UNIVERSITÄT
BAYREUTH

Optimierung von Kamerapositionen zur Überwachung teils unbekannter Umgebungen

Diplomarbeit
von
Maria Hänel

Mathematisches Institut
Fakultät für Mathematik, Physik und Informatik
Universität Bayreuth

1. Gutachter:	Prof. Dr. Lars Grüne
2. Gutachter:	Prof. Dr. Dominik Henrich
Tag der Einreichung:	19. April 2010

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Man stelle sich die Situation vor, es werde in einem Raum mit einem Industrieroboter gearbeitet, der im Boden verankert ist. Seinen gewaltigen Arm bewege er in Sekundenbruchteilen von einer Ecke des Raums in die andere. Ohne Probleme könnte er einen Menschen mitreißen und gegen die Wand schleudern. Aufgrund der kurzen Zeit hätte Letzterer noch nicht einmal die Möglichkeit auszuweichen, geschweige denn den Roboter anzuhalten.

Um dies zu verhindern, soll eine solche Gefahr frühzeitig erkannt werden. Jedes Mal, wenn die Distanz zwischen Roboter und Mensch zu klein wird, soll dem Roboter der Befehl erteilt werden, langsamer zu fahren - im Extremfall sogar anzuhalten. Dazu soll ein Netz mit einer festen Anzahl an Kameras den Ort bewachen. Das Netz gilt es so einzustellen, dass der Abstand zwischen Menschen und Gefahr in einer Sicherheitszone „so gut wie möglich“ beobachtet werden kann. Dabei wird der Begriff „Einstellung“ für die Positionierung und Orientierung der Kameras verwendet.

Die Beobachtung des Abstandes wird erschwert durch die Tatsache, dass der Mensch während seines Aufenthaltes in der Sicherheitszone in Nischen hinter sichtblockierenden Hindernissen versteckt sein könnte - sei es zum Beispiel, weil unter einem Tisch etwas repariert werden muss, hinter einer Wand geputzt oder dem (ebenfalls sichtblockierenden) Roboter Teile gereicht werden müssen. In solchen Fällen kann der Mensch jedoch nicht von gewöhnlichen Sensoren registriert werden, und deswegen werden auch keine Schutzmaßnamen ergriffen. Folglich könnte der Roboter mit einem einzigen unvorsichtigen Zug dem Menschen Schaden zufügen. Die Kameras müssen den Abstand zwischen Menschen und Gefahr also konservativ abschätzen, das heißt Letzterer darf den tatsächlichen Abstand nicht übersteigen.

Das Ziel dieser Arbeit ist es, ein Programm zu entwickeln, das bei einer gegebenen Anzahl von Kameras und Informationen zu der Umgebung (wie z.B. Größe des Raumes, Hindernisse), sowie Informationen zu dem Menschen (z.B. Größe, Aufenthaltswahrschein-

lichkeit), die beste Einstellung des Kameranetzes findet. Dabei bezieht sich die Wertung „beste“ auf die Höhe des Fehlers, der beim Schätzen des Abstandes zwischen Mensch und Gefahr von den Kameras gemacht wird.

Sobald eine gute Einstellung der Kameras für eine Szene gefunden worden ist, sollen die Kameras fest installiert werden. Dies ermöglicht die Optimierung der Kameraeinstellungen offline, also vorab zu bearbeiten. Die Rechenzeit des entwickelten Programms ist deswegen sekundär.

1.2 Stand der Forschung

In allen Bereichen, in denen Kameras eingesetzt werden, ist eine optimale Kameranetzeinstellung von großer Bedeutung. Dabei handelt es sich sehr oft um einen Kompromiss zwischen einer guten Bildqualität einerseits, das heißt wie nah und detailliert das zu Beobachtende erscheint, und andererseits der Fülle der Informationen, die durch eine Kamera eingefangen werden können. Je nach Aufgabenfeld gibt es unterschiedliche Möglichkeiten der Handhabung dieses Kompromisses:

Zum einen gibt es die Aufgabe, die von den Kameras erfassten Oberflächen [10], Bahnen [1, 6, 7] oder Anzahl von Objekten [14, 15, 13] zu maximieren. Damit verwandt ist das Problem, zusätzlich zu der Standortsuche der Kameras deren Anzahl zu minimieren [5, 14, 15, 13]. Dabei ist vorgegeben, dass alle relevanten Gegenstände beobachtet werden müssen (in 2D ist dies das „Art Gallery Problem“). Die angegebenen Veröffentlichungen stellen nur eine kleine Auswahl dar.

Innerhalb dieser Aufgabenfelder wird die Fülle der Informationen bewertet, die durch ein Kameranetz bei gegebener Positionierung der Kameras gewonnen wird. Das ist hilfreich bei Überwachungsnetzwerken, deren Aufgabe es ist zu entscheiden, *ob* ein zu beobachtendes Objekt die Szene betreten hat (sofern dieses erkannt werden kann). Das hier diskutierte Thema verlangt zusätzlich die Beantwortung der Frage „wo?“und „wie weit?“.

Eine Distanz zweier Objekte anhand Kameras zu berechnen, ist eine Aufgabenstellung der „Photogrammetry“. In unserem Falle handelt es sich um den Bereich der „Closed-Range Photogrammetry“, bei der Distanzen kleiner als 100 Meter betrachtet werden. Bei der Positionierung und Orientierung der Kameras soll häufig der Fehler minimiert werden, der bei einer betrachteten Distanzmessung entsteht. Meist wird jedoch die Phrase „Potogrammetric Network Design“ verwendet, um auszudrücken, dass der Fehler minimiert werden soll, der beim Projizieren eines Bildes zurück in den Raum, das heißt bei der Rekonstruktion, auftritt. Dabei wird angenommen, dass freie Sicht auf das zu Rekonstruierende gewährt ist. Das Optimieren von Kamerapositionen zur Distanzbestimmung wird bereits in [19, 18, 16, 17] aufgegriffen. Ähnlich zu [9] werden einzelne Punkte betrachtet, mit dem Unterschied, dass die Flächenrekonstruktion berücksichtigt wird. Auch in [4] besteht die Aufgabe darin, ein Objekt mit minimalem Fehler zu lokalisieren, welches nicht von anderen Objekten verdeckt werden kann. In vielen dieser Veröffentlichungen wird außerdem problemspezifisch vereinfacht. So wird etwa die Anzahl der Kameras auf zwei beschränkt. Oder es werden Nebenbedingungen an die Positionen der Kameras gestellt; die Kame-

ras dürfen beispielsweise nur auf der Oberfläche einer Kugel um das zu rekonstruierende Objekt positioniert werden.

Da die optimale Kamerapositionierung bezüglich des Beobachtungsfehlers ein komplexes Problem darstellt, bei dem frühere Resultate aufgrund solcher Spezialisierungen nicht auf unsere Aufgabenstellung übertragbar sind, muss spezifischer gesucht werden. [23] kommt der Intention am nächsten, durch das Background-Subtraction-Verfahren die Verdeckungen von Objekten zu bestimmen, um sie zu lokalisieren. Diese Veröffentlichung geht davon aus, dass durch die Minimierung der Stellen, an denen sich Objekte befinden können, auch gleichzeitig die Lokalisierung eines Objektes erleichtert wird. Um die Dimension des Problems in den Griff zu bekommen, wird in [23] allerdings ebenfalls vereinfacht: Zum einen ist der Sichtkegel der Kameras ein Halbraum. Die Kameras werden in die Mitte des Raumes gerichtet, deswegen ist lediglich die Position einzustellen. Zum anderen werden keine Hindernisse in der Szene betrachtet. Im Gegensatz dazu werden in [3] statische Hindernisse betrachtet, allerdings ist hier lediglich eine Auswahl aus bereits vorhandenen und fest installierten Kameras möglich.

In dieser Arbeit soll im Gegensatz zu den genannten Forschungsergebnissen der allgemeine Fall betrachten werden, in dem Kameras einer gegebenen Anzahl als Sensornetz dienen, welche in einer Umgebung uneingeschränkt beliebig aufgestellt und orientiert werden dürfen. Im Gegensatz zu vielen Vereinfachungen aus bereits vorhandener Literatur wird die Herausforderung angenommen, statische und dynamische sichtblockierende Hindernisse zu berücksichtigen. Die Distanz zwischen zwei Objekten soll dennoch konservativ abgeschätzt werden.

1.3 Aufgabenstellung

Mathematisch formuliert sei in einer Umgebung $U \subset \mathbb{R}^3$ eine Sicherheitszone $Z \subset U$ gegeben. Letztere soll überwacht werden. In dieser Zone Z befinden sich Gefahren (zum Beispiel der Roboter), deren Punkte in $G \subset Z$ zusammengefasst sind. Außerdem enthält die Zone Z mehrere unbekannte Objekte (zum Beispiel mehrere Menschen), deren Abstände zu den Gefahren überwacht werden sollen. Die Punkte innerhalb dieser Objekte werden im unbekannten Kollektiv $O_u \subset Z$ zusammengefasst. Bewegungen und Deformierungen des Kollektivs sind nicht bekannt, daher die Namensgebung „unbekannt“.

Weiterhin gibt es $m = 1, ..., M$ Hindernisse in der Sicherheitszone Z, die für das unbekannte Kollektiv O_u undurchdringbar sind und die Sicht von Kameras teilweise blockieren. Diese Hindernisse sind statisch oder dynamisch und werden mit $O_{s_j} \subset U,\ j = 1, ..., m$ und $O_{d_j} \subset U,\ j = (m+1), ..., M$ bezeichnet. Die Punkte innerhalb der statischen Objekte werden im statischen Kollektiv O_s, die Punkte innerhalb der dynamischen im dynamischen Kollektiv O_d zusammengefasst.

Zur Optimierung bezüglich des Beobachtungsfehlers gibt es n Kameras. Sie sollen innerhalb der Umgebung U fest installiert werden. Welchen Teil der Umgebung eine einzelne Kamera $i \in \{1, ..., n\}$ überwacht, kann separat durch deren Einstellung $e_i \in E$ variiert werden. Eine Einstellung aus E bestimmt die Positionierung und Orientierung einer Ka-

mera.

Im Zusammenhang mit den genannten Angaben der Umgebung U, Sicherheitszone Z, Gefahren G und verschiedenen Kollektiven $O_{u/s/d}$ ist die Einstellung des Netzwerkes, das heißt eine Einstellungskombination $(e_1, ..., e_n) \in E^n$ gesucht, so dass der Abstand zwischen dem unbekannten Kollektiv O_u und den Gefahren in G konservativ und möglichst genau bestimmt werden kann, damit der Fehler zwischen geschätzem und tatsächlichem Abstand also möglichst klein wird.

Der Fehler, der bei der Abschätzung des Abstandes einer bestimmten Kameranetzeinstellung $(e_1, ..., e_n)$ gemacht wird, wird in Kapitel 2 modelliert. Die Funktion, die die Kameranetzeinstellung auf jenen Fehler abbildet, bewertet die Güte der Kameranetzeinstellungen und wird daher als Zielfunktion des Optimierungsproblems verwendet. In zwei Erweiterungen des Problems werden in diesem Kapitel Hindernisse und andere Schwierigkeiten betrachtet und in die Zielfunktion des Optimierungsproblems integriert. Die erweiterte Zielfunktion ist nichtlinear und nicht stetig. Sie wird in Kapitel 3 numerisch umgesetzt und durch eine schwarmintelligente Metaheuristik gelöst, die sukzessiv Lösungen in der Nähe von Kameranetzeinstellungen untersucht, welche bereits als gute Kameranetzeinstellungen identifiziert worden sind. Im anschließenden Kapitel 4 werden die Ergebnisse der Untersuchungen der Umsetzung bezüglich des Erfolgs, der Laufzeit, der Anzahl der Iterationen der Methaheuristik und des Speicherverbrauch dargestellt und diskutiert.

Kapitel 2

Analytische Problemstellung

Das Ziel ist es, eine möglichst optimale Einstellung des Kameranetzes zu erhalten, so dass die geschätzte Distanz zwischen unbekanntem Kollektiv und Gefahren möglichst wenig von der tatsächlichen abweicht. Im Hinblick darauf werden hier die Angaben aus Abschnitt 1.3 mathematisch definiert und die besagte Abweichung als Fehler modelliert. Durch das Kameranetz soll der Abstand zwischen zwei Mengen innerhalb einer Umgebung konservativ geschätzt werden, das heißt die Schätzung soll immer kleiner oder gleich dem tatsächlichen Abstand sein.

Zur Erreichung des Ziels wird in diesem Kapitel die Theorie für die praktische Umsetzung aus Kapitel 3 in drei Etappen ausgearbeitet. Wie die Kameras die Schätzung vornehmen, wird über die Betrachtung der Beschaffenheit von Objekten und Einstellungen des Kameranetzes in folgendem Abschnitt 2.1 erarbeitet. Das entstehende Problem dient als Grundproblem der folgenden zwei Erweiterungen. Zum einen werden in Abschnitt 2.2 sichtblockierende Hindernisse der Umgebung hinzugefügt. Zum anderen werden in Abschnitt 2.3 Gefahren betrachtet, die selbst sichtblockierende Hindernisse sein können.

2.1 Das Grundproblem

Eine Sicherheitszone $Z \subset U \subset \mathbb{R}^3$ innerhalb einer Umgebung U enthält eine Menge von gefährlichen Punkten $G \subset Z$. Das Kameranetzwerk in der Umgebung soll möglichst genau abschätzen, ob genügend Abstand zu diesen Punkten gehalten wird. Dazu sei die Sicherheitszone außerdem Teil der durch die Kameras abbildbaren Menge A der Umgebung: $Z \subset A \subset U$. Die Bedeutung von A wird erst in Kapitel 2.2 im Zusammenhang mit Hindernissen offensichtlich. Im angesprochenen Kapitel enthält diese Menge jene Punkte, die nicht innerhalb bekannter Hindernissen liegen, und übernimmt damit einen Teil der Rolle der Sichtblockierung. Für den hier vorliegenden Fall ohne Hindernisse sei $A := U$. Wir werden feststellen, dass es sich um ein Problem der Art

$$\text{Minimiere } F(x) \text{ unter den Nebenbedingungen: } x \in X$$

mit der Zielfunktion $F : X \to \mathbb{R}$ auf dem Definitionsbereich X der Variablen x handelt. Um dieses Problem konkret zu formulieren, wird zunächst die Beschaffenheit von Objekten betrachtet, deren Abstand zu den Punkten aus G geschätzt werden soll. In der Annahme, ein solches Objekt sei in der Umgebung und könne durch die Kameras sensorisch erfasst werden, wird daraufhin aus den Parametern einer Kamera der Bereich hergeleitet, in dem sich das Objekt befinden kann. Aus der Gesamtheit der Bereiche aller Kameras werden die möglichen Aufenthaltsorte eines unbekannten Objektes und zugehörige Abstände zu den gefährlichen Punkten bestimmt; d.h. die möglichen Aufenthaltsorte dienen als Modell des unbekannten Kollektivs und werden dazu benötigt, eine mathematisch präzise Formulierung des Grundproblems anzugeben.

2.1.1 Unbekannte Objekte und ihre Parameter

In der Umgebung U bewegen und deformieren sich Objekte, die keinen Schaden nehmen dürfen - die sogenannten *unbekannten Objekte*. Wir fassen diese zusammen unter dem Begriff *unbekanntes Kollektiv*. Wie das unbekannte Kollektiv angeordnet und geformt ist, wird angegeben durch einen objektspezifischen (mehrdimensionalen) Parameter $a \in \mathbb{R}^k$. Er bestimmt die Oberfläche des unbekannten Kollektivs. Mit dieser Oberfläche umschließt das Kollektiv Punkte aus der Umgebung U, dabei bezeichne $O_u(a)$ das unbekannte Kollektiv mit Oberflächenparameter a: $O_u(a) \subset U$. Die Verteilung des Parameters gibt das Wahrscheinlichkeitsmaß $P : 2^{\mathbb{R}^k} \to [0, 1]$ an. Im Folgenden verwenden wir $O_u := O_u(a)$ zur Vereinfachung der Darstellung.

Solange ein Objekt des unbekannten Kollektivs die Sicherheitszone in der Umgebung betreten hat, das heißt, wenn $\forall a \in \mathbb{R}^k$ mit $P(a) > 0$ gilt $O_u(a) \cap Z \neq \emptyset$, soll sichergestellt werden, dass das Objekt nicht zu Schaden kommt. Das geschieht dadurch, dass Sicherheitsmaßnamen sofort in die Wege geleitet werden, sobald der Abstand zwischen jedem beliebigen gefährlichen Punkt $g \in G \subset Z$ und jedem beliebigen Punkt $u \in O_u \cap Z$ zu klein erscheint. Dabei entspricht der Abstand der 2-Norm $d(x_1, x_2) = ||x_2 - x_1||_2$.

$$d(G, O_u) = \min\{d(g, u) | g \in G, u \in O_u \cap Z\} \quad \textit{tatsächlicher Abstand}$$

Sind die Kameras dabei nicht optimal gestellt, kann sich der gemessene Abstand deutlich von dem tatsächlichen Abstand unterscheiden.

Der folgende Abschnitt beschreibt den Aufbau einer Kamera und deren Wirkungsweise. Dies ist essenziell für die darauffolgende Bestimmung eines Modells des unbekannten Kollektivs, dessen Abstand zu den gefährlichen Punkten geschätzt werden soll. Aus dem gemessenen und dem tatsächlichen Abstand wird dann eine Abweichung berechnet, die es zum Schluss zu minimieren gilt.

2.1.2 Die Kamera als Sensor

Das Kameranetz, das den Abstand zwischen Gefahren und dem unbekannten Kollektiv bestimmt, bestehe aus n Kameras. Die Idee des Verfahrens ist, ein grobes Modell des unbekannten Kollektivs zu erstellen, zu dem der besagte Abstand gemessen wird. Der Fehler der Schätzung liegt dabei in der Erstellung des Modells: Jede dieser Kameras muss erkennen können, wo sich definitiv kein Objekt des unbekannten Kollektivs aufhält. Sobald demzufolge eine einzige Kamera die Information liefert, ein Punkt sei frei von unbekannten Objekten, gehört dieser Punkt zum freien Bereich. In unserer Betrachtung entsteht ein Modell des unbekannten Kollektivs aus den übrigen Bereichen, nämlich jene, die nicht als „frei“ gekennzeichnet worden sind.

Um diese Bereiche in der Umgebung einzuteilen und das Modell später durch die Zielfunktion zu beurteilen, werden einige Definitionen benötigt, die die Bereichseinteilung mit den frei wählbaren Variablen der Kamerapositionierung in Verbindung bringen. Wir beginnen mit einer abstrakten Definition einer Kamera, jene wird im Laufe des Kapitels verfeinert:

Definition 2.1
Sei U eine Umgebung. Eine *Kamera* definiert eine Funktion

$$\kappa : (E \times A) \to S \ .$$

Dabei entspricht E dem Raum der einstellbaren Kameraparameter und S einer Menge von sensorischen Werten. $A \subset U$ kennzeichnet dabei wiederum die durch die Kamera abbildbaren Punkte der Umgebung.

Diese Definition lässt erkennen, dass sich die Einstellungen einer Kamera auf den Sensorwert, der zu einem Punkt der Umgebung gehört, auswirken. Der Sensorwert ist wiederum relevant für die Bestimmung von Bereichen, die frei von unbekannten Objekten sind. Deswegen konkretisieren wir im Folgenden E und S. Dabei wird der Begriff „Kamera“ nicht im ursprünglichen Sinne der Bilderstellung verwendet. Eine besseres Verständnis von der Aufgabe einer Kamera in vorliegendem Falle bekommt man, indem man sich die Kamera als Instrument zur Einteilung der Umgebung in besagte Bereiche vorstellt. Demnach können später auch Punkte der Umgebung abgebildet werden, die im ursprünglichen Sinne durch ein Hindernis blockiert sind.

Die einstellbaren Kameraparameter

Kommen wir zuerst zu den Kameraparametern, die die Sicht von einer Kamera κ beeinflussen. Ein Grundmodell sieht die Kameraposition, -orientierung und den Öffnungswinkel ihres Sichtbereichs vor. Während die Öffnung für jede Kamera fest vorgegeben ist, sind Kameraposition und -orientierung frei wählbare Variablen. In Abschnitt 2.1.4 wird dann gefordert, diese Freiheitsgrade so zu wählen, dass die Güte der Kamerastellung optimiert wird. Um dieses Ziel zu erreichen, muss E zunächst durch Kameraposition und -orientierung konkretisiert werden:

$$E = \overline{U} \times [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Dabei entspricht $\overline{U} \subset U$ einem Teil der Umgebung, aus der der Ortsvektor der Kameraposition gewählt werden darf. Wir nehmen zunächst an, $\overline{U} = U$. Der Ortsvektor der Kameraposition wird weiterhin auch Fokus der Kamera genannt $f \in \overline{U}$.

Die Orientierung wird normalerweise mit Hilfe von drei Winkeln angegeben, deren Bezeichnungen aus der Flugzeugnavigation stammen: „Rollen-Nicken-Gieren“, oder im Englischen auch „roll-pitch-yaw“, beschreiben in dieser Reihenfolge wie in Abbildung 2.1 den Querneigungswinkel um die Normale der Bildebene (x-Achse), den Winkel um die Querachse (y-Achse) $-\frac{\pi}{2} \leq \rho \leq \frac{\pi}{2}$ und den Azimuth $-\pi \leq \phi \leq \pi$ um die z-Achse. Im Zusammenhang mit der Kamerapositionierung sind auch die Begriffe „swing-tilt-pan“gebräuchlich. Im Rahmen dieser Arbeit wird die einfachste Form einer Kameradarstellung mit kegelförmigem Sichtbereich betrachtet. Stelle man sich die Kamera als Lichtquelle vor, so wird gleich klar, dass das Drehen um die Rotationsachse des Kegels den Sichtbereich nicht verändert. Das heißt die Kamera deckt durch das „Rollen“den gleichen Bereich ab; deswegen wird dieser Parameter vernachlässigt. Somit sind pro Kamera fünf Freiheitsgrade festlegbar.

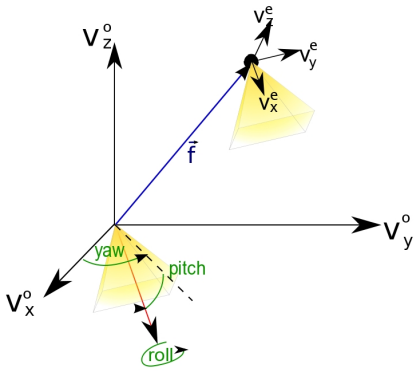

Abbildung 2.1: Kameraorientierung (grün), -positionierung (blau)

Zunächst soll geklärt werden, welchen Bereich eine Kamera mit ihrem kegelförmigen Sichtbereich abdeckt. Dazu wird aus den Kameraparametern ein neues Koordinatensystem eingeführt:

Definition 2.2
Seien $v_x^0, v_y^0, v_z^0 \in \mathbb{R}^3$ orthogonale Einheitsvektoren. Sei weiterhin $u = u_x^0 v_x^0 + u_y^0 v_y^0 + u_z^0 v_z^0 \in \mathbb{R}^3$.

1. Als *Weltkoordinaten von u* bezüglich v_x^0, v_y^0 und v_z^0 werden die kartesischen Koordinaten (u_x^0, u_y^0, u_z^0) bezeichnet

2. Als *Kamerakoordinaten* der Einstellung $e = (f, \phi, \rho) \in E$ bezeichnen wir (u_x^e, u_y^e, u_z^e) mit $u = u_x^e v_x^e + u_y^e v_y^e + u_z^e v_z^e$ im affinen Unterraum $f + \mathbb{R}^3$ bezüglich der Basis

$$\begin{aligned} v_x^e &= Rot_z(\phi) \cdot Rot_y(\rho) \cdot v_x^0 + f \\ v_y^e &= Rot_z(\phi) \cdot Rot_y(\rho) \cdot v_y^0 + f \\ v_z^e &= Rot_z(\phi) \cdot Rot_y(\rho) \cdot v_z^0 + f \end{aligned}$$

$$\text{mit } Rot_z(\phi) = \begin{pmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ und } Rot_y(\rho) = \begin{pmatrix} cos\rho & 0 & sin\rho \\ 0 & 1 & 0 \\ -sin\rho & 0 & cos\rho \end{pmatrix}$$

3. Daraus entstehen die Transformationen:

$$\begin{aligned} \text{(Kamera} \to \text{Welt)} \quad T_e^0 : \quad \mathbb{R}^3 &\to \mathbb{R}^3 \\ u^e &\mapsto Rot_z(\phi) \cdot Rot_y(\rho) \cdot u^e + f \\ \text{(Welt} \to \text{Kamera)} \quad T_0^e : \quad \mathbb{R}^3 &\to \mathbb{R}^3 \\ u^0 &\mapsto Rot_y(-\rho) \cdot Rot_z(-\phi) \cdot (u^0 - f) \end{aligned}$$

Definition 2.3
Sei $e \in E$ die Einstellung der Kamera κ und u^e die Beschreibung eines Punktes $u \in \mathbb{R}^3$ in Kamerakoordinaten $u^e = (u_x^e, u_y^e, u_z^e)$.

1. Es ist ein einseitiger Kegel definiert mit Radius $\beta \in [0, \pi]$:
$cone_+(\beta) =$

$$= \begin{cases} \mathbb{R}^3 & für\ \beta = \pi \\ \left\{u^e \in \mathbb{R}^3 \mid u_x^e \geq 0\right\} & für\ \beta = \frac{\pi}{2} \\ \left\{u^e \in \mathbb{R}^3 \mid u_x^e \geq 0, \sqrt{(u_y^e)^2 + (u_z^e)^2} \leq \tan\beta \cdot u_x^e\right\} & \beta \in [0, \frac{\pi}{2}) \\ \left\{u^e \in \mathbb{R}^3 \mid u_x^e \geq 0 \ \vee \ \left(u_x^e < 0, \sqrt{(u_y^e)^2 + (u_z^e)^2} \geq \tan(\pi - \beta) \cdot u_x^e\right)\right\} & \beta \in (\frac{\pi}{2}, \pi) \end{cases}$$

2. Sei außerdem $0 \leq \alpha \leq 2\pi$. Es heißt, u^e ist *im Sichtbereich von κ mit Einstellung e und Öffnungswinkel α*, wenn

$$u^e \in cone_+\left(\frac{\alpha}{2}\right)$$

Ein kegelförmiger Sichtbereich, im Gegensatz zu einem pyramidenförmigen, hat zusätzlich den Vorteil, dass der Öffnungswinkel eindimensional ist. Das erspart ebenfalls Rechenzeit, wenn auch nur gerinfügig, da der Öffnungswinkel keinen Freiheitsgrad der Zielfunktion darstellt. Auf Weltkoordinaten übertragen bedeutet der Sichtbereich folgendes:

Bemerkung 2.4
Sei κ eine Kamera mit Einstellung $e \in E$ und Öffnungswinkel α. Sei T_0^e die Transformation, die einen Punkt von Weltkoordinaten in Kamera-Koordinaten transformiert und sei $u^0 \in U \subset \mathbb{R}^3$ in Weltkoordinaten gegeben. u^0 ist innerhalb des Sichtkegels, wenn

$$T_0^e(u^0) \in \left\{ u^e \in \mathbb{R}^3 \text{ in Kamerakoordinaten} \mid u^e \in cone_+\left(\frac{\alpha}{2}\right) \right\}$$

Wir schreiben kürzer:

$$u^0 \in (T_0^e)^{-1}\left(cone_+\left(\frac{\alpha}{2}\right)\right)$$

Die oben angesprochene Abdeckung des Raumes durch die Kamera wird zwar von Linsendistorsionen, also Verkrümmungen des Abbildes der Linse, beeinflusst, diese entfallen jedoch für das Modell der Kamera. Das entstandene Modell entspricht dem einer Lochbildkamera wie in [1, S. 264]. Die Abdeckung einer Kamera kann zusätzlich eingeschränkt werden durch Angabe der Brennweite *(focal length)* der Linse oder der Tiefe des Feldes (der Abstand zwischen dem nächsten sichtbaren Objekt zum weitesten *(Depth of Field)*). Aus Gründen der Rechenaufwandsersparnis wird auch hier auf ein Erweitern des Modells der Lochbildkamera verzichtet.

Sensorische Werte

Das Ziel ist es, mit Hilfe eines Kameranetzes den Abstand zwischen dem unbekannten Kollektiv und gefährlichen Punkten abzuschätzen, indem die Kameras die unbekannten Objekte eingrenzen und daraus ein Modell erstellen. Nun sind die Sensordaten von einer Kamera κ auszuwerten, um die durch κ abbildbaren Punkte $w^0 \in A$ in drei Bereiche einzuteilen, aus denen in Abschnitt 2.1.3 das Modell des unbekannten Kollektivs hergeleitet wird. Das Verfahren ist [11] entnommen, wobei die Notation entsprechend angepasst wird.

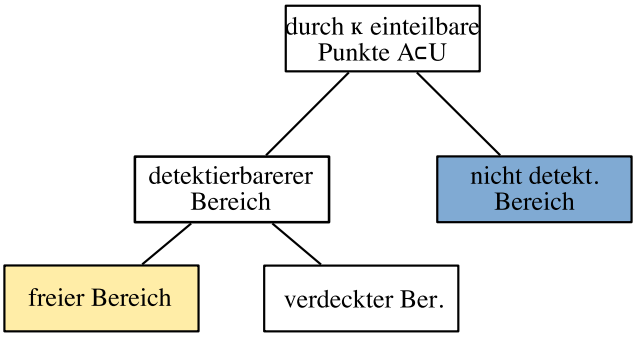

Abbildung 2.2: Einteilung der Umgebung in Bereiche

Die Kamera κ ordnet nach Einstellung der Kameraparameter aus E jedem Punkt in der Umgebung einen Wert aus S zu. Wie das Schaubild 2.2 zeigt, entstehen dadurch die drei Bereiche „frei“, „verdeckt“ und „nicht detektierbar“.

An dieser Stelle wird aus einer Kamera im eigentlichen Sinne (Sensor zur Bilderstellung) ein Instrument der Raumeinteilung. Dies geschieht durch ein Background-Subtraction--Verfahren. Die Kamera macht ein Referenzbild (im Sinne der Bilderstellung) von der Umgebung (Abbildung 2.3, erstes Bild), bevor ein Objekt des unbekannten Kollektivs die Sicherheitszone betritt. Die Menge A kann durch eine Kamera also zuerst in die Punkte, die auf dem Referenzbild der Kamera erscheinen (alle Punkte innerhalb des Sichtkegels ausgehend von der Kamera bis hin zur Oberfläche des ersten Objektes), und die übrigen eingeteilt werden. Diejenigen, die auf dem Referenzbild erscheinen, sind *detektierbar* und werden weiterhin unterschieden:

Dazu wird der Pixelwert eines Bildes zu einem späteren Zeitpunkt (Abb.2.3 zweites Bild) von dem Wert des Pixels abgezogen, das im Referenzbild an der gleichen Stelle liegt. Ist der Wert gleich null, so hat sich an der Szene nichts verändert, ist dies nicht der Fall ($= 0 \pm$ Tolerationsrauschen), so ist ein unbekanntes Objekt erschienen. Dieses spezielle Background-Subtraction-Verfahren nennt man auch „Differenzbild-Verfahren“. Der Bereich indem ein Objekt erschienen ist, wird *verdeckter* Bereich genannt. Dabei sei immer vorrausgesetzt, dass sich das unbekannte Kollektiv zu jeder Zeit vom Hintergrund abhebt (zum Beispiel farblich), so dass der Unterschied im Differenzbild erkennbar ist. Diese Bereichseinteilung wird im folgenden Abschnitt mathematisch formuliert.

