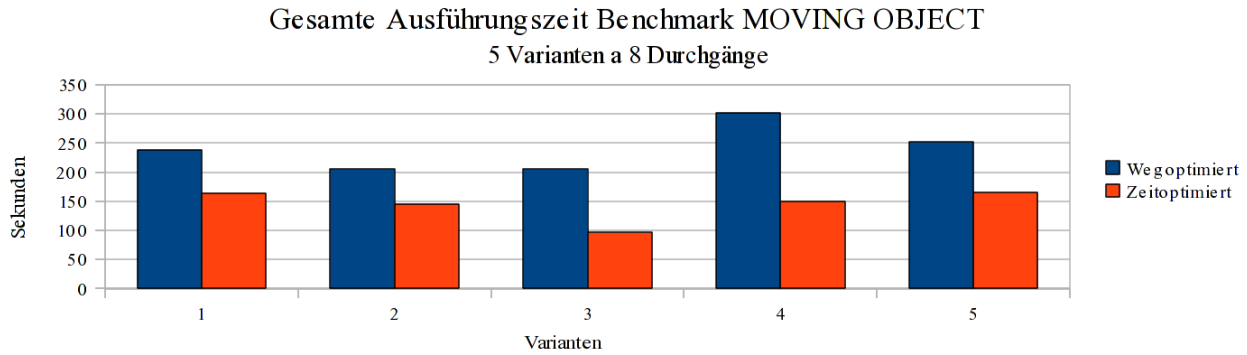


Abbildung 5.12: Gesamte Ausführungszeiten für acht Durchgänge der Bewegung von v_0 nach v_1 in Sekunden im Benchmark MOVING OBJECT. Die Parameter Kollisionstestanzahl (bzw. Anzahl Distanzberechnungen) und Objektanzahl/-geschwindigkeit werden variiert. Die Varianten 1 bis 3 umfassen ein Objekt (siehe Abbildung 5.9) und jeweils 16,32 und 64 Kollisionstests/Distanzberechnungen; die Variante 4 umfasst ein Objekt wie die Varianten 1 bis 3, jedoch bei halbierter Geschwindigkeit; die Variante 5 umfasst 3 Objekte ebenfalls bei halbierter Geschwindigkeit (siehe Abbildung 5.11).

Zusammenfassung

Insgesamt lässt sich durch den Einsatz des zeitoptimierenden Planers in den hier getesteten Szenarien von sowohl einfachen bzw. komplexen statischen Umgebungen als auch dynamischen Umgebungen eine deutliche Verbesserung der gesamten Ausführungzeit erreichen (bis ca. 50 %), wenn wie im Rahmen der Mensch-Roboter-Koexistenz/ Kooperation die Geschwindigkeit des Roboterarms basierend auf dem Abstand zum nächsten Hindernis geregelt wird. Wie im Diagramm in Abbildung 5.12 zu erkennen, profitiert hier der zeitoptimierende Planer von einer steigenden Anzahl an Distanzberechnung, während eine erhöhte mögliche Anzahl an Kollisionstests vom wegoptimierenden Planer nicht genutzt wird (siehe hierzu auch Abbildung 5.6), da ein kollisionsfreier Pfad mit einer geringeren Anzahl an Kollsionstests gefunden wird, als ein zeitoptimaler Pfad und da sich bei einem bewegten Objekt der zeitoptimierte Pfad global verschieben kann, während der wegoptimierte Pfad im Allgemeinen lediglich lokal verschoben wird.

5.2 Realwelt-Experimente

Im Rahmen des in Kapitel 1.5.2 beschriebenen Demonstratoraufbaus wurden verschiedene Tests des Gesamtsystems durchgeführt, welche alle vorgestellten Komponenten erfassen inklusive der nicht detaillierter dargestellten Differenzbild-berechnung. In realen Umgebungen kommt es aufgrund vielfältiger Einflüsse im Bereich der Bilderzeugung und -verarbeitung zu sehr unterschiedlichen, zum Teil temporären Fehlersituationen, welche die Leistung des Planers stark beeinflussen können und einen direkten Vergleich des Einflusses verschiedener Parameter erschwert. Die Effekte dieser Störungen und deren Behandlung wird in einer parallelen Promotionsprojekt am Lehrstuhl eingehender untersucht, um die reale Verfügbarkeit des Systems zu steigern. Neben der

Steigerung der Qualität der Bildverarbeitung kann hier auch die (rechenaufwändigere) Rekonstruktion des Arbeitsraumes Vorteile für die Planung bringen und wird daher ebenfalls in einer parallelen Promotionsprojekt untersucht.

In Abbildung 5.13 ist eine Sequenz einer realen Planungssituation zu sehen, in der die Blockierung des ursprünglichen Pfades eine Umplanung der Roboterbewegung erfordert. Die Planung erfolgt mit den vollen sechs Freiheitsgraden des Roboters in einem statischen Graphen von 5000 Knoten und ca. 20 Kanten pro Knoten, womit sich eine gute Abdeckung des Arbeitsraum des Roboters erreichen lässt. Es werden Zykluszeiten von unter 100 ms erreicht, was eine Framerate von 10-15 Hz ermöglicht. Die in den oben dargestellten Simulationsexperimenten beobachteten Effekte lassen sich auch in realen Umgebungen nachvollziehen, der Einsatz des wegoptimierenden Planers erhöht hier im Allgemeinen die Zykluszeiten und führt bei dynamischen Objekten zu häufigem Stillstand, wenn zu nahe an ein unbekanntes Objekt heran gefahren wird. Der zeitoptimierende Planer wählt frühzeitig eine Ausweichbewegung mit größerem Hindernisabstand und reduziert damit deutlich die Gefahr eines Stillstands und ermöglicht eine schnelle Umfahrung des unbekannten Objekts. Der vergrößerte Abstand erhöht zudem die gefühlte Sicherheit des menschlichen Kollegens trotz erhöhter Verfahrgeschwindigkeit.

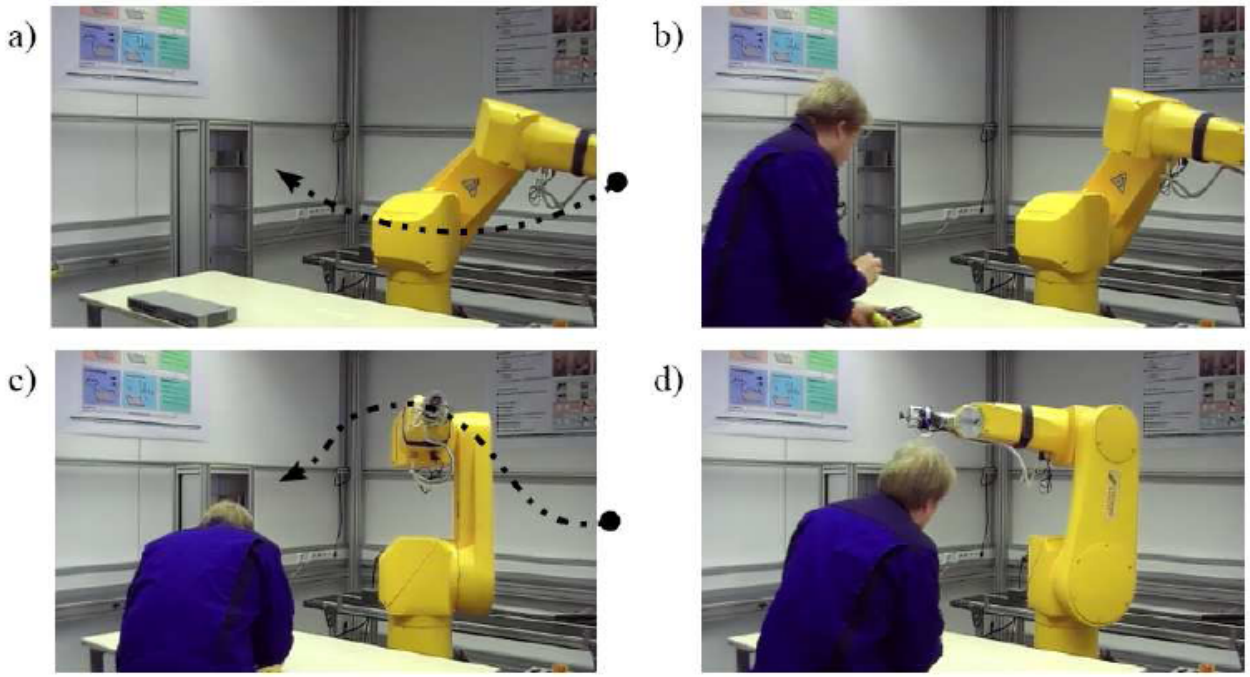


Abbildung 5.13: Auschnitt einer Start-Ziel-Bewegung eines Realwelt-Experiments mit zeitoptimierenden Bahnplaner. Der Industrieroboterarm verfährt mit einer maximalen Geschwindigkeit von ca. 750mm/s am TCP. Die Geschwindigkeitsregelung ist linear zum Hindernisabstand zwischen der Minimaldistanz d_{min} = 175 mm und d_{max} = 800mm. ***a)*** *Ausgangssituation mit leerer Zelle, die gestrichelte Linie deutet die Bahn an, die der Roboter ohne Hindernis wählen würde. Das Ziel ist die „Bearbeitungsstation" im Hintergrund.* ***b)*** *Ein dynamisches Hindernis betritt die Szene, die Reaktion des Roboterarm erfolgt sofort mit einem Abweichen von der ungestörten Bahn.* ***c)*** *und* ***d)*** *zeigen den weiteren Verlauf der Ausweichbewegung (angedeutet durch die gestrichelte Linie in Bild c)).*

Die Unterschiede zwischen zeit- und wegoptimierendem Planer wurden in einer Experimentreihe untersucht, in der das Szenario aus Abbildung 5.13 iteriert durchgeführt wurde und die jeweilige Zeit vom Startpunkt der Bewegung bis zum Abschluss aufgezeichnet wurde. Die Ergebnisse sind in einem Scatter-Plot in Abbildung 5.14 dargestellt. Durch die schwierige Reproduzierbarkeit realer Umgebungen schwanken die Zeiten zwar stark, zeigen aber dennoch eindeutige Unterschiede auf. Der zeitoptimierende Planer kann in diesem Szenario aufgrund des verfügbaren Freiraums deutliche Vorteile gegenüber dem wegoptimierendem Planer erreichen, indem durchweg höhere Verfahrgeschwindigkeiten erreicht werden. Der wegoptimierende Planer benötigt im Schnitt 50,3 Sekunden für die Bewegung von Start zu Ziel, der zeitoptimierende Planer 17,7 Sekunden. Die minimale Fahrtzeit für die untersuchte Bewegung beträgt ohne Hindernisse 6,8 Sekunden. Somit erreicht der zeitoptimierende Planer im Schnitt eine lediglich um das in etwa 2,5-fache gesteigerte Verfahrzeit trotz des nötigen Umweges. Der zeitoptimierende Planer verfährt in der Nähe des Hindernisses, da so die kürzeste Wegstrecke erreicht wird. Schon bei kleinen Bewegungen des Hindernisses kommt es daher häufig zur Unterschreitung des Mindestabstands und damit zum Stillstand des Roboters.