Abbildung 2.3: Background-Subtraction-Verfahren: Erstes Bild ist Referenzbild, zweites Bild ist zu einem späteren Zeitpunkt aufgenommen worden, drittes Bild ist Differenzbild (Bilder am Lehrstuhl „AI 3“ verfügbar)

Definition 2.5
Sei κ eine Kamera des Kameranetzes, sei $e = (f, \rho, \phi) \in E$ fest. Sei $0 \leq \alpha \leq 2\pi$ der Öffnungswinkel der Kamera und sei $w^0 \in U$. Die Menge der sensorischen Werte einer Kamera bestehe aus den Elementen

$$S = \left\{\text{Frei } F, \text{Verdeckt } V, \text{Nicht-detektierbar } \overline{D}\right\}$$

1. Man sagt, w^0 erscheint *nicht auf dem Referenzbild* einer Kamera mit Einstellung e, wenn
$$\exists \lambda \in [0,1) \ : \ \left(f + \lambda \cdot (w^0 - f)\right) \notin \left(A \cap (T_0^e)^{-1}\left(cone_+\left(\frac{\alpha}{2}\right)\right)\right)$$

2. $\kappa(e, w^0) := \overline{D}$, wenn $w^0 \in A$ und w^0 ist nicht auf dem Referenzbild der Kamera

3. Ist 2. nicht der Fall, so heißt w^0 detektierbar $\kappa(e, w^0) = D$ mit $D = F \vee V$. Für die weitere Unterscheidung sei $w^e = T_0^e(w^0)$ die Abbildung des Punktes in Kamerakoordinaten und $P \subset D^1 = \{v^e \in \mathbb{R}^3 \mid ||v^e||_2 = 1\}$ das zugehörige Pixel $w^e \in P$.

   (a) Sei $p \in P$ der Punkt, an dem die Farbwertdifferenz gemessen wird. Man sagt, die *Farbwertdifferenz des Pixels P zweier Bilder dieser Kamera ist ungleich null*, wenn
   $$\exists u \in O_u,\ \lambda(u) \geq 0\ :\ f + \lambda(u) \cdot (p - f) = u$$

   (b) $\kappa(e, w^0) := V$ wenn die Farbwertdifferenz des Pixels ungleich null ist.

   (c) $\kappa(e, w^0) := F$ sonst

Man beachte an dieser Stelle, dass die in Definition 2.5 genannten Sensorwerte zu einer Kamera ohne Tiefensensor gehören. Eine Kamera mit Tiefensensor ermittelt zusätzlich zu der Farbwertdifferenz den Abstand des erschienenen unbekannten Kollektivs zum Kamerafokus. Ein Punkt innerhalb dieses Abstandes muss dann nicht $\kappa(e, w^0) = V$ zur Folge haben. Diese Anpassung verändert die weiteren Betrachtungen nicht, weswegen weiter von einer Kamera ohne Tiefensensor ausgegangen wird. Durch die obige Definition kann nun die Einteilung der Umgebung vorgenommen werden:

Definition 2.6
Es gelten die Voraussetzungen wie in Definition 2.5.

1. $B_{\overline{D}}^{\kappa(e,.)} := \left\{w^0 \in A \mid \kappa(e, w^0) = \overline{D}\right\} = A \setminus (T_0^e)^{-1}\left(cone_+\left(\frac{\alpha}{2}\right)\right)$
   heißt *nicht-detektierbarer Bereich* von κ mit Einstellung e.

2. $B_D^{\kappa(e,.)} := \left\{w^0 \in A \mid \kappa(e, w^0) = D\right\}$ heißt der *detektierbare Bereich* der Kamera.

3. $B_F^{\kappa(e,.)}(O_u) := \left\{w^0 \in B_D^{\kappa(e,.)} \mid \kappa(e, w^0) = F\right\}$ heißt der *freie Bereich* der Kamera.

4. $B_V^{\kappa(e,.)}(O_u) := \left\{w^0 \in B_D^{\kappa(e,.)} \mid \kappa(e, w^0) = V\right\}$ heißt der *verdeckte Bereich.*

Korollar 2.7
Aus Definition 2.5 folgt: $A = B_D^{\kappa(e,.)} \cup B_{\overline{D}}^{\kappa(e,.)}$, wobei $B_D^{\kappa(e,.)}$ und $B_{\overline{D}}^{\kappa(e,.)}$ disjunkt sind. Das heißt wiederum:
$$B_D^{\kappa(e,.)} = A \setminus B_{\overline{D}}^{\kappa(e,.)} = A \cap (T_0^e)^{-1}\left(cone_+\left(\frac{\alpha}{2}\right)\right)$$

Korollar 2.8
Nach den Definitionen 2.5 (3. Teil) und 2.6 gilt: $B_F^{\kappa(e,.)}(O_u)$ und $B_V^{\kappa(e,.)}(O_u)$ sind disjunkt und
$$B_F^{\kappa(e,.)}(O_u) \cup B_V^{\kappa(e,.)}(O_u) = B_D^{\kappa(e,.)}$$

Das erste Bild der Abbildung 2.4 zeigt eine Kamera ohne Tiefensensor mit dem Differenzbild aus 2.3 und die disjunkten Bereiche: detektierbar und nicht detektierbar (blau). Der detektierbare Bereich ist unterteilt in den freien Bereich (gelb) und den verdeckten Bereich

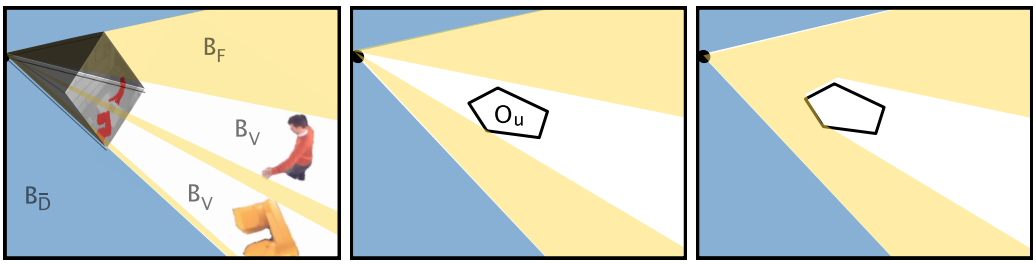

Abbildung 2.4: Farben wie in Abbildung 2.2; nicht detektierbarer Bereich: blau; verdeckter Bereich: weiß; freier Bereich: gelb; O_u: schwarz umrahmt

(weiß). Man kann erkennen, dass die verdeckten Bereiche durch die Pixel (rot) zustandekommen, deren absolute Farbwertdifferenz größer null gewesen ist. Diese Szene ist im zweiten Bild der Abbildung 2.4 abstrahiert abgebildet: Die Kamera wird hier durch einen Punkt dargestellt. Betrachtet man Kameras mit Tiefensensor, kann der Abstand zwischen einem Objekt und der Kamera gemessen werden, d.h. der Raum zwischen Objekt und Kamera wird als „frei“erkannt, wie im letzten Bild der Abbildung illustriert ist.

2.1.3 Eingrenzung der Objekte eines unbekannten Kollektivs

Betrachtet man mehrere unbekannte Objekte, so ist nicht bekannt, welches Objekt ein Pixel verdeckt. Bei Kameras ohne Tiefensensor kann außerdem nicht festgestellt werden, wie weit ein unbekanntes Objekt von der Kamera entfernt ist (Abbildung 2.5, erstes Bild). Es sind zwei Kameras nötig, um dieses Objekt in 2D gut einzugrenzen (Zweites Bild).

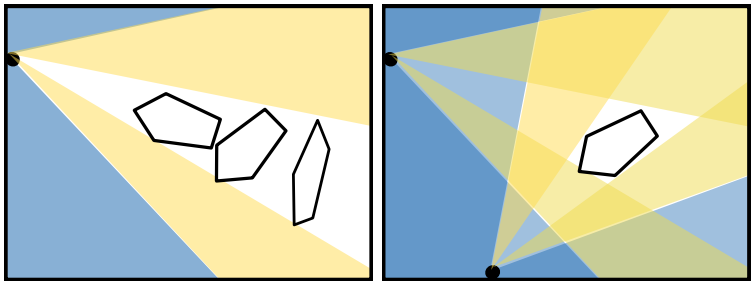

Abbildung 2.5: Farben wie in Abbildung 2.2; nicht detektierbarer Bereich: blau; verdeckter Bereich: weiß; freier Bereich: gelb; O_u: schwarz umrahmt

Nun wird das Modell des unbekannten Kollektivs aus der Bereichseinteilung der Menge A hergeleitet. Aus den Korollaren 2.7 und 2.8 ist bekannt, dass die durch die Kamera κ abbildbaren Punkte einer Umgebung durch die Abbildung disjunkt zerlegt werden können in

$$A = B_D^{\kappa(e,.)} \cup B_{\overline{D}}^{\kappa(e,.)} = B_F^{\kappa(e,.)}(O_u) \cup B_V^{\kappa(e,.)}(O_u) \cup B_{\overline{D}}^{\kappa(e,.)}$$

Zudem gilt außerdem die Generalannahme $U = A$ aus Abschnitt 2.1. Hält sich ein unbekanntes Objekt innerhalb der Sicherheitszone $Z \subset A \subset U$ auf, das heißt $Z \cap O_u \neq \emptyset$,

so kann man sicher sein, dass es nicht innerhalb des freien Bereichs jeder Kamera liegt; dies kann jedoch nicht konkretisiert werden. Zu dieser Annahme kommen wir, da im nicht detektierbaren Bereich keine Aussage über einen Aufenthalt möglich ist und der verdeckte Bereich durch ein unbekanntes Kollektiv entstanden ist. Dazu folgendes Lemma:

Lemma 2.9
Sei κ eine Kamera mit Einstellung $e = (f, \phi, \rho) \in E$, deren Pixel nur einen Punkt enthalten $P = \{p\}$, und sei $O_u \subset A$. Dann gilt

$$O_u \subset \left(B_V^{\kappa(e,.)}(O_u) \cup B_{\overline{D}}^{\kappa(e,.)} \right)$$

Beweis:

Wähle $u^0 \in O_u$ beliebig, so gilt $u^0 \in A = B_D^{\kappa(e,.)} \cup B_{\overline{D}}^{\kappa(e,.)}$.
Da $B_D^{\kappa(e,.)} \cap B_{\overline{D}}^{\kappa(e,.)} = \emptyset$, folgt daraus $u^0 \in B_D^{\kappa(e,.)} \ \vee \ u^0 \in B_{\overline{D}}^{\kappa(e,.)}$.

- *Ist $u^0 \in B_{\overline{D}}^{\kappa(e,.)}$, so gilt auch*

$$u^0 \in \left(B_V^{\kappa(e,.)}(O_u) \cup B_{\overline{D}}^{\kappa(e,.)} \right).$$

- *Ist $u^0 \in B_D^{\kappa(e,.)}$:*
  Nach Definition 2.6 gilt $\kappa(e, u^0) = D$ mit $D = F \vee V$.
  Sei $u^e = T_0^e(u^0)$ die Abbildung des Punktes in Kamerakoordinaten, sei $\lambda_{u^0} \geq 0$ so gewählt, dass $f + \lambda_{u^0} \cdot (u^e - f) = u^0$.
  Weiterhin sei $P \subset D^1$ das zugehörige Pixel $u^e \in P$.
  Da $P = \{p\}$, wird an $u^e = p$ die Farbwertdifferenz gemessen.
  Wähle nun aus Definition 2.5 Teil 2.a $u = u^0$ und $\lambda(u) = \lambda_{u^0}$, so gilt mit Definition 2.5 Teil 2.b: $\kappa(e, u^0) = V$.

$$\Rightarrow u^0 \in B_V^{\kappa(e,.)}(O_u) \text{ (Definition 2.6)}$$

$$\Rightarrow u^0 \in \left(B_V^{\kappa(e,.)}(O_u) \cup B_{\overline{D}}^{\kappa(e,.)} \right)$$

Aus diesen beiden Fallunterscheidungen folgt

$$O_u \subset \left(B_V^{\kappa(e,.)}(O_u) \cup B_{\overline{D}}^{\kappa(e,.)} \right)$$

□

Diese Einteilung kann für alle Kameras mit Einstellungen $e_i \in E, \ i = 1, ..., n$ vorgenommen werden $\kappa_i(e_i, .), \ i = 1, ..., n$, deswegen wird der Aufenthaltsort zunehmend eingeschränkt. Zu diesem eingeschränkten Bereich wird im folgenden Abschnitt der Abstand berechnet.

Korollar 2.10
Aus Lemma 2.9 folgt für ein Kameranetz aus den Kameras $\kappa_i,\ i = 1, ..., n$ mit Einstellungen $e_i \in E,\ i = 1, ..., n$:

$$\begin{aligned} O_u &\subset \bigcap_{i=1}^{n} \left(B_V^{\kappa_i(e_i,.)}(O_u) \cup B_{\overline{D}}^{\kappa_i(e_i,.)} \right) = \bigcap_{i=1}^{n} \left(A \setminus B_F^{\kappa_i(e_i,.)}(O_u) \right) \\ &= A \setminus \left(\bigcup_{i=1}^{n} B_F^{\kappa_i(e_i,.)}(O_u) \right) \end{aligned}$$

O_u ist außerhalb aller freien Bereiche, deswegen sagt man O_u wird *eingegrenzt von den vereinigten freien Bereichen* und schreiben der Kürze wegen:

$$B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u) := \bigcap_{i=1}^{n} \left(B_V^{\kappa_i(e_i,.)}(O_u) \cup B_{\overline{D}}^{\kappa_i(e_i,.)} \right)$$

Eigentliches Ziel war es, den Abstand zwischen unbekanntem Kollektiv und Gefahrenpunkten gut zu schätzen, damit zum richtigen Zeitpunkt Sicherheitsschritte in die Wege geleitet werden können. Als Modell dient dazu die Eingrenzung des Kollektivs durch die vereinigten freien Bereiche.

In Abschnitt 2.1.1 wird angenommen, dass Sicherheitsschritte nur dann in die Wege geleitet werden, wenn ein unbekanntes Objekt die Sicherheitszone tatsächlich betreten hat, also wenn $O_u \cap Z \neq \emptyset$. Dann gilt für alle $u \in (O_u \cap Z)$ auch $u \in B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u)$ (siehe Beweis zu 2.9) und damit :

$$\left(Z \cap B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u) \right) \neq \emptyset$$

Damit ist das Modell des unbekannten Kollektivs nicht leer, das heißt, es gibt immer einen Punkt, zu dem der *geschätzte Abstand*

$$d\left(G, Z \cap B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u)\right) = \min\left\{ d(g,u) | g \in G, u \in \left(Z \cap B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u) \right) \right\}$$

berechnet werden kann. Geschätzer und tatsächlicher Abstand motivieren das Optimierungsproblem in Abschnitt 2.1.4.

2.1.4 Mathematische Formulierung

Gesucht ist eine Einstellung der Kameras, bei der der tatsächliche Abstand von gefährlichen Punkten aus G zum unbekannten Kollektiv O_u von dem Abstand zum Modell von O_u möglichst wenig abweicht. Der Unterschied zwischen dem geschätzen Abstand, abgekürzt durch $d(g, B_{\overline{D} \vee V})$, und dem tatsächlichen, abgekürzt durch $d(g, O_u)$, ist in Abbildung 2.6 anhand eines gefährlichen Punktes g und zwei Kameras (schwarze Punkte) illustriert. Die freien Bereiche (gelb) schließen manche Punkte für die Erstellung des Modells des unbekannten Kollektivs aus, d.h. die weißen Bereiche innerhalb der Sicherheitszone ergeben das Modell, zu dem wiederum eine der beiden Distanzen berechnet wird.

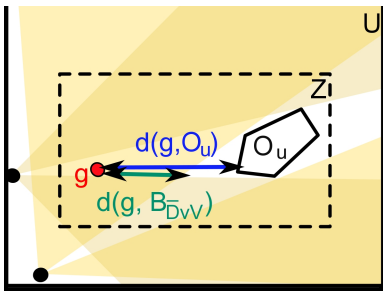

Abbildung 2.6: Zwei Kameras (schwarze Punkte) mit freien Bereichen (gelb), Modell des unbekannten Kollektivs (nicht gelb= weiß), gefährlicher Punkt $g \in G$ (rot), Abstand zum Objekt des unbekannten Kollektivs (blau), Abstand zum Modell (grün)

Man erinnere sich an den Oberflächenparameter a des unbekannten Kollektivs $O_u(a)$. Die Verteilung von a sei P. Seien $0 \leq \alpha_i \leq 2\pi$ die Öffnungswinkel der Kameras $i = 1, ..., n$. Die Einstellung einer Kamera κ_i, $i = 1, ..., n$ sei bezeichnet als $e_i \in E$.

Optimierungsproblem *Grundproblem (GP)*

Minimiere:

$$\int_{a \in \mathbb{R}^k} \left(d(G, O_u(a)) - d\left(G, Z \cap B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u(a))\right)\right)^2 dP(a)$$

unter den Nebenbedingungen:

$$e_i \in E \qquad \forall i \in \{1, ..., n\}$$

Im Folgenden wird auf diese Formulierung immer wieder zurückgegriffen. Dabei werden die vereinigten freien Bereiche mit Hilfe logischer Implikationen abgeändert und Aspekte der gefährlichen Punkte hinzugefügt.

2.2 Hindernisse in der Umgebung

Im letzten Kapitel wurde erörtert, wie das Kameranetz die Objekte des unbekannten Kollektivs durch seine vereinigten freien Bereiche eingrenzt, so dass der Abstand abgeschätzt werden kann. Nun soll das Ganze in einem Zeitabschnitt $[t_0, t_1]$ betrachetet, wobei t_0 der Zeitpunkt ist, zu dem das Referenzbild gemacht wird. Der Oberflächenparameter $a(t)$ ist von nun an also von der Zeit abhängig, ebenso wie seine Verteilung $P(a(t)) \in [0, 1]$ und die dadurch induzierte Verteilung des unbekannten Kollektivs. Um das unbekannten Kollektiv $O_u(a(t))$ abzukürzen, wird es weiterhin auch O_u genannt.

In diesem Kapitel wird zusätzlich die Tatsache berücksichtigt, dass Hindernisse die Sicht der Kameras teilweise blockieren. Nach der Definition von Hindernisobjekten wird dazu auf die Veränderungen der Umgebung und deren Bestandteilen eingegangen und die Beeinträchtigung der Eingrenzung des unbekannten Kollektivs diskutiert.

2.2.1 Veränderungen durch Anwesenheit von Hindernissen

Hindernisobjekte werden in verschiedene Kategorien eingeteilt. Die Grundeinteilung erfolgt in Hinblick auf die Unterscheidung, ob ein solches Objekt im Referenzbild vorkommt. Je nach Mächtigkeit des Background-Subtraction-Verfahrens zählen zu den im Referenzbild vorkommenden Hindernissen mehr als nur statische Objekte. In der Betrachtung dieses Problems sind die Hindernisse im Referenzbild jedoch nur auf die statischen beschränkt. Die Hindernisobjekte werden somit in dieser Betrachtung durch das Kriterium unterschieden, ob sie konstant mit der Zeit sind.

Definition 2.11
Sei U eine Umgebung. Sei t_0 der Zeitpunkt, zu dem das Referenzbild gemacht wird. Ein Hindernisobjekt ist eine zusammenhängende Teilmenge $O(t) \subset U$, abhängig von der Zeit $t \geq t_0$, die für unbekannte Objekte nicht penetrierbar ist, das heißt

$$O(t) \cap O_u = \emptyset$$

Man unterscheidet zwei Arten. Sei κ eine Kamera und E ihre möglichen Einstellungen.

1. Das Hindernisobjekt heißt *statisches Hindernisobjekt*, wenn es konstant mit der Zeit t ist

   $$O(t) \equiv O(t_0), \quad \forall t > t_0$$

   und wenn es ein $s \in O(t_0)$ (um genauer zu sein, liegt s auf dem Rand $^{\partial}O(t_0)$) und eine Kameraeinstellung $e \in E$ gibt, so dass s (nach Definition 2.5 Teil 1) auf dem Referenzbild erscheint. Wir schreiben dann

   $$O(t) \equiv O$$

2. Ist dies nicht der Fall, so heißt das Hindernisobjekt *dynamisch*.

Es wird von nun an von den statischen Hindernissen O_{s_j} mit $j = 1, ..., m$ und dynamischen $O_{d_j}(t)$ mit $j = (m+1), ..., M$ ausgegangen. Die Objekte einer Kategorie können im Folgenden auch mit den Begriffen *statisches Kollektiv* $O_s = \bigcup_{j=1}^{m} O_{s_j}$ und *dynamisches Kollektiv* $O_d(t) = \bigcup_{j=m+1}^{M} O_{d_j}(t)$ bezeichnet werden. Im folgenden Abschnitt werden die Auswirkungen von zwei Veränderungen näher betrachtet, die dem freien Bereich durch die Anwesenheit dieser Objekte wiederfahren:

Bei statischen Objekten ist die Oberfläche konstant und bekannt. Dies ist etwa bei einem Regal oder am Boden verankerten Tisch der Fall. Zu den statischen Hindernissen zählen aber auch sichtblockierende Begrenzungen, wie etwa Boden und Wände. Nach obiger Definition stehen statische Objekte bereits in der Umgebung, wenn ein Referenzbild von den Kameras gemacht wird. Das heißt, die Farbwertdifferenz wird weder durch das statische Kollektiv noch durch Objekte des unbekannten Kollektivs dahinter verändert, wohl aber durch welche davor. Der detektierbare Bereich muss dementsprechend, wie in Abbildung 2.7 illustriert, um die Punkte erweitert werden, die aus Sicht der Kamera „hinter“ den Objekten liegen, welche potentiell auf dem Referenzbild vorkommen können (in diesem Fall sind das nur statische).

Zu diesem Zweck wird nun die Menge der durch eine Kamera abbildbaren Punkte $A \subset U$ verwendet: A enthält von nun an nur die Punkte der Umgebung, die außerhalb des statischen Kollektivs liegen, welche potentiell im Referenzbild vorkommen können. Dadurch greift die Definition eines nicht detektierbaren Punktes 2.5 (Teil 2.) für alle Punkte hinter dem statischen Kollektiv.

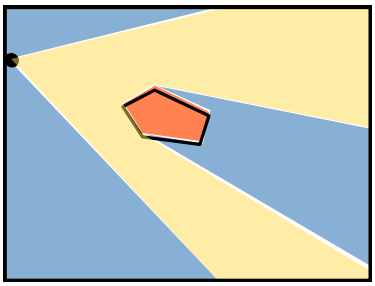

Abbildung 2.7: Nicht detektierbarer Bereich (blau) bei statischem Hindernis (rötlich)

Definition 2.12
Gegeben seien die Voraussetzungen aus Definition 2.11. Die abbildbaren Punkte $A \subset U$ verändern sich durch die Anwesenheit eines statischen Kollektivs $O_s \subset U$ zu

$$A := U \setminus O_s$$

Dynamische Hindernisse bewegen und deformieren sich wie Objekte des unbekannten Kollektivs, aber ihre Oberfläche zur Zeit t ist bekannt. Ein solches Hindernisobjekt könnte etwa ein Roboter auf einer vordefinierten Bahn sein. Würde man ein dynamisches Hindernis ins Referenzbild aufnehmen, würde es, sobald es sich bewegt, nicht nur an der neuen

Position verdeckte Pixel erzeugen, sondern auch an der ursprünglichen Position. Das Hindernis darf dementsprechend nicht im Referenzbild enthalten sein, kann aber eventuell in einem der folgenden Bilder zum Vorschein kommen, siehe Abbildung 2.3 (Zweites und drittes Bild). Es schränkt nicht wie die statischen Hindernisse den detektierbaren Bereich ein, sondern wie die unbekannten Objekte den freien Bereich. Eine Kamera kann daher weder vor dem dynamischen Hindernis ein unbekanntes Objekt detektieren noch dahinter. Dies ist in Abbildung 2.8 dargestellt.

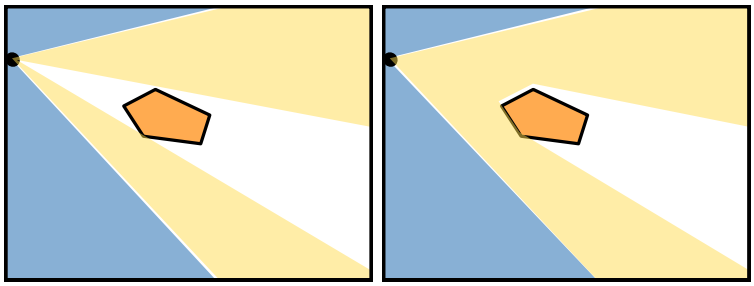

Abbildung 2.8: Zusätzlicher verdeckter Bereich (weiß) bei dynamischem Hindernis (orange) zu einer festen Zeit ohne (Bild 1) und mit (Bild 2) Abstandssensor

Definition 2.5 Teil 3 Ergänzung

Gegeben seien die Vorraussetzungen aus Definition 2.5 (Teil 3. a)) und sei zusätzlich $O_d(t) = \bigcup_{j=(m+1)}^{M} O_{d_j}(t) \subset U$. Sei $p \in P$ der Punkt des Pixels P, an dem die Farbwertdifferenz gemessen wird.
Die *Farbwertdifferenz des Pixels zweier Bilder dieser Kamera ist ungleich null*, wenn

$$\exists u \in (O_u \cup O_d(t)) \;\; \exists \lambda(u) \geq 0 \;\; : \;\; f + \lambda(u) \cdot (p - f) = u.$$

Bemerkung 2.13
Die Definition 2.6 und das Korollar 2.10 des verdeckten und freien Bereichs hängen über die veränderte Definition 2.5 mit den sich in der Zeit t verändernden dynamischen Hindernisobjekten zusammen. Um das zu kennzeichnen, werden folgende Bezeichnungen benutzt:

$B_F^{\kappa(e,.)}(O_u, t) := \left\{ w^0 \in B_D^{\kappa(e,.)} \mid \kappa(e, w^0) = F \right\}$ freier Bereich, abhängig von Zeit t

$B_V^{\kappa(e,.)}(O_u, t) := \left\{ w^0 \in B_D^{\kappa(e,.)} \mid \kappa(e, w^0) = V \right\}$ verdeckter Bereich, abhängig von t

$B_{\overline{D} \vee V}^{\kappa_{i=1\ldots n}(e_i,.)}(O_u, t) := \bigcap_{i=1}^{n} \left(B_V^{\kappa_i(e_i,.)}(O_u, t) \cup B_{\overline{D}}^{\kappa_i(e_i,.)} \right)$ Eingrenzung des unbek. Objektes

2.2.2 Auswirkungen der Veränderungen

Das Modell eines unbekannten Kollektivs in einer Umgebung mit Hindernissen herzuleiten, stellt einen komplexeren Vorgang dar als jener in Abschnitt 2.1.3. Die Definitionen, die zur Berücksichtigung von Hindernissen gemacht wurden, verändern die Bereiche der Kameras, nicht aber die Aussage von Lemma 2.9. Aus der Tatsache, dass Hindernisobjekte für

Objekte des unbekannten Kollektivs nicht penetrierbar sind, ergibt sich sogar noch eine weitere Einschränkung des Bereichs, in dem sich ein unbekanntes Objekt aufhalten kann:

Lemma 2.9 Ergänzung

Sei κ eine Kamera mit Einstellung $e = (f, \phi, \rho) \in E$, deren Pixel nur einen Punkt enthalten $P = \{p\}$.
Seien zusätzlich $O_s \subset U$ ein statisches Kollektiv, das A nach Definition 2.12 einschränkt und $O_d(t) = \bigcup_{j=m+1}^{M} O_{d_j}(t) \subset U$ ein dynamisches und die daraus folgende Erweiterung von Definition 2.5 Teil 3 und sei $O_u \subset U$, dann gilt:

$$O_u \ \subset \ \left(B_V^{\kappa(e,.)}(O_u, t) \cup B_{\overline{D}}^{\kappa(e,.)}\right) \setminus O_d(t)$$

Bei mehreren Kameras $\kappa_i, \ i = 1, ..., n$ mit Einstellungen $\kappa_i(e_i, .), \ e_i \in E$ gilt:

$$O_u \ \subset \ B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u, t) \setminus O_d(t)$$

Wir schreiben kurz:

$$B_{\overline{D} \vee V \setminus O_{s/d}}^{\kappa_{i=1...n}(e_i,.)}(O_u, t) := B_{\overline{D} \vee V}^{\kappa_{i=1...n}(e_i,.)}(O_u, t) \setminus O_d(t)$$

Beweis:

Wähle $u^0 \in O_u$ beliebig, dann gibt es zwei Folgerungen:

1. $\Rightarrow$ $u^0 \notin O_s$ *(Definition der Hindernisse 2.11)*

$\Rightarrow$ $u^0 \notin U \setminus A$ *(Definition der Menge A 2.12)*
$\Rightarrow$ $u^0 \in A = B_D^{\kappa(e,.)} \cup B_{\overline{D}}^{\kappa(e,.)}$
$\Rightarrow$ $u^0 \in \left(B_V^{\kappa(e,.)}(O_u, t) \cup B_{\overline{D}}^{\kappa(e,.)}\right)$ *(Beweis wie zu Lemma 2.9)*

2. $\Rightarrow$ $u^0 \notin O_{d_j}(t), \ \forall j = m+1, ..., M$ *(Definition der Hindernisse 2.11)*

$$\Rightarrow u^0 \notin \bigcup_{j=m+1}^{M} O_{d_j}(t) = O_d(t)$$

Aus 1 und 2 folgt: $u^0 \in \left(B_V^{\kappa(e,.)}(O_u, t) \cup B_{\overline{D}}^{\kappa(e,.)}\right) \setminus O_d(t)$ $\square$

Die Verfeinerung der Bereichseinteilung kann in Abbildung 2.9 nachvollzogen und mit dem Szenario aus Abbildung 2.10 verglichen werden. Zunächst werden die abbildbaren Punkte A der Umgebung durch das statische Kollektiv eingeschränkt und danach eingeteilt in detektierbar und nicht detektierbar. Je nach dem, ob ein dynamisches Objekt bezüglich einer Kameraperspektive hinter oder vor einem statischen liegt, ist dieses Teil

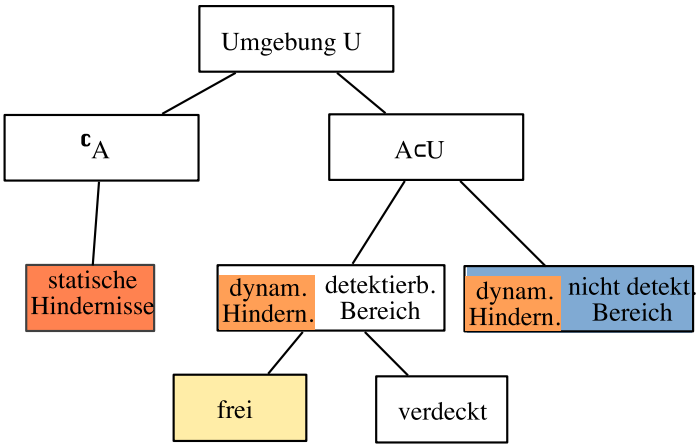

Abbildung 2.9: Einteilung der Umgebung in Bereiche

des nicht detektierbaren oder detektierbaren Bereichs. Die gleiche Unterscheidung kann mit unbekannten Objekten gemacht werden. Beide Objektarten, unbekannt und dynamisch, hinterlassen nur im detektierbaren Bereich Verdeckungen. Es ist aus dem Lemma 2.9 (Ergänzung) am Ende des Abschnittes 2.2.1 bekannt, dass das Modell des unbekannten Kollektivs weder statische beziehungsweise dynamische Hindernisse enthält, noch den freien Bereich einer Kamera.

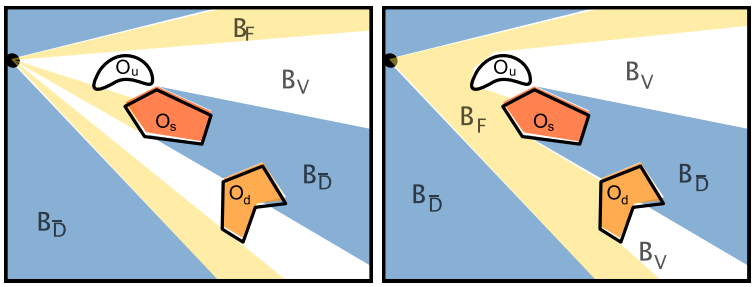

Abbildung 2.10: Mögliches Szenario: Nicht detektierbar hinter statischem (blau), verdeckt hinter dynamischem und unbekanntem Objekt (weiß), frei (gelb), ohne (Bild 1) und mit (Bild 2) Abstandssensor

2.2.3 Mathematische Formulierung

Gesucht sind wiederum Kameraplatzierung und -orientierung, so dass die Abstandsabweichung zwischen dem tatsächlichen Abstand G zu O_u und dem Abstand zwischen G und der Eingrenzung von O_u durchschnittlich und erwartungsgemäß möglichst klein ist. Im Modell des unbekannten Kollektivs wird hier berücksichtigt, dass Hindernisse in der

Umgebung sind. Es seien dazu Vorraussetzungen gegeben wie in Abschnitt 2.1.4.

Sei t_0 der Zeitpunkt des Referenzbildes und seien weitere Bilder der Kameras zu einem Zeitpunkt $t \in [t_0, t_1]$ zu betrachten. Sei die Verteilung des Oberflächenparameter $P(a(t)) \in [0, 1]$ des unbekannten Kollektivs abhängig von der Zeit. Seien zusätzlich ein statisches Kollektiv O_s und ein dynamisches $O_d(t)$ gegeben, das sich mit der Zeit t verändert.

Optimierungsproblem *Erweitertes Problem mit Hindernissen (EPH)*

Minimiere:

$$\frac{1}{t_1 - t_0} \int\limits_{t \in [t_0, t_1]} \int\limits_{a \in \mathbb{R}^k} \left(d(G, O_u(a(t))) - d\left(G, Z \cap B^{\kappa_{i=1\ldots n}(e_i,.)}_{\overline{D} \vee V \setminus O_{s/d}}(O_u(a(t)), t)\right)\right)^2 dP(a(t))\, dt$$

unter den Nebenbedingungen:

$$e_i \in E \qquad \forall i \in \{1, ..., n\}$$

2.3 Auftretende Schwierigkeiten im Zusammenhang mit dem Roboter

Möchte man den Abstand zwischen einem Menschen als Objekt des unbekannten Kollektivs und einem Roboter mit einem Kameranetz kontrollieren, so stellt man fest: Der Roboter ist gleichzeitig ein dynamisches Hindernis und eine Gefahrenquelle für den Menschen.

$$\text{Œ}\ G(t) = O_{d_1}(t)$$

In diesem Kapitel wird zunächst versucht mit einem Beispiel die Frage zu klären, warum dies ein Problem darstellt. Anschließend wird eine mögliche Lösung des Problems vorgestellt.

Durch diese Aufgabenstellung werden zwei Veränderungen evident. Die erste Veränderung stellt noch keine Schwierigkeit dar: Die Gefahrenquellen sind nicht mehr statisch, sondern bewegen sich dynamisch mit der Zeit t. Die zweite Veränderung hingegen zieht eine Umformulierung der Zielfunktion nach sich:

Stelle man sich eine Kamera als Lichtquelle vor, so werfen statische sowie dynamische Hindernisse einen Schatten in den Lichtkegel. Bei Statischen besteht jener aus nicht-detektierbarem Bereich und bei Dynamischen aus verdecktem Bereich (siehe Abbildungen 2.7 und 2.8). Ohne weitere Informationen muss in diesen Bereichen ein Teil des unbekannten Kollektivs vermutet werden. Wird der Abstand zu einem dieser Hindernisse überwacht, so muss man aus Sicherheitsgründen davon ausgehen, dass das unbekannte Kollektiv das Hindernis berührt, also der Abstand der beiden gleich null ist.

Dieses einfache Beispiel zeigt, dass der Abstand zwischen dem Modell des unbekannten Kollektivs und einem Hindernis als Gefahrenquelle oft null sein kann. Selbst bei mehreren Kameras können diese Bereiche an dem Hindernisobjekt, das die Gefahrenquelle darstellt, vorkommen. Man denke an eine Kugel, die nie durch einen Polyeder dargestellt, also auch nie durch endlich viele Kameras eingegrenzt, werden kann. Eine kleine Abhilfe schaffen in [11] die Plausibilitätsbedingungen. Sie verwenden Eigenschaften der Objekte des unbekannten Kollektivs, wie Größe oder Volumen, um manche Bereiche logisch auszuschließen, in denen sich unbekannte Objekte aus Sicht der Kameras aufhalten könnten und die sich ineffizient gegenüber der Berechnung des Abstandes herausstellen könnten. So sind dann nicht mehr unendliche viele Kameras für die Approximation der Kugel nötig, da die überschüssigen Bereiche wegrationalisiert werden, sofern sie das Höchstvolumen nicht erreicht haben.

Um die Plausibilitätsbedingungen zu definieren, sei $\mathbb{P}$ die Menge aller größten zusammenhängenden Untermengen des Modells $B^{\kappa_{i=1\ldots n}(e_i,.)}_{\overline{D}\vee V\setminus O_{s/d}}(O_u,t)$ des unbekannten Kollektivs.

Definition 2.14
$\mathbb{P} :=$

$$\left\{H \subset B^{\kappa_{i=1\ldots n}(e_i,.)}_{\overline{D}\vee V\setminus O_{s/d}}(O_u,t) \text{ zus.häng. } \middle|\ \nexists K \subset B^{\kappa_{i=1\ldots n}(e_i,.)}_{\overline{D}\vee V\setminus O_{s/d}}(O_u,t) \text{ zus.häng. mit } H \subsetneq K\right\}$$

Bemerkung 2.15
Es gilt $\bigcup_{H\in\mathbb{P}} H = B^{\kappa_{i=1\dots n}(e_i,.)}_{\overline{D}\vee V\setminus O_{s/d}}(O_u,t)$

Definition 2.16

1. Mittels der *charakteristischen Eigenschaft* $c:\mathbb{P}\to\{0,1\}$ des unbekannten Kollektivs

$$c(H)=\begin{cases} 0 & O_u\cap H=\emptyset \\ 1 & \text{sonst} \end{cases},$$

   kann überprüft werden, ob $H\in\mathbb{P}$ unbekannte Objekte beinhalten kann.

2. Sei c eine charakteristische Eigenschaft. Eine *Plausibilitätsbedingung* p_c ist eine Funktion, die alle Bereiche aus $\mathbb{P}$ auf die Eigenschaft c überprüft:

$$p_c:\bigcup_{H\in\mathbb{P}} H\mapsto \bigcup_{H\in\mathbb{P},c(H)=1} H$$

3. Mehrere Plausibilitätsbedingungen $p_{c_1},\dots,p_{c_C}$ werden zusammengefaßt zu der gesamten Plausibilitätsabprüfung:

$$p_{1,\dots,C}=(p_{c_1}\circ\dots\circ p_{c_C}):\bigcup_{H\in\mathbb{P}} H\mapsto \bigcup_{\substack{H\in\mathbb{P}\\ c_1(H)=1\\ \vdots\\ c_C(H)=1}} H$$

Dadurch wird das Optimierungsproblem EPH noch um die Möglichkeit erweitert, bekannte Informationen über das unbekannte Kollektiv zu nutzen, um die Optimierung zu präzisieren.

Optimierungsproblem *Erweitertes Problem Plausibilität (EPP)*

Minimiere:

$$\frac{1}{t_1-t_0}\int_{t\in[t_0,t_1]}\int_{a\in\mathbb{R}^k}\left(d(G(t),O_u(a(t)))-d\left(G(t),p_{1,\dots,C}\left(Z\cap B^{\kappa_{i=1\dots n}(e_i,.)}_{\overline{D}\vee V\setminus O_{s/d}}(O_u(a(t)),t)\right)\right)\right)^2 dP(a(t))dt$$

unter den Nebenbedingungen:

$$e_i\in E\qquad \forall i\in\{1,\dots,n\}$$

Im letzten Kapitel ist die Optimierung eines Kameranetzes verfeinert worden, welches den Abstand zwischen dem unbekannten Kollektiv und Gefahrenquellen beobachten soll. Inzwischen werden nicht nur sichtblockierende Hindernisse und sich bewegende Gefahrenquellen berücksichtigt, Gefahrenquellen können nun auch selbst Hindernisse sein. Im folgenden Kapitel wird die numerische Umsetzung dieses Problems besprochen.

Kapitel 3

Numerische Umsetzung

In Kapitel 2 wird das EPP in drei Schritten erarbeitet. Es handelt sich dabei um ein Problem der Form

$$\text{Minimiere } F(x) \text{ unter den Nebenbedingungen: } x \in X$$

wobei die *Zielfunktion* F die Einstellung x eines Kameranetzwerkes bewertet. Dabei setzt sich die Einstellung des Netzwerkes aus den einzelnen Einstellungen der Kameras zusammen $X = E^n$. Ein *Löser* des Problems sucht sukzessiv nach Netzwerkeinstellungen, die besser als die aktuelle bewertet werden, also einen kleineren Zielfunktionswert haben.

In diesem Kapitel wird das EPP numerisch umgesetzt. Die Umsetzung erfolgt in zwei Schritten. Zunächst wird die Implementierung der Zielfunktion in Abschnitt 3.1 besprochen, die die Auswahl des Lösers im Anschluss (vgl. Abschnitt 3.2) deutlich eingrenzt. Im letzt genanntem Abschnitt wird auch die Wirkunsweise des ausgewählten Lösers dargelegt. Anschließend werden Verbesserungen der Implementierung in Abschnitt 3.3 mit Hilfe des Rechenaufwandes diskutiert.

3.1 Umsetzung der Zielfunktion

In diesem Abschnitt wird eine mögliche Implementierung der Zielfunktion angesprochen. Dazu wird diese zuerst zerlegt. Die Zerlegung bildet ein Konzept, mit Hilfe dessen der exakte Aufbau der Klassen zu diskutieren ist.

Zuerst wird die Zerlegung der Zielfunktion betrachtet. Seien dazu in der Zeit $t \in [t_0, t_1]$ die Gefahrenpunkte der Menge $G(t) \subset Z$ gegeben und sei $O_u(a(t)) \subset U$ ein unbekanntes Kollektiv, so dass für alle $a(t)$ mit $P(a(t)) > 0$ gilt

$$O_u(a(t)) \cap Z \neq \emptyset.$$

Sei $B^{\kappa_{i=1,...,n}(e_i,.)}_{\overline{D} \vee V \setminus O_{s/d}}(O_u(a(t)), t)$ der Bereich, in dem die Kameras κ_i $i = 1, ..., n$ mit der Einstellung $\kappa_i(e_i, .), e_i \in E$ unbekannte Objekte vermuten, also das Modell des unbekannten Kollektivs. Sei außerdem $p_{1,...,C}$ eine Plausibilitätsabprüfung.

Man stelle zuerst fest, dass die Zielfunktion eine Komposition einer Abstandsberechnung d zu einer durch Kameraeinstellungen $(e_1, ..., e_n)$ und der Plausibilitätsabprüfung $p_{1,...,C}$ aus dem Kapitel 2.3 veränderbaren Menge ist. Die Veränderung der Menge durch Kameraeinstellungen beschreibe die Funktion v:

$$\begin{aligned} v: \quad E^n \times \mathbb{R}^k \times [t_0, t_1] &\to 2^Z \\ ((e_1, ..., e_n), a(t), t) &\mapsto \left(Z \cap B^{\kappa_{i=1,...,n}(e_i,.)}_{\overline{D} \vee V \setminus O_{s/d}}(O_u(a(t)), t) \right) \end{aligned}$$

$$\begin{aligned} d_{G(t)}: \quad 2^Z &\to \mathbb{R} \\ A \subset Z &\mapsto \min\{d(g, a) \mid g \in G, a \in A\} \end{aligned}$$

$F : E^n \to \mathbb{R}$, $(e_1, ..., e_n) \mapsto$

$$\frac{1}{t_1 - t_0} \int\limits_{t \in [t_0, t_1]} \int\limits_{a \in \mathbb{R}^k} \left[d_{G(t)}(O_u(a(t))) - \left(d_{G(t)} \circ p_{1,...,C} \circ v \right) ((e_1, ..., e_n), a(t), t) \right]^2 dP(a(t))\, dt \quad (3.1)$$

Die Zerlegung der Zielfunktion in Formel (3.1) dient zur Strukturierung der Klassen, die zur Lösung des Problems verwendet werden. Die Klasse „ObjectiveFunction“ fügt alles zusammen, das zur Berechnung des Zielfunktionswertes nötig ist. Dabei ist die Simulation der Veränderung der Bereiche v in Abhängigkeit der Kameraeinstellungen in der Klasse „SightAnalysis“ ausgelagert. Im Folgenden werden diese beiden Klassen genauer betrachtet.

In den Abbildungen A.1 und A.2 im Anhang A wird ein Überblick über die wichtigsten Klassen gegeben, die zur Implementierung hergenommen worden sind. Den erstellten Klassen liegt ein Geometrie-Paket Herrn Dipl. Inf. Stefan Kuhn zu Grunde, in dem sowohl die Angabe von Punkten und Vektoren eines Vektorraumes, als auch die von geometrischen Figuren wie einer Kugel oder aber auch die Einteilung des Raumes erleichtert werden.

3.1.1 „ObjectiveFunction“ - Bestimmen der Distanzabweichung

In den nun folgenden Ausführungen wird die Distanzabweichung und die damit verbundenen Klassen der Implementierung bestimmt. Um die Zielfunktion numerisch umsetzen zu können, müssen zwei Diskretisierungen vorgenommen werden. Die erste Diskretisierung betrifft die beiden Parameter a und t, denn das numerische Integrieren kann nicht über den ganzen Definitionsbereich der beiden geschehen. Die zweite Diskretisierung entsteht aus der Tatsache, dass eine überabzählbare Menge (wie das unbekannte Kollektiv $O_u(a(t))$ oder sein Modell $(p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t))$ nicht in endlicher Zeit abgesucht und auf Inklusionen beziehungweise Verdeckungen untersucht werden kann. Es wird dafür eine endliche Menge M an Punkten (in 3.1.2) vorgegeben, die abgearbeitet wird. All dies ist umgesetzt in der Klasse „ObjectiveFunction“ und wird zusammen mit deren Unterklassen im Folgenden detailliert dargestellt.

Zur Berechnung des Zielfunktionswertes werden die Abstandsabweichungen aller Zeiten $t \in [t_0, t_1]$ und aller Oberflächenparameterwerte $a(t) \in \mathbb{R}^k$ gemittelt. Numerisch setzt man dies durch eine gewichtete Summe über diskrete Zeitschritte $t_h \in [t_0, t_1]$ $h = 1, ..., H$ und eine diskrete Verteilung $P(a_k(t_h)) \in [0, 1]$ $k = 1, ..., K$ um. Aus der Zielfunktion (3.1) entsteht dann die Funktion:

$(e_1, ..., e_n) \mapsto$

$$\frac{1}{t_1 - t_0} \sum_{h=1}^{H} \sum_{k=1}^{K} P(a_k(t_h)) \cdot \left[d_{G(t_h)}(O_u(a_k(t_h))) - (d_{G(t_h)} \circ p_{1,...,C} \circ v)\,((e_1, ..., e_n), a_k(t_h), t_h) \right]^2 \tag{3.2}$$

Ein diskreter Schritt der Bahn $O_d(t_h)$ des dynamischen Kollektivs wird *Konfiguration* genannt. Die durch P induzierte diskrete Verteilung des unbekannten Kollektivs enthält die *Ereignisse* $O_u(a_k(t_h))$. Um das Programm etwas zu vereinfachen, werden nicht alle $H \cdot K$ Ereignisse gespeichert, sondern weniger, nämlich alle nicht-kongruenten $\overline{a_l}$ $l = 1, ..., L$.

Beispiel 3.1

$h = 1$: Ereignisse $a_1(1), a_2(1), a_3(1)$

$h = 2$: Ereignisse $a_1(2), a_2(2), a_3(2)$, wobei $a_2(1) = a_2(2)$ und $a_3(1) = a_3(2)$.

Gespeichert wird:

$l =$	1	2	3	4
$\overline{a_l} =$	$a_1(1)$	$a_2(1)$	$a_3(1)$	$a_1(2)$

Die Zuodrnung von gegebenen Konfiurationen zu gegebenen Ereignissen erfolgt im Weiteren durch die Gewichte

$$w_{h,l} = \begin{cases} P(a_k(t_h)) & \exists l \in \{1, ..., L\}, \quad s.d.\ \overline{a_l} = a_k(t_h) \\ 0 & sonst \end{cases}.$$

Damit ergibt sich die folgende diskretisierte Zielfunktion:

$$(e_1, ..., e_n) \mapsto \sum_{h=1}^{H} \sum_{l=1}^{L} w_{h,l} \cdot \left[d_{G(t_h)}(O_u(\overline{a_l})) - (d_{G(t_h)} \circ p_{1,...,C} \circ v)\left((e_1, ..., e_n), \overline{a_l}, t_h\right) \right]^2 \tag{3.3}$$

Die Gewichte beinhalten Informationen über die zeitliche Abhängigkeit der dynamischen Hindernisobjekte sowie über die Verteilung der unbekannten Kollektive. Da der Löser die Zielfunktionswerte untereinander vergleicht, zählt allerdings nur der Unterschied der Gewichte bis auf Multiplikation mit Konstanten. Aus diesem Grund kann der Term $\frac{1}{t_1 - t_0}$ vernachlässigt werden. Da es sich hier um eine Anwendung handelt, bei der eine Kollision zu *jedem Zeitpunkt* ausgeschlossen werden soll, ist es wenig sinnvoll, einen Unterschied zwischen den Gewichte w_{h,l_0} zu einer festen Verteilung $l_0 = 1, ..., L$ zu machen, es sei denn ein bestimmter Zeitschritt $h \in \{1, ..., H\}$ soll im Zusammenhang mit l_0 nicht betrachtet werden. Sofern trotzdem keine Gleichverteilung der Ereignisse und Konfigurationen gewünscht ist, d.h. die Gewichte unterschiedlichen Wert haben sollen, können diese der Klasse „ObjectiveFunction“ über eine Methode bereitgestellt werden.

Zur Berechnung von (3.3) benötigt man außerdem Informationen über das unbekannte Kollektiv, über die Hindernisobjekte des statischen und dynamischen Kollektivs und über deren Abstand zu jeder Zeit $h = 1, ..., H$ und jedem Ort $l = 1, ..., L$. Für alle Orte und Zeiten müssen außerdem die Bereiche konstruiert werden, in denen die Kameras das unbekannte Objekt vermuten.

Die Informationen über die drei Kollektive „unbekannt“, „statisch“und „dynamisch“ bezieht die Klasse „ObjectiveFunction“ einer von mehreren Instanzen der Klasse „Environment“, die ebenfalls durch Methoden bereitgestellt werden müssen. Die Konfigurationen des dynamischen Kollektivs werden durch einen Vektor von „Environments“ angegeben. Die Ereignisse der durch den Oberflächenparameter induzierten Verteilung des unbekannten Kollektivs müssen ebenfalls durch einen Vektor von „Environments“ bereitgestellt werden.

Jede Instanz eines „Environments“ hat nicht zwingend zusammenhängende „Objekte“. Die Punkte, die innerhalb der „Objekte“ liegen, sind durch ein Oberflächenmodell gekennzeichnet. Dieses Oberflächenmodell besteht aus „Facetten“ - hier Dreiecken - und begrenzt damit eine zusammenhängende Menge an Punkten.

Durch die angehängte Klasse „Inclusiontest“ kann ein „Objekt“ selbst entscheiden, ob ein Punkt der Umgebung innerhalb des gleichen Objektes liegt. Die Klasse „Occlusiontest“ ist erstellt worden, um die Distanz dieses Objektes zu einem Punkt zu bestimmen, oder aber auch um zu entscheiden, ob das „Objekt“ einen Kamerastrahl zu einem Punkt unterbricht. Da in EPP $G(t_h) = O_{d_1}(t_h)$ gefordert wird, ist damit auch die Möglichkeit gegeben,

$$d_{G(t_h)}(z) \quad \forall z \in V \subset Z$$

zu berechnen, solange V endlich ist, d.h. dass V durch $d_{G(t_h)}$ in endlicher Zeit abgearbeitet werden kann. An dieser Stelle ist zu beachten, dass die Implementierung mehrere dynamische Objekte, die nur teilweise zu der gefährlichen Menge G zählen, nicht vorsieht. Hier wird zur Vereinfachung das gesamte dynamische Kollektiv herangezogen: $G(t_h) = O_d(t_h)$

Es stellt sich nun die Frage, welche Punkte zur Abstandsberechnung herangezogen werden müssen. Das heißt, bei gegebener Kameranetzeinstellung $(e_1, ..., e_n) \in E^n$ ist die Menge M für alle $l = 1, ..., L$ und für alle $h = 1, ..., H$ so zu bestimmen, dass folgende Menge endlich ist:

$$M_{v,l,h}(e_1, ..., e_n) = \{z \in M \mid z \in v\,((e_1, ..., e_n), \overline{a_l}, t_h)\} \tag{3.4}$$

Die Bearbeitung der genannten Frage geschieht im Abschnitt 3.1.2. Es erfolgen nun die Konsequenzen aus der Bildung dieser Menge M. Um der Zielfunktion die Möglichkeit zu bieten, eine exakte Schätzung des unbekannten Kollektivs mit einer guten Bewertung auszustatten (also null), müssen die gleichen Punkte M zur Bestimmung des tatsächlichen Abstandes herangezogen werden.

$$M_{O,l} = \{z \in M \mid z \in O_u(\overline{a_l})\} \tag{3.5}$$

Während die Bestimmung der Menge (3.5) ausreicht, um den tatsächlichen Abstand des unbekannten Kollektivs zu den Gefahrenquellen zu berechnen, sind die Kalkulationen zwischen der Bildung der Menge (3.4) bis hin zum geschätzten Abstand in mehreren Schritten abzuarbeiten. Sobald nämlich diese Menge gebildet worden ist, müssen zum Zwecke der Plausibilitätsabprüfung zusammenhängende Bereiche daraus erkannt werden, die als „Cluster“ bekannt sind. Um den Begriff „zusammenhängend“ auf die Menge M auszudehnen, definieren wir:

Definition 3.2
Sei M wie in (3.4) und (3.5) gefordert.

1. Eine *Nachbarschaftsliste* von $\overline{m} \in M$ ist eine Menge, abhängig von $\overline{m}$, mit der Eigenschaft:
$$N_{\overline{m}} \subset M$$

2. Eine Menge $A \subset M$ ist *zusammenhängend*, falls für zwei beliebige $a^0, a^* \in A$ gilt:
$$\exists a_1 ... a_i \in A : a^* \in N_{a^i}, a_i \in N_{a_{i-1}}, ..., a_2 \in N_{a_1}, a_1 \in N_{a^0}$$

Die zusammenhängenden Bereiche der Menge (3.4) lassen sich demnach durch einen Algorithmus rekonstruieren, der ausgehend von einem Punkt in M die Nachbarschaftslisten sukzessiv absucht. Ist ein betrachteter Punkt auch in der Menge (3.4), gehört er einem „Cluster“ an, dessen Nachbarschaftliste wiederum durchsucht werden muss. Andernfalls wird der Punkt nicht weiter betrachtet.

Jede Instanz eines „Clusters“ wird daraufhin auf Plausibilität überprüft und verworfen, sofern er nicht plausibel ist. Die Punkte, die innerhalb der nicht-verworfenen „Cluster“ liegen, sind für die Bestimmung des geschätzten Abstandes ausschlaggebend.

Mit diesen Angaben kann die Klasse „ObjectiveFunction“ somit eigenständig die Distanzen - und damit die Distanzabweichung - von den angegebenen „Environments“ berechnen. In folgendem Abschnitt sollen Möglichkeiten eruiert werden, die beiden Mengen (3.5) und (3.4) und die Auswahl der Punkte für eine Nachbarschaftsliste zu bestimmen.

3.1.2 „SightAnalysis“ und Mengenbestimmung

In diesem Abschnitt soll die Menge M bestimmt werden, aus der die beiden Mengen $M_{O,l}$ aus Formel (3.5) und $M_{v,l,h}(e_1, ..., e_n)$ aus Formel (3.4) hervorgehen. Dazu betrachten wir zuerst, wie $M_{v,l,h}(e_1, ..., e_n)$ aufgebaut sein muss, erzeugen daraus die Menge M und die zur Umsetzung benötigten Klassen. Letztere sind in der Klasse „SightAnalysis“ enthalten. Ohne eine Instanz dieser Klasse soll deswegen keine Instanz der „ObjectiveFunction“ erzeugt werden können.

Um das Modell des unbekannten Kollektivs und somit auch $M_{v,l,h}(e_1, ..., e_n)$ erstellen zu können, müssen der „SightAnalysis“ ein Netzwerk mit Kameraeinstellungen (in Klasse „CameraNetwork“), Informationen zur Sicherheitszone Z (in „DiscreteSurveillanceArea“), das statische „Environment“ und jeweils ein Ereignis $l \in \{1, ..., L\}$ des unbekannten und eine Konfiguration $h \in \{1, ..., H\}$ des dynamischen „Environments“ mitgegeben werden. Das Modell des unbekannten Kollektivs enthält die Punkte des folgenden Bereichs:

$$\begin{aligned} v\left((e_1, ..., e_n), \overline{a_l}, t_h\right) &= \left(Z \cap B^{\kappa_{i=1,...,n}(e_i,.)}_{\overline{D} \vee V \setminus O_{s/d}}(O_u(\overline{a_l}), t_h)\right) \\ &= \left(Z \cap \bigcap_{i=1}^{n} \left(B^{\kappa_i(e_i,.)}_{V}(O_u(\overline{a_l}), t_h) \cup B^{\kappa_i(e_i,.)}_{\overline{D}}\right) \setminus O_d(t_h)\right) \end{aligned} \tag{3.6}$$

In der Computergraphik wird eine derartige nicht-konvexe Schnittmengenbildung zum Beispiel durch Diskretisieren des betrachteten Raumes gelöst. Im Anschluss an die Diskretisierung wird jedes diskrete Stück darauf überprüft, ob es größtenteils in der Menge $v\left((e_1, ..., e_n), \overline{a_l}, t_h\right)$ enthalten ist.

Diese Diskretisierung nimmt die Klasse „DiscreteSurveillanceArea“ innerhalb der „SightAnalysis“ vor. Eine Ableitung dieser Klasse ist der „VoxelCuboid“ (dem Geometrie-Paket entnommen). Er erstellt eine quaderförmige Sicherheitszone Z, die in x-Richtung aus $r_x \in \mathbb{N}$, in y-Richtung aus $r_y \in \mathbb{N}$ und in z-Richtung aus $r_z \in \mathbb{N}$ gleichen Quadern besteht. Diese werden Voxel

$$\nu_{i,j,k} \subset Z \quad \text{mit} \quad i \in \{1, ..., r_x\},\ j \in \{1, ..., r_y\},\ k \in \{1, ..., r_z\}$$

genannt.

Das Tupel (r_x, r_y, r_z) ist die Auflösung (resolution) der Diskretisierung. Der Mittelpunkt eines jeden Voxels wird zur Abstandsberechnung hergenommen. Ein durch Voxel diskretisierter Raum, in dem nur Voxelmittelpunkte auf ein „Belegtheit“ getestet werden, wird in der Literatur auch also „Occupancygrid“ bezeichnet.

Durch das Diskretisieren entsteht eine endliche Menge von Punkten aus Z, nämlich die Menge der Mittelpunkte von genau $r_x \cdot r_y \cdot r_z$ Voxeln. Diese wird zur Bestimmung von den beiden Mengen (3.5) und (3.4) herangezogen:

$$M = \{z \in Z \mid \text{z ist Mittelpunkt von } \nu_{i,j,k}\ ,\ i = 1, ..., r_x,\ j = 1, ..., r_y,\ k = 1, ..., r_z\} \tag{3.7}$$

Die Bestimmung von $M_{O,l}$ ist damit gesichtert. Bei der ersten Erstellung der Ereignisse des unbekannten Kollektivs oder der Konfigurationen des dynamischen wird für alle $l = 1, ..., L$ ein Feld der Größe $r_x \cdot r_y \cdot r_z$ abgearbeitet. Jeder Voxelmittelpunkt wird überprüft, ob er, bei Betrachtung eines gegebenen Ereignis l der Verteilung, innerhalb des unbekannten Kollektivs liegt. Diese Informationen werden allerdings nicht abgespeichert, sondern unmittelbar zur Abstandsberechnung $d_{G(t_h)}(M_{O,l})$ herangezogen. Das Feld $d_{G(t_h)}(M_{O,l})$ wird in der Größe $H \times L$ gespeichert.

Zur Erstellung von $M_{v,l,h}(e_1, ..., e_n)$ sind weitere Informationen unabdingbar: In der Instanz der Klasse „CameraNetwork“ sind beim Erzeugen der „SightAnalysis“ sofort n Instanzen der Klasse „Camera“ gespeichert, die neben den Kameraeinstellungen auch andere Funktionen übernehmen, wie die Transformation in Kamerakoordinaten oder in Weltkoordinaten oder die Überprüfung, ob ein Punkt innerhalb des Kamerasichtkegels liegt. Die Einstellungen der Kamera sind private Variablen, die allerdings durch eine Methode frei wählbar sind. Dies ist wichtig, da der Löser dieser Anwendung eine beliebige Kameraeinstellung fordern darf, um so ein Minimum zu finden.

Die „SightAnalysis“ verfügt über Zeiger auf das statische und auf eine Konfiguration $h \in \{1, ..., H\}$ des dynamischen, sowie auf ein Ereignis $l \in \{1, ..., L\}$ des unbekannten Kollektivs bzw. „Environments“. Mit Hilfe der Klassen „Occlusiontest“ und „Inclusionest“ der „Environments“ sind nun alle Informationen vorhanden, die zum Bestimmen des Bereichs $v\left((e_1, ..., e_n), \overline{a_l}, t_h\right)$ und damit auch der Menge $M_{v,l,h}(e_1, ..., e_n)$ benötigt werden.

Dies geschieht durch die private Methode „evaluate()“ der Klasse „SightAnalysis“ . Darin wird ein Punkt der „DiscreteSurveillanceArea“ zuerst daraufhin überprüft, ob er innerhalb des statischen oder dynamischen „Environments“ liegt. Ist dies nicht der Fall, werden sukzessiv alle Kameras bearbeitet: Ist der Punkt im Sichtkegel der Kamera, interessiert die Anordnung von den Hindernissen, jenem Punkt und Kamerafokus. Die Gerade durch den Kamerafokus und den betrachteten Punkt kann in verschiedenen Situationen von einem der „Environments“ unterbrochen werden. Sobald die Anordnung einer der Situationen in Tabelle 3.1 entspricht, wird der Punkt von der Kamera gesehen und gehört damit nicht zum Bereich, in dem diese das unbekannte Objekt vermutet.

Anordng. 1:	stat. Env.	Fokus	Punkt		
Anordng. 2:	dynam. Env.	Fokus	Punkt		
Anordng. 3:	unbek. Env.	Fokus	Punkt		
Anordng. 4:		Fokus	Punkt	stat. Env.	
Anordng. 5:		Fokus	Punkt	stat. Env.	dynam. Env.
Anordng. 6:		Fokus	Punkt	stat. Env.	unbek. Env.

Tabelle 3.1: Anordnungsmöglichkeiten für den Kamerafokus, Hindernisobjekte, unbekannte Objekte und einem von der Kamera sichtbaren Punkt, wenn Fokus, Punkt und Objekte auf einer Geraden liegen

Nachdem die Menge $M_{v,l,h}(e_1, ..., e_n)$ erstellt worden ist, wird sie zur Weiterverarbeitung an die „ObjectiveFunction“ geliefert. In der weiteren Verarbeitung müssen durch Nachbarschaftslisten zusammenhängende Bereiche erstellt werden. Hierfür muss noch geklärt

werden, welche Nachbarn ein Punkt der Menge M hat:

Definition 3.3
Sei $\overline{m} \in M$ Voxelmittelpunkt von $\overline{m} \in \nu_{i,j,k}$.

1. Ein Voxel $\nu_{i,j,k}$ ist *existent*, solange $i \in \{1, ..., r_x\}, j \in \{1, ..., r_y\}, k \in \{1, ..., r_z\}$
2. Die Nachbarschaftliste von $\overline{m}$ ist definiert durch

$$N_{\overline{m}} = \{m \in M \quad | \quad m \text{ Voxelmittelpunkt von } \nu_{i\pm1,j\pm1,k\pm1} \vee \nu_{i\pm1,j\pm1,k} \vee \nu_{i\pm1,j,k\pm1} \vee \nu_{i,j\pm1,k\pm1} \vee \nu_{i\pm1,j,k} \vee \nu_{i,j\pm1,k} \vee \nu_{i,j,k\pm1} \text{ sofern diese existent sind}\}$$

Durch die Klasse „ObjectiveFunction“ wird demnach durch Diskretisierung der Sicherheitszone in die Menge M und der Zeit $h = 1, ..., H$ sowie der Verteilung $l = 1, ..., L$ der unbekannten Kollektive eine numerische Lösung der Zielfunktion durch $F_{M,H,L} : E \to \mathbb{R}$ angenähert:

$$F_{M,H,L} : (e_1, ..., e_n) \mapsto \sum_{h=1}^{H} \sum_{l=1}^{L} w_{h,l} \cdot \left[d_{G(t_h)}(M_{O,l}) - (d_{G(t_h)} \circ p_{1,...,C})(M_{v,l,h}(e_1, ..., e_n)) \right]^2 \tag{3.8}$$

Zum Lösen des ganzen Problems muss nun ein Löser gefunden werden, der mit dieser Annäherung arbeiten kann. Die Auswahl des Lösers anhand von Eigenschaften der so erstellten Näherung soll Inhalt des nun folgenden Kapitels sein.