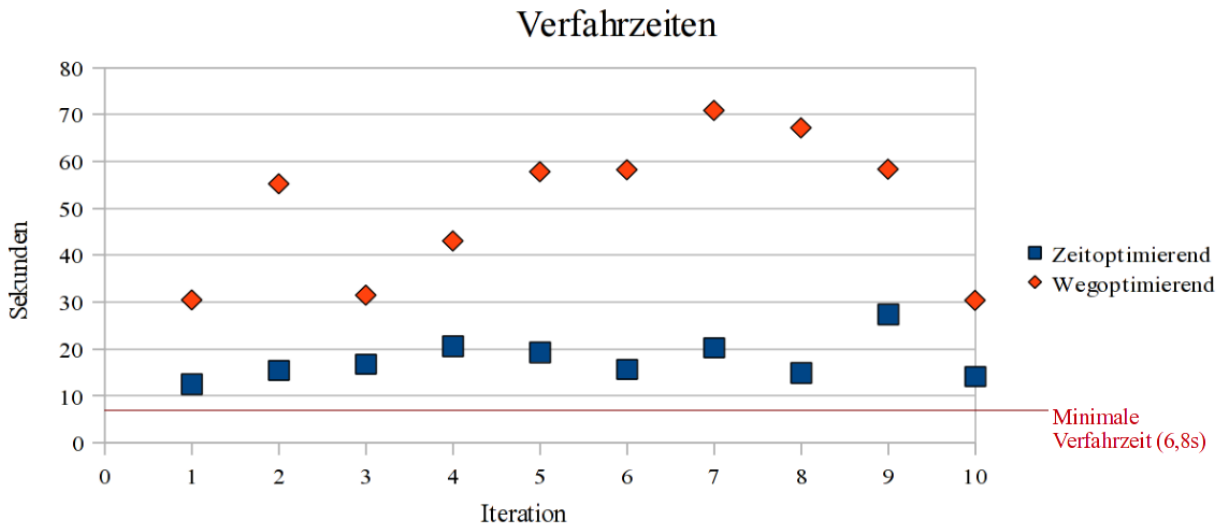


Abbildung 5.14: Verfahrzeiten des Roboters für eine Start-Ziel-Planungsaufgabe in einer realen Umgebung. Es wurden für jeden Planertyp 10 Versuche durchgeführt. Im untersuchten Szenario stört ein dynamisches Objekt (Mensch) die Bahn des Roboters (siehe Abbildung 5.13). Die minimale Verfahrzeit für die gegebene Bewegung (6,8s) ist durch die rote Linie markiert.

Weiterhin wurden während dieses Experiments die durch den Planer verbrauchte Anzahl an Kollisionstests bzw. Distanzmessungen aufgezeichnet. Maximal standen den Planervarianten pro Planungszyklus 64 Kollisionstests zur Verfügung. Abbildung 5.15 zeigt den Verlauf der Nutzung dieses Kontingents für beide Planer über die Dauer einer Start-Ziel-Bewegung.

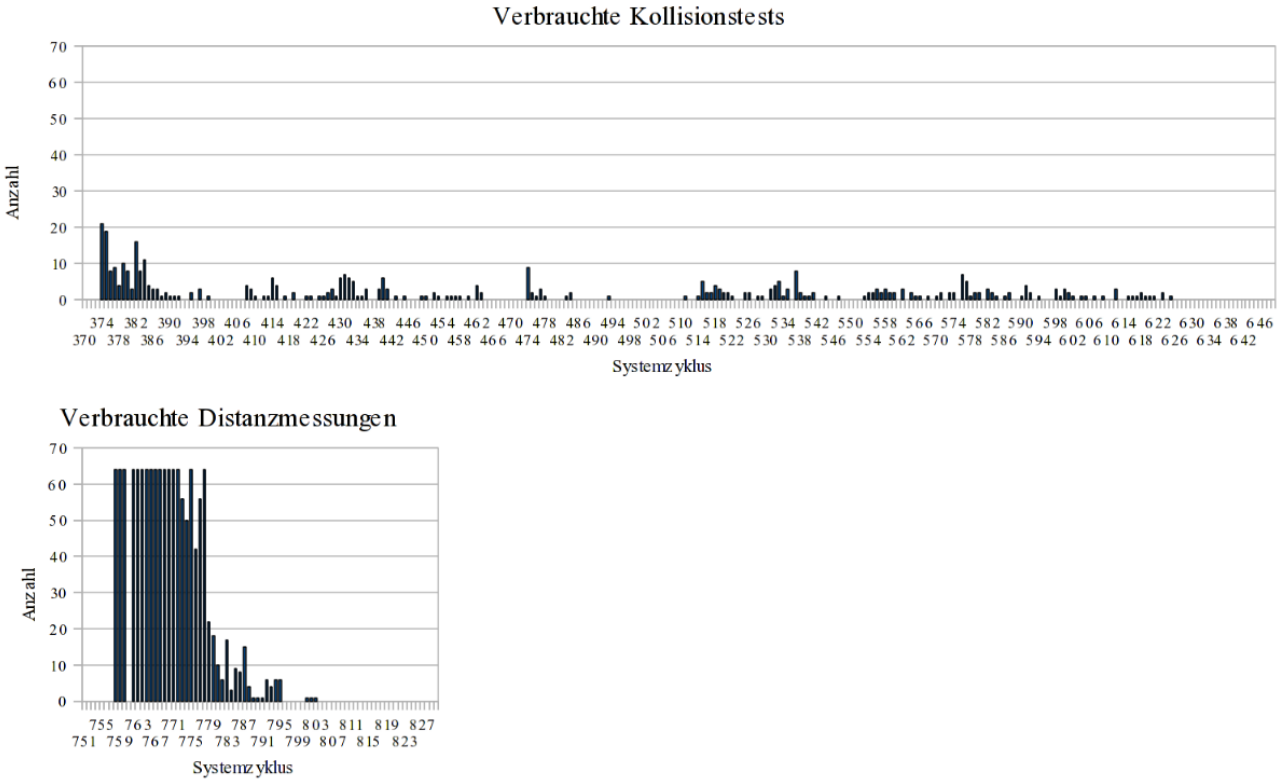


Abbildung 5.15: Ausschnitt des Verlaufs der verbrauchten Kollisionstests (im Falle des wegoptimierenden Planers) bzw. Distanzmessungen (im Falle des zeitoptimierenden Planers). Dargestellt ist der Verlauf für eine Start-Ziel-Bewegung des Experiments über eine Reihe von konsekutiv nummerierten Systemzyklen. Jeder Zyklus steht für 100 msec.

Ebenso wie in den Simulationsexperimenten (siehe Abbildungen 5.3 und 5.6) zeigt sich hier, dass der zeitoptimierende Planer mehr Informationen der Umgebung erfasst und im Allgemeinen die Anzahl der zur Verfügung stehenden Distanzmessungen voll ausschöpft.

5.3 Revalidierungsstrategien

Die in Kapitel 3.4 genannten Revalidierungsstrategien für invalidierte Kanten des Graphen wurden in einem Simulationsexperiment einem Vergleich bezüglich verschiedener Hindernisszenarien unterzogen, welche in Abbildung 5.16 dargestellt sind. Die gewählten Szenarien sollen unterschiedliche Eigenschaften der Revalidierungsstrategien heraus arbeiten. Die Konstruktion bestimmter Eigenschaften des Konfigurationsraumes ist in der zweidimensionalen Simulation leichter und wurde daher als Simulationsvariante ausgewählt.