3.2 Der Löser

Bei der Umsetzung des EPP fehlt noch ein entscheidender Teil, ein Löser zu Minimierungszwecken. Bei der Auswahl des Lösers muss in erster Linie berücksichtigt werden, welche Eigenschaften Zielfunktion und Nebenbedingungen erfüllen. Zunächst werden diese beiden Kriterien in Abschnitt 3.2.1 untersucht. Darauf folgt die Auswahl aus der Vielfalt der Löser in Abschnitt 3.2.2. Anschließend wird in Abschnitt 3.2.3 beschrieben, wie der ausgewählte Löser vorgeht, um das Minimum zu finden.

3.2.1 Untersuchung der Zielfunktion und der Nebenbedingungen

Zuerst werden die Problemstellen unserer Formulierung in Augenschein genommen. Hier werden sowohl die der Nebenbedingungen, als auch die der Zielfunktion angesprochen:

Nebenbedingungen

Die Nebenbedingungen eines Problems sind Beschränkungen der Variablenwerte der Zielfunktion. Im vorliegenden Problem erfolgt die Angabe der Variablen durch die Bereitstellung der zulässigen Einstellungen der Kameras. Bislang wurde angenommen, dass die einzige Beschränkung der Einstellungen $e_1, ..., e_n$ mit $\overline{U} = U$ folgende ist:

$$e_1, ..., e_n \in E = \overline{U} \times [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Ist der Definitionsbereich von Variablen eines Problems konvex, haben viele Löser die Möglichkeit Optimalität zu garantieren. Darunter fallen zum Beispiel Innere-Punkte-Verfahren, die innerhalb des abzusuchenden Bereiches immer in eine Richtung suchen, die am lohnensten erscheint. Ist der abzusuchende Bereich nicht konvex, kann es sein, dass das Optimum innerhalb des Bereiches liegt, der Weg dahin jedoch außerhalb. Dadurch verschätzen sich viele Löser. Ein Bereich B ist konvex, wenn $\forall p_1, p_2 \in B$ gilt

$$\tau p_1 + (1 - \tau) p_2 \in B \quad \forall \tau \in [0, 1]$$

Obige Beschränkung der Kameraeinstellungen ist demnach konvex, sobald $\overline{U}$ konvex ist. Das erleichtert manchen Lösern das Suchen nach einem Optimum. Es gibt allerdings einige sinnvolle nicht-konvexe Beschränkungen von Kameraeinstellungen:

Beispiel 3.4

Wir untersuchen hier die Einschränkungen der Kameraeinstellungen beim Anbringen von Kameras nur an einer Wand oder an der Decke eines Quader-förmigen Raumes. Das heißt wir betrachten eine Untermenge $\overline{U} \subset U$ der Umgebung: $U = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1] \in \mathbb{R}^3$ mit:

$$\overline{U} = \underbrace{[x_0, x_1] \times [y_0, y_1] \times \{z_1\}}_{\text{Decke}} \cup \underbrace{[x_0, x_1] \times \{y_1\} \times [z_0, z_1]}_{\text{Wand}}$$

Dieser Bereich $\overline{U}$ ist nicht konvex, denn mit $\overline{x} \in [x_0, x_1]$ und $\overline{y} \in [y_0, y_1]$ und $\underline{x} \in [x_0, x_1]$ und $\underline{z} \in [z_0, z_1]$, sind zwei Punkte definiert, deren Verbindungslinie nicht in $\overline{U}$ liegt:

$$p_1 = \begin{pmatrix} \overline{x} \\ \overline{y} \\ z_1 \end{pmatrix}, p_2 = \begin{pmatrix} \underline{x} \\ y_1 \\ \underline{z} \end{pmatrix}, \quad \tau p_1 + (1-\tau)p_2 = \begin{pmatrix} \tau\overline{x} + (1-\tau)\underline{x} \\ \tau\overline{y} + (1-\tau)y_1 \\ \tau z_1 + (1-\tau)\underline{z} \end{pmatrix} \notin \overline{U}$$

für alle $\tau \in (0, 1)$

Es ist demnach sinnvoll, einen Löser zu finden, der auch mit nicht-konvexen Nebenbedingungen arbeiten kann. Die Nebenbedingung für „Anbringen der Kameras nur an Wänden oder der Decke“ kann ein Löser allerdings nicht einfach in der Mengenschreibweise wie oben verarbeiten. Dazu muss ein Ungleichungssystem angegeben werden, das nun für dieses Beispiel kurz erarbeitet wird.

Zuerst wird angegeben, in welchem Definitionsbereich sich die Kameraeinstellungen bewegen:

$$e_1, ..., e_n \in E = U \times [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$$

mit $U = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1] \in \mathbb{R}^3$. Für eine Kameraeinstellung $e = (f, \phi, \rho)$, bei der der Fokus $f = (f_x, f_y, f_z)$ an der Decke ist, gilt:

$$f_z - z_1 = 0 \quad \Leftrightarrow \quad f_z - z_1 \leq 0, \quad -f_z + z_1 \leq 0$$

Ein Fokus an der Wand $y = y_1$ bedeutet:

$$f_y - y_1 = 0 \quad \Leftrightarrow \quad f_y - y_1 \leq 0, \quad -f_y + y_1 \leq 0$$

Diese Ungleichungen sind linear. Möchte man nun ausdrücken, dass die Kamera sowohl an der Decke als auch an der einen Wand $y = y_1$ befestigt werden soll, d.h. beide Gleichungen erfüllt sein sollen, können die Gleichungen beziehungweise Ungleichungen so an den Löser übergeben werden. Eine Oder-Verknüpfung kann zum Beispiel durch diese nichtlineare Bedingung erreicht werden:

$$(f_z - z_1) \cdot (f_y - y_1) = 0$$

Ein lineares Optimierungsproblem wird automatisch zu einem nicht-linearen, wenn eine der Nebenbedingungen nicht-linear ist. Dies ist hier allerdings nicht ausschlaggebend für die Auswahl des Lösers, da in folgendem Abschnitt festgestellt wird, dass schon die Zielfunktion nicht-linear ist. Weitere Schwierigkeiten in Bezug auf die Zielfunktion werden nun besprochen, die die Auswahl des Lösers einschränken.

Zielfunktion

Die Zielfunktion wird im Folgenden in der sowohl nach Ort und Zeit, als auch nach Voxel diskretisierten Zerlegung (Gleichungen (3.3) und (3.7)) in Form der Gleichung (3.8) untersucht. Ausgehend vom Grundproblem GP über die beiden erweiterten Probleme EPH und

EPP bis hin zur Diskretisierung werden dabei die Aspekte der Stetigkeit, Differenzierung und Konvexität betrachtet, die bei der Auswahl des Lösers von Bedeutung sind.

Schon die Betrachtung der Zielfunktion des GP lässt erkennen, dass es sich hier um ein nicht lineares Optimierungsproblem (NLP) handelt. Die Zielfunktion enthält nämlich nichtlineare Teile, wie etwa die Minima der Abstände und Quadrierungen. Weiterhin kommt es schon im Grundproblem mit einem einzelnen Punkt in G zu Unstetigkeiten, je nach Lage dieses Punktes und Konstellation des unbekannten Kollektivs. Diese Unstetigkeiten sind darauf zurückzuführen, dass die Zielfunktion F eine Komposition von mehreren Funktionen d mit p und v ist. Sind die Komponenten stetig, ist auch F stetig. Ein Beispiel, in dem sich jedoch die Unstetigkeit in der Funktion v als Unstetigkeit in der Zielfunktion F fortpflanzt, ist in Abbildung 3.1 aufgezeigt.

Beispiel 3.5

Wir nehmen dazu an, G bestehe nur aus einem Punkt g und verändere sich nicht mit der Zeit. Weiterhin habe die Verteilung des unbekannten Kollektivs nur ein Ereignis. Das erste Bild zeigt die Ausgangsabstände. Der Abstand zu dem unbekannten Kollektiv ist konstant mit Veränderung der Kameraparameter. g liegt zuerst auf dem Rand von Z und auf dem Rand des freien Bereichs der Kamera, so dass der Abstand zu den Bereichen, in denen O_u vermutet wird, größer null ist. Nach geringer Verschiebung des Fokusses df oder nach geringer Neigung um $d\phi$, entsteht an der Stelle g ein nicht-detektierbarer Bereich, in dem sich das unbekannte Objekt aufhalten könnte. Der Abstand ist null.

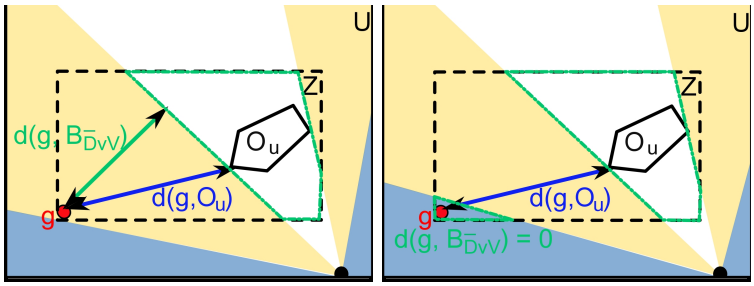

Abbildung 3.1: Veranschaulichung der Unstetigkeit von v und damit F in 2D

Dieses konstruiert erscheinende Beispiel ist in der Realität keine Seltenheit. Tatsächlich kann g bei einer oder mehreren Kameras auch im Inneren von Z liegen, solange ein verdeckter oder nicht-detektierbarer Bereich durch die Anpassung eines Kameraparameters so verändert wird, dass er aus der betrachteten Zone verschwindet oder hineinbefördert wird. Ist der veränderte Bereich näher als alle anderen, so entsteht die Unstetigkeit. Eine Möglichkeit, die Stetigkeit an dieser Stelle wiederherzustellen, beruht darauf, nur Bereiche um g zu betrachten, die näher sind als der minimale Abstand zum Rand der Sicherheitszone. Die Illustration dieser Möglichkeit ist an dieser Stelle nicht nötig, da sie spätestens durch die Erweiterung zum EPP verschwindet.

Ein weiterer Aspekt zur Auswahl eines Lösers ist die Konvexität der Zielfunktion. Ist eine Zielfunktion auf einer kompakten Menge streng konvex, so gibt es höchstens ein Minimum. D.h. falls ein lokales Minimum existiert, so ist dieses das globale Optimum. Um diesen Aspekt zu erarbeiten, brauchen wir die Beobachtung, dass der Abstand zu dem in Kapitel 2 formulierten Modell des unbekannten Kollektivs kleiner als der tatsächliche Abstand zum unbekannten Kollektiv ist, also eine konservative Abschätzung erfolgt ist:

Satz 3.6
Für einen Zeitpunkt $t \in [t_0, t_1]$, in dem Z beobachtet wird, sei $a(t)$ so gewählt, dass $O_u(a(t)) \cap Z \neq \emptyset$. Es sei $(e_1, ..., e_n) \in E^n$ eine Einstellung des Kameranetzwerkes. Es gilt:

$$d_{G(t)}(O_u(a(t))) \geq (d_{G(t)} \circ p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t)$$

Beweis:

Seien $\overline{g} \in G(t)$ und $\overline{u} \in O_u(a(t))$ die Punkte, zu denen der tatsächliche Abstand gemessen wird:

$$d(\overline{g}, \overline{u}) = d_G(O_u(a(t))) = \min\{d(g, u) | g \in G, u \in O_u(a(t))\}$$

Da nach Lemma 2.9 (Ergänzung) aus Abschnitt 2.2.2 die Inklusion $O_u(a(t)) \subset v\,((e_1, ..., e_n), a(t), t)$ gilt und $p_{1,...,C}$ so konstruiert worden ist, dass $O_u(a(t)) \subset (p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t)$, lässt sich folgern dass:

$$\overline{u} \in (p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t)$$

Daraus folgt für den geschätzten Abstand

$$\begin{aligned}
&\min\{d(g, u) |\ g \in G(t),\ u \in (p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t)\} \\
&= \min\left\{d_{G(t)}(\{\overline{u}\})\ ,\ \min\{d(g, u) | g \in G(t), u \in (p_{1,...,C} \circ v)\,((e_1, ..., e_n), a(t), t)\}\right\} \\
&\leq d_{G(t)}(\{\overline{u}\}) = d_{G(t)}(O_u(a(t))).
\end{aligned}$$

□

Eine Funktion F ist konvex, wenn $\forall \tau \in [0, 1]$ und $\forall (e_1, ..., e_n)_1, (e_1, ..., e_n)_2 \in E^n$ gilt:

$$F\left(\tau \cdot (e_1, ..., e_n)_1 + (1 - \tau) \cdot (e_1, ..., e_n)_2\right) \leq \tau \cdot F\,(e_1, ..., e_n)_1 + (1 - \tau) \cdot F\,(e_1, ..., e_n)_2$$

Wie nachfolgendes Gegenbeispiel zeigt, ist die Zielfunktion jedoch nicht konvex.

Beispiel 3.7

Wir nehmen wie in Beispiel 3.5 an, G bestehe nur aus einem Punkt g, der sich nicht verändere, genau wie das unbekannte Kollektiv, also gilt

$$d_{G(t)}(O_u(a(t))) = \overline{d}\ \text{const}$$

Der Fokus der einzigen Kamera bewege sich auf der Bahn $\tau \cdot e_1 + (1-\tau) \cdot e_2$ so, dass der Bereich, in dem die Kamera das unbekannte Objekt vermutet, zwischendurch g enthält, jedoch nicht am Anfang oder am Ende der Bewegung. In Abbildung 3.2 vermuten die Kameras das unbekannte Objekt im verdeckten Bereich. Es gilt nach Satz 3.6:

$$\overline{d} \geq (d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(e_1, a(t), t) > 0 \;\; \textit{und} \;\; \overline{d} \geq (d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(e_2, a(t), t) > 0$$

Für die Kameraeinstellung $\overline{e} = \tau \cdot e_1 + (1-\tau) \cdot e_2 \in E$ mit $\tau \in [0,1]$ so dass $g \in (p_{1,\ldots,C} \circ v)\,(\overline{e}, a(t), t)$, gilt:

$$(d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(\overline{e}, a(t), t) = 0$$

Das bedeutet für die Zielfunktion:

$$\begin{aligned} F(\tau \cdot e_1 + (1-\tau) \cdot e_2) &= \left[\overline{d} - (d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(\overline{e}, a(t), t)\right]^2 \\ &= \overline{d}^2 \\ &> \tau \cdot \left[\overline{d} - c_1\right]^2 + (1-\tau)\left[\overline{d} - c_2\right]^2 \quad \forall \overline{d} \geq c_1, c_2 > 0 \end{aligned}$$

Wählt man nun $c_1 = (d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(e_1, a(t), t)$ und $c_2 = (d_{G(t)} \circ p_{1,\ldots,C} \circ v)\,(e_2, a(t), t)$, so sieht man sofort, dass die Konvexitätsbedingung nicht erfüllt ist; es gilt die strikte Ungleichung

$$F(\tau \cdot e_1 + (1-\tau) \cdot e_2) > \tau \cdot F(e_1) + (1-\tau) \cdot F(e_2)$$

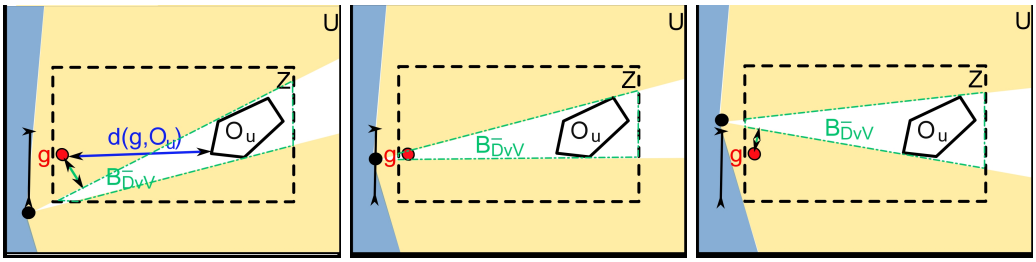

Abbildung 3.2: Veranschaulichung der Konvexität F in 2D

Auch bei der Anwesenheit von Hindernissen sowie bei der Betrachtung von Plausibilitätsbedingungen sind diese Beispiele aktuell. Eine Unstetigkeit tritt auf, sobald ein Bereich, in dem das unbekannte Objekt vermutet wird, so klein wird, dass er von den Plausibilitätschecks erfasst und ausgeschlossen wird, oder außerhalb der Sicherheitszone verschwindet. Mit den Unstetigkeiten von $p \circ v$ durch die Plausibilitätsbedingungen entstehen weitere nicht-konvexe Stellen, bei denen der betrachtete gefährliche Punkt durch den verdeckten oder nicht-detektierbaren Bereich manövriert wird.

Zusätzlich ist die Sicherheitszone nun in Voxel diskretisiert. Will man zu einer diskreten Menge den Abstand berechnen, wie eben zu $M_{v,l,h}(e_1, ..., e_n) = M \cap v((e_1, ..., e_n), a(t), t)$, so springt der Abstand bei stetiger Änderung von $v((e_1, ..., e_n), a(t), t)$ von einem Punkt aus M zum anderen. Zwischen den Sprüngen bleibt der Abstand konstant. Dadurch besteht die Zielfunktion aus stückweise stetigen Funktionen, die zwischen den Unstetigkeiten konstant ist, d.h. deren Ableitung null ist. Auch das ist ein Hindernis für viele Löser, denn die Ableitung nutzen viele Algorithmen, um eine Richtung zu finden, in der als nächstes nach dem Optimum gesucht wird. Ein Punkt, an dem die Ableitung verschwindet, d.h. null wird, könnte somit fälschlich als Optimum angenommen werden.

All diese aufgezeigten Merkmale der Zielfunktion haben schwerwiegende Folgen für die Auswahl des Lösers, worauf in folgendem Abschnitt eingegangen werden soll.

3.2.2 Auswahl des Lösers

Für das EPP mit der diskretisierten Zielfunktion (3.8) wird ein Löser gesucht. Da der Zielfunktionswert über eine Simulation der Kamerastellung und deren Sichtbereiche berechnet wird, ist die Zielfunktion nicht als explizite Formel vorhanden und kann auch nicht auf eine bestimmte gebracht werden. Hinzu kommt, dass die nicht-lineare Zielfunktion stellenweise unstetig, nicht-differnzierbar und nicht konvex ist. Die Simulation ist weiterhin äußerst rechenaufwendig, so dass der eingesetzte Löser in wenigen Schritten zum Ziel kommen sollte.

Eine Zusammenstellung der Verfahren der nichtlinearen Programmierung ist in Abbildung 3.3 aus [2, 21] zusammengetragen, wobei diese Auflistung nicht den Anspruch der Vollständigkeit stellt, sondern lediglich zur Illustration der Bandbreite aktueller Optimierungsverfahren dient.

Die gängigsten NLP-Löser nutzen Eigenschaften des Problems, etwa die Konvexität, Differenzierbarkeit oder zumindest Stetigkeit, um ein lokales oder globales Optimum (zum Beispiel im konvexen Falle) zu approximieren. Unter diese Kategorie von Lösern fallen spezielle nichtlineare Optimierungsverfahren, sowie numerische Suchverfahren und „Sequential-Quadratic-Programming“. Da schon das Grundproblem 2.1.4 weder stetig noch konvex ist, entfällt dieser Teil der deterministischen Programme als Löser.

Da die gängigsten NLP-Löser demnach nicht zum Minimieren der Abstandsabweichung geeignet sind, wird das hier besprochene Problem als ein Spezialfall der diskret-und-nichtlinearen Optimierungsprobleme (Mixed Integer Non Linear Programs - MINLP) betrachtet. Die Löser eines MINLPs können prinzipiell eingeteilt werden in deterministische und nicht-deterministische bzw. stochastischen Programme. Beispielsweise kann der Definitionsbereich einer Funktion, die an abzählbar vielen Stellen größer null und sonst gleich null ist, beim Maximieren entweder an allen Stellen oder nur an ausgewählten durchsucht werden. Um letztere Suche effizient zu gestalten, muss zusätzliches Wissen über den Definitionsbereich einfließen (zum Beispiel stochastische Informationen).

Wie in [21] beschrieben, können deterministische Programme - sowohl die der NLPs also auch zum Beispiel Branch and Bound, Decompositions, etc. - meist die globale Optimalität

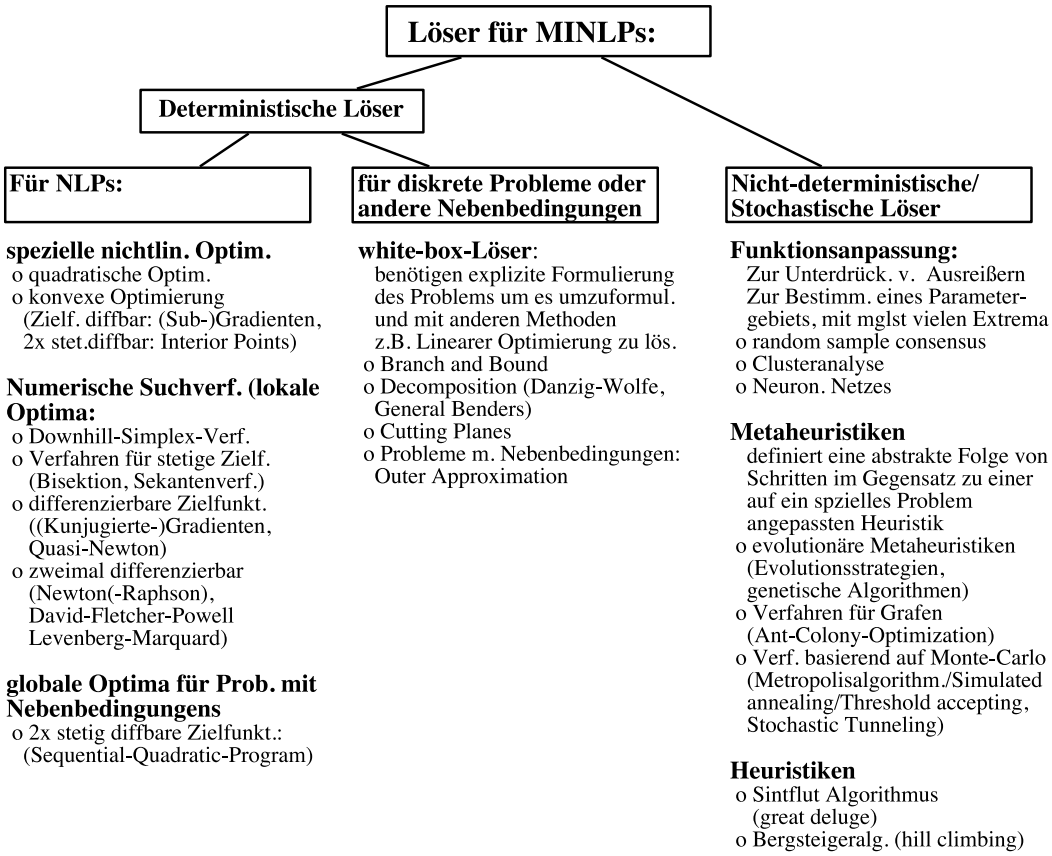

Abbildung 3.3: Optimierungsverfahren zusammengetragen aus [2, 21]

garantieren, brauchen dazu aber entweder Vorwissen oder mehr Rechenleistung. Das heißt, die Implementierung vieler deterministischen Löser benötigt eine explizite Formulierung des Problems (white box solver), die nicht durch eine Simulation ersetzt werden kann. So ist es z.B. beim „Branch and Bound“ unabdingbar, das Problem in mehrere Teilprobleme zu zerlegen.

Da der Löser nur durch die Simulation der Kameraeinstellungen an den Zielfunktionswert gelangen kann, wird hier dagegen das Konzept des „Black-Box-Solvers“ benötigt. Die Methaheuristiken der nicht-deterministischen Programme gehören zu diesen Lösern. Im Gegenzug zu den Heuristiken definieren sie eine abstrakte Folge von Schritten, die nicht auf ein spezielles Problem zugeschnitten ist. Das Problem wird nicht auf den Löser, sondern die einzelnen Schritte des Lösers werden auf das Problem angepasst. Zudem suchen die nicht-deterministischen Programme nicht den ganzen Raum der Variablen ab, sondern beschränken sich auf die lohnenden Stellen. Das führt allerdings wiederum dazu, dass die Globalität nicht garantiert werden kann.

3.2.3 Der Ant-Colony-Algorithmus

Zu den Methaheuristiken gehört auch der Ant-Colony-Algorithmus. In dem Projekt MIDACO wurde dieser in [20] für restringierte MINLPs erweitert. Trotz der Zugehörigkeit des Algorithmus zu den „Black-box-Lösern“, bei denen nur geringe Kenntnis über die Wirkungsweise selbst von Nöten ist, um ihn zu benutzen, wird hier kurz eine Einführung in den Algorithmus gegeben.

Der original Ant-Colony-Algorithmus lehnt sich an die natürliche Futtersuche von einem Ameisenschwarm um ihren Hügel an. Während eine Ameise auf Futtersuche ist, legt sie eine Spur von Pheromonen; das sind Wirkstoffe, die andere Ameisen zur Orientierung verwenden. Diese Pheromone nehmen mit fortschreitender Zeit ab und der Pfad wird weniger attraktiv für die anderen. Je kürzer jedoch der Weg, umso mehr Ameisen können ihn in einer vorgegebenen Zeitspanne beschreiten und desto öfter werden die Pheromone aufgefrischt, desto mehr Ameisen werden auf den Weg aufmerksam und so weiter. Dieses Verhalten ist in Bezug auf die Optimierung unter die Schwarmintelligenz gegliedert. Die Summe der Ameisen entdecken den kürzesten Weg.

Die Metaheuristik des Basis-Ant-Colony-Algorithmus imitiert dieses natürliche Verhalten, um ein kombinatorisches Optimierungsproblem zu lösen. Obwohl diese Veranschaulichung für den Ant-Colony-Algorithmus für MINLPs nicht mehr gilt, wird hier kurz die Idee geschildert.

Algorithmus: *ACO Metaheuristik*

while not Stopkriterium erfüllt **do**

    Lösung durch Pheromone konstruieren
    Pheromone aktualisieren
    Dämonaktivitäten

endwhile

Sei das Tupel $(\mathbb{L}, f)$ eine Instanz des kombinatorischen Optimierungsproblems mit einer Menge von Lösungen $\mathbb{L}$ und den dazugehörigen Kosten $f : \mathbb{L} \to \mathbb{R}$, deren Minimum gesucht wird. Man assoziiere nun jede Lösung mit einem Weg durch den Graphen G, also eine Aneinanderreihung von Knoten des Graphen durch Kanten. Die imaginären Ameisen suchen anhand einer Pheromontabelle, die von den Gewichten der Kanten abhängt, und die wiederum von den Kosten der Lösungen, einen Weg durch diesen Lösungsgraphen und hinterlassen wiederum Pheromone. Dies geschieht, bis ein Stopkriterium erfüllt ist, wie zum Beispiel das Überschreiten einer maximalen Anzahl von Iterationsschritten oder einer Zeitschranke. Um dem Algorithmus die Möglichkeit zu geben, Aktionen auszuführen, die einzelne Ameisen nicht erledigen können, wie etwa die Pheromone von außen zu manipulieren oder lokale Suchen innerhalb des Algorithmus zuzulassen, gibt es am Ende jeder Iteration die Dämonaktivität.

Umsetzung der Metaheuristik für den kontinuierlichen Fall

Im kontinuierlichen Fall verliert die biologische Visualisierung an Bedeutung, trotzdem werden die Lösungen ebenfalls inkrementell erstellt und deren Pheromone aktualisiert. In diesem Abschnitt wird erläutert, welche Form ein Pheromon hier annimmt und wie die Schritte der Metaheuristik am konkreten Beispiel der Kameraplatzierung umgesetzt werden.

Ein essenzieller Bestandteil des Ant-Colony-Algorithmus für den kontinuierlichen Fall ist das Lösungsarchiv (LA). Das Archiv entspricht einer Tabelle, in der die letzten k Lösungen, also Kamerapositionen und -orientierungen zusammen mit deren Zielfunktionswerten und einem Residuumswert aufgehoben werden. Der Residuumswert gibt an, wie stark die Nebenbedingungen des Problems verletzt werden. Das Lösungsarchiv ist absteigend sortiert nach der Attraktivität der Lösungen, berechnet aus dem Residuum und dem Zielfunktionswert. Eine attraktivere Lösung steht demnach weiter vorne (hat einen kleineren Tabellenindex). Das Lösungsarchiv ist besonders bedeutsam für die Berechnung einer neuen Lösung und deren Pheromone.

l	$\mathbf{s_l} = \left(\begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}_1 \quad \cdots \quad \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}_n \begin{matrix} \phi_1 \\ \rho_1 \end{matrix} \quad \begin{matrix} \phi_n \\ \rho_n \end{matrix} \right)_l$							**F**($\mathbf{s_l}$)	**Res**($\mathbf{s_l}$)
	1	2	$\cdots$	i	$\cdots$	$5n-1$	$5n$		
1	$(s^1$	s^2	$\cdots$	s^i	$\cdots$	s^{5n-1}	$s^{5n})_1$	F_1	Res_1
	μ_1^1	μ_1^2		μ_1^i		μ_1^{5n-1}	μ_1^{5n}		
2	$(s^1$	s^2	$\cdots$	s^i	$\cdots$	s^{5n-1}	$s^{5n})_2$	F_2	Res_2
	μ_2^1	μ_2^2		μ_2^i		μ_2^{5n-1}	μ_2^{5n}		
$\vdots$				$\vdots$				$\vdots$	
k	$(s^1$	s^2	$\cdots$	s^i	$\cdots$	s^{5n-1}	$s^{5n})_k$	F_k	Res_k
	μ_k^1	μ_k^2		μ_k^i		μ_k^{5n-1}	μ_k^{5n}		

$\omega_l^i = \frac{k-l+1}{\sum_{j=1}^{k} j}$	$\sigma^i = \frac{\max_{g,h=1,\ldots,k,g\neq h} \lvert s_g^i - s_h^i \rvert \; - \min_{g,h=1,\ldots,k,g\neq h} \lvert s_g^i - s_h^i \rvert}{\#generationen}$

Abbildung 3.4: Lösungsarchiv (LA) für die Kameraeinstellung und die daraus berechneten Pheromone $(\omega_l^i, \sigma_l^i, \mu_l^i)$

Neue Lösungen werden nun anhand einer Dichtefunktion gewählt. Eine der populärsten Dichtefunktionen ist die der Standardnormalverteilung. Diese Funktion hat unter anderem die Vorteile, dass sie leicht zu implementieren ist und schnelle Stichprobengenerierung ermöglicht. Die Stichproben einer Standardnormalverteilung sammeln sich allerdings lediglich um einen Erwartungswert. Zum Suchen in mehreren Bereichen, setzen wir eine Verteilung für die i-te Variable aus L eindimensionalen Gaußfunktionen $g_l^i(x)$ zusammen.

$$G^i(x) = \sum_{l=1}^{L} \omega_l^i \cdot g_l^i(x) = \sum_{l=1}^{L} \omega_l^i \cdot \frac{1}{\sigma_l^i \sqrt{2\pi}} \cdot \exp(-\frac{(x-\mu_l^i)^2}{2 \cdot {\sigma_l^i}^2})$$

Dargestellte Funktion ist durch die Tripel $(\omega_l^i, \sigma_l^i, \mu_l^i)$ mit $i = 1, ..., 5n$ und $l = 1, ..., L$ charakterisiert, wobei ω_l^i die Gewichte, σ_l^i die Standardabweichung und μ_l^i den Erwartungswert des l-ten Suchbereichs der i-ten Variable bezeichnen. In vorliegendem Falle gibt es fünf Variablen pro Kamera, d.h. insgesamt $5n$ Variablen. Eine Ameise - also eine neue Lösung - wird generiert, indem für jede Variable i ein Erwartungswert l nach den Gewichten ω_l^i ausgesucht wird und eine Stichprobe um ihn herum bezüglich der Standardabweichung entnommen wird. Jedesmal, wenn eine neue Lösung generiert wird, wird sie ihrer Attraktivität nach im Archiv eingeordnet, die darauffolgenden Lösungen fallen um einen Platz nach hinten und die letzte fällt aus dem Archiv. Die Anzahl der im Archiv gespeicherten Lösungen k könnte sich deswegen zum Beispiel an der Anzahl L der Suchbereiche orientieren.