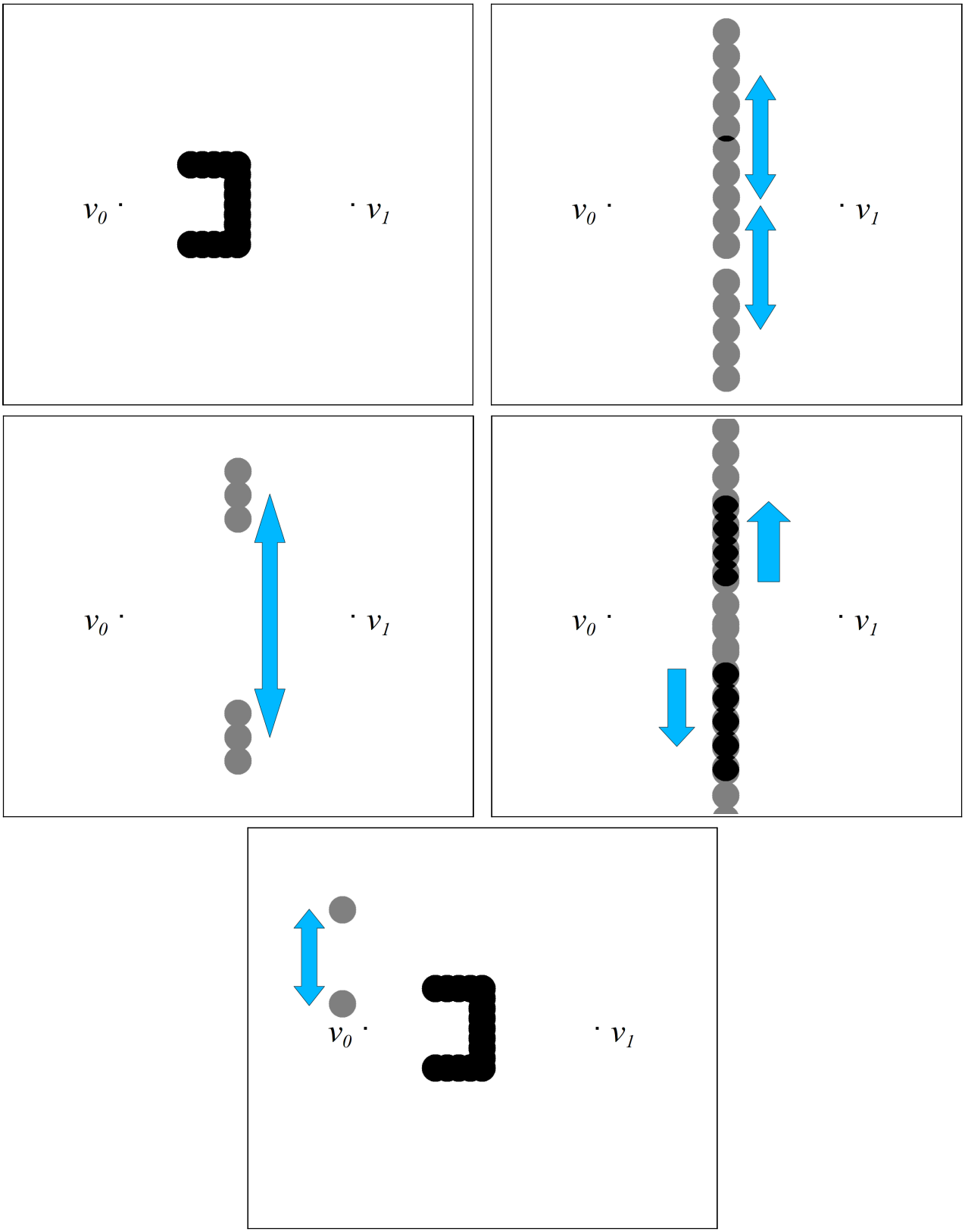


Abbildung 5.16: Simulationsszenarien zu den Revalidierungsstrategien. Die Planung startet in v_0 mit dem Ziel v_1 und wechselt nach Erreichen von Knoten v_1 das Ziel auf v_0. Dies wird in jedem Experiment dreimal iteriert. ***Oben links****: Statische Falle (TRAP), Szenario S.* ***Oben rechts****: zwei Objekte bewegen sich vom Rand des Konfigurationsraums zum Zentrum, den direkten Weg versperrend, Szenario M1.* ***Mitte links****: Ein Objekt bewegt sich zyklisch hin und her, Szenario M2.* ***Mitte rechts****: Zwei Objekte wandern von der Mitte zum Rand, den direkten Weg freigebend, Szenario M3.* ***Unten****: statische Falle mit bewegtem Objekt, das keinen Einfluss auf den kollisionsfreien Pfad zwischen v_0 und v_1 hat, Szenario SM.*

Verschiedene Messwerte wurden aufgezeichnet, darunter die durchschnittlich invalidierten Kanten I_{dst}, die verfahrene durchschnittliche Wegstrecke (im Konfigurationsraum in Grad) L_{dst} über alle Iterationen und die durchschnittlich für den Weg von Start zum Ziel verbrauchte Zeit T_{dst}. Die durchschnittlich invalidierten Kanten geben einen Hinweis auf den Freiraum im Konfigurationsraum, der dem Planer zur Verfügung steht. Je höher dieser Wert, desto stärker ist der Freiraum eingeschränkt. Die verfahrene Wegstrecke und die verbrauchte Zeit geben lediglich einen Hinweis auf die Qualität des gefundenen Weges bezüglich des Kriteriums Wegoptimierung. Die Werte können in dynamischen Szenarien stark schwanken, abhängig von den Entscheidungen, die der Planer basierend auf den in jedem Schritt zur Verfügung stehenden Informationen trifft. Diese Entscheidungen können positiv oder negativ sein abhängig davon, wie sich die Hindernisse bewegen, da deren Bewegung nicht vorhergesagt wird. Hierbei spielt auch eine Rolle, dass der wegoptimierende Planer Wege wählt, die nahe an den Hindernisobjekten vorbei führen und es daher durch die Bewegungen der Objekte oft geschieht, dass die Minimaldistanz unterschritten wird und der Roboter still steht.

		Szenario S	Szenario M1	Szenario M2	Szenario M3	Szenario SM
Keine Revalidierung	I_{dst}	2049,5	2558,1	1543,1	AkP	2049,8
	L_{dst}	180,2	279,9	211,1	AkP	180,2
	T_{dst}	49,7	28,5	13,4	AkP	49,9
Ziel-Revalidierung	I_{dst}	1641	1880,4	187,8	AkP	1695,3
	L_{dst}	202	334,2	151,2	AkP	200,1
	T_{dst}	56,9	50,7	12,3	AkP	59,7
Auszeit-Revalidierung	I_{dst}	AZ	248	222,5	471,2	AZ
	L_{dst}	AZ	117,8	153,1	153,4	AZ
	T_{dst}	AZ	35,7	12,2	15	AZ
Sensorbasierte Revalidierung	I_{dst}	2060	303,6	82,1	299,5	AZ
	L_{dst}	180,6	167,9	141,9	151,6	AZ
	T_{dst}	49,8	35,9	11,9	14,9	AZ

Tabelle 5.1: Versuchsergebnisse für die Kantenrevalidierungsstrategien 2D-Simulation (AkP = Abbruch, da kein Pfad gefunden wurde, AZ = Abbruch wegen zyklischen Verhaltens, welches das Erreichen des Zieles unmöglich macht, solange das das Verhalten erzeugende Hindernisobjekt an derselben Stelle verharrt)

Beim Blick auf die Ergebnisse in Tabelle 5.1 ist zunächst im wesentlichen auffällig, wo eine Revalidierungsstrategie komplett versagt. Dies ist der Fall, wenn der Planer keinen Pfad zum Ziel findet (AkP) oder in ein zyklisches Verhalten (AZ) verfällt, welches im Effekt ebenso dazu führt, dass das Ziel nicht erreicht wird. Im Falle AkP finden die Planer keinen Weg zum Ziel, obwohl dieser existiert, da das Hinzufügen neuer Punkte ausgeschaltet wurde. Hierdurch sollen bestimmte Effekte der Revalidierungsstrategien herausgearbeitet werden, die für die Praxis relevant sind. Denn das dann nötige Hinzufügen von Punkten zum Erreichen des Zieles ist eine aufwendige Operation, die unter Umständen eine lange Laufzeit hat, wenn viele Punkte hinzugefügt werden müssen, bis das Ziel erreicht wird. Dies sollte daher vermieden werden und eine Revalidierungsstrategie gewählt werden, die dies vermeidet.

Die Strategien „Keine Revalidierung“ und „Ziel-Revalidierung“ führen im Szenario M3 dazu, dass kein Weg zum Ziel gefunden wird, da alle Kanten belegt sind, an denen sich die Hindernisobjekte (zeitweise) aufhalten. Prinzipiell kann dies in dynamischen Umgebungen bei diesen beiden Strategien immer auftreten, wenn die Hindernisobjekte alle Kanten des Graphen im Konfigurationsraum zeitweise berühren. Die Strategie Ziel-Revalidierung reduziert dieses Problem insofern etwas, als dass durch die Revalidierung aller Kanten bei Erreichen des Ziels keine monoton steigende Anzahl invalidierter Kanten auftritt, wie dies bei der Strategie „Keine Revalidierung“ der Fall ist.

Das zyklische Verhalten der Strategien „Auszeit-Revalidierung“ und „sensorbasierte Revalidierung“ kommt daher, dass Kanten revalidiert werden, welche dauerhaft nicht kollisionsfrei sind aufgrund der assoziierten statischen Objekte und dass diese Kanten wiederholt durch den Planer für das Erreichen des Ziels ausgewählt werden. Das hier gewählte Beispiel der statischen Falle (TRAP) ist zur Erzeugung dieses Verhaltens gut geeignet. Bei der Strategie „Auszeit-Revalidierung“ ist hierbei die Zeitdauer entscheidend, für die die Invalidierung einer Kante aufrecht erhalten wird (die sogenannte Auszeit). Wenn diese ausreichend verlängert wird, so kann der Roboter schließlich eine beliebig komplexe Falle überwinden, da keine Kante revalidiert wird, die mit dieser Falle assoziiert ist, bevor der Roboter die Falle verlassen hat. Diese Verlängerung der Auszeit führt allerdings zu einer Annäherung an das Verhalten der Strategien „Keine Revalidierung“ und „Ziel-Revalidierung“ mit den entsprechend verbundenen Nachteilen.

Die Strategie „Sensorbasierte Revalidierung“ zeigt diesen Nachteil für statische Fallen nicht, da sie die Bewegung der Hindernisobjekte abschätzt und dementsprechend die Revalidierung der Kanten wählt. In statischen Situationen werden somit keine Revalidierungen vorgenommen. Es können jedoch Situationen konstruiert werden, in denen eine statische Falle mit einem dynamischen Objekt kombiniert für diese Strategie ebenfalls zu einem zyklischen Verhalten führen (Szenario SM).

Bei der Strategie „Keine Revalidierung“ fällt bezüglich der mittleren Zeitdauer T_{dst} auf, dass diese Strategie für das Szenario M1 die kürzeste Zeitdauer liefert. Dies liegt in diesem Szenario daran, das der Planer den Bereich der sich bewegenden Objekte in der Mitte des Konfigurationsraums nahezu vollständig abtastet und mit invalidierten Kanten belegt und daher immer Wege wählt, welche an dieser belegten Zone vorbei führen. Dadurch wird im Durchschnitt in hoher Hindernisabstand gewährleistet, welcher sich günstig auf die Zeitdauer T_{dst} auswirkt, da eine hohe Robotergeschwindigkeit erreicht wird. Diese Strategie ist daher von Vorteil, wenn sich die Dynamik der Objekte auf einen bestimmten Bereich des Konfigurationsraumes beschränkt, welcher das Erreichen des Ziels zulässt und wenn die Objekte eine hohe Aufenthaltswahrscheinlichkeit an diesen Stellen haben. Gilt dies nicht, so entstehen durch lange Verfahrwege Nachteile, wenn z.B. der Konfigurationsraum hindernisfrei ist (hierzu wurde kein spezifisches Szenario entworfen).

Insgesamt ist die Strategie „Sensorbasierte Revalidierung“ zu bevorzugen, da sie in dynamischen Umgebungen die komplette Blockade des Ziels vermeidet (wenn auch nicht verhindert) und in statischen Umgebungen kein zyklisches Verhalten aufweist wie die Strategie „Auszeit-Revalidierung“. Die Kombination aus statischer Falle und dynamischem Hindernis wird hierbei als eher selten auftretend angenommen. Die sensorbasierte Revalidierung kann eine Blockade des Ziels nicht verhindern (obwohl ein Pfad existiert), da sie nicht exakt die Kanten revalidiert, für welche dies möglich wäre, da sie aufgrund der Objektbewegung kollisionsfrei geworden sind.

5.4 Platzierung der Distanzberechnungen für den zeitoptimierenden Planer

Wie in Kapitel 4.8 auf Seite 171 beschrieben, gibt es Varianten, die Platzierung der Distanzberechnungen während der Graphensuche zu streuen, d.h. nicht die unberechneten Knoten lediglich der Reihenfolge nach zu berechnen, wie sie aus der OPEN-Liste herausfallen. Dort wurden drei weitere Möglichkeiten dargestellt, mittels derer die unberechneten Knoten einer weiteren Auswahl unterzogen werden.

Die Untersuchungen hierzu wurden in einem speziellen Szenario in der 2D-Simulation durchgeführt (siehe Abbildung 5.17). Dieses Szenario soll den Planer dazu zwingen, einen Bereich mit bewegten Hindernissen zu queren. Dazu dienen die statischen Hindernisse über und unter dem Bereich der bewegten Hindernisse. Die vier Platzierungsvarianten wurden dann für jeweils 3, 10, 20 und 40 Iterationen zwischen den Knoten v_0 und v_1 im Planer verwendet und die Verfahrzeit aufgezeichnet, welche die Zeit umfasst, die benötigt wird, um alle Iterationen der Planung auszuführen.

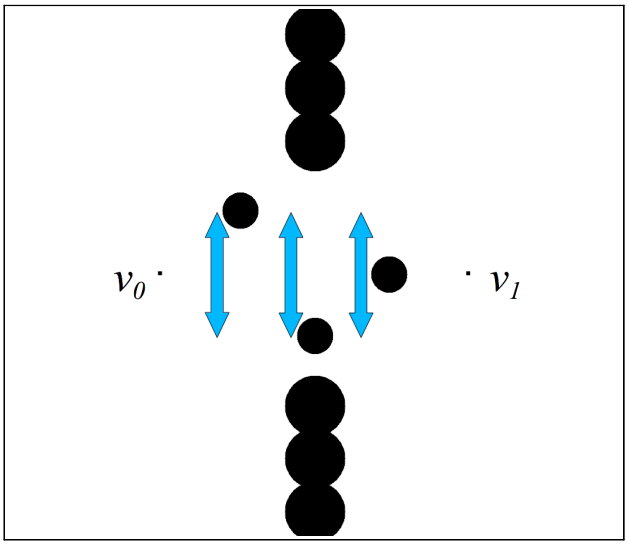


Abbildung 5.17: Simulationsszenario zur Platzierung der Distanzberechnungen. Die Planung startet in v_0 mit dem Ziel v_1 und wechselt nach Erreichen von Knoten v_1 das Ziel auf v_0. Dies wird in jedem Experiment mehrfach iteriert (3,10,20,40 Iterationen). Zwischen den Zielknoten ist ein Feld bewegter Objekte angeordnet, welches oben und unten durch statische Objekte gerahmt wird.

In den Diagrammen in Abbildung 5.18 sind die Ergebnisse der Experimente dargestellt. Die Varianten umfassen: A = Basisvariante (Jeder expandierte Knoten wird berechnet), B = Zufalls-Selektor (5 % der expandierten Knoten werden berechnet), C = Intervallbetrag-Selektor und D = Kostenorientierter Intervallbetrag-Selektor (s. S. 171ff). Die Verfahrzeit im linken Diagramm ist dabei auf die jeweils längste Verfahrzeit normiert, da sonst die Darstellung aufgrund der unterschiedlichen Iterationsanzahlen unübersichtlich geworden wäre. Im rechten Diagramm ist der durchschnittliche Anteil an verwendeten Distanzberechnung dargestellt, anteilig an den maximal Verwendbaren pro Zyklus.

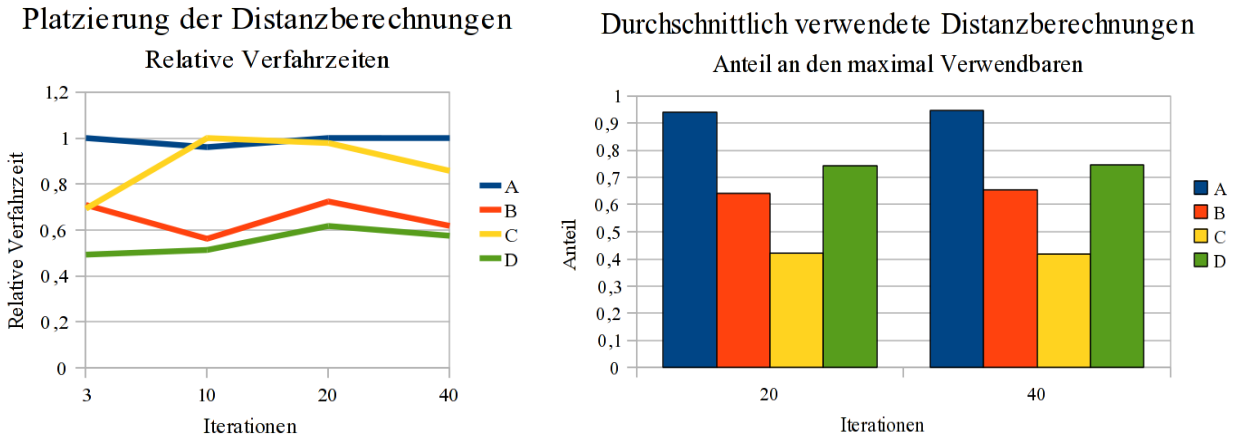


*Abbildung 5.18: Messwertdiagramme der Simulation zur Platzierung der Distanzberechnungen. **Links**: Relative Verfahrzeiten der Planung mit unterschiedlichen Platzierungsvarianten (A,B,C,D), normiert auf die jeweils längste Verfahrzeit. **Rechts**: Für die Simulationen mit 20 und 40 Iterationen wurde aufgezeichnet, welcher Anteil der maximal möglichen Distanzmessungen pro Planungszyklus in der jeweiligen Variante verwendet wurden, im Durchschnitt über alle Zyklen.*

Die Basisvariante A hat in diesem Szenario nahezu immer die schlechteste Laufzeit,

da sie die Distanzberechnung nahe an der aktuellen Roboterposition konzentriert (dies ist natürlich auch von der lokalen Knotendichte des Graphen abhängig) und somit die Entscheidungen des Planers auf „sehr“ lokalen Informationen basieren. Die Verfahrzeiten dieser Variante sind nahezu doppelt so lange wie die der besten Variante D.

Ähnlich schlechte Verfahrzeiten wie Variante A hat Variante C, bei der die Distanzberechnungen auf Knoten platziert werden, deren Distanzintervall einen gewissen Betrag überschreitet. Hierdurch werden die Distanzberechnung sehr verteilt platziert und nicht auf den zuletzt berechneten Knoten, da deren Intervall durch Anpassungen aufgrund von Objektbewegungen erst wieder anwachsen muss, bis es erneut zur Berechnung ausgewählt wird. Somit finden häufig in der Nähe der aktuellen Roboterposition keine Berechnungen statt, was zu lokalen Fehlentscheidungen aufgrund mangelnder Information führt. Auch ist der durchschnittlich verwendete Anteil an Distanzberechnung (Abb. 5.18 rechts) bei dieser Methode am geringsten. Dieser Wert alleine lässt noch keine Aussagen auf die Leistung der jeweiligen Variante bezüglich der Verfahrzeiten zu, da er für die Variante A maximal ist, jedoch gibt er einen Hinweis darauf, dass zu wenige Distanzberechnungen eingesetzt werden. Das Verhalten ist hier über die Wahl des Betragsschwellwertes steuerbar, nähert sich jedoch bei einem sehr kleinem Wert der Basisvariante an.

Die besten Laufzeiten erzeugen die Varianten B und D, welche Distanzberechnungen sowohl in der Nähe und mit abnehmender Dichte entfernt von der aktuellen Roboterkonfiguration platzieren. Diese sind daher in der Praxis vorzuziehen, zumal sie lediglich einen vernachlässigbaren zusätzlichen Rechenaufwand im Vergleich zur Basisvariante erfordern.

6 Zusammenfassung und Ausblick

Zusammenfassung

Die Zusammenarbeit von Mensch und (Industrie-)Roboter ist ein wichtiges Ziel der aktuellen Forschung in der Robotik und möchte die jeweiligen Vorteile von Mensch (Adaptivität, Problemlösungsfähigkeiten, hochentwickelte Sensorik, etc.) und Roboter (Kraft, Ausdauer, Präzision, Wiederholgenauigkeit, etc.) in einem gemeinsamen Prozess nutzen. Den vielen existierenden Ansätzen gemeinsam ist die Beobachtung, dass der Roboter eine Gefährdung für den Menschen darstellt, insbesondere dann, wenn große Kräfte und gefährliche Werkstücke im Spiel sind. Um die Zusammenarbeit dennoch zu ermöglichen sind Sicherheitskonzepte nötig, die den gemeinsamen, gleichzeitigen Aufenthalt von Mensch und Roboter in einem Arbeitsraum so realisieren, dass einerseits die Sicherheit des Menschen gewährleistet ist und andererseits eine hohe Verfügbarkeit des Robotersystems besteht. Die Forderung nach hoher Verfügbarkeit ergibt sich aus der notwendigen Wirtschaftlichkeit für die breite Akzeptanz. Diese Arbeit präsentiert einen Lösungsansatz zu dieser Aufgabenstellung mit dem Schwerpunkt auf der Verfügbarkeit durch eine echtzeitfähige weg- und zeitoptimierende Bahnplanung zum Umgang mit Blockaden durch dynamische Hindernisse, wie beispielsweise dem Menschen. Der Sicherheitsaspekt wird durch die dreidimensionale Detektion des Menschen oder anderer Hindernisse addressiert.