Da das Tripel $(\omega_l^i, \sigma_l^i, \mu_l^i)$ die Dichtefunktion komplett definiert und das Wissen speichert, um eine neue Lösung zu generieren, die Ameise sozusagen leitet, wird es ein „Pheromon“ im Sinne von ACO genannt. Wie in Abbildung 3.2.3 dargestellt, stellen die Daten im LA die nötigen Informationen zum Aktualisieren der Pheromone zur Verfügung. Jede Variable einer Lösung stellt den Erwartungswert eines Suchraumes dar. Die Standardabweichung einer Variable bildet sich aus den Unterschieden der im LA gespeicherten Lösungen. Die Gewichte sind so gewählt, dass sie eine linear absteigende Priorität der Lösungen im Archiv implizieren. Deswegen bedeutet Aktualisieren des LAs mit einer besseren Lösung auch gleichzeitig eine Verbesserung der Pheromonspur. Die hier angestellten Überlegungen sind nachzulesen in [21].

Das MIDACO Projekt

Die Erweiterung des Ant-Colony-Algorithmus aus [22] soll Thema dieses Abschnittes sein. In der Erweiterung werden zwei Modifizierungen des Algorithmus gemacht, um den ihn an ein „Mixed Integer Nonlinear Programm“ (MINLP) anzupassen, also ein Problem mit Nebenbedingungen und zusätzlichen diskreten Variablen.

Ein solches Problem hat sowohl kontinuierliche als auch diskrete Variablen, x_{kont} und x_{int}. Die erste Modifikation ist aus der Erkenntnis entstanden, dass alle Lösungen einer diskreten Variable s^i $i = 1, ..., k$ des Lösungsarchivs dem gleichen Wert entsprechen könnte, dadurch die Standardabweichung null ist und keine anderen Variablenwerte mehr konstruiert werden. Da die Aufgabenstellung allerdings keine diskreten Schritte der Kame-

Optimierungsproblem *MINLP*

Minimiere:

$$f(x)$$

unter den Nebenbedingungen:

$$\begin{aligned} g_i(x) &= 0 && i = 1...m_{eq} \in \mathbb{N} \\ g_i(x) &\leq 0 && i = (m_{eq}+1)...m \in \mathbb{N} \\ x = (x_{kont}, x_{int}) & && x_{kont} \in \mathbb{R}^{n_{kont}} \\ & && x_{int} \in \mathbb{N}^{n_{int}} \end{aligned}$$

rapositionierung und -orientierung und damit keine diskreten Variablen vorsieht, wenden wir uns der zweiten Modifikation aus [22] zu.

Der Grundgedanke des Algorithmus ist, für den Löser zusätzlich zur Formulierung des Problems nur einen einzigen Parameter $\Omega \in \mathbb{R}$ anpassen zu müssen; dieser Parameter wird in [22] *oracle* genannt. Um das Ziel zu erreichen, wird eine „penalty"-Methode verwendet. Darin werden die Nebenbedingungen als Strafterme in die Zielfunktion aufgenommen. Je mehr die Nebenbedingungen verletzt werden, desto größer sind die zugehörigen Strafterme und desto größer - also schlechter - ist auch der Zielfunktionswert. Beispiele zu Straftermen sind ebenfalls in [22] zu finden. Die Strafterme der Nebenbedingungen g werden im *Residuum* $res_g(x)$ zusammengefasst. Das Problem wird so umformuliert, dass es zusätzlich in Abhängigkeit von Ω ist:

$$\min \overline{f}(x) \equiv 0$$

Nebenbedingungen:

$$\begin{aligned} g_0(x) &= f(x) - \Omega \\ g_i(x) &= 0 \quad i = 1...m_{eq} \in \mathbb{N} \\ g_i(x) &\leq 0 \quad i = (m_{eq}+1)...m \in \mathbb{N} \end{aligned}$$

Ist x^* Lösung des MINLPs, so ist es auch die Lösung von diesem Problem, wenn $\Omega = f(x^*)$. In dieser Formulierung sind Zielfunktion f und Nebenbedingungen g direkt vergleichbar.

Aus den entstandenen Nebenbedingungen kann der Strafterm

$$p(x) = \alpha \cdot |f(x) - \Omega| + (1 - \alpha) \cdot res_g(x)$$

formuliert werden. Dieser setzt den Wert der Zielfunktion mit den restlichen Nebenbedingungen in Relation. Dabei ist $\alpha \in [0, 1]$ ein dynamisches Gewicht, das die umgewandelte Zielfunktion $|f(x) - \Omega|$ betont, wenn diese größer als das Residuum ist, und umgekehrt.

Das Minimieren des Termes p löst ebenfalls das MINLP. Diese Methode wird *basic oracle penalty methode* genannt. Die Grundlage dieser Methode und und die Berechnung von

α können nachgelesen werden in [22]. Die Methode geht davon aus, dass Ω der kleinste Zielfunktionswert ist, der zulässig ist. In [22] wird aber auch diskutiert, dass die Methode auch für Probleme interessant ist, bei dem ein $\Omega \geq f(x^*)$ angegeben werden kann.

Zusammenfassend verwendet der Algorithmus ein $\Omega \geq f(x^*)$, für das bewiesen werden kann, dass es eine zulässige Lösung $\overline{x}$ mit $f(\overline{x}) = \Omega$ gibt. Er passt die Formulierung des Problems in drei Schritten an dieses etwas ungenaue Ω an, so dass zulässige Zielfunktionswerte sowohl größer als auch kleiner berücksichtigt werden und unzulässige Lösungen weiterhin bestraft werden. Der so entstandene Strafterm wird in [22] *extended oracle penalty function* genannt.

Diese Methode wird in dem MIDACO-Projekt zusammen mit dem Ant-Colony-Algorithmus verwendet, um die Nebenbedingungen zu bewältigen. MIDACO ist allerdings nicht mehr auf die Angabe des Wertes Ω angewiesen, sondern verwaltet dessen Verwendung selbst. Für den ersten Durchlauf des Ant-Colony-Algorithmusses wird der Wert Ω größer angenommen, als der Wert in Hinblick auf das zugehörige Problem sein kann. Danach folgen weitere Durchläufe, in denen der Parameter immer an die beste Lösung angepasst wird, wenn diese einen geringeren Zielfunktionswert liefert. In [20] werden weitere Veränderungen des Lösers besprochen, welche am Schluss jeder Iteration eine lokale deterministische Suche als Dämonaktivität anfügen. Diese kommen allerdings nicht im MIDACO-Projekt zum Einsatz.

3.3 Rechenaufwand des Programms

Bevor das Programm in die Testphase gegeben wird, werden hier die Angaben zu der Laufzeit und Genauigkeit der gesamten Implementierung dargelegt. Zuerst betrachten wir in 3.3.1 die Dimension der in Kapitel 3.1 erstellten diskretisierten Zielfunktion und den Zeitaufwand der Auswertung. Mit diesen Erkänntnissen wird der Aufwand des gesamten Programmes in Abschnitt 3.3.2 mit Hilfe des Landau-Symbols $\mathcal{O}$ abgeschätzt. Zum Schluss werden in Abschnitt 3.3.3 Verbesserungen der Implementation in Hinblick auf die Laufzeit und die Genauigkeit vorgeschlagen.

3.3.1 Dimension und Zeitaufwand

In folgendem Abschnitt wird die Dimension und der Zeitbedarf der Implementierung aus den Kapiteln 3.1 und 3.2 analysiert. Es wird zuerst auf die Dimension der Zielfunktion, dann auf den Aufwand der Auswertung der Zielfunktion eingegangen.

Es bestehe das Netzwerk aus n Kameras. Jede Kamera hat drei Parameter zur Einstellung der Positionierung und zwei zur Einstellung der Orientierung. Zusammen ergibt das $n \cdot 5$ freie Parameter, die dem Löser als Optimierungsvariablen dienen.

Zur Berechnung des Zeitaufwandes werden weiterhin folgende Angaben benötigt: Die Kameras beobachten eine Sicherheitszone mit der Auflösung (r_x, r_y, r_z) der drei Koordinatenachsen der Weltkoordinaten, bestehend aus $r_x \cdot r_y \cdot r_z = r$ Voxeln. Im statischen „Environment“ seien statische Hindernisobjekte mit insgesamt f_s Facetten. Im dynamischen Falle bewege sich das Kollektiv mit der Zeit auf einer Bahn; ist die Zeit diskret $t_h,\ h = 1, ..., H$, nennen wir ein „Environment“ zum Zeitpunkt t_h *Konfiguration.* Jede Konfiguration habe $f_d(h)$ Facetten.

Die Verteilung des Oberflächenparameters des unbekannten Kollektivs sei für die diskreten Schritte $l = 1, ..., L$ mit $P(\overline{a_l})$ angegeben. Die durch den Oberflächenparameter induzierte Verteilung des Kollektivs ist deswegen auch diskret, ein „Environment“ des unbekannten Kollektivs an der Stelle l heiße Ereignis. Die Anzahl der Facetten eines Ereignisses l sei $f_u(l)$.

Mit diesen Informationen können wir den Aufwand der Zielfunktion sukzessiv angeben. Der erste Bestandteil der diskretisierten Zielfunktion berechnet den tatsächlichen Abstand $d_{G(t_h)}(M_{O,l})$ von einem Ereignis der Verteilung l des unbekannten Kollektivs (im vorliegenden Falle: Environment) zu den dynamischen Objekten einer gegebenen Konfiguration h. Um $M_{O,l}$ zu erstellen, wird der Mittelpunkt jedes Voxels in M überprüft, ob er innerhalb einer Instanz der Klasse „Object“ des unbekannten „Environments“ liegt. Dies geschieht, indem ein Vergleich an der Normale der Facetten stattfindet, die am nächsten liegt.

$$\text{Erstellung von } M_{O,l}\text{: } \ r \cdot f_u(l)$$

Um $d_{G(t_h)}(M_{O,l})$ dann für alle $h = 1, ..., H$ zu berechnen, müssen die Abstände aller Punkte, die in $M_{O,l}$ als *innerhalb* gekennzeichnet sind, zu allen Facetten der aktuellen Konfiguration h des dynamischen „Environments“ $f_d(h)$ minimiert werden. $M_{O,l}$ hat die

Größe von M, also auch r. Zusammen mit der Erstellung von $M_{O,l}$ ist der Aufwand der Berechnung des Abstandes Folgender:

$$\text{Berechnung } d_{G(t_h)}(M_{O,l}): \quad r \cdot f_u(l) + r \cdot f_d(h) \tag{3.9}$$

Ein weiterer Bestandteil der Berechnung der diskreten Zielfunktion ist die Berechnung des geschätzten Abstandes. Dazu wird zuerst der Aufwand der Erstellung der Menge $M_{v,l,h}(e_1, ..., e_n)$ dargelegt:

Bei der Erstellung von $M_{v,l,h}(e_1, ..., e_n)$ für ein $l \in \{1, ..., L\}$ und für ein $h \in \{1, ..., H\}$ wird ein Punkt der „DiscreteSurveillanceArea“ zuerst daraufhin überprüft, ob er innerhalb des statischen oder des betrachteten dynamischen Environments h liegt. Ist dies nicht der Fall, werden sukzessiv alle Kameras abgearbeitet: Ist der Punkt im Sichtkegel der Kamera, wird zuerst bestimmt, in welchem Verhältnis jedes der drei „Environments“ die Strecke zwischen Kamerafokus und betrachteten Punkt unterbricht. Je nach Konstellation wird entschieden, ob der Punkt in dieser Kamerasicht verdeckt ist oder nicht.

$$\text{Inklusion und Verdeckung:} \quad r \cdot [f_s + f_d(h) + n \cdot (f_s + f_d(h) + f_u(l))]$$

Nach dargestellter Simulation geht ein Nachbarschaftssuchalgorithmus jedes Voxel der Sicherheitszone ab, um Cluster für die Plausibilitätsbedingungen zu suchen. Letztere betrachten jeden Cluster, deren Anzahl $c \leq (\frac{r_x}{2} \cdot \frac{r_y}{2} \cdot \frac{r_z}{2})$ sei.

$$\text{Nachbarschaft und Plausibilität:} \quad r + c$$

Die Berechnung des Abstandes zu der entstehenden Menge geschieht ähnlich der Berechnung (3.9) mit dem Aufwand „$r \cdot f_d(h)$“. Daraus ergibt sich der gesamte Aufwand von der Erstellung des Modells des unbekannten Kollektivs und der zugehörigen Distanzmessung:

Berechnung $d_{G(t_h)}(M_{v,l,h})$:

$$\big[r \cdot [f_s + f_d(h) + n \cdot (f_s + f_d(h) + f_u(l))] \ + \ r + c\big] \ + \ r \cdot f_d(h) \tag{3.10}$$

Die Rechenaufwände in (3.9) und (3.10), müssen für alle $l = 1, ..., L$ und alle $h = 1, ..., H$ aufgewandt werden, um nur einen einzigen Iterationsschritt des Lösers durchzuführen. Im folgenden wird ein gesamter Iterationsschritt des Lösers betrachtet.

$$\begin{array}{lll} \sum_{h=1}^{H}\sum_{l=1}^{L} & \big[r\ f_u(l) + r\ f_d(h) & +r \cdot [f_s + f_d(h) + n \cdot (f_s + f_d(h) + f_u(l))] + r + c \ + \ r\ f_d(h)\big] \\ & M_{O,l} \quad + Abst. & + \quad M_{v,l,h} \qquad\qquad\qquad\qquad\qquad\qquad + \ Abst. \\ & & Inklusion + \ Verdeckung \qquad\qquad + Plausibil. \end{array} \tag{3.11}$$

Der Löser braucht aber weitaus mehr als einen Iterationsschritt, um in die Nähe des Optimums zu gelangen. Deswegen ist es wichtig, den Aufwand und zugehörigen Faktor eines Iterationsschrittes möglichst klein zu halten. Eine Möglichkeit ist, sich die Information, die berechnet worden ist, zu merken. Dies ist teilweise schon implementiert:

1. Die Erstellung von $M_{O,l}$ und Berechnung des Abstandes $d_{G(t_h)}(M_{O,l})$ geschieht, sobald der Klasse „ObjectiveFunction“ die dynamischen Konfigurationen bzw. unbekannten Ereignisse zur Verfügung gestellt werden. Dies wird nicht in jedem Iterationschritt des Lösers ausgeführt, sondern apriori berechnet.

2. Die Bestimmung der Menge $M_{v,l,h}(e_1, ..., e_n)$ kann nicht vorab geschehen. Trotzdem können während der Berechnung von Inklusion und Verdeckung bezüglich eines Voxelmittelpunktes die Informationen *innerhalb des statischen*, *innerhalb des dynamischen* „Environments“, *sichtbar trotz statischer*, *sichtbar trotz dynamischer* und *sichtbar trotz unbekannter Verdeckung* in einem Feld gespeichert werden. Für jede Information wird jeweils ein Feld der Größe von M angelegt, in der alle Kameras berücksichtigt sind. Diese Felder bleiben bestehen, solange sich nichts an den relevanten Klassen ändert.

   Es ändert sich beispielsweise die Information *innerhalb des statischen Environments* nicht, wenn sich das dynamische oder unbekannte „Environment“ verändert hat. Dieses Feld wird zwar während des ersten Iterationsschrittes des Lösers geändert, aber danach nie wieder.

Durch die erste Modifizierung wird der gesamten Aufwand der Berechnung (3.9) von $d_{G(t_h)}(M_{O,l})$ in jedem Iterationsschritt des Lösers eingespart. Anstelle dieses Aufwandes muss der zugehörige Abstandswert nur noch in einem Feld der Größe $L \cdot H$ nachgeschlagen werden. Um dies zu kennzeichen verwenden wir den Aufwand „1“. Durch die zweite Modifizierung findet offensichtlich der Aufwand bezüglich der Prüfung auf statische Inklusionen, also der erste Term f_s in (3.10), nur in der ersten Iteration des Lösers statt und kann in der Betrachtung einer folgenden Iteration vernachlässigt werden.

$$\sum_{h=1}^{H} \sum_{l=1}^{L} \left[1 + r \cdot [f_d(h) + n \cdot (f_s + f_d(h) + f_u(l))] \ + \ r + c \ + \ r \ f_d(h)\right]$$

Ausgehend von diesen Einsparungen sind dies weitere Verbesserungen des Aufwandes eines einzigen Iterationsschrittes, die durch die zweite Modifizierung entstanden sind:

a) Das Feld *sichtbar trotz statischer Verdeckung* muss angepasst werden, sobald sich statische Hindernisse oder das Kameranetzwerk verändern.

b) Das Feld *innerhalb eines dynamischen* Hindernisses muss angepasst werden, sobald sich die Konfiguration des dynamischen „Environments“ ändert, das Feld *sichtbar trotz dynamischer Verdeckung*, sobald sich Kameranetzwerk, statische Hindernisse oder dynamische Hindernisse ändern.

Der Aufwand, der sich für die Zielfunktion ergibt, ist folgender:

$$\begin{aligned} a) \quad & r\ n\ f_s + \quad \sum_{h=1}^{H}\sum_{l=1}^{L}\left[\ 1 + r\cdot[f_d(h) + n\cdot(f_d(h)+f_u(l))]\ +\ r + c\ +\ r\ f_d(h)\right] \\ b) \quad & r\ n\ f_s + \quad \sum_{h=1}^{H}\left[r\ f_d(h) + r\ n\ f_d(h) + \sum_{l=1}^{L}\left[\ 1 + r\ n\ f_u(l)\ +\ r + c\ +\ r\ f_d(h)\right]\right] \\ & \underbrace{stat.Verd.} \qquad \underbrace{dyn.Inkl + Verdeck.} \qquad \underbrace{Abst. + unbk.Verd. + Plausb. + Abst.} \end{aligned} \tag{3.12}$$

Im Vergleich zum Aufwand (3.11) ist (3.12) deutlich reduziert worden. Es lassen sich allerdings noch mehr Verbesserungen eines Iterationsschrittes machen, wie sich in Abschnitt 3.3.3 erweist. Vor diesen Betrachtungen wird kurz auf die Abschätzung des Aufwandes der gesamten Iteration in O-Notation eingegangen.

3.3.2 Aufwand der gesamten Iteration

Zusammenfassend wird nun der Aufwand der gesamten Iteration des Lösers betrachtet. Davon ausgehend, dass die Anzahl der Iterationsschritte bekannt sei, wird der Aufwand mit Hilfe des Landau-Symbols $\mathcal{O}$ abgeschätzt. Bei dieser Betrachtung werden neben der Iterationsanzahl nur durch den Benutzer beeinflussbare Größen der Zielfunktionswertberechnung betrachtet, beeinflussbare Größen des Lösers werden vernachlässigt.

Die Anzahl der Iterationsschritte sei dabei I. Der Vollständigkeit wegen sei hier erwähnt, dass zur Bildung der Menge $M_{O,l}$ und zur Messung des zugehörigen Abstandes zu den gefährlichen Punkten der Term (3.9) zu Beginn des Algorithmus, d.h. vor der Iteration aufgewendet werden muss. Da die Mengenbildung sowie die Abstandsmessung nicht innerhalb der Iteration geschieht, wird der zugehörige Aufwand in den Betrachtungen der Iteration des Lösers vernachlässigt. Andererseits erinnere man sich an die zweite Modifizierung des Algorithmus aus letztem Abschnitt. Die statischen Inklusionen, die in der Betrachtung (3.12) vernachlässigt worden sind, müssen mit dem Aufwand „$r \cdot f_s$“ im ersten Iterationsschritt berücksichtigt werden. Zusammen mit dem Aufwand eines Iterationsschrittes (3.12) ergibt sich daraus der folgende Aufwand der gesamten Iteration:

$$\begin{aligned} & r\cdot f_s + \quad I\cdot \quad \left\{ r\ n\ f_s + \sum_{h=1}^{H}\left[r\ f_d(h) + r\ n\ f_d(h) + \sum_{l=1}^{L}\left[\ 1 + r\ n\ f_u(l)\ +\ r + c\ +\ r\ f_d(h)\right]\right]\right\} \\ & \underbrace{st.Inkl.} \qquad \underbrace{st.Verd.} \quad \underbrace{dyn.Inkl + Verdeck.} \qquad \underbrace{Abst. + unbk.Verd. + Clst. + Plsb. + Abst.} \end{aligned} \tag{3.13}$$

Um dies mit Hilfe des Landau-Symbols $\mathcal{O}$ auszudrücken, muss folgende Überlegung gemacht werden. Mit Hilfe von

$$f_d^{max} := \max_{h\in\{1,\ldots,H\}}\{f_d(h)\} \quad \text{und} \quad f_u^{max} := \max_{l\in\{1,\ldots,L\}}\{f_u(l)\}$$

kann ausgehend von Aufwand (4.2) folgende Abschätzung gemacht werden:

$$\begin{aligned}
& r \cdot f_s + I \cdot \left\{ r\ n\ f_s + \sum_{h=1}^{H} \Big[r\ f_d(h) + r\ n\ f_d(h) + \sum_{l=1}^{L} \big[\ 1 + r\ n\ f_u(l)\ +\ r + c\ +\ r\ f_d(h) \big] \Big] \right\} \\
\leq\ & r \cdot f_s + I \cdot \left\{ r\ n\ f_s + \sum_{h=1}^{H} \Big[r\ f_d^{max} + r\ n\ f_d^{max} + \sum_{l=1}^{L} \big[\ 1 + r\ n\ f_u^{max}\ +\ r + r\ +\ r\ f_d^{max} \big] \Big] \right\} \\
=\ & r \cdot f_s + I \cdot \left\{ r\ n\ f_s + H \cdot \Big[r\ f_d^{max} + r\ n\ f_d^{max} + L \cdot \big[\ 1 + r\ n\ f_u^{max}\ +\ 2r\ +\ r\ f_d^{max} \big] \Big] \right\} \\
=:\ & \mathcal{A}(I, r, n, f_s, f_d^{max}, f_u^{max})
\end{aligned}$$

Man fasst diesen Ausdruck mit Hilfe des Landau-Symbols $\mathcal{O}$ derart zusammen:

$$\mathcal{A}(I, r, n, f_s, f_d^{max}, f_u^{max}) \ =\ \mathcal{O}\Big(\ I \cdot r \cdot \{\ n\ f_s\ +\ H\ (n + L)\ f_d^{max}\ +\ H\ L\ n\ f_u^{max}\ \}\ \Big) \tag{3.14}$$

3.3.3 Verbesserungen der Implementierung der Zielfunktion

Der Inhalt folgender Ausführungen sollen die Verbesserungen der Implementierung der Zielfunktion sein. Die Veränderungen, die gemacht werden können, betreffen nicht nur die Laufzeit, sondern auch die Genauigkeit der Simulation im Hinblick auf die Wirklichkeit.

Beschleunigen der Laufzeit

Zuerst interessieren wir uns für das Resultat aus den Modifizierungen (3.12). Diese Laufzeit kann noch weiter beschleunigt werden. Dabei ist interessant, ob sich die Verbesserung nur auf den Faktor des Aufwandes oder auf das asymptotische Verhalten auswirkt, welches durch die O-Notation erfasst wird. Beginnen wir mit der Faktorverkleinerung:

- Eine erhebliche Verbesserung der Laufzeit wird dadurch erreicht, dass nicht alle Konfigurationen H mit allen Ereignissen L bearbeitet werden. Dies geschieht zum Beispiel, indem jeder Iteration des Löesers, also der Auswertung jeder Kameranetzeinstellung, der aktuell minimale Zielfunktionswert mitgegeben wird. Es genügt, nur so viele Konfigurationen bzw. Ereignisse zu betrachten, wie deren Wert insgesamt dem aktuell minimalsten Zielfunktionswert übersteigt. Sobald dies der Fall ist, kann abgebrochen werden, da sich der Zielfunktionswert nicht verbessert hat. Dies ändert allerdings den schlechtesten Fall nicht, in dem alle Konfigurationen und alle Ereignisse abgearbeitet werden müssen. Die Veränderung betrifft also den Faktor der Laufzeit.
- Zur Zeit wird die Abstandsabweichung unabhängig von den Gewichten berechnet (außer dieses ist null). Eine Verbesserung des Faktors der Laufzeit wird in Verbindung mit mit erster Verbesserung dadurch erreicht, dass man die Konfigurationen bzw. Ereignisse der „Environments“ mit Priorität des höchsten Gewichtes absteigend bearbeitet.

- Offensichtlich ist es von Vorteil, bei der Berechnung des geschätzten Abstandes nicht alle Voxel der „Discrete Surveillance Area“ abarbeiten zu müssen. In einer weiterführenden Arbeit könnte das Programm so abgeändert werden, dass während der Berechnung der Distanzabweichung einer Konfiguration und eines Ereignisses zuerst die Voxel nahe der gefährlichen Punkte abgearbeitet werden. Dies läuft allerdings auf einen Online-Algorithmus hinaus, der für jedes hinzugenommene Voxel zuerst die Inklusionen und Verdeckungen prüft, und dieses dann an einen eventuell schon vorhandenen Cluster heftet, der auf Plausibilität überprüft wird. Erst wenn der Cluster so groß ist, dass er trotz der Plausibilitätsbedingungen anerkannt wird, kann dazu ein Abstand gemessen werden. Die restlichen Voxel würden dann nicht betrachtet werden, da der Abstand zwischen gefährlichen Punkten und Cluster schon minimal ist. Es handelt sich auch hier um die Verbesserung des Faktors der Laufzeit.

- Zur Verbesserung des Faktors der Laufzeit kann der ausgewählte Löser MIDACO sämtliche Iterationsschritte parallelisieren. Zur Zeit wird jede Iteration des Lösers an eine einzige Instanz der Klasse „ObjectiveFunction“ übergeben. Dies ist beim parallelen Rechnen nicht mehr möglich, die gespeicherten Daten gingen verloren. Wenn bekannt ist, wieviele parallele Stränge berechnet werden sollen, muss für jeden Strang eine Instanz angelegt werden. Auch innerhalb der Zielfunktionsimplementierung können Berechnungen parallel laufen, die allerdings noch nicht implementiert sind: Zum Beispiel bei der Entscheidung, ob ein Voxelmittelpunkt innerhalb eines Objektes liegt oder von einer Kameranetzeinstellung verdeckt ist, muss jede Facette jedes Objektes innerhalb der „Environments“ getestet werden; dies kann parallelisiert werden. Der Algorithmus wäre ein gutes Einsatzgebiet für Berechnungen auf einer Graphikkarte.

- Eine Verbesserung des asymptotischen Verhaltens erreicht man durch geeignete Datenstrukturen. Diese Datenstrukturen betreffen die Abfragen, ob ein Voxel innerhalb eines Hindernisses liegt bzw. von einem Objekt verdeckt werden. Für die Verdeckungen kann zum Beispiel ein Oct-Tree hergenommen werden; für Inklusionen ein BSP-Tree. Das Prinzip stammt aus der Computergraphik [8] und wird am Beispiel des Octrees erleutert.

  Für jede Instanz der Klasse „Environment“ muss es einen eigenen Baum geben. In unserem Falle kann dies auch für jedes Objekt geschehen. In der ersten Verzweigung wird der Raum in acht Teile geteilt (daher der Name). Dabei gibt es die drei Kritierien: vorne-hinten, rechts-links, oben-unten. Die nächste Verzweigung geschieht in einem Oktanten der Sicherheitszone, dadurch entstehen wieder acht Teile, usw. Hat man das gewünschte Stopkriterium erreicht, (zum Beispiel die Auflösung der „Discrete Surveillance Area“oder die Unterschreitung einer Facettenanzahl in jedem kleinsten Octanten), bestehen die Blätter des Baumes dann aus Zeigern, die auf die Facetten des „Environments“gerichtet sind. Bei der Betrachtung, welche Facette den Strahl einer Kamera zu einem Voxel unterbricht, wird jedes Voxel, das der Strahl durchquert im Baum nachgeschlagen. Nur diejenigen Facetten werden getestet, deren Zeiger in den zugehörigen Blättern des nachgeschlagenen Teilbaumes liegen.

  Dadurch, dass ein solcher Baum eine abgeleitete Klasse des „Occlusiontests“ ist, kann

diese Idee einfach verwirklicht werden. Hat der Algorithmus vorher ein asymptotisches Verhalten von $O(f)$, mit der Anzahl der Facetten f des Environments, so besitzt er nach der Umstellung ein asymptotisches Verhalten von $O(\log f)$.

Verbesserung der Genauigkeit der Zielfunktion

Anfänglich (in Kapitel 2) werden zwei Beschränkungen gemacht, die die Laufzeit zwar verbessern, jedoch der Genauigkeit der Bewertung entgegenstehen. Diese beiden sind hier erneut aufgelistet:

- Zur Zeit wird der Sichtbereich einer Kamera als ein Kegel mit der Grundfläche eines Kreises betrachtet. Dies ist in Wirklichkeit jedoch nicht der Fall. Der Bereich den eine Kamera abdeckt, ist meist pyramidenförmig. In der Klasse „ObjectiveFunction“ lässt sich dies leicht implementieren. Man leite dazu von der Klasse „Camera“ einfach eine neue Klasse ab. Auch die Ränder des Bildes einer solchen Kamera sind meist verzerrt. Des weiteren sind Kameras mit Tiefensensoren angesprochen worden. Um ein Netzwerk von Tiefenkameras zu implementieren, muss allerdings zusätzlich in die Methode „evaluate()“ der „SightAnalysis“ eingegriffen werden. Zu ändern ist im Prinzip die Abfrage der Tabelle 3.1.
- Es wäre weiterhin der Genauigkeit wegen von Vorteil, die Schnittmengenbildung in (3.6) möglichst nicht nur approximativ durch Voxel darzustellen, sondern algebraisch durchzuführen. Ein Schritt in diese Richtung ist die Bildung von sogenannten „Irregular Space Partitions“ aus [12]. Diese teilen die Sicherheitszone in Polyeder, ähnlich den Voxeln; jene sind lediglich weder alle gleich groß noch regelmäßig. Die Bildung der „Irregular Space Partitions“ ist abhängig von den Kameraeinstellungen. Zum Beispiel ist der gesamte Bereich, der der Bildung eines Clusters auf Voxelbasis entspricht, ein solches. Eine Möglichkeit zur Integration einer so eingeteilten Sicherheitszone gibt es auch in der Implementierung der Zielfunktion; man füge zu der Klasse „Discrete Surveillance Area“ eine davon abgeleitete Klasse hinzu. Eine grobe Richtlinie, an die man sich bei der Berechnung von $M_{v,l,h}$ und $M_{O,l}$ halten sollte, wenn M aus „Irregular Space Partitions“ besteht, ist: Verwende nicht die Mittelpunkte zur Distanzbestimmung sondern Ecken, Kanten bzw. Begrenzungsflächen. Momentan erweisen sich diese Überlegungen allerdings als irrelevant, da zur Zeit erst versucht wird, die Partitions durch einen hochauflösenden Voxelraum anzunähern.

Die in diesem Kapitel angestellten Überlegungen betreffen die Implementierung eines Algorithmusses, der das Problem EPP aus dem vorherigen Kapitel lösen soll. Die Zielfunktion wurde diskretisiert und für diese Diskretisierung wurde ein Löser gesucht. Daraufhin wurde die Laufzeit einer Zielfunktionsauswertung betrachtet. In folgendem Kapitel können dadurch die Testdurchläufe des Programms besser bewerten werden.

Kapitel 4

Experimente und Auswertung

In den nun folgenden Ausführungen werden die Testläufe des im vorigen Kapitel entwickelten Programms ausgewertet. Als Abbruchkriterium des Lösers werden dabei zwei Möglichkeiten in Betracht gezogen. Der Löser erreicht ein durch den Benutzer des Programms spezifizierbares Minimum der Zielfunktion, oder überschreitet eine maximale Zeitschranke. Angesichts dieser Stopkriterien können folgende Gegenstände untersucht werden:

- Ist die Genauigkeit erreicht?
- Welche Laufzeit hat das gesamte Programm/ haben einzelne Teile des Programms?
- Wie viele Iterationsschritte werden benötigt?
- Welche Speicherauslastung wird maximal oder minimal benötigt?