Die Detektion des Menschen und beliebiger anderer a priori *unbekannter Objekte* wird hier mit Hilfe eines Kamerasystems realisiert, welches eine große Menge von Informationen bezüglich des gemeinsamen Arbeitsraums mit geringer Latenz liefern kann. Der Arbeitsraum wird dabei von einer Anzahl stationären Kameras beobachtet. Die stationäre Befestigung ermöglicht den Einsatz effizienter „Change Detection“-Verfahren (z.B. Differenzbildverfahren) und die Kalibrierung der Kameras. Die Kalibrierung und die unterschiedlichen Perspektiven der Kameras werden in einer Fusion der Informationen der einzelnen Kamerabilder verwendet, um einen 3D-Kollisionstest bzw. eine 3D-Distanzberechnung zwischen unbekannten Objekten (z.B. der Mensch) und Roboter zu realisieren. Mit Hilfe von Modellen bekannter Objekte (Roboter und Zellenaufbauten) können Verdeckungen unbekannter Objekte im Kollisionstest und der Abstandsinformation

mit einberechnet und korrekt berücksichtigt werden. Die Berechnungen finden hierbei vollständig und recheneffizient im Bildraum statt. Der Kollisionstest und die Distanzberechnung von Mensch zu Roboter können aufgrund eines Robotermodells für beliebige Positionen des Roboters berechnet werden, was für die Bahnplanung benötigt wird.

Die Bahnplanung soll zur Erhöhung der Verfügbarkeit eine Ausweichbewegung des Roboterarms erzeugen, wenn der Mensch die vorgesehene Roboterbahn blockiert. Ein echtzeitfähiges Verhalten ist gefordert, um die Reaktion frühzeitig zu ermöglichen und um auf die Dynamik des Arbeitsraumes reagieren zu können. Gleichzeitig soll der Planer im Sinne der Verfügbarkeit einen Weg zum Ziel finden, falls ein solcher existiert (Vollständigkeit). Das in dieser Arbeit entwickelten Konzept erreicht diese Ziele für eine wegoptimierende und eine zeitoptimierende Variante des Planers.

Die echtzeitfähige Reaktion wird dadurch erreicht, dass der Bahnplaner in jedem Zyklus des Systems basierend auf den jeweils aktuellen Umweltinformationen ausgeführt wird und durch verschiedene Maßnahmen in seiner Laufzeit begrenzt wird. Das entwickelte Bahnplanerkonzept verbindet eine globale Bahnplanung auf randomisierten Graphen mit der Begrenzung von teuren Kollisionstests oder Distanzberechnungen, um statistische Vollständigkeit mit einer hohen Anzahl von Roboter-Freiheitsgraden und kurze Laufzeiten zu erreichen. Die geometrische Information über bekannte statische Objekte (z.B. Zellenaufbauten) werden verwendet, um einen randomisierten Graph offline zu erzeugen, welcher die Kollisionsfreiheit des Roboters mit diesen Objekten berücksichtigt, sodass zur Laufzeit lediglich die Kollision mit sensorisch erfassten Objekten berücksichtigt werden muss. Durch die Begrenzung der Kollisionstests bzw. Distanzberechnungen muss auf Teilen des Graphen geplant werden, für die keine exakten Informationen über die Umwelt vorliegen. Die Informationen über die nicht berechneten Teile des Graphen werden daher durch Schätzungskonzepte erzeugt. Zusätzlich werden die durch den Kollisionstest bzw. die Distanzberechnung erlangten Informationen über die Umwelt durch Speicherung im Graph von Systemzyklus zu Systemzyklus beibehalten, um die Informationen über die Umwelt sukzessive auszuweiten und somit die statistische Vollständigkeit zu erreichen. Diese Informationspersistenz wird dabei an die Arbeitsraumdynamik adaptiert, indem Informationen beibehalten werden, wenn die Arbeitsraumdynamik niedrig ist und verworfen bzw. angepasst, wenn die Dynamik hoch ist. Die Dynamik wird dabei global aus den Bildinformationen ermittelt.

Basierend auf dem Kollisionstest wurde obiges Konzept für die Realisierung eines wegoptimierenden Planers verwendet und basierend auf der Distanzberechnung wurde ein entsprechend angepasstes Konzept für einen zeitoptimierenden Planer untersucht. Die Zeitoptimierung muss dabei berücksichtigen, dass die Robotergeschwindigkeit aus

Sicherheitsgründen basierend auf dem Abstand zum nächsten Hindernis geregelt wird. Hierbei werden durch den Planer Streckenlänge und Hindernisabstand miteinander verrechnet. Beide Planer wurden durch Experimente in realen und simulierten Umgebungen getestet. Vor allem die durch den zeitoptimierenden Planer erzeugten Bahnen bieten eine hohe Ausführungsgeschwindigkeit durch optimierte Hindernisabstände und größere Ergonomie durch die Erhöhung der gefühlten Sicherheit bei großem Abstand von Mensch und Roboter. Dadurch ist das Konzept insbesondere im Rahmen der Mensch-Roboter-Koexistenz/-Kooperation gut einsetzbar und verbindet hohe Reaktivität mit aus Verfügbarkeitsgründen wichtiger Vollständigkeit.

Ausblick

Das hier vorgestellte System realisiert die Koexistenz von Mensch und Roboter unter besonderer Berücksichtigung der Verfügbarkeit. Für die weitere Entwicklung wäre eine detailliertere Betrachtung der Möglichkeiten der Kooperation zwischen Mensch und Roboter sinnvoll. Hierbei ist zu beachten, dass der effiziente, aber einfache Differenzbildansatz, welcher hier für die Objekterkennung verwendet wurde, zunächst darauf ausgelegt ist, unabhängige Arbeiten im gleichen Raum zu unterstützen, also das, was als Koexistenz bezeichnet wird. Dies liegt daran, dass das erzeugte Referenzmodell von einem festen zyklischen Prozess des Roboters ausgeht, für den die erwarteten Bilder des Arbeitsraumes bekannt sind. Während einer Kooperation von Mensch und Roboter können jedoch unvorhergesehene Situationen und Zustände als Folge der Realisierung komplexer Aufgabenstellungen auftreten. Diese werden im Falle einer einfachen Differenzbildberechnung als Vordergrund(-objekte) erkannt und führen bei Interpretation als zu vermeidendes Hindernis zum Stillstand des Robotersystems während der Kooperation. Der während der Kooperation erwünschte Kontakt mit unbekannten Objekten wird somit durch das hier vorgestellte System vermieden. Erste prototypische Tests zur Kooperation zwischen Mensch und Roboter durch Kraftführung wurden bereits im SIMERO-Projekt durchgeführt, in dessen Rahmen auch diese Arbeit entstanden ist. Hierzu wurde für die Kraftführung ein unüberwachter Bereich um den Greifer definiert, den der Mensch durch eigene Überwachung gegen unerwünschte Kollisionen absichert [Kuhn06]. Dieser rudimentäre Ansatz müsste verbessert werden, indem in einem zweiten Schritt nach der Differenzbilderzeugung die detektierten Objekte weiter unterschieden werden müssten in Klassen von Objekten mit erwünschtem Kontakt und solche ohne. Alsdann könnte mit diesen Klassen von Objekten unterschiedlich verfahren werden. Allerdings darf auch mit Objekten, mit denen im Rahmen der Aufgabe durch den Roboter Kontakt aufgenommen werden muss, im Allgemeinen nicht rüde umgegangen werden, was heißt, dass auch hier eine Abstandsberechnung gewährleistet werden muss, welche eine „sanfte“ Annäherung ermöglicht. Da diese Annäherung bei der Kooperation häufig auch in engen Räumen

stattfindet, müsste hier unter Umständen weitergehende Sensorik zum Einsatz kommen, beispielsweise Kamerasysteme oder Abstandsensoren auf dem Roboterarm. Um die Kooperation von Mensch und Roboter generell voran zu treiben, wäre es sinnvoll, dem Roboter verschiedene Grundfertigkeiten anzutrainieren, welche die Bausteine einer komplexeren Kooperationsaufgabe bilden („Kooperationsskills“) und diese dann online durch einen geeigneten Planer anhand der aktuell detektierten Intention des Menschen im Kontext der Aufgabenstellung zu verknüpfen.

Die hier gegebene Aufgabenstellung umfasste nicht die Realisierung eines sicheren Detektionssystems für unbekannte Objekte wie den Menschen im Sinne der entsprechenden Sicherheitsnormen. Die Sicherheit des Systems ist daher nicht gewährleistet. Hier ist ein integrierter Ansatz nötig, der die gesamte Kette der Umweltbedingungen, Sensordatenaquisition, Vorverarbeitung, Fusion und resultierender Aktion betrachtet und unter dem Aspekt der sicheren Funktionalität beim Ausfall von Teilkomponenten betrachtet. Hierbei können auch prinzipielle Bedingungen untersucht werden, unter denen die sensorische Erfassung des Arbeitsraumes mit vertretbarem Aufwand sicher erfolgen kann. Diese Fragestellung wird in einer parallelen Promotionsprojekt (siehe auch [Ober10]) am gleichen Lehrstuhl verfolgt.

Im Bereich des Bahnplaners könnte eine Kombination der vorgestellten Methoden mit einer modifizierten Version der tabellenbasierten Rekonstruktion der Belegung der Graphknoten und -kanten untersucht werden. Hierbei würde aus Gründen der Echtzeitfähig keine vollständige Tabelle zum Einsatz kommen, sondern eine Cache (ähnlich zu [Jaillet04]), der lediglich die häufig verwendeten Relationen von belegten Arbeitsraumbereichen zu den dadurch kollisionsbehafteten Knoten und Kanten enthält. Dieser wäre günstiger als eine vollständige Tabelle, da er in der Größe begrenzt werden kann und somit echtzeitfähig abarbeitbar wäre. Da dies dann keine vollständige Information liefern würde, müsste ein tatsächlicher Kollisionstest nachgelagert werden. Aus dessen Ergebnissen ließen sich dann online die Cache-Inhalte aktualisieren.

7 Literaturverzeichnis

Fettdruck: Eigene Publikationen.

[Adams02] F. Adams: „Kollege Roboter wird teamfähig“, Zeitschrift IEE Nr.9/2002, Hüthig Fachverlag, Heidelberg, 2002

[Althaus04] P. Althaus, H. Ishiguro, T. Kanda, T. Miyashita, H. Christensen: „Navigation for Human-Robot Interaction Tasks“, Proceedings of the 2004 IEEE International Conference on Robotics and Automation 2004 (ICRA 04), April 26-May 1, New Orleans, USA, 2004

[Alvarado02] M. Alvarado: „A risk assessment of human-robot interface operations to control the potential of injuries/losses at the XYZ Manufacturing company“, A research paper submitted in partial fulfillment of the requirements for the master in science degree in risk control, University of Wisconsin-Stout, May 2002

[Ameling96] Ameling, W. (Hrsg.): „Flexible Handhabungsgeräte im Maschinenbau“; Ergebnisse aus dem Sonderforschungsbereich 208, VCH Verlag, 1996

[Awais10] M. Awais, D. Henrich: "Human-Robot Collaboration by Intention recognition using Probabilistic State Machines", 19th International Workshop on Robotics in Alpe-Adria-Danube Region - RAAD 2010, 23-25 June 2010, Budapest, Hungary

[Baerveldt92] A.J. Baerveldt: „Cooperation between Man and Robot: Interface and Safety“, Proceedings of the IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, 1-3 September 1992

[Baginski99] B. Baginski: ”Motion Planning for Manipulators with Many Degrees of Freedom – The BB-Method”, Dissertation, AG Echtzeitsysteme und Robotik, TU München, 1999

[Bekris07] K. Bekris, L. Kavraki: „Greedy but safe replanning under kinodynamic constraints.“ In Proceedings of the IEEE International Conference on Robotics and Automation, pages 704–710, Rome, Italy, April 2007

[Berg06] J. van den Berg, D. Ferguson, J. Kuffner: “Anytime Path Planning and Replanning in Dynamic Environments”, Proceedings of the 2006 IEEE International Conference Robotics and Automation, May 15-19, Orlando, USA, 2006

[Berg08] J. van den Berg, M. Overmars: „Planning Time-Minimal Safe Paths Amidst Unpredictably Moving Obstacles“, The International Journal of Robotics Research, Vol. 27, No. 11-12, pp. 1274-1294, 2008

[Bohlin00] R. Bohlin, L.E.Kavraki: “Path Planning Using Lazy PRM”, Proceedings of the 2004 IEEE International Conference on Robotics and Automation, 521-528 vol.1, April 24-28, San Francisco, USA, 2000

[Bischoff99] Bischoff, A.; “Echtzeit Kollisionsvermeidung für einen Industrieroboter durch 3D-Sensorüberwachung“, Diplomarbeit Fernuniversität Hagen, 1999

[Braune05] I. Braune, J. Grabinger: „Method for the Monitoring of a Monitored Zone“, US Patent Application Publication, US 2005/0232465 A1, Oct. 2005

[Breckweg05] A. Breckweg: „Assistenzroboter für die Produktion: Stand der Technik – Herausforderungen – Potenziale“, in 4. Workshop für OTS-Systeme in der Robotik (Sichere Mensch-Roboter-Interaktion ohne trennende Schutzsysteme), Stuttgart, Nov. 2., 2005

[Brock99] O. Brock and O. Kathib: “Elastic Strips: A framework for integrated planning and execution”, in P. Corke and J. Trevelyan (Eds.), Proceedings of the International Symposium on Experimental Robotics, Volume 250 of Lecture Notes in Control and Information Sciences, pp. 328-338, Springer Verlag, 1999

[Brock00] O. Brock: „Generating robot motion: the integration of planning and execution“, Thesis, Stanford University, Stanford, CA, USA, 2000, ISBN:0-599-65839-8

[Burns07] B. Burns, O. Brock: „Single-query motion planning with utility-guided random trees.“, IEEE International Conference on Robotics and Automation, Rome, Italy, 2007

[Busse06] L. Busse: „Unüberwachtes Lernen zur Hindernisdetektion in Roboterzellen“, Bachelor Thesis, University of Bayreuth,2006

[Capek17] Capek, Karel: „Opilek“, 1917

[Carpenter91] G. Carpenter, S. Grossberg, D. Rosen: „ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition“, Neural Networks, Vol. 4, pp. 493-504, 1991

[Casas06] J. R. Casas, J. Salvador. „Image-based Multi-view Scene Analysis using ‘Conexels’“, The 1st HCSNet Workshop on the Use of Vision in HCI (VisHCI’06), pp 19-28, Canberra/Australia, November 1-3, 2006.

[Choset05] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun: „Principles of Robot Motion: Theory, Algorithms, and Implementations“, A Bradford Book, The MIT Press, 2005

[DeLuca06] A. De Luca, A. Albu-Schäffer, S. Haddadin, G. Hirzinger: „Collision Detection and Safe Reaction with the DLR-III Lightweight Manipulator Arm“, Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 9 - 15, Beijing, China, 2006

[Donald93] B. Donald, P. Xavier, J. Canny, J. Reif: “Kinodynamic motion planning”, Journal of the ACM, vol. 40 no. 5, pp. 1048-1066, 1993

[Duda01] R. Duda, P. Hart, D. Stork: „Pattern Classification“, Second Edition, John Wiley & Sons, New York, 2001

[Ebert03] Ebert, Dirk: „Bildbasierte Erzeugung kollisionsfreier Transferbewegungen für Industrieroboter“, Dissertation, Universität Kaiserslautern, 2003

[Elhabian08] Elhabian S., El-Sayed K., Ahmed S.: „Moving Object Detection in Spatial

Domain using Background Removal Techniques – State-of-Art", Recent Patents on Computer Science, vol. 1, no. 1, pp. 32-54, Jan 2008

[Feddema94] Feddema, J.T.; Novak, J.L.: "Whole Arm Obstacle Avoidance for Teleoperated Robots", In: IEEE Robotics and Automation Proceedings, pp.3303 – 3309, 1994

[Ferber99] Ferber, J.: „Multi Agent Systems", Addison-Wesley, New York, Bonn, Tokyo, ISBN 0-201-36038-9, 1999

[Fiorini96] P. Fiorini, Z. Shiller: „ Time Optimal Trajectory Planning in Dynamic Environments", Proceedings of the IEEE International Conference on Robotics and Automation, 22-28 Apr, Minneapolis, USA, 1996

[Fischer09] M. Fischer, D. Henrich: „3D Collision Detection for Industrial Robots and Unknown Obstacles using Multiple Depth Images", German Workshop on Robotics, June 9-10, Braunschweig, Germany, 2009

[**Gecks04**] T. Gecks, D. Henrich: „SIMERO: Camera Supervised Workspace for Service Robots" ASER 2004, 2nd Workshop on Advances in Service Robotics, Feldafing, Germany, 20-21 May 2004

[**Gecks05**] T. Gecks, D. Henrich: „Human-robot cooperation: Safe Pick-and-Place Operations", 14th IEEE International Workshop on Robot and Human Interactive Communication, August 13-15, Nashville, USA, 2005

[**Gecks06**] T.Gecks, D. Henrich: „Multi-Camera Collision Detection allowing for Object Occlusions", 37th International Symposium on Robotics (ISR 2006) and 4th German Conference on Robotics (Robotik 2006); München, Germany May 15th to 17th, 2006

[**Gecks07**] T. Gecks, D. Henrich, „Path Planning and Execution in Fast-Changing Environments with Known and Unknown Obstacles", International Conference on Intelligent Robots and Systems, 29 October – 2 November, San Diego, USA, 2007

[**Gecks09**] T. Gecks, D. Henrich: „Sensor-based Online Planning of Time-optimized Paths in Dynamic Environments", GWR09 German Workshop on Robotics, Braunschweig, Germany, June 9-10, 2009

[Geraerts03] R.Geraerts, M. Overmars: "Sampling Techniques for Probabilistic Roadmap Planners", Technical report UU-CS-2003-041, Institute of information and computing sciences, Utrecht University, 2003

[Geraerts04] R. Geraerts and M.H. Overmars: "Clearance based path optimization for motion planning", in Proceedings of the International Conference on Robotics and Automation, pp. 2386-2392, Vol. 3, April 26 – May 1, New Orleans, USA, 2004

[Göger06] D. Göger, K. Weiß, K. Burghart, H. Wörn: „Sensitive Skin for a Humanoid Robot", Proceedings of the International Workshop on Human-Centered Robotic Systems (HCRS'06), Munich, 2006

[Graf08] J. Graf, H. Wörn: „Combined Optical Flow Estimation and Active Contours for Safe Human-Robot Cooperation", in Proceedings of the 8th Asia-Pacific Conference On Control and Measurement (APPCM), 2008

[Guan06] L. Guan, S. Sinha, J.-S. Franco, M. Pollefeys: „Visual Hull Construction in the Presence of Partial Occlusion", Third International Symposium on 3D Data Processing, Visualization, and Transmission, 14-16 June 2006, pp.