Die Untersuchungsgegenstände werden in diesem Kapitel bezüglich folgender Parameter untersucht: Anzahl der Kameras, Anfangseinstellungen des Kameranetzwerkes, sowie die Beschränkung des Einstellungsraumes E^n, Voxelraumauflösung, Anzahl der Konfigurationen und Ereignisse oder Facetten der verschiedenen Kollektive, Anzahl Objekte und deren unterschiedliche Positionen im Raum.

Dazu wird in Abschnitt 4.1 zunächst die Ausgangssituation mit Hardware und Software der verwendeten Systeme betrachtet, sowie der prinzipielle Basisaufbau der Testläufe. In jedem Testlauf wird nur einer der oben beschriebenen Untersuchungsparameter variiert. In Abschnitt 4.2 befindet sich die Auswertung der Testläufe.

4.1 Ausgangssituation

Bevor die Testläufe im einzelnen besprochen werden, wird zuerst aufgezeigt, mit welcher Hardware und Software die Testläufe durchgeführt worden sind (Abschnitt 4.1.1). Darauf folgt ein Überblick über den Aufbau der Testläufe, welche anhand der Untersuchungsparameter unterschieden werden (Abschnitt 4.1.2). Auf den genannten vorbereitenden Überlegungen kann die im Abschnitt 4.2 dargestellte Auswertung aufbauen.

4.1.1 Verwendete Hardware und Software

Es ist innerhalb eines WAP-Pools mit 10 Rechnern mit der gleichen Hardware- und Softwarekonfiguration gearbeitet worden. Jeder Rechner hat zwei Kerne, wobei ein Testlauf nur auf einem Kern eines Rechners durchgeführt worden ist.

Auf diesen Rechnern läuft das Betriebssystem SuSE Linux Version 10.0 (x86_64) ($: cat /etc/SuSE-release). Das Programm ist in der Entwicklungsumgebung *eclipse* Version 3.2.2 entwickelt und mit dem Compiler „gcc Version 4.0.2 20050901 (prerelease)“ mit der Optimierungsstufe „-O3“ kompiliert. In den Tabellen 4.1 und 4.2 befinden sich Teile der Ausgabe der Abfragen „cat /proc/cpuinfo“ und „cat /proc/meminfo“, die die Information zur Hardware liefern. Die beiden Befehele sind nicht unter Volllast aufgerufen worden. Deswegen wird die Leistung eines Kernes mit c.a. 1GHz angezeigt. Die Leistung ist jedoch dynamisch und beträgt bei Last 2,8 GHz.

model name:	AMD Opteron(tm) Processor 254
cpu MHz:	1004.631
cache size:	1024 KB
clflush size :	64
cache_alignment :	64

Tabelle 4.1: Ein Teil der Ausgabe der Abfrage $: cat /proc/cpuinfo des ersten Kerns

MemTotal:	4038428 kB
MemFree:	886856 kB
Buffers:	431016 kB
Cached:	2079360 kB
SwapCached:	0 kB
Active:	1431004 kB
Inactive:	1138428 kB
SwapTotal:	12586916 kB
SwapFree:	12586916 kB

Tabelle 4.2: Ein Teil der Ausgabe der Abfrage $: cat /proc/meminfo

Diese Hardware- und Softwarekonfiguration wird für alle in folgendem Abschnitt dargestellten Testläufe verwendet.

4.1.2 Aufbau der Testläufe

Da die Zielfunktion nicht konvex, beziehungsweise monoton ist, kann es mehrere lokal optimale Lösungen für das Optimierungsproblem EPP geben. Um eines dieser Minimas zu finden geht der Löser stochastisch vor. Selbst, wenn die gleiche lokal minimale Lösung gefunden wird, ist es wahrscheinlich, dass die betrachteten Untersuchungsgegenstände unterschiedliche Werte aufweisen. Die Untersuchungen werden deswegen in Gruppen von 20 Testläufen der gleichen Untersuchungsparameterwerte durchgeführt. Im folgenden verwenden wir den Begriff „Testlauf“ für einen Durchlauf des Lösers, den Begriff „Gruppe von Testläufen“ für mehrere Testläufe der gleichen Parameterwerte und den Begriff „Untersuchung“ für mehrere Gruppen, in denen nur ein Parameter variiert worden ist.

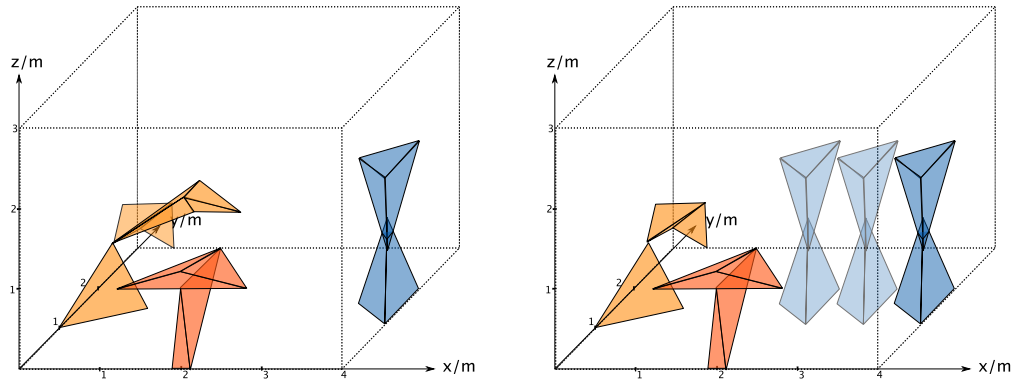

Abbildung 4.1: Testszenario: acht statische, 24 dynamische und 24 unbekannte Facetten, zwei statische, sechs dynamische und sechs unbekannte Objekte, in zwei Konfigurationen (links) und drei Ereignissen (rechts), in einem Raum mit 4m x 3m x 3m Umfang

Solange in den einzelnen Untersuchungen keine anderen Angaben gemacht werden, wird folgender Basisaufbau der Testläufe verwendet. Betrachtet wird stets eine quaderförmige Sicherheitszone der Größe 4m x 3m x 3m. Die Zone enthält im Basisaufbau, wie in Abbildung 4.1 aufgezeigt, ein statisches Kollektiv (rot) mit acht Facetten an zwei Objekten, ein dynamisches (orange) mit 24 Facetten an insgesamt sechs Objekten in zwei Konfigurationen und ein unbekanntes (blau) mit insgesamt 24 Facetten an sechs Objekten in drei Ereignissen. Die Sicherheitszone ist unterteilt in einen Voxelraum der Auflösung 16x12x12. Der Löser darf die Kameras in der ganzen Sicherheitszone platzieren und beliebig orientieren. Die Starteinstellung einer Kamera ist dabei zufällig aus diesem Raum, nämlich

$$E = \overline{U} \times [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}], \text{ mit } \overline{U} := Z$$

entnommen. In diesem Aufbau ist eine Gruppe von Testläufen gestartet worden, diese Gruppe dient als Basis zum Vergleich mehrerer Untersuchungen.

Es gibt folgende zwei Stopkriterien für alle Testläufe: Das erste Stopkriterium ist eine maximale Zeitschranke von drei Stunden. Außerdem wird das Programm abgebrochen, wenn die Zielfunktion einen angegebenen Wert unterschreitet. Dieser Wert ist in den hier betrachteten Untersuchungen so konstruiert, dass er von der maximalen Genauigkeit abhängt, die durch die Auflösung des Voxelraumes möglich ist. Der Abbruchwert hängt dabei nicht von dem Fehler des Modells, sondern von dem Fehler ab, der durch die Disktretisierung entstanden ist. Seien dazu die Kantenlängen eines Voxels $dim_x/dim_y/dim_z$, seien außerdem $f, g \in Z$ zwei beliebige Punkte, wobei g ein gefährlicher Punkt ist und $f \in \nu$ Teil des Voxels ν . Der größte Fehler, der bei der Distanzbestimmung durch die Diskretisierung in solche Voxel gemacht wird unter der Annahme, ν wird zur Distanzbestimmung hergenommen, ist folgender

$$
\begin{aligned}
\max \quad & \left\{ \sum_l \sum_h \omega_{l,h} \cdot \left(d_g(\text{Mittelp. von } \nu) - d_g(f)\right)^2 \right\} \\
\leq \; & \sum_l \sum_h \omega_{l,h} \cdot \max \left\{ \left(d_g(\text{Mittelp. von } \nu) - d_g(f)\right)^2 \right\} \\
= \; & \sum_l \sum_h \omega_{l,h} \cdot \max \left\{ \left(d_{\text{Mittelp. von } \nu}(f)\right)^2 \right\} \\
= \; & \sum_l \sum_h \omega_{l,h} \cdot \left(d_{\text{Mittelp. von } \nu}(\text{beliebiger Eckpunkt von } \nu)\right)^2 \\
= \; & \left(d_{\text{Mittelp. von } \nu}(\text{beliebiger Eckpunkt von } \nu)\right)^2 \\
= \; & dim_x^2 + dim_y^2 + dim_z^2
\end{aligned}
\tag{4.1}
$$

Dies ist die Abweichung zwischen der Distanz zu einem Punkt aus diesem Voxel und dem im Modell verwendeten Mittelpunkt des Voxels. Ist dieser Wert durch die Zielfunktion erreicht, wird das Programm ebenfalls abgebrochen. Wie sich in Abschnitt 4.2.1 herausstellt, muss dieses Abbruchkriterium für Probleme mit Nebenbedingungen (also die in 3.2.1 dargestellte Situation) noch dadurch erweitert werden, dass der Wert des Residuums ebenfalls im Kriterium berücksichtigt wird.

Zunächst werden hier die möglichen Veränderungen der Untersuchungsparameter dargestellt, die einzelne Untersuchungen beeinflussen, um anschließend die Auswertung der Untersuchungen vorzunehmen.

1. Zu jeder Kameraanzahl aus {3, 4, 5, 6, 7, 8, 9} wird eine Gruppe von Testläufen gestartet.
2. Die Kameraeinstellungen werden auf der einen Seite durch Einschränken des Definitionsbereiches selbst, auf der anderen Seite durch Nebenbedingungen vorgeschrieben. Es werden die Vorschriften „An der Decke“ und „Im oberen Viertel“ betrachtet.
3. Die Voxelraumauflösung wird in sechs Gruppen untersucht: $16 \times 12 \times 12$, $20 \times 15 \times 15$, $24 \times 18 \times 18$, $28 \times 21 \times 21$, $32 \times 24 \times 24$ und $36 \times 27 \times 27$.

4. Jeweils in fünf Gruppen wird die Facettenanzahl der statischen, dynamischen bzw. unbekannten Objekte untersucht. Die Objektanzahl bleibt dabei gleich.
   (a) Die Anzahl der statischen Facetten beträgt: 8, 68, 128, 188, 248, 308
   (b) Die Anzahl der dynamischen Facetten beträgt: 24, 84, 144, 204, 264, und 324
   (c) Die Anzahl der unbekannten Facetten beträgt: 24, 84, 144, 204, 264, und 324
5. Von Interesse ist weiterhin eine Untersuchung der Korrelation zwischen der Anzahl der Objekte und besagten Untersuchungsgegenständen. Dazu wird ein ausgewähltes Objekt jedes Kollektivs an eine zufällige Stelle im Raum dubliziert, verdreifacht, etc. Dabei bleibt die x-Koordinate des Objekts bestehen. Durch die Lage der Kollektive in der Sicherheitszone ist die Überschneidung von Objekten verschiedener Kollektive ausgeschlossen, nicht aber die von Objekte eines einzigen Kollektivs. Als Startpunkt der Kameras wird $((2m; 1.5m; 1.5m), 0, 0)$ verwendet. Die drei Untersuchungen belaufen sich auf folgende Variationen:
   (a) das statische Objekt dubliziert, verdreifacht, vervierfacht, verfünffacht
   (b) das dynamische Objekt dubliziert, verdreifacht, vervierfacht, verfünffacht; es wird nur eine Konfiguration verwendet
   (c) das unbekannte Objekt dubliziert, verdreifacht, vervierfacht, verfünffacht; es wird nur ein Ereignis verwendet
6. Beim Untersuchen der Anzahl der Konfigurationen und Ereignisse werden nur zwei dynamische Objekte pro Konfiguration beziehungsweise Ereignis betrachtet. Die zusätzlichen Konfigurationen bzw. Ereignisse werden auf einer linearen Bahn zwischen den bereits bekannten Konfigurationen bzw. Ereignissen hinzugefügt. Untersucht werden
   (a) eine, zwei, drei, vier und fünf Konfigurationen (mit zwei Objekten pro Konfiguration) und drei Ereignisse
   (b) ein, zwei, drei, vier und fünf Ereignisse und drei Konfigurationen

4.2 Auswertung der Testläufe

Die zuletzt angesprochenen Veränderungen der Untersuchungsparameter werden hier ausgewertet und graphisch illustriert. Die Untersuchungsgegenstände werden grundsätzlich auf vertikaler Achse gegen die Veränderung des Untersuchungsparameters auf horizontaler Achse angetragen. Dabei wird eine Gruppe von Testläufen in einer Spalte dargestellt.

Zur Veranschaulichung der Anzahl der Iterationsschritte eines Testlaufs und der benötigten Zeit eines Testlaufs werden sogenannte „Scatter-Plots“ verwendet. Diese stellen alle Testläufe einer ganzen Untersuchung dar. Ein einzelner Testlauf entspricht dem Symbol einer grauen Raute. Der Mittelwert einer Gruppe ist in der Spalte der zugehörigen Gruppe mit einem Viereck verdeutlicht. Durch das Viereck und mehrere Rauten kann demnach der Mittelwert und zugehörige Varianz abgelesen werden. Mehrere Mittelwerte werden zudem durch eine gestrichelte graue Linie verbunden. Diese Linie kann eine Tendenz des Untersuchungsgegenstandes mit fortschreitendem Untersuchungsparameter darstellen. Allerdings ist der Mittelwert und die Darstellung der Varianz von zwanzig Testläufen bei einer überabzählbaren Menge an Kameraeinstellungen nur begrenzt statistisch aussagekräftig.

Der Mittelwert einer gesamten Gruppe von Testläufen wird auch in den Säulendiagrammen dargestellt, die zur Verdeutlichung der Dauer verschiedener Programmteile eines Iterationsschrittes verwendet werden. Die Programmteile, deren Zeitbedarf gemessen wird, gehen aus dem in Abschnitt 3.3.2 angegebenen Aufwand der gesamten Iteration des Lösers hervor. Der Aufwand beträgt:

$$\underset{st.Inkl.}{r \cdot f_s +} \quad \underset{st.Verd.}{I \cdot} \quad \underset{dyn.Inkl + Verdeck.}{\left\{ r\ n\ f_s + \sum_{h=1}^{H} \Big[r\ f_d(h) + r\ n\ f_d(h) + \sum_{l=1}^{L} \big[\ 1 + r\ n\ f_u(l)\ +\ r + c\ +\ r\ f_d(h) \big] \Big] \right\}} \quad \underset{Abst. + unbk.Verd. + Clst. + Plsb. + Abst.}{}$$
(4.2)

Genau genommen wird die Dauer eines Programmteils für eine bestimmte Konfiguration $h \in \{1, ..., H\}$ und ein bestimmtes Ereignis $l \in \{1, ..., L\}$ gemessen; dies geschieht in der Genauigkeit von $10msec$. Sei wiederum

$$f_d^{max} := \max_{h \in \{1,...,H\}} \{f_d(h)\} \quad \text{und} \quad f_u^{max} := \max_{l \in \{1,...,L\}} \{f_u(l)\}$$

Das Säulendiagramm stellt den Mittelwert des Zeitbedarfs folgender Komponenten dar:

DistUnkn „tatsächliche Distanzmessung“: diese Komponente sollte den geringsten Zeitbedarf besitzen, da sie nur durch das Nachschlagen in einer Tabelle mit dem asymptotischen Verhalten des Aufwandes $\mathcal{O}(1)$ bestimmt ist.

Cluster „Nachbarschaftssuchalgorithmus zum Clustern“: besitzt in der Analyse einen Aufwand des asymptotischen Verhaltens $\mathcal{O}(r)$

Plausible „Plausibilitätsabprüfungen“: besitzt in der Analyse einen Aufwand des asymptotischen Verhaltens $\mathcal{O}(c)$ mit $c \leq (\frac{r_x}{2} \cdot \frac{r_y}{2} \cdot \frac{r_z}{2})$

DistClust „geschätzte Distanzmessung“mit dem Aufwand des asymptotischen Verhaltens $\mathcal{O}(r \cdot f_d^{max})$

Visible „Sichtbarkeitsanalyse der SightAnalysis“: diese Komponente beinhaltet sämtliche Inklusions- und Verdeckungsanalysen, die in Zusammenhang mit einem festen $l \in \{1, ..., L\}$ und einem festen $h \in \{1, ..., H\}$ gemacht werden. Das asymptotische Verhalten des Aufwandes dieser Komponente entspricht $\mathcal{O}\big(\ r\ n \cdot \{ f_s\ +\ H\ f_d^{max}\ +\ H\ L\ f_u^{max}\ \}\ \big)$. Dabei ist zu beachten, dass der Faktor, der in der Betrachtung mit dem Landau-Symbol vernachlässigt worden ist, die Größenordnung $(\frac{1}{H} \cdot \frac{1}{L})$ hat.

Betrachtet man die Anzahl der benötigten Iterationsschritte eines Testlaufs, so stellt man fest, dass dieser Mittelwert sehr wohl statistisch aussagekräftig ist. Diese Diagramme ergeben so trotz häufiger Unterschreitung der 10 $msec$ die Tendenz der Aufwandsabschätzung. Dabei ist zu beachten, dass der „Nachbarschaftssuchalgorithmus zum Clustern“, die „Plausibilitätsabprüfungen“ und die „geschätzte Distanzmessung“ nicht nur von den Untersuchungsparametern abhängen, der Zeitbedarf dieser Komponenten verändert sich vielmehr zusätzlich in Abhängigkeit von der Größe und Genauigkeit des Modells des unbekannten Kollektivs.

Im Zusammenhang mit dem Speicherbedarf des Programms ist Mittelwert oder Varianz der Ergebnisse belanglos, interessanter ist der minimale und maximale Wert aller Testläufe einer Gruppe. Dazu wird während eines Testlaufes gleichzeitig alle fünf Sekunden der virtuelle Speicherbedarf in Kilobyte (1024 Byte) abgerufen ($: ps ux, Spalte vsz). Der minimale Wert einer Gruppe wird daher in einem Diagramm mit dem maximalen Wert derselben verbunden. Bei genauerer Betrachtung kommt dem minimalen Wert keine große Bedeutung zu, da die Messungen des Speicherbedarfs kurz vor den zugehörigen Testläufen gestartet worden sind. Der minimale Wert müsste bei einer kontinuierlichen Messung daher verschwindend gering sein. Da die Messungen allerdings nur alle fünf Sekunden wiederholt werden, ist anzunehmen, dass ein Wert zwischen dem Speicherbedarf des Programms selbst und dem maximalen Wert (nach Allokation vieler Variablen) als minimal festgestellt wird.

Die Anzahl der Testläufe einer Gruppe, deren Ergebnis die gewünschte Genauigkeit in drei Stunden erreicht hat, ist ein einfaches Kriterium, das die Qualität des Programms für einen gegebenen Zeitraum, das heißt die Leistung des Programms, beurteilt. In Säulendiagrammen wird dieses Kriterium veranschaulicht. Die Anzahl der erfolgreichen Testläufe einer Gruppe wird über der entsprechenden Säule vermerkt.

Im Folgenden werden die Untersuchungsparameter Schritt für Schritt bearbeitet. Dabei werden zuerst Veränderungen bezüglich der Kameras betrachtet (siehe Abschnitt 4.2.1). Diese betreffen in dieser Reihenfolge die Kameraanzahl und Kamerastandorte. Darauf folgt eine Untersuchungreihe bezüglich der Auflösung des Voxelraumes (Abschnitt 4.2.2). In Abschnitt 4.2.3 werden die statischen, dynamischen und unbekannten Kollektive bezüglich der Positionierung ihrer Objekte untersucht. Der Abschnitt 4.2.4 betrachtet statische, dynamische und unbekannte Facettenanzahlen. Zuletzt werden die Untersuchungsgegenstände bezüglich der Anzahl der dynamischen Konfigurationen und unbekannten Ereignisse untersucht (siehe Abschnitt 4.2.5)

4.2.1 Testläufe bezüglich der Kameras

Die Auswertungen im Hinblick auf zwei verschiedene Kameravariationen stehen in diesem Teil im Mittelpunkt der Betrachtung. Im ersten Abschnitt wird die Anzahl der Kameras variiert; im zweiten werden Auswertungen bezüglich der Nebenbedingungen und Definitionsbereichseinschränkungen dargelegt.

Anzahl der Kameras

In diesem Abschnitt wird die Kameraanzahl variiert. Die erste Auffälligkeit dieser Untersuchung ist im Diagramm 4.2 (links) aufgezeigt. Man kann erkennen, dass die Anzahl der benötigten Iterationsschritte in der betrachteten Parameterwahl vor allem bei geringer Anzahl an Kameras hoch ist. Neben der möglichen Fehleranfälligkeit aufgrund geringer Testdurchläufe kann dies dadurch entstanden sein, dass wenige Kameras ein ungenaueres Modell des unbekannten Kollektivs erzeugen und deswegen die gewünschte Genauigkeit erst recht spät erreicht wird.

Mit der sinkenden Iterationsschrittanzahl ist in dieser Untersuchung auch die benötigte Zeit eines gesamten Testlaufs verringert. Der zugehörige Scatter Plot 4.2 (rechts) hat deswegen eine ähnliche Form wie vorheriger Plot. Nur für diese Untersuchung wurde die Zeitschranke des Abbruchkriteriums hochgesetzt, so dass Zeiten über 10 800 000 Millisekunden gemessen werden können. Der Testlauf der längsten Dauer benötigte umgerechnet 3 Stunden und 39 Minuten.

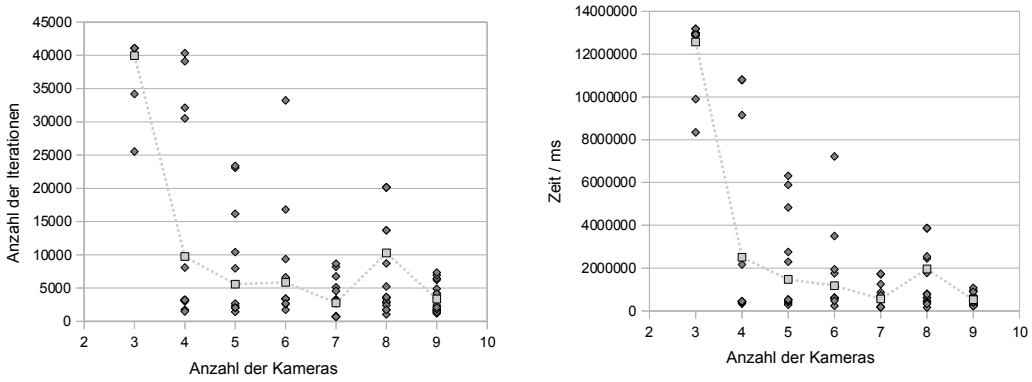

Abbildung 4.2: Scatter Plot: Die Anzahl der Iterationsschritte (links) und die tatsächliche Laufzeit in Millisekunden (rechts) in Abhängigkeit der Kameraanzahl

Auffällig ist zudem, dass das Mittel der benötigten Zeit eines Iterationsschrittes mit Vergrößerung der Kameraanzahl zu sinken scheint, wie in Abbildung 4.3 zu sehen ist. Ein Blick auf die Untersuchung der benötigten Zeit der einzelnen Komponenten des Programmes in Abbildung 4.4 löst dieses Rätsel: Mit steigender Anzahl an Kameras sinkt die Zeit, die für den Nachbarschaftssuchalgorithmus zum Clustern benötigt wird. Diese Aussage bekräftigt noch einmal die Vermutung, dass das Modell des unbekannten Kollektivs durch

eine größere Anzahl an Kameras exakter wird. Aus diesem Diagramm ist desweiteren zu entnehmen, dass die Dauer der Sichtbarkeitsanalyse der Klasse „SightAnalysis“, wie in der Aufwandsabschätzung in Kapitel 3.3.1 und am Anfang dieses Kapitels 4.2 vermutet, mit der Anzahl der Kameras steigt. Da zu erwarten ist, dass das asymptotische Verhalten der Dauer des Clusterns für große Anzahlen n einen konstanten Wert behält, zumindest aber immer größer null sein muss (untere Schranke), wird der Zeitaufwand der Distanzmessung von genau einer Konfiguration und einem Ereignis trotzdem mit großen n steigen. Damit steigt der Zeitaufwand eines Iterationsschrittes und auch der der gesamten Iteration, solange die Anzahl der Iterationsschritte nicht fällt. Dieses Verhalten wird durch die Analyse (3.14) bestätigt.

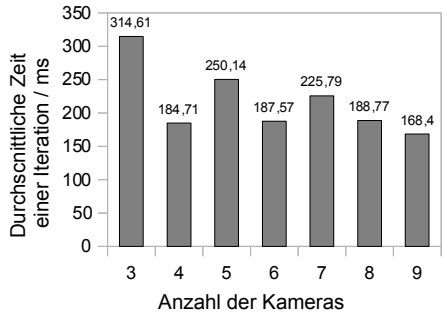

Abbildung 4.3: Diagramm: Die Laufzeit eines Iterationsschrittes in Millisekunden in Abhängigkeit der Kameraanzahl

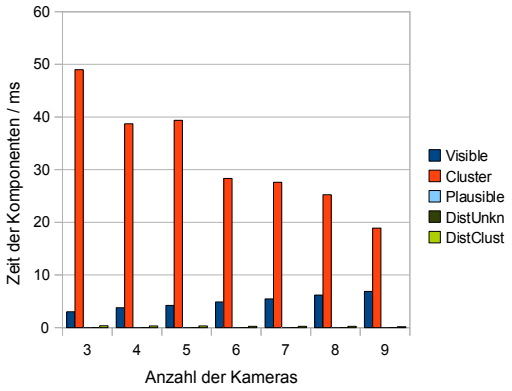

Abbildung 4.4: Diagramm: Die Laufzeit einer Komponente eines Iterationsschrittes in Millisekunden in Abhängigkeit der Kameraanzahl

Bei den Testläufen der drei Kameras haben nur zwei erfolgreich in drei Stunden die gewünschte Genauigkeit erreicht. (Abbildung 4.5) Drei Kameras scheinen demnach nicht zur Modellerstellung dieser Szene geeignet zu sein. Erst sechs Kameras haben alle zwanzig Testläufe erfolgreich beendet. In den folgenden Untersuchungen werden daher immer sechs Kameras verwendet.

Man kann im Diagramm 4.6 erkennen, dass der Speicherbedarf mit der Anzahl der Kameras steigt. Der Speicherzuwachs begründet sich wie folgt: Für jede Kamera werden zusätzlich zu den in Abschnitt 3.3.1 am Ende erwähnten Feldern zum Speichern der Inklusion und Verdeckung drei Felder der gleichen Größe angelegt. Das heißt, ein Feld der Größe des Voxelraumes gehört zu einer Kamera und einem Kollektiv. Eine Zelle dieses Feldes enthält die relative Distanz, mit der das entsprechende Kollektiv die Strecke zwischen Voxel und Kamera unterbricht. Wird die Strecke nicht durch betrachtetes Kollektiv unterbrochen, enthält die Zelle den Wert „-1“. Die Anzahl der Kameras geht dementsprechend linear in den Speicherbedarf ein.

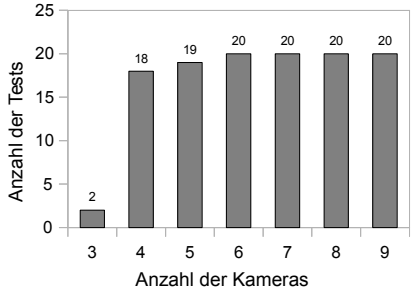

Abbildung 4.5: Diagramm: Die Anzahl der Testläufe, welche die gewünschte Genauigkeit innerhalb drei Stunden erreicht haben, in Abhängigkeit der Kameraanzahl

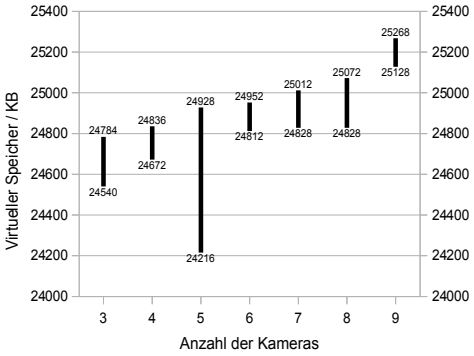

Abbildung 4.6: Diagramm: Der Bedarf an virtuellem Speicher in kB in Abhängigkeit der Kameraanzahl

Definitionsbereicheinschränkungen und Nebenbedingungen

Nach der Abhandlung über die Veränderungen der Untersuchungsgegenstände in Abhängigkeit der Anzahl der Kameras wird hier ein kurzer Überblick über Untersuchungen bezüglich spezieller Definitionsbereichseinschränkungen sowie Beschränkungen der Kameraeinstellungen durch Nebenbedingungen gegeben. In beiden Fällen werden Gruppen von Testläufen bezüglich der zwei Kategorien „Im oberen Viertel“ und „An der Decke“ untersucht und werden mit einer Gruppe von Testläufen des Basisaufbaus ohne Beschränkungen verglichen. Im Falle der Nebenbedingungen werden beim Untersuchen von „Im oberen Viertel“ Ungleichungen der Form „$f_z \geq \frac{3}{4} \cdot 3m$“ für die z-Koordinate f_z des Fokusses aller Kameras verwendet, beim Untersuchen von „An der Decke“ Gleichungen der Form „$f_z = 3m$“.

Der Augenmerk sei zunächst auf die Gesamtzeit aller Iterationsschritte eines Testlaufs gelenkt. In Scatter Plot 4.7 ist zu erkennen, dass alle Testläufe der Definitionsbereichsbeschränkung die maximale Zeitschranke von drei Stunden überschreiten, bevor sie die gewünschte Genauigkeit erreichen. Das Gleiche gilt für die mit Gleichungen als Nebenbedingungen beschränkten Testläufe. Offensichtlich sind alle Beschränkungen zu stark einschränkend; dadurch hat ein Kameranetz mit keiner Einstellung die Möglichkeit das Modell so genau zu beschreiben, dass der dabei eingetretene Fehler die gewünschte Ge-

nauigkeit erreicht. Nur eine Kameraplatzierung im oberen Viertel der Sicherheitszone beschränkt durch Nebenbedingungen führt in allen Fällen vor den drei Stunden zu einem Abbruch des Programms. Geht es um die Anzahl der Testläufe, die die gewünschte Genauigkeit erreicht haben, so lassen sich für diesen Fall lediglich sechs feststellen. Das in Gleichung (4.1) hergeleitete Abbruchkriterium ist offensichtlich nicht vollständig. Nur wenn das Residuum der Nebenbedingungen des Problems kleiner einer zu definierenden Konstanten ist, dürfte der Algorithmus mit dem Abbruchkriterium aus (4.1) beendet werden.