413-420, Chapel Hill, USA, 2006

[Hart68] P.E. Hart, N.J. Nilsson, B. Raphael: „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): pp. 100–107, 1968

[Hartley04] R. Hartley and A. Zisserman: „Multiple View Geometry in Computer Vision“, Second Edition, Cambridge University Press, ISBN: 0521540518, 2004

[Heilig02] Heiligensetzer, P.: „Sichere Mensch-Roboter Kooperation für Roboter im niedrigen Traglastbereich“ In: OTS-Systeme in der Robotik, Reihe BKM Berichte, Herbert Utz Verlag, München, 2002

[Hein03] Hein, B.: „Automatische offline Programmierung von Industrierobotern in der virtuellen Welt“, Dissertation, Universität Karlsruhe, 2003

[Heinzmann99] J. Heinzmann, A. Zelinsky: „A Safe-Control Paradigm for Human-Robot Interaction“, In: Journal of Intelligent and Robotic Systems, Issue 25, 1999, Kluwer Academic Publishers, pp. 295-310, 1999

[Henrich92] D. Henrich, X. Cheng: „Fast Distance Computation for On-line Collision Detection with Multi-Arm Robots“, in Proceedings of the IEEE International Conference on Robotics and Automation, 12.-14. May, Nice, France, 1992

[Henrich98] D. Henrich, H. Wörn and C. Wurll: „On-line path planning with optimal C-space discretization“, in Proceedings of the International Conference on Intelligent Robots and Systems, Victoria, Canada, October 12-16, 1998

[**Henrich04**] D. Henrich, D. Ebert, T. Gecks: "Bildbasierte Kollisionstests für Randomized-Roadmap-Bahnplaner", VDI Robotik 2004, München, Deutschland, June 17.-18., 2004

[**Henrich08**] D. Henrich, T. Gecks: „Multi-camera Collision Detection between Known and Unknown Objects“, 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), Sept 7–11, Stanford/USA, 2008

[**Henrich08b**] D. Henrich, M. Fischer, T. Gecks, S. Kuhn: „Sichere Mensch/Roboter-Koexistenz und Kooperation“, ROBOTIK 2008, München, Germany, 2008

[Holleman00] C. Holleman, L. E. Kavraki: „A Framework for Using the Workspace Medial Axis in PRM Planners“, In Proceedings of the International Conference on Robotics and Automation, pp. 1408-1413, April 24-28, San Francisco, USA, 2000

[Hoover99] A. Hoover, B.D. Olsen: „Path planning for mobile robots using a video camera network“, Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 19-13 September, Atlanta, USA, 1999

[Hosoda95] K. Hosoda, K. Sakamoto, M. Asada: „Trajectory Generation for Obstacle Avoidance of Uncalibrated Stereo Visual Servoing without 3D Reconstruction“, Proceedings of the IEEE International Conference on Intelligent Robots and Systems 1995, Pittsburgh, 1995

[Jaillet04] L. Jaillet, T. Simeon: “A PRM-based Motion Planner for Dynamically Chaning Environments”, Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai International Center, Sendai, Japan September 28 - October 2, 2004

[Kallmann04] M. Kallmann, M. Mataric: „Motion Planning Using Dynamic Roadmaps“, Proceedings of the 2004 IEEE International Conference on Robotics & Automation, New Orleans, USA, 2004

[Kavraki96] L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars: „Probabilistic roadmaps for path planning in high-dimensional configuration spaces“, IEEE Transactions on Robotics and Automation 12 (4): pp. 566–580, 1996

[Keck08] M. Keck, J.W. Davis: „3D occlusion recovery using few cameras“, IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2008, 23-28 June, Anchorage, USA, 2008

[Khatib86] O. Khatib: “Real-time obstacle avoidance for manipulators and mobile robots”, in International Journal of Robotics Research Vol.5 (1), pp. 90-98, 1986

[Kim09] Y. M. Kim, C. Theobalt, J. Diebel, J. Kosecka, B. Micusik, S. Thrun: „Multi-view Image and ToF Sensor Fusion for Dense 3D Reconstruction“, In Proceedings of the IEEE International Workshop on 3-D Digital Imaging and Modeling (3DIM 2009), co-hosted with ICCV 2009

[Koditschek90] D.E. Koditschek, E. Rimon: „Robot navigation functions on manifolds with boundary“, Journal of Advances in Applied Mathematics, 11:412-442, 1990

[Koenig02] S. Koenig, M. Likhachev: „D* Lite“, 18th national conference on Artificial Intelligence, Edmonton, Alberta, Canada, Pages: 476 – 483, 2002

[Koenig04] S. Koenig, M. Likhachev, D. Furcy: „Lifelong Planning A*“, International Journal of Artificial Intelligence, Vol. 155, no. 1-2, pp. 93-146. May 2004

[**Kuhn06**] S. Kuhn, T. Gecks, D. Henrich: “Velocity control for safe robot guidance based on fused vision and force/torque data". In: IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp. 485-492, Heidelberg, Germany, Sep 3-6, 2006.

[Kuhn09] S. Kuhn, D. Henrich: „Multi-View Reconstruction of Unknown Objects within a Known Environment“, Proceedings of the 5th International Symposium on Visual Computing (ISVC 2009), Las Vegas, USA, 2009

[Kurzweil90] Kurzweil, Ray: „The Age of Intelligent Machines“, MIT Press, 1992, ISBN: 0262610795

[Ladikos08] A. Ladikos, S. Benhimane, N. Navab: „Efficient visual hull computation for real-time 3D reconstruction using CUDA“, IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008, 23-28 June, Anchorage, USA, 2008

[Lavalle98] S.M. Lavalle: "Rapidly-exploring random trees: A new tool for path planning", Technical Report 98-11 Computer Science Dept, Iowa State University, 1998

[Lavalle00] S.M. Lavalle, J.J. Kuffner: „Rapidly-Exploring Random Trees: Progress and Prospects“, Journal of Algorithmic and Computational Robotics: New Directions, 2000

[Lee03] J.-H. Lee, H. Hashimoto: „Controlling mobile robots in distributed intelligent sensor network“, IEEE Transactions on Industrial Electronics, Vol. 50, Issue: 5, pp. 890- 902, 2003

[Leou92] J. J. Leou, Y. L. Chang, J.S. Wu: „Robot operation monitoring for collision avoidance by image sequence analysis“, In: Pattern Recognition, Vol. 25,

No. 8, pp. 855-867, 1992

[Leven00] P. Leven, S. Hutchinson: „Toward Real-Time Path Planning in Changing Environments“, In Proceedings of the Workshop on Algorithmic Foundations of Robotics, Dartmouth, USA, 2000

[Lim00] H. Lim, K. Tanie: „Human Safety Mechanisms of Human-Friendly Robots: Passive Viscoelastic Trunk and Passively Movable Base“, In: The International Journal of Robotics Research Vol. 19 No. 4 April 2000 pp.307-335, Sage Publications, 2000

[Lin04] M. C. Lin, D. Manocha: „Collision and proximity queries“, in Handbook of Discrete and Computational Geometry, 2nd edition, ed. by J.E. Goodman, J. O'Rourke, Chapman Hall / CRC, New York, 2004

[Lindemann05] S. R. Lindemann and S. M. LaValle: “Smoothly blending vector fields for global robot navigation”, In Proceedings of the 44th IEEE Conference on Decision & Control, pp. 3353-3559, December 12-15, Sevilla, Spain, 2005

[Liu06] H. Liu, X. Deng, H. Zha, D. Ding, “A Path Planner in Changing Environments by Using W-C Nodes Mapping Couples with Lazy Edges Evaluation”, Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 9-15, Beijing, China, 2006

[Lumelsky93] Lumelsky, V.; Cheung, E.: “Real-Time Collision Avoidance in Teleoperated Whole-Sensitive Robot Arm Manipulators”, IEEE Transactions on Systems, Man and Cybernetics, Vol.23 No.1, pp.194-203,1993

[Magnor04] B. Goldluecke, M. Magnor: „Space-time isosurface evolution for temporally coherent 3D reconstruction“, Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. I-350 - I-355 Vol.1, 2004

[Markou03] Markou M., Singh S.: „Novelty detection: a review“, Signal Processing Volume 83, Issue 12, Pages 2481-2521, December 2003

[Martin99] Martín, P., Millán, J.: „Learning of Sensor-Based Arm Motions while Executing High-Level Descriptions of Tasks”, In: Autonomous Robots Vol. 7, Aug. 1999, Kluwer Academic Publishers, pp. 57-75, 1999

[Meisel91] Meisel, A.; Föhr, R.; Ameling, W.:“3D-Kollisionsschutzsensor auf der Basis von CCD-Kameras“, In SENSOR 91, Seiten 157-170, Nürnberg, Mai 1991

[Meisel94] Meisel A: “3D-Bildverarbeitung für feste und bewegte Kameras”, Vieweg Verlag, Reihe Forschritte der Robotik Nr. 21, 1994

[Mezouar02] Y. Mezouar, F. Chaumette: „Path planning for robust image-based control“, IEEE transactions on robotics and automation, vol. 18, no4, pp. 534-549, 2002

[Moore88] B. Moore: „ART1 and pattern clustering“, Proceedings of the 1988 Connectionist Models Summer School, p174-185, Morgan Kaufmann, San Mateo CA, 1988

[Moravec90] Moravec, Hans: „Mind Children - Future of Robot and Human Intelligence“, Harvard University Press 1990, ISBN: 0674576187

[Morhard02] D. Morhard: „Realisierung eines OTS-Systems bei einem Zulieferer der Kunststoffindustrie“, In: OTS-Systeme in der Robotik, Reihe BKM Berichte, Herbert Utz Verlag, München, pp. 8.1 – 8.12, 2002

[Mure-Dubois08] J. Mure-Dubois, H. Hugli: „Fusion of time of flight camera point clouds“, in Proceedings of the IEEE Workshop on Multi-camera and Multi-model Sensor Fusion Algorithms and Applications (M2SFA2), Marseille, France, 2008

[Niem97] W. Niem: „Error Analysis for Silhouette-Based 3D Shape Estimation from Multiple Views“, in Proceedings of the International Workshop on Synthetic - Natural Hybrid Coding and Three Dimensional Imaging, Rhodos, September, 1997

[Novak92] Novak, J.L.; Feddema, J.T.: ”A Capacitance-Based Proximity Sensor for Whole Arm Obstacle Avoidance”, IEEE Proceedings of the Intl. Conf. on Robotics and Automation, pp. 1307-1314, 1992

[Ober10] A. Ober, D. Henrich: „A Safe Fault Tolerant Multi-View Approach for Vision-Based Protective Devices”, Conference for Security, Safety and Monitoring in Smart Environments, 29 August - 1 September, Boston (USA), 2010

[Oberer-Treitz08] S. Oberer-Treitz, C. Meyer, A. Verl: „Kollisionsbewertung bei der Mensch-Roboter-Kooperation - Auswertung einer Crash Test Versuchsreihe mit einem Industrieroboter und einem Crash Test Dummy“, ROBOTIK 2008, München, Germany, 2008

[Okada09] N. Okada, J. Qiu, K. Nakamura, E. Kondo: „Multiple self-organizing maps for a visuo-motor system that uses multiple cameras with different fields of view“, Proceedings of the Fourteenth International Symposium on Artificial Life and Robotics 2009, B-Con Plaza, Beppu, Oita, Japan, 2009

[Philippsen05] R. Philippsen, R. Siegwart: „An Interpolated Dynamic Navigation Function“, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), April 18-22, Barcelona, Spain, 2005

[Piccardi04] Piccardi M.: „Background subtraction techniques: a review”, Proceedings of the International Conference on Systems, Man and Cybernetics, pp. 3199-3104, Oct. 2004.