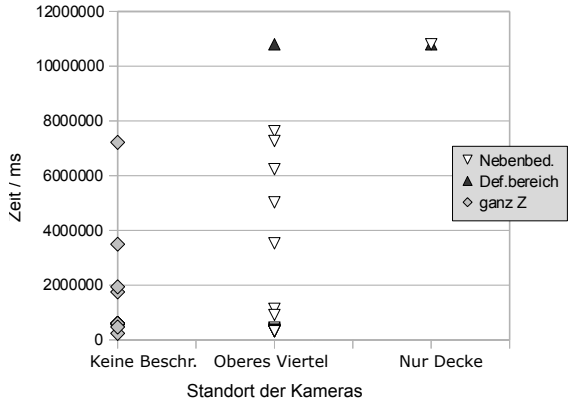

Abbildung 4.7: Scatter Plot: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Einschränkungen des Kamerastandorts

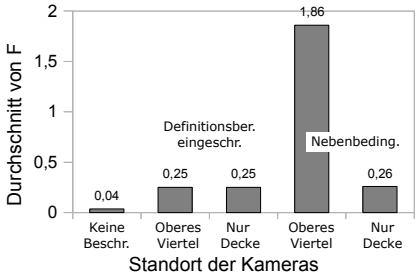

Abbildung 4.8: Diagramm: Der erreichte Zielfunktionswert in Abhängigkeit der Einschränkungen des Kamerastandorts

Ein Blick auf den Mittelwert der Zielfunktion aller Testläufe einer Gruppe (Abbildung 4.8) bestärkt die Vermutung, dass das erlangte Minimum der Zielfunktion des eben beschriebenen Falles deutlich ungenauer als das der anderen Fälle ist. Die erste Spalte zeigt auch hier den Basisaufbau. Während in den anderen Fällen, das heißt in den Fällen der Definitionsbereichsbeschränkung und im Falle der Gleichung als Nebenbedingung durchschnittlich eine mittelmäßige Abweichung der Distanzmessung (Zielfunktion) von $0.25m^2$ erzielt wird, ist die Spalte der Ungleichungen als Nebenbedingungen zu hoch. Dass der durchschnittliche Zielfunktionswert im Falle der Beschränkung durch eine Gleichung nicht ausartet, kann möglicherweise an einer geringen Verbesserung des Residuums in den Testläufen liegen und sich der Zielfunktionswert demzufolge zwingend verkleinern muss. Um diese Untersuchung zu verbessern, müssten die Testläufe noch einmal unter der Betrachtung des Residuums wiederholt werden.

4.2.2 Testläufe bezüglich der Auflösung

Im vorliegenden Abschnitt der Arbeit wird der Voxelraum in sechs Auflösungen betrachtet. Dabei wird mit Ausnahme der Voxelraumauflösung der in Abschnitt 4.1.2 beschriebene Basisaufbau verwendet. Die Voxelraumauflösung reicht von 16x12x12 bis 36x27x27. Sie wird jeweils um vier Voxel in der ersten Koordinate, um drei Voxel in der zweiten und dritten Koordinate erhöht.

Ein Blick auf die Anzahl der Testläufe, die die gewünschte Genauigkeit in drei Stunden erreicht haben, verrät, dass diese Anzahl mit der Verfeinerung des Voxelraumes (also Vergrößerung der Auflösung) fällt (Abbildung 4.9). Für die letzten drei Gruppen von Testläufen ist offensichtlich nicht genug Zeit zur Verfügung. Keiner der Testläufe dieser drei Gruppen hat die gewünschte Genauigkeit erreicht.

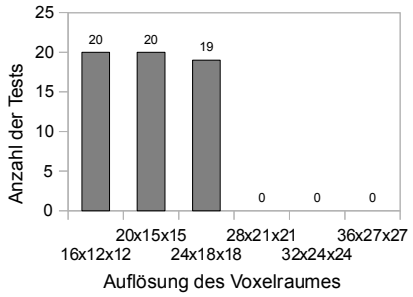

Abbildung 4.9: Diagramm: Die Anzahl der Testläufe, welche die Genauigkeit in drei Stunden erreicht haben, in Abhängigkeit der Auflösung des Voxelraums

Dieses Phänomen ist auch im Scatter Plot 4.10 (rechts) in Hinblick auf die Gesamtzeit aller Testläufe der Untersuchung sichtbar. Je größer die Auflösung, desto mehr Zeit wird für die einzelnen Testläufe benötigt. Dies wird auch in Abschnitt 3.3.2 in der Abschätzung des Zeitaufwandes (3.14) mit Hilfe des Landau-Symbols $\mathcal{O}$ deutlich. Außerdem ist erkennbar, dass mit feinerer Auflösung durchschnittlich weniger Iterationsschritte gemacht werden (Abbildung 4.10 links). Der Grund dafür liegt sicherlich auch in der fehlenden Zeit, die zum erfolgreichen Abschluss benötigtigt wird.

Wird der Zeitverbrauch der einzelnen Komponenten des Programmes einer genaueren Betrachtung unterzogen, so stellt man fest, dass mit Vergrößern der Auflösung nicht nur die Sichtbarkeitsanalyse verlängert wird, sondern auch die Berechnung des Abstandes zu den Clustern (Abbildung 4.11 links) und vor allem aber der Nachbarschaftssuchalgorithmus des Clusterns selbst (Abbildung 4.11 rechts). Die zeitlichen Verlängerungen scheinen nichtlinear zu sein.

Diese Beobachtung wird durch die Aufwandsabschätzung vom Beginn dieses Kapitels 4.2 bestätigt. Die drei Komponenten hängen zwar linear von der Anzahl r der Voxel ab. Jedoch wird r nichtlinear verändert. Zum Beispiel verändert sich der Aufwand der Abstandsmessung von einer Konfiguration $h \in \{1, ..., H\}$ zu den Clustern durch die Vergrößerung des

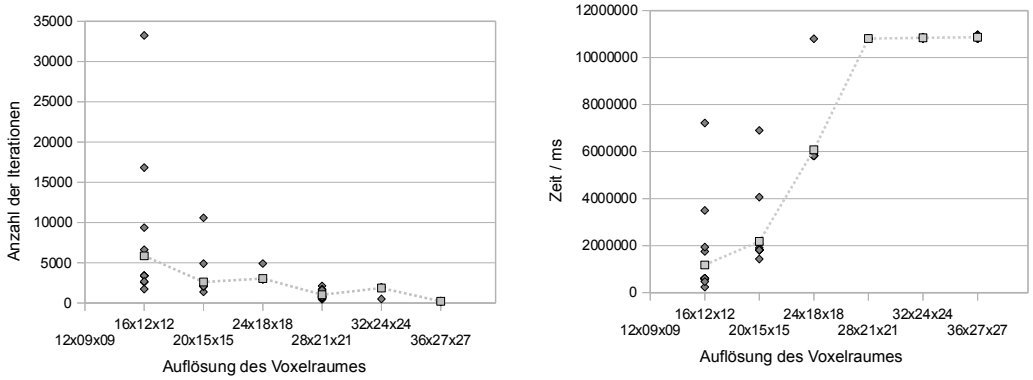

Abbildung 4.10: Scatter Plot: Die Anzahl der Iterationsschritte (links) und tatsächliche Laufzeit in Millisekunden (rechts) in Abhängigkeit der Auflösung des Voxelraums

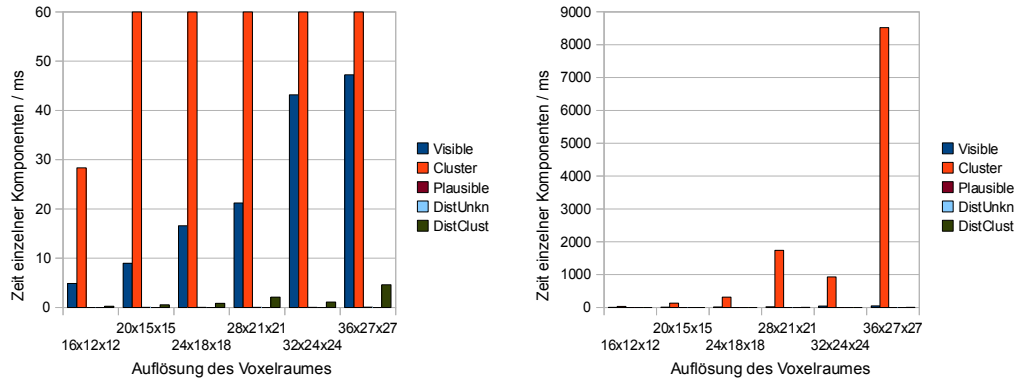

Abbildung 4.11: Diagramm: Die tatsächliche Laufzeit in Millisekunden mit vergrößerter Darstellung (links) in Abhängigkeit der Auflösung des Voxelraums

Voxelraumes um 4 in x-Richtung und um 3 in jeweils y- und z-Richtung so:

$$\begin{aligned} r \cdot f_d(h) &= (r_x^{neu} \cdot r_y^{neu} \cdot r_z^{neu}) \cdot f_d(h) \\ &= \Big((r_x^{alt} + 4) \cdot (r_y^{alt} + 3) \cdot (r_z^{alt} + 3)\Big) \cdot f_d(h) \end{aligned}$$

Im Diagramm 4.12 sind die durchschnittlichen Ergebnisse der Zielfunktion eingetragen, die sich in Abhängigkeit der Auflösung ergeben. Die Ergebnisse der letzten drei Gruppen von Testläufen sind im Durchschnitt nicht zu gebrauchen. (In einem Raum von 4m x 3m x 3m ist der Abstand zum unbekannten Kollektiv durchschnittlich um $\sqrt{1,5}m$ bzw. $\sqrt{1,49}m$ bzw. $\sqrt{2,29}m$ verfehlt) Da das Abbruchkriterium (4.1) abhängig von der Auflösung des Voxelraumes ist, müsste die durchschnittlich erreichte Genauigkeit bei unbeschränkter Dauer eigentlich sinken.

Die Messung des virtuellen Speichers - aufgetragen in Diagramm 4.13 - verrät ebenfalls einen nichtlinearen Zuwachs an Speicherbedarf hinsichtlich der Vergrößerung der Voxel-

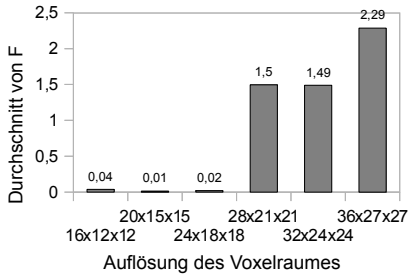

Abbildung 4.12: Diagramm: Der gemittelte Zielfunktionswert bei einer Beschränkung der Dauer auf drei Stunden in Abhängigkeit der Auflösung des Voxelraums

raumauflösung. Zuvor bereits erwähnte Felder aus Abschnitt 3.3.1 und Abschnitt 4.2.1 sind alle in der Größe der Voxelraumauflösung gespeichert, daher ist der Zuwachs kubisch.

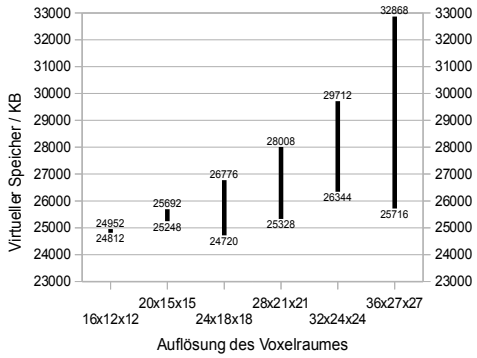

Abbildung 4.13: Diagramm: Der Bedarf an virtuellem Speicher in kB in Abhängigkeit der Auflösung des Voxelraums

4.2.3 Testläufe bezüglich der Objektanzahl

In diesem Abschnitt wird die Objektanzahl, bzw. Positionierung aller Kollektive untersucht. Dabei werden die Kameras am Anfang nicht zufällig platziert und orientiert, sondern alle in der Mitte der Sicherheitszone angebracht und in Richtung der x-Achse orientiert.

Es gibt drei Untersuchungen mit je fünf oder sechs Gruppen von Testläufen. Ein ausgewähltes Objekt eines Kollektivs wird in einer Gruppe von Testläufen dubliziert und zufällig in der Sicherheitszone platziert. In der nächsten Gruppe von Testläufen wird es verdreifacht usw.. Die Position bezüglich der x-Achse bleibt dabei konstant. In der Untersuchung bezüglich der dynamischen Objektplatzierung wird nur eine Konfiguration, in der Untersuchung bezüglich der unbekannten Objektplatzierung nur ein Ereignis betrachtet.

Die Messungen des Speicherbedarfs ergeben bei sämtlichen Objektvariationen das selbe Ergebnis: Bis auf wenige Ausreißer nach unten nimmt der Speicherbedarf zwischen 24808kB und 24972kB ein; das sind ca. 24,3MB.

Statisches Kollektiv

Zuerst seien die zufällig platzierten statischen Objekte betrachtet. Hier ist das untere der beiden Objekte aus der Abbildung 4.1 vervielfältigt worden. Die beiden Scatter Plots 4.14 dieser Untersuchung lassen erkennen, dass die Anzahl der Iterationsschritte und somit auch die Laufzeit des Programms mit der Anzahl der zufällig platzierten statischen Objekte gestreut wird. Die Vermutung liegt nahe, dass die Modellbildung des unbekannten Kollektivs mit diesem Untersuchungsparameter ungenauer wird. Das liegt vermutlich an der Lage der statischen Objekte; diese sind zentriert bezüglich der x-Achse. Die Kameras auf der einen Seite des Kollektivs verursachen auf der anderen Seite einen nicht detektierbaren Bereich, der das Modell des unbekannten Kollektivs vergrößert. Dieser Bereich nimmt an Größe zu, je mehr statische Objekte sich in der Szene befinden.

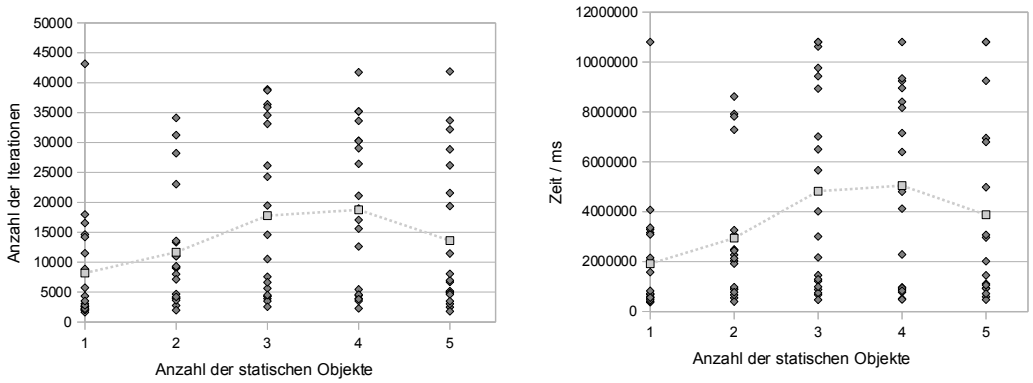

Abbildung 4.14: Scatter Plot: Die Anzahl der Iterationsschritte (links) und tatsächliche Laufzeit in Millisekunden (rechts) in Abhängigkeit der Anzahl der zusätzlichen statischen Objekte

Das Diagramm 4.15 bestätigt diese Vermutung. Je mehr die Verdeckungen und der nicht detektierbarer Bereich in einer Szene vorherrschen, desto mehr Zeit wird zum Clustern benötigt. Die anderen Komponenten des Programms sind nahezu konstant.

Im Diagramm 4.16 wird die Anzahl der Testläufe, welche die gewünschte Genauigkeit in drei Stunden erreicht haben, bezüglich der Anzahl der zusätzlichen statischen Objekte aufgetragen. Die zuvor genannte Streuung der tatsächlich verbrauchten Zeit kann auch daran abgelesen werden, dass die Menge der erfolgreichen Testläufe mit der Anzahl der zusätzlichen statischen Objekte fällt. Bei dargestellter Betrachtung bedarf es einer gewissen Vorsicht: Der Unterschied der erfolgreichen Testläufe verschiedener Säulen beträgt nur „zwei“.

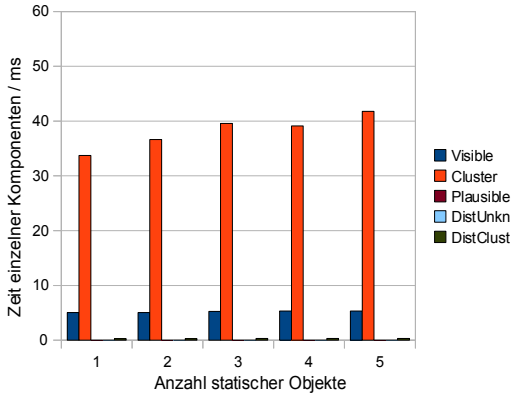

Abbildung 4.15: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der zusätzlichen statischen Objekte

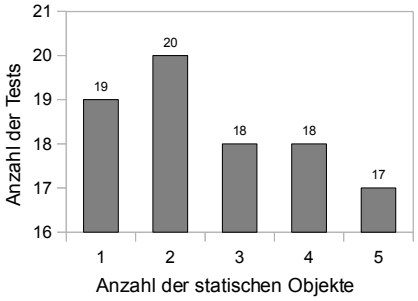

Abbildung 4.16: Diagramm: Die Anzahl der Testläufe, welche die Genauigkeit innerhalb drei Stunden erreicht haben, in Abhängigkeit der Anzahl der zusätzlichen statischen Objekte

Dynamisches Kollektiv

Beim Betrachten der dynamischen Objektanzahl ist der kleinste Tetraeder des in Abbildung 4.1 orange dargestellten dynamischen Kollektivs des Basisaufbaus vervielfältigt worden. Damit die Objektanzahl mit der des statischen Kollektivs verglichen werden kann, wird nur diejenige Konfiguration des Roboters verwendet, bei der die beiden oberen Tetraeder des dynamischen Kollektivs in die Mitte der Sicherheitszone zeigen.

Im Gegensatz zum statischen Kollektiv hält sich die Streuung der Elemente in den Scatter Plots des dynamischen Kollektivs in Grenzen, jedoch kann ein Trend wahrgenommen: Verrät zum Beispiel der Mittelwert der Iterationsschrittanzahl in Scatter Plot 4.17 (links) einer Gruppe von Testläufen noch keine exakte Tendenz, so stellt man diese spätestens beim Betrachten des minimalen Wertes fest. Je mehr dynamische Objekte in der Umgebung sind, desto mehr Iterationsschritte werden mindestens benötigt, um zur gewünschten Genauigkeit zu gelangen. Im Scatter Plot über die benötigte Zeit (rechts daneben) konkretisiert sich dieses Ergebnis. Je mehr dynamische Objekte zufällig in der Umgebung verteilt sind, desto mehr Testläufe erreichen die maximale Zeitschranke von drei Stunden (10800000 Millisekunden).

Zur Verdeutlichung ist in Diagramm 4.18 die Anzahl der erfolgreichen Testläufe gegen die Anzahl der zusätzlichen dynamischen Objekte angetragen. Die erkennbare drastische Reduzierung verdankt der Löser vermutlich unmittelbar den zusätzlichen dynamischen Objekten. Diese erzeugen einen verdeckten Bereich in direkter Nähe zu sich. Durch das veränderte Modell wird die Distanzberechnung fehlerhaft. Dieser fehlerbehaftete verdeckte Bereich kann mit zunehmender Anzahl von dynamischen Objekten nicht in gegebener Zeit mit der gegebenen Anzahl von Kamerasbeseitigt werden.

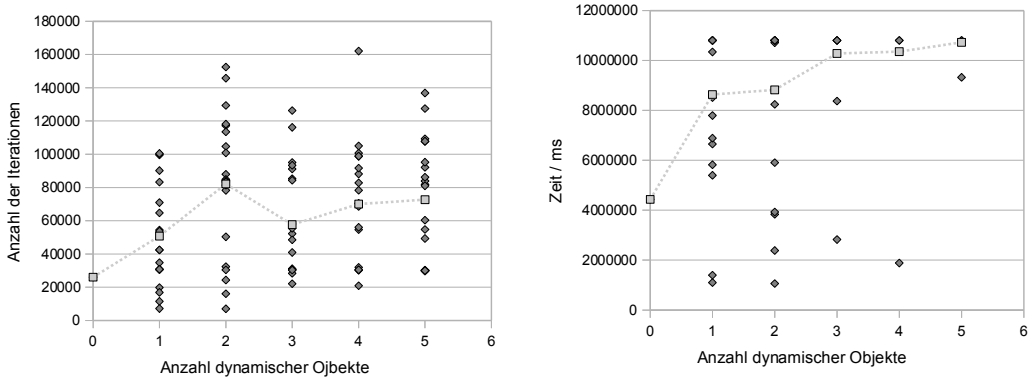

Abbildung 4.17: Scatter Plot: Die Anzahl der Iterationsschritte (links) und tatsächliche Laufzeit in Millisekunden (rechts) in Abhängigkeit der Anzahl der zusätzlichen dynamischen Objekte

Die Theorie der fehlerhaften Distanzmessung findet auch in Diagramm 4.19 Bestätigung. Schon bei einem zusätzlichen dynamischen Objekt ist der Wert der Zielfunktion „0.82“; der Fehler der Distanzmessung ist demnach $\sqrt{0.82m^2} \approx 0.9m$.

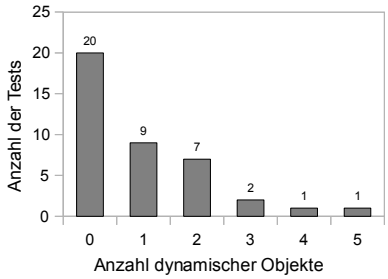

Abbildung 4.18: Diagramm: Die Anzahl der in drei Stunden erfolgreichen Testläufe in Abhängigkeit der Anzahl der zusätzlichen dynamischen Objekte

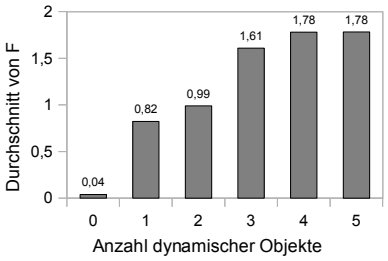

Abbildung 4.19: Diagramm: Der gemittelte Zielfunktionswert in Abhängigkeit der Anzahl der zusätzlichen dynamischen Objekte

In Diagramm 4.20 ist verdeutlicht, dass der Anstieg des Zeitbedarfs eines gesamten Testlaufs nicht die Folge eines erhöhten Zeitanspruchs des Clusterns sein kann. Diese Zeiten sind unregelmäßig. Einen wenn auch geringen Anstieg zeigt allerdings der Zeitbedarf der Distanzmessung zum Modell (DistClust). Die Anzahl der Voxel in den Clustern, die zu dieser Distanzmessung hergenommen werden, hat sich offensichtlich erhöht. Auch dies bestätigt, dass das Modell des unbekannten Kollektivs größer und damit ungenauer geworden ist.

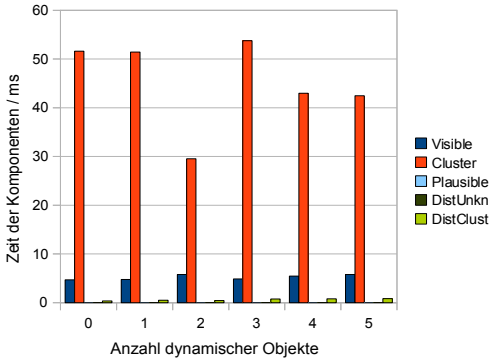

Abbildung 4.20: Diagramm: Die Laufzeit in Millisekunden in Abhängigkeit der Anzahl der zusätzlichen dynamischen Objekte

Unbekanntes Kollektiv

Grundlage einer weiteren Untersuchung ist die Vervielfältigung unbekannter Objekte. Um wiederum den Vergleich zu den Untersuchungen der beiden anderen Kollektive herstellen zu können, wird genau ein Ereignis des unbekannten Objektes betrachtet. Während sich an den Scatter Plots der Untersuchung hinsichtlich der Anzahl der Iterationsschritte und des Zeitbedarfs kein Trend bezüglich der Objektanzahl des unbekannten Kollektivs feststellen lässt - diese Plots sind demnach auch nicht aufgeführt - zeichnet sich bei den Zeitmessungen der einzelnen Komponenten (Abbildung 4.21) die gleiche Tendenz ab wie beim statischen Kollektiv. Das Clustering dauert länger, je mehr Objekte in der Szene sind. Auch hier kann die Ursache einerseits darin liegen, dass das Modell ungenauer gebildet wird. Andererseits umschließen mehr unbekannte Objekte der gleichen Art auch mehr Voxel, die beim Clustern berücksichtigt werden müssen.

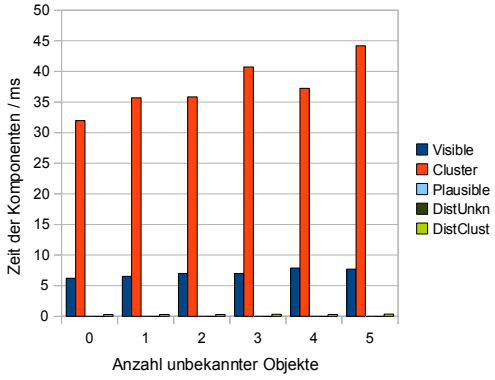

Abbildung 4.21: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der zusätzlichen unbekannten Objekte

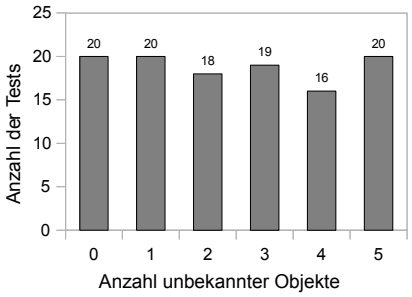

Abbildung 4.22: Diagramm: Die Anzahl der innerhalb drei Stunden erfolgreichen Testläufe in Abhängigkeit der Anzahl der zusätzlichen unbekannten Objekte

Eine kleine Bestätigung der ersten Theorie findet man in der Betrachtung der erfolgreichen Testläufe dieser Untersuchung in Diagramm 4.22. Während anfänglich die gewünschte Genauigkeit in allen Testläufen erreicht wird, scheint das Erreichen dieser Genauigkeit mit Erhöhen der Anzahl unbekannter Objekte zunehmend unsicherer. Wie man an der letzten Säule erkennen kann, können jedoch keine härteren Aussagen diesbezüglich getroffen werden.

4.2.4 Testläufe bezüglich der Facettenanzahl

Dieser Abschnitt behandelt die Abhängigkeit der Untersuchungsgegenstände von der Facettenanzahl der einzelnen Kollektive. Es werden drei Untersuchungen mit je sechs Gruppen von Testläufen anberaumt, eine pro Kollektiv. Beginnend mit dem Basisaufbau werden sechs verschiedene Anzahlen von Facetten des jeweiligen Kollektivs untersucht, die sich alle mindestens um den Betrag „60“ unterscheiden. Betrachtet werden 8 bis 308 Facetten bezüglich des statischen Kollektivs und 24 bis 324 Facetten bezüglich des dynamischen sowie unbekannten Kollektivs. Um die Oberfläche der Kollektive nicht zu verändern, wird für diese Untersuchung eine Facette ausgewählt und im Objekt immer an die gleiche Stelle vervielfältigt.

Im Diagramm 4.23 ist der Speicherbedarf bei Vergrößerung der Facettenanzahl im statischen Kollektiv dargestellt. Bis auf die Variation der minimalen Werte, kann ein geringer Speicherbedarfszuwachs bezüglich einer vergrößerten Facettenanzahl aufgezeichnet werden. Ein solcher Zuwachs kann mit Werten und Steigung der gleichen Größenordnung auch in den Untersuchungen bezüglich der dynamischen und unbekannten Facetten festgestellt werden.

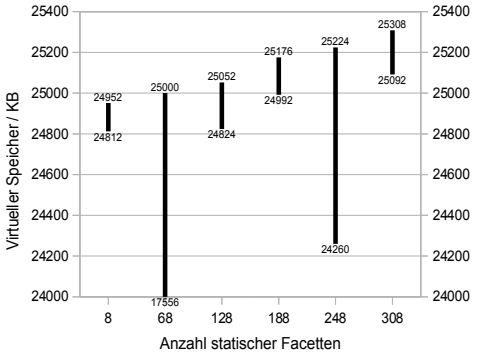

Abbildung 4.23: Diagramm: Der Bedarf an virtuellem Speicher in kB in Abhängigkeit der Anzahl der gesamten statischen Facetten

Auch die Scatter Plots 4.24 der Vervielfältigung statischer Facetten sind beispielhaft für die der anderen beiden Kollektive. Die Streuung der Testergebnisse der Anzahl der Iterationsschritte und der Gesamtzeit der Testläufe bleibt bei allen Gruppen von Testläufen gering. Dies ist keine überraschende Entdeckung: Das Modell des unbekannten Kollektivs wird durch die Vervielfältigung von bereits vorhandenen Facetten, welche die Oberfläche

der Kollektive nicht verändern, nicht beeinflusst. Die Trendlinie des Erwartungswertes zeigt in beiden Plots weder einen erheblichen Anstieg noch ein erhebliches Gefälle.

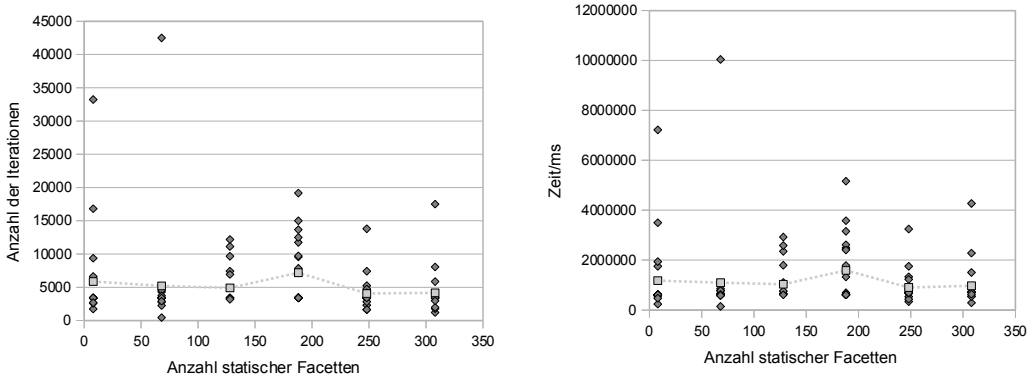

Abbildung 4.24: Scatter Plot: Die Anzahl der Iterationsschritte (links) und tatsächliche Laufzeit in Millisekunden (rechts) in Abhängigkeit der Anzahl der gesamten statischen Facetten

Zunächst soll die Facettenanzahl des Statischen Kollektivs in Bezug auf den Zeitbedarf einzelner Komponenten in Augenschein genommen werden. Eine Facette des statischen Kollektivs wird in der ersten Gruppe von Testläufen 60 mal vervielfältigt, in der zweiten 120 mal, in der dritten 180 mal usw. Die Untersuchung des Zeitbedarfs der einzelnen Komponenten ergibt, dass der Zeitbedarf des Clusterns nur geringe Unterschiede aufweist, was wiederum für eine ähnliche Bildung des Modells des unbekannten Kollektivs spricht. Ein kleiner Trend zeigt sich jedoch trotzdem: Bei der Sichtanalyse der „SightAnalysis“ werden alle Facetten des statischen Kollektivs daraufhin untersucht, ob sie die Strecke zwischen einem Voxel und einer Kamera unterbrechen. Dies zeigt sich an einem erhöhten Zeitbedarf dieser Komponente in Bezug auf die Erhöhung der Facettenanzahl (Diagramm 4.25 (Visible)).

Man beachte nun die Facetten des dynamischen Kollektivs in zwei Konfigurationen. Um einen Vergleichswert zu schaffen, wird in der ersten Gruppe von Testläufen in jeder Konfiguration des Kollektivs eine Facette verdreißigfacht. In der zweiten Gruppe wird die Facette versechzigfacht, u.s.w. Bei einem Zuwachs von dreißig Facetten in jeder Konfiguration des dynamischen Kollektivs, erhält man einen Zuwachs von 60 Facetten des gesamten dynamischen Kollektivs pro Gruppe.

Wie im statischen Falle zeigen die Scatter Plots kaum Veränderungen in Bezug auf die Vergrößerung der Facettenanzahl des dynamischen Kollektivs, die Messung des Zeitbedarfs einzelner Komponenten in Diagramm 4.26 allerdings schon. Der Dauer der Sichtanalyse bei Vergrößerung der Facettenanzahl des dynamischen Kollekivs scheint linear in einer ähnlichen Größenordnung zu steigen wie bei Vergrößerung der Facettenanzahl des statischen. Die Dauer der tatsächlichen Distanzbestimmung (DistUnkn) ist davon nicht beeinflusst, da diese Berechnung vor sämtlichen Iterationsschritten des Lösers geschieht; Dies ist in Abschnitt 3.3.1 in der Veränderung von Gleichung (3.11) in Gleichung (3.12) beschrieben.