[Pilger05] Pilger J.: „Machbarkeitsstudie zur Personendetektion mit Kamerasystemen im Arbeitsumfeld von Industrierobotern“, Fachhochschule Bonn-Rhein-Sieg, Bachelorarbeit, 2005

[Quick96] G. Quick, P. Müller: „A Novel Approach to Collision Avoidance of Robots Using Image Planes“, In: Robotics and Manufacturing – Recent Trends in Research and Applications, Vol. 6, pp. 567-573, 1996

[Quinlan93] S. Quinlan and O. Kathib: “Elastic bands: Connecting path planning and control”, In Proceedings of the International Conference on Robotics and Automation, Vol.2, pp802-807, 1993

[Radke04] Radke R. J., Andra S., Al-Kofahi O., Roysam B.: „Image change detection algorithms: A systematic survey“, in IEEE Transactions on Image Processing, 2004

[Rohrmüller08] F. Rohrmüller, M. Althoff, D. Wollherr, M. Buss: „Probabilistic Mapping of Dynamic Obstacles Using Markov Chains for Replanning in Dynamic Environments“, Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS08), Nice, France, 2008

[SafetyEye06] Patent DE 10 2006 057 605 A1. Verfahren und Vorrichtung zum

Überwachen eines dreidimensionalen Raumbereichs. PILZ GmbH & Co. KG, 2006, http://www.safetyeye.com.

[Schiavi09] R. Schiavi, F. Flacco, A. Bicchi: „Integration of Active and Passive Compliance Control for Safe Human-Robot Coexistence“, IEEE International Conference on Robotics and Automation, May 12 - 17, Kobe, Japan, 2009

[Schulz03] O. Schulz: „Image-based 3D-surveillance in Human-Robot-Cooperation“, in Modern Trends in Manufacturing. Second International CAMT Conference, Wroclaw, Poland: Oficyna Wydawnicza Politechniki Wroclawskiej, pp.327-34, Feb. 20-21, 2003

[Sisbot05] E. A. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, S. Woods: „Navigation in the presence of humans“, Proceedings of the 2005 5th IEEE-RAS International Conference on Humanoid Robots, December 5-7,Tsukuba, Japan, 2005

[Som05] F. Som: „Sichere Steuerungstechnik für den OTS-Einsatz von Robotern“, in 4. Workshop für OTS-Systeme in der Robotik (Sichere Mensch-Roboter-Interaktion ohne trennende Schutzsysteme), Stuttgart, Nov. 2., 2005

[Spingler02] Spingler, J., Thiemermann, S.: „Direkte Mensch-Roboter Kooperation in der flexiblen Montagezelle“, In: Robotik 2002, VDI-Berichte Nr. 1679, pp. 191-195, 2002

[Steinhaus99] Peter Steinhaus, Markus Ehrenmann, Rüdiger Dillmann: MEPHISTO: A Modular and Existensible Path Planning System Using Observation, ICVS, 1999

[Svoboda05] T. Svoboda, D. Martinec, T. Pajdla: „A Convenient Multicamera Self-Calibration for Virtual Environments“, Presence: Teleoperators and Virtual Environments. Vol. 14, no. 4, pp. 407-422, 2005

[Thiem02] Thiemermann, S.: „team@work – Direkte Mensch-Roboter Kooperation“, In: OTS-Systeme in der Robotik, Reihe BKM Berichte, Herbert Utz Verlag, München, pp. 4.1 – 4.5, 2002

[Thiem05] S. Thiemermann: „Direkte Mensch-Roboter-Kooperation in der Kleinteilmontage mit einem SCARA-Roboter“, Dissertation, Universität Stuttgart, 2005

[Vannoy04] J. Vannoy and J. Xiao: „Real-time adaptive and trajectory-optimized manipulator motion planning“ In Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2004), Sendai, Japan, 2004

[Vieth08] M. Vieth, O. Zilken, R. Herpers, D. Reinert: „Vision Based Hand and Finger Detection at Machines with Manual Operation“, ROBOTIK 2008, München, Germany, 2008

[Wegerif92] Wegerif, D.: “Sensor-based whole-arm obstacle avoidance for kinematically redundant robots”,In: Proceedings of SPIE Vol 1828 Sensor Fusion V, pp.417-426, 1992

[Winkler07] B. Winkler: „Safe Space Sharing Human-Robot Cooperation Using a 3D Time-of-Flight Camera“, International Robots and Vision Show, 2007

[Yang06] Y. Yang, O. Brock: „Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation“, in Robotics: Science and Systems,

August 16th-19th, Philadelphia, USA, 2006

[Zeller97] M. Zeller, R. Sharma, K. Schulten: "Motion planning of a pneumatic robot using a neural network", IEEE Control Systems Magazine, Volume: 17 Issue:3, pp: 89 - 98, June 1997

[Zettl02] Zettl, H.: „Verwirklichte OTS-Systeme und Gedanken zur Weiterentwicklung“, In: OTS-Systeme in der Robotik, Reihe BKM Berichte, Herbert Utz Verlag, München, pp. 9.1 – 9.13, 2002

URL-Referenzen:

[SFB588] http://www.sfb588.uni-karlsruhe.de

Normen

[EN 13849-1] „Sicherheit von Maschinen – Sicherheitsbezogene Teile von Steuerungen – Teil 1: Allgemeine Gestaltungsleitsätze“

[EN 61496-1] „Sicherheit von Maschinen - Berührungslos wirkende Schutzeinrichtungen, Teil 1: Allgemeine Anforderungen und Prüfungen“, (IEC 61496-1:2004, modifiziert); Deutsche Fassung EN 61496-1:2004

[ISO 10218] „Industrieroboter - Sicherheitanforderungen - Teil 1: Roboter“, (ISO 10218-1:2006, einschließlich Berichtigung 1:2007); Deutsche Fassung EN ISO 10218-1:2008

8 Anhänge

8.1 Simulationsumgebung – Hardwarekonfigurationen

Konfigurationsname	S1
CPU	AMD Athlon 64 X2 Dual Core Processor 3800+ 2.0 Ghz, 2 x 512 KByte L2-Cache
RAM	2GByte
Software(-umgebung)	OpenSuse 11.0, gcc 4.3, glibc 2.8

Konfigurationsname	S2
CPU	AMD Sempron Processor 3000+ 2.0 Ghz, 512 KByte L2-Cache
RAM	512 MB
Software(-umgebung)	OpenSuse 11.0, gcc 4.3, glibc 2.8

8.2 Optimierung des LPA

In Experimentläufen zeigte sich, dass der Expansionsschritt des LPA (Funktion *expandNode*(*v*, ***COLLECTEDNODES***) in Algorithmus 4.13, Seite 169) sehr viele Knoten zu ***COLLECTEDNODES*** hinzufügt und die anschließende Aktualisierung dieser Knoten via *updateState*(...) einen hohen Rechenaufwand erzeugt, da alle Nachbarn der Knoten betrachtet werden. Dies ist häufig unnötig, da in vielen Fällen nur der expandierte Knoten *v* Ursache der Aktualisierung ist. Hier kann man für die Nachbarn v_i von *v* verschiedene Fälle unterscheiden:

1) v_i ist noch nie in ***OPEN*** gewesen und hat keinen Vorgängerknoten:
   der Knoten muss sich aus all seinen Nachbarn den günstigsten Vorgängerknoten raussuchen (unbedingter Aufruf von *updateState*(...))
2) v_i ist schon in ***OPEN*** gewesen und hat einen Vorgängerknoten:
   dann muss der expandierte Knoten *v* selbst betrachtet werden:
   1) Die heuristischen Kosten zum Endknoten v_z haben sich geändert (Kapitel 4.7), dann muss der Knoten immer aktualisiert werden, da sich Verschiebungen in der Sortierung der Knoten ergeben könnten. Dies ist auch ein wesentlicher Unterschied zum klassischen A*, bei dem die Kosten zum Ziel für einen Knoten als konstant angenommen werden. (unbedingter Aufruf von *updateState*(...))
   2) *key*(*v*) ist unterkonsistent:
      wenn *v* der Vorgängerknoten von v_i ist, muss sich v_i aus all seinen Nachbarn den günstigsten Vorgängerknoten suchen, da der Weg über den Knoten v eventuell nicht mehr der günstigste ist, da dessen Kosten steigen. (bedingter Aufruf von *updateState*(...))
   3) *key*(*v*) ist überkonsistent (nach Expansion konsistent):
      der Pfad von *v* nach v_i kann nun eventuell der günstigste sein für v_i, das kann man direkt prüfen und nur im Fall einer Verbesserung des Schlüssels *key*(v_i) den Knoten v_i in OPEN einfügen, ansonsten ist keine Aktion erforderlich (kein Aufruf von *updateState*(...))

Diese Optimierung erzeugt Laufzeitvorteile von beispielsweise einem Faktor 10 bei 20 Kanten pro Knoten; es werden ca. 50% der Aktualisierungen von Nachbarknoten vermieden. Im nachfolgenden Algorithmus 8.1 ist die nach obiger Unterscheidung modifizierte Version des Expansionsschrittes (die Funktion *expandNode*(...) in Algorithmus 4.13) gegeben:

```
(1)  expandNode(v, COLLECTEDNODES)
(2)      bool isUnderConsistent = false
(3)      if(g(v) > rhs(v))
(4)          g(v) = rhs(v)
(5)      else
(6)          isUnderConsistent = true
(7)          g(v) = ∞
(8)          insert v in COLLECTEDNODES
(9)
(10)     forall v_i in neighbours(v)
(11)         if v_i has not been in OPEN before in this planning cycle
(12)             insert v_i in COLLECTEDNODES
(13)         else
(14)             if h(v_i, v_z) changed   // heuristic to-end costs have changed ?
(15)                 insert v_i in COLLECTEDNODES
(16)             else
(17)                 if isUnderConsistent and v is predecessor of v_i
(18)                     insert v_i in COLLECTEDNODES
(19)                 else      // v has been overconsistent
(20)                     if g(v) + c(v, v_i) < rhs(v_i)  // do we need to reopen?
(21)                         remove v_i from OPEN, when it is in OPEN
(22)                         rhs(v_i) = g(v) + c(v, v_i)
(23)                         insert v_i into OPEN with key(v_i)
```

Algorithmus 8.1: Modifikation des Expansionsschritts der Graphensuche zur Optimierung der Ausführungsgeschwindigkeit