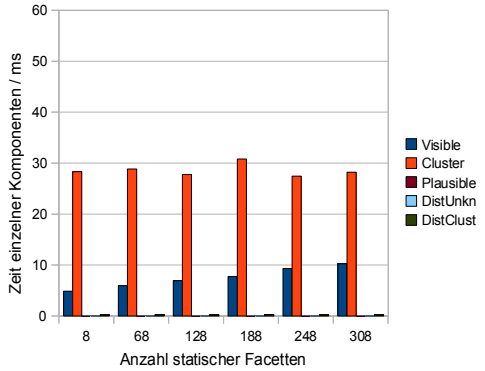

Abbildung 4.25: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten statischen Facetten

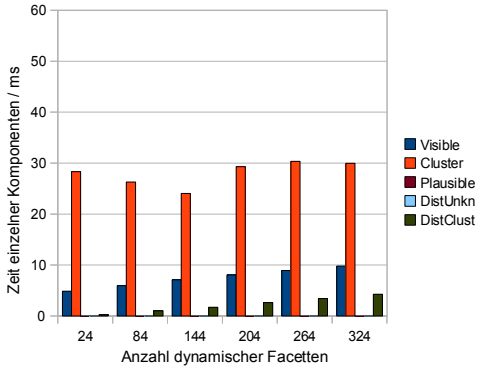

Abbildung 4.26: Scatter Plot: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten dynamischen Facetten

In der nächsten Untersuchung sollen drei Ereignisse des unbekannten Kollektivs verwendet werden. Um ebenfalls auf einen Zuwachs von 60 Facetten pro Gruppe von Testläufen zu kommen, wird hier eine Facette verzwanzigfacht. Wie im statischen Falle kristallisiert sich hier nur bezüglich des Zeitbedarfs einer einzigen Komponente ein klarer Trend heraus, nämlich bezüglich des Zeitbedarfs der Sichtanalyse (Visible) in Diagramm 4.27. Je mehr Facetten im unbekannten Kollektiv vorhanden sind, desto mehr müssen in der Sichtanalyse überprüft werden, daher der Zeitbedarfsanstieg. Es fällt weiterhin auf, dass dieser Anstieg steiler ist als der des statischen Kollektivs.

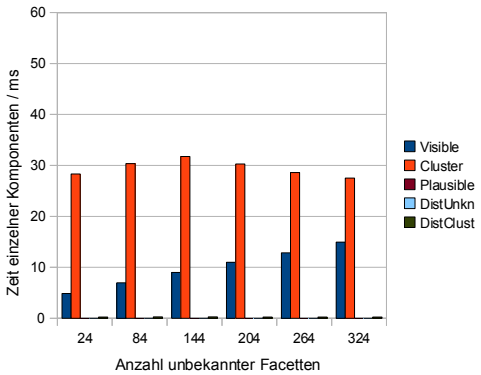

Abbildung 4.27: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten unbekannten Facetten

Die Diagramme 4.25, 4.26 und 4.27 der drei Untersuchungen bezüglich der Facettenanzahl bestätigen die Analyse des Zeitbedarfs der Sichtanalyse vom Beginn dieses Kapitels 4.2 aus folgenden Gründen: Der Zuwachs der statischen Facetten sei im weiteren Verlauf mit Δf_s, der Zuwachs der Facettenanzahl des dynamischen Kollektivs mit Δf_d^{max} und der der

Facettenanzahl des unbekannten Kollektivs mit Δf_u^{max} bezeichnet. Es gilt:

$$\Delta f_s = 60, \quad \Delta f_d^{max} = 30 \quad und \quad \Delta f_u^{max} = 20$$

Damit gilt für den Aufwand der Sichtanalyse:

$$\mathcal{O}\Big(r\ n \cdot \{f_s + \Delta f_s\ +\ H\ f_d^{max}\ +\ H\ L\ f_u^{max}\ \}\Big) =$$

$$= \mathcal{O}\Big(r\ n \cdot \{f_s\ +\ H\ (f_d^{max} + \Delta f_d^{max})\ +\ H\ L\ f_u^{max}\ \}\Big) \text{ denn } H = 2$$

$$\leq \mathcal{O}\Big(r\ n \cdot \{f_s\ +\ H\ f_d^{max}\ +\ H\ L\ (f_u^{max} + \Delta f_u^{max})\ \}\Big) \text{ denn } H \cdot L = 6 \geq \frac{\Delta f_s}{\Delta f_u^{max}}$$

Die Messungen entsprechen nicht nur in diesem Punkt der Theorie: Beim Anstieg der dynamischen Facettenanzahl kann zusätzlich ein Anstieg des Zeitbedarfs der Distanzbestimmung zu den Clustern verzeichnet werden. Ob der Zeitbedarf der anderen Komponenten wie in der Analyse vorhergesagt konstant ist, kann jedoch aufgrund verschwindender Werte nicht festgelegt werden.

4.2.5 Testläufe bezüglich der Ereignisse und Konfigurationen

In den letzten beiden Untersuchungen soll festgestellt werden, wie sich die Anzahl der Konfigurationen und Ereignisse auf die Untersuchungsgegenstände auswirkt. Betrachtet werden jeweils ein bis fünf Konfigurationen oder Ereignisse. Damit die beiden Untersuchungen verglichen werden können, wird das dynamische Kollektiv nur mit zwei Objekten pro Konfiguration ausgestattet. In der Abbildung 4.1 werden diese beiden durch den untersten Tetraeder und den kleinsten dargestellt; das Mittelstück wird für diese Betrachtung entfernt. Bei der Variation der Konfigurationen werden drei Ereignisse festgelegt, bei der Variation der Ereignisse drei Konfigurationen.

Der Speicherbedarf beider Untersuchungen beläuft sich bis auf wenige Ausnahmen in sämtlichen Gruppen von Testläufen zwischen 24672kB und 24960 kB. Ein Zuwachs an Speicherbedarf kann in der Größenordnung von einer einstelligen Konfigurations- bzw. Ereignisanzahl nicht festgestellt werden.

Sowohl die zunehmende Anzahl an Konfigurationen als auch die an Ereignissen hat zur Folge, dass im Scatter Plot der Dauer einer gesamten Iteration eines Testlaufs (Plot 4.28: Konfigurationenvariation, Plot 4.29: Ereignisvariation) ein leichter Aufwärtstrend stattfindet. Dies ist vor allem an der Minimalzeit der Testläufe einer Gruppe zu erkennen, wenn auch nur geringfügig.

Dass der steigende Zeitbedarf nicht an einer steigenden Anzahl von Iterationsschritten liegt, kommt auch in den beiden Diagrammen zum Vorschein, die den Mittelwert des Zeitbedarfs eines einzigen Iterationsschrittes über die Gruppe von Testläufen bemessen (Diagramm 4.30 bei Konfigurationsvariation, Diagramm 4.31 bei Ereignisvariation). Es herrscht in etwa direkte Proportionalität zwischen der Anzahl der Konfigurationen/Ereignisse und des Zeitbedarfs. Nur die letzte Säule des zweiten Diagramms verzeichnet einen Gefälle.

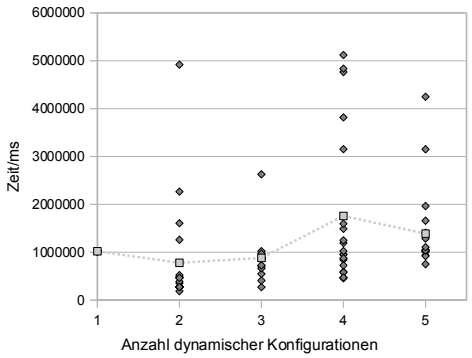

Abbildung 4.28: Scatter Plot: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten dynamischen Konfigurationen

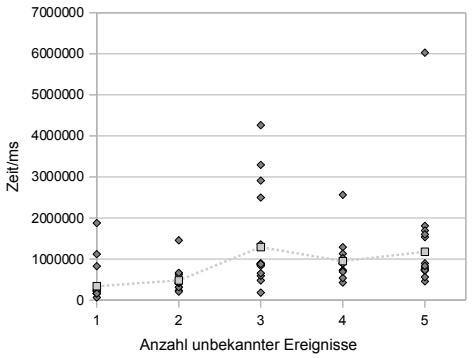

Abbildung 4.29: Scatter Plot: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten unbekannten Ereignisse

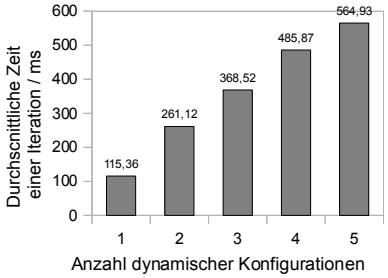

Abbildung 4.30: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten dynamischen Konfigurationen

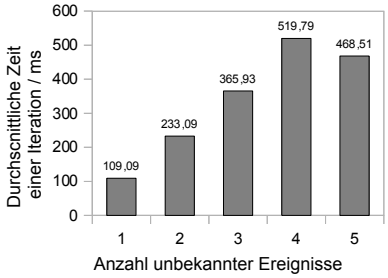

Abbildung 4.31: Diagramm: Die tatsächliche Laufzeit in Millisekunden in Abhängigkeit der Anzahl der gesamten unbekannten Ereignisse

Im Diagramm 4.33 wird der Grund hierfür ersichtlich: Das Clustern hat in dieser Gruppe von Testläufen nicht so lange gedauert wie in den vorangehenden Testläufen. In diesem Diagramm ist bis auf den Ausreißer der letzten Säule ein Anstieg des Zeitbedarfs des Clusterns verzeichnet. Dieser Trend ist im Zusammenhang mit dem dynamischen Kollektiv in Diagramm 4.32 nicht zu erkennen. Die Theorie der Zeitaufwandsanalyse verbindet ein Anstieg des Zeitbedarfs einzelner Komponenten des Programms nicht mit einem Anstieg der Anzahl der Konfigurationen bzw. Ereignisse. Im Gegenteil: Bei einer Betrachtung von vielen statischen oder dynamischen Facetten sollte die durchschnittliche Dauer der Sichtanalyse wegen des am Anfang dieses Kapitels erwähnten Faktors $\frac{1}{H} \cdot \frac{1}{L}$ beim Anstieg der Konfigurations- und Ereignisanzahl fallen. Dieses Phänomen käme natürlich in der Betrachtung der durchschnittlichen Dauer eines gesamten Iterationsschrittes nicht zum Vorschein!

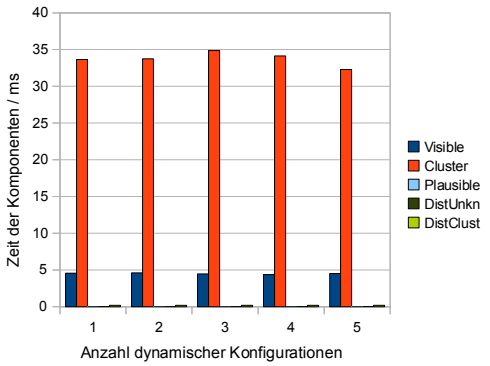

Abbildung 4.32: Diagramm: Der Zeitbedarf einzelner Komponenten des Programms in Abhängigkeit der Anzahl der gesamten dynamischen Konfigurationen

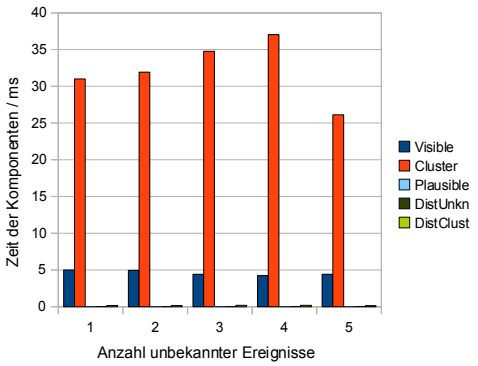

Abbildung 4.33: Diagramm: Der Zeitbedarf einzelner Komponenten des Programms in Abhängigkeit der Anzahl der gesamten unbekannten Ereignisse

4.3 Fazit

In den letzten Abschnitten sind die Gegenstände „Testlauf erfolgreich im Sinne von dem Erreichen der Genauigkeit in drei Stunden“, „Zeitbedarf des gesamten Programms, der Iterationsschritte im Mittel und einzelner Komponenten daraus“, „Anzahl der Iterationsschritte “ und „minimaler und maximaler Bedarf an virtuellem Speicher“ in Bezug auf Veränderungen der Kameras, Auflösung des Voxelraumes und Veränderungen der Kollektive untersucht worden.

Die gewünschte Genauigkeit wird in vielen Fällen in drei Stunden erreicht. Mit Vorsicht zu genießen sind dabei eine zu geringe Anzahl an Kameras und eine zu starke Beschränkung der Einstellungen, so dass das Modell des unbekannten Kollektivs nicht mehr genau genug erstellt werden kann. Auch die Beschränkung durch Ungleichungen führt in wenigen Fällen zu einem erfolgreichen Lauf, denn diese Beschränkung wird im zweiten Abbruchkriterium (4.1) nicht berücksichtigt. Zu Voxelraumauflösungen von mehr als 24x18x18 ist in einer maximalen Zeit von drei Stunden ohne die Optimierungen des Programms aus Abschnitt 3.3.3 ebenfalls nicht anzuraten. Mit der Verstreuung eines Kollektivs in dieser Szene in y- und z-Richtung ist ebenfalls die Anzahl der Tests mit ungenügender Genauigkeit gewachsen. Ausgeartet ist dies vor allem im dynamischen Falle.

Die Laufzeitanalysen der einzelnen Programmkomponenten haben folgendes ergeben: In der Genauigkeit von 10 *msec* fällt die Plausibilitätsabprüfung und die tatsächliche Distanzmessung nicht ins Gewicht. Bei den anderen Komponenten ergibt sich stets ein Wert über Null. Der Mittelwert der Sichtanalyse nimmt stets einen ein- bis zweistelligen Wert an. Im Falle der Objekt- und Konfigurations- bzw. Ereignisveränderungen bleibt dieser Wert annähernd konstant. Bei der Veränderung der Kamera- und Facettenanzahlen steigt dieser Wert annähernd linear, bei der Vergrößerung der Voxelraumauflösung steigt er sogar nichtlinear. Der Zeitbedarf des Clusterns ist in allen Fällen sehr unregelmäßig, jedoch mit zwei bis vier Stellen stets der höchste Wert. Während dieser nur im Falle der zunehmenden Kameraanzahl fällt, bleibt er bei Veränderungen der Facetten-, Konfigurations-, Ereignis- und dynamischen Objektanzahl annähernd konstant. Er steigt bei Erhöhung der Objektanzahl des statischen und unbekannten Kollektivs und bei Erhöhung der Auflösung des Voxelraumes sogar kubisch. Der Zeitbedarf der Distanzmessung zu den erstellten Clustern ergibt im Falle der Auflösung, der dynamischen Objektanzahl und der dynamischen Facettenanzahl einen Anstieg und verändert sich in den anderen Fällen nicht. Dieser Zeitbedarf hat mit Ausnahme der Plausibilitätsabprüfung und der tatsächlichen Distanzmessung stets den geringsten Wert.

Der Zeitbedarf eines ganzen Iterationsschrittes ergibt unabhängig von der Dauer dieser einzelnen Komponenten nur im Falle der Konfigurations- und Ereignisanzahlerhöhung einen Anstieg. Die Scatter Plots der Zeit und der Anzahl der Iterationsschritte sind meist ähnlich strukturiert. Sie unterscheiden sich nur dann erheblich, wenn viele Testläufe die gegebene Zeitschranke von drei Stunden erreicht haben. Ansonsten ist die Verwendung der Verbindungslinie der Mittelwerte als Trendanzeige hinreichend erfüllt. Die Streuung der Anzahl der Iterationsschritte weist meist auf einen erhöhten Zeitbedarf des Clusterns hin.

Bei der Messung des Speicherbedarfs ist der Trend des minimalen Werts häufig nicht erkennbar. Beim maximalen können folgende Unterschiede gemacht werden: Die Vergrößerung der Objektanzahl, Konfigurationsanzahl und Ereignisanzahl, sowie die Einschränkung der Kameraeinstellungen ist in dieser Größenordnung nicht relevant. In allen anderen Fällen steigt der maximale Wert. Deutlich nichtlinear erhöht sich der Speicherbedarf bei Verfeinerung der Voxelraumauflösung, nämlich kubisch.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Diese Arbeit ist in Zusammenhang mit dem DFG-Projekt „SIMERO-2“ des Lehrstuhls für Angewandte Informatik 3 der Universität Bayreuth entstanden. Die Ziele des SIMERO-Projekts sind die Analyse, Entwicklung und Auswertung von Konzepten zur sicheren Koexistenz und Kooperation von Mensch und Roboter. Dabei kommen Sensoren zum Einsatz, deren Platzierung für die Genauigkeit der Konzepte von großer Bedeutung ist. Das Ziel speziell dieser Arbeit ist es, ein Programm zu entwickeln, das bei einer gegebenen Anzahl von Kameras und Informationen zur Umgebung (wie etwa Größe des Raumes und Hindernisse), sowie Informationen zu unbekannten Objekten (z.B. Größe und Aufenthaltswahrscheinlichkeiten eines Menschen) die beste Einstellung des Kameranetzes findet. Die Wertung „beste“ bezieht sich dabei darauf, wie genau ein Modell der unbekannten Objekte durch die Kameras erstellt werden kann, und mit welchem Fehler dadurch die Distanz zwischen Gefahrenpunkten und den unbekannten Objekten berechnet wird.

In Kapitel 2 wird die Problemstellung diesbezüglich analytisch ausgearbeitet. Zuerst werden als Gefahrenquellen nur einzelne Punkte angenommen und keine Hindernisse betrachtet. Dies wird durch die Schwierigkeit der Hindernisse erweitert. Die hinzugenommenen Hindernisse sind sowohl statischer als auch dynamischer Natur, von ihnen ist aber zu jeder Zeit bekannt, wo sie sich befinden. Im letzten Schritt wird das Problem gelöst, dass Hindernisse auch Gefahrenquellen sein können. Das Produkt dieser Analyse ist die Zielfunktion eines nichtlinearen, nicht stetigen Optimierungsproblems.

Diese Zielfunktion wird in Kapitel 3 umgesetzt sowie an einen Löser gekoppelt. Letzter ist eine schwarmintelligente Methaheuristik. In diesem Teil der Arbeit befinden sich auch die Aufwandsabschätzungen bezüglich der Umsetzung der Zielfunktion. Es hat sich herausgestellt, dass die Implementierung der Zielfunktion bereits bezüglich der Laufzeit optimiert ist, wenn auch zu einem geringen Teil. Darauf aufbauend werden mehrere Verbesserungsvorschläge bezüglich Laufzeit und Genauigkeitsanforderungen angegeben.

Im Kapitel 4 ist besonderes Augenmerk auf die Laufzeit in Abhängigkeit verschiedener Un-

tersuchungsparameter der Kameras, Kollektive und Auflösung gelegt und findet durch Experimente Bestätigung. In diesem Zusammenhang werden ebenfalls Messungen bezüglich des virtuellen Speichers, und bezüglich des Erfolgs der Tests erörtert.

5.2 Potential und Grenzen des Programms

In der Literaturrecherche zu diesem Thema sind nur zwei Veröffentlichungen gefunden worden, welche Photogrammetrie mittels des Differenzbildverfahrens betreiben. Viele Vereinfachungen aus diesen Veröffentlichungen werden in dieser Arbeit als Herausforderungen angenommen. Neben dem Hauptziel die Genauigkeit einer Distanzmessung zu maximieren, d.h. den Fehler zu minimieren, sind mehrere kleinere Ziele erreicht worden: Mit dem hier entwickelten Programm hat man die Möglichkeit, die Kameras trotz statischer und sogar dynamischer Hindernisse überall in der Umgebung zu positionieren und orientieren. Der Sichtkegel kann für alle Kameras durch einen Parameter separat eingestellt werden.

Bei der Umsetzung ist auf die Dimension des Problems wenig Rücksicht genommen worden. Das Programm ist somit nur für Offline-Berechnungen bestimmt. Der Zeitbedarf des Programms hängt von der Facettenanzahl und Konfigurations- bzw. Ereignisanzahl, sowie in erster Linie von der Voxelraumauflösung ab. In einer Voxelraumauflösung der Größenordnung von 24x18x18 ist das Erreichen einer Genauigkeit des Abbruchkriteriums (4.1) bei mindestens sechs gegebenen Kameras in drei Stunden mit 2,8 GHz gut möglich. Dabei kommt die Genauigkeit des Ergebnisses stark auf die Platzierung der Hindernisse in der Umgebung an. Hilfreich könnte die Wahl einer geeigneten Anfangseinstellung der Kameras bezüglich der Hindernisse sein, so dass nur noch die Feineinstellungen durch das Programm gemacht werden müssen. Das heißt eine Wand in der Mitte des Raumes könnte ein Problem darstellen, wenn alle Kameras auf einer Seite der Wand starten. Für starke Beschränkungen der Kameraeintellungen ist das Programm vorerst nicht tauglich.

Durch die Verwendung von Oberklassen und Templates kann das Programm sowohl um beschleunigende Klassen erweitert werden, als auch um Klassen, die die Betrachtung eines speziellen Problems verfeinern. So können durch zusätzliche Klassen etwa verschiedene Kamera- sowie Sicherheitszonenmodelle hinzugefügt werden. Zur Verbesserung des Programms können die am Ende des Kapitels 3 erwähnten Möglichkeiten in Betracht gezogen werden. Zur leichteren Handhabung ist es außerdem angenehm, die Kollektive durch ein CAD-Programm erstellen zu können. Dies ist ebenfalls durch eine Schnittstelle in der Implementierung ermöglicht; die Objekte eines Kollektivs können über das STL-Format ausgetauscht werden. Um die Genauigkeit und Laufzeit des entwickelten Programms weiterhin zu verbessern, sollte das Programm in einer weiterführenden Arbeit auf Szenen mit einer anderen Sicherheitszone, anderen Hindernissen und Kameraeinstellungen getestet werden. Dabei ist das Augenmerk zum Beispiel auf die Fragestellung zu richten, ob sich zwischen einer Szene mit gegebener Sicherheitszone, Anzahl der Facetten, Objekte und Konfigurationen bzw. Ereignisse eine Aussage bezüglich der zu verwendenden Anzahl der Kameras und der zu verwendenden Auflösung des Voxelraumes machen lässt.

Anhang A

Klassen und Code

Die Zerlegung der Zielfunktion in Formel (3.1) in dem Kapitel 3.1 dient der Strukturierung der Klassen, die das Programm aufbauen. Folgende Abbildungen zeigen einen Überblick über die wichtigsten Klassen gegeben, die zur Implementierung hergenommen worden sind. Den erstellten Klassen liegt ein Geometrie-Paket von Herrn Dipl.-Inf. Stefan Kuhn zu Grunde, in dem die Angabe von Punkten („Point3D“) und Vektoren („Vektor3D“) eines Vektorraumes, sowie die Angabe geometrischer Figuren wie einer Kugel oder die Einteilung des Raumes („VoxelCuboid“), erleichtert werden. Auch „Dimension3D“ gehört dazu.

Im ersten Bild sind die Variablen der Klassen aufgeführt, dabei bedeuten die Zeichen +, # und − in dieser Reihenfolge: *public*, *protected* und *private*. Neben den Datentypen aus oben angesprochenem Geometriepaket werden in der „SightAnalysis“ zudem der Typ „visibility-Vector“, was dem Datentyp „vector<bool>“ in der Größe der Anzahl der Voxel entspricht, und der Typ „intersectionDistance“ verwendet, was dem Datentyp „vector<double>“ der gleichen Größe entspricht. Im zweiten Bild werden die Methoden der gleichen Klassen aufgezeigt. Dabei ist zu beachten, dass der Buchstabe *c* „const“ und der Buchstabe v „virtual“ bedeutet.

Wie bereits in Kapitel 3.1 erwähnt, fügt die Klasse „ObjectiveFunction“ alles zusammen, was zur Berechnung des Zielfunktionswertes nötig ist. Dabei ist die Simulierung der Veränderung der Bereiche v in Abhängigkeit der Kameraeinstellungen in die Klasse „SightAnalysis“ ausgelagert.

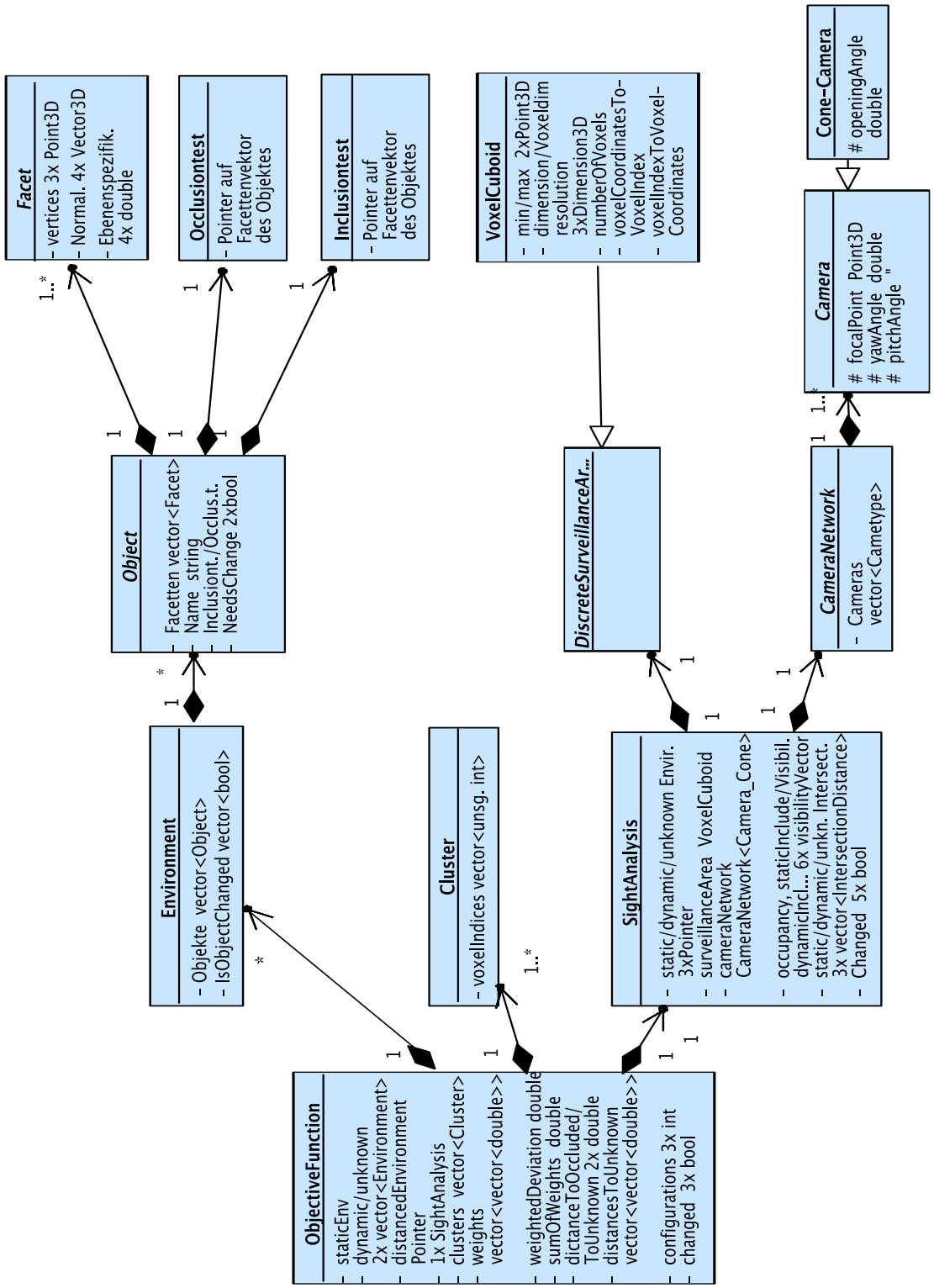

Abbildung A.1: Klassendiagramm: Variablen

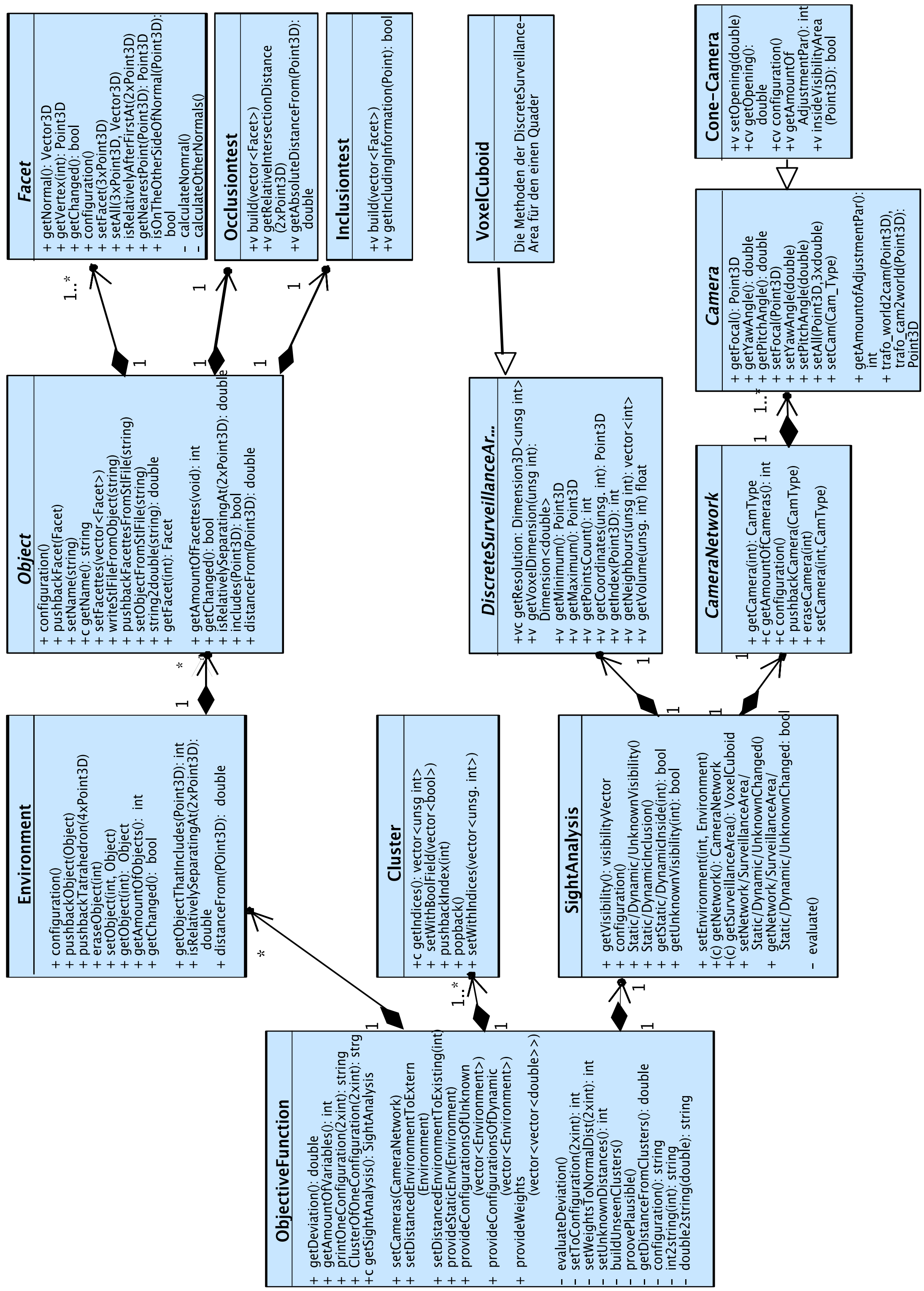

Abbildung A.2: Klassendiagramm: Methoden

Anhang B

Ordnerstruktur der DVD

In diesem Anhang befindet sich die Ordnerstruktur der mitgelieferten DVD. Dabei bedeutet der Buchstabe „*d*“ vor der Datei *directory* oder Verzeichnis.

Diplomarbeit.pdf

d **Klassen des Programms**

In diesem Ordner befinden sich alle Header- und Source-Dateien, die zur Umsetzung von Nöten gewesen sind. Sie werden im Folgenden kurz aufgelistet.

d geometry
- VoxelSpaceDefinition.h
- Voxel.h
- Vector3D.h
- SphereModel.h
- SphereModel.cpp
- Sphere.h
- Point3D.h
- Point2D.h
- Dimension3D.h
- Dimension2D.h

Camera_Cone.h
Camera.h
CameraNetwork.h
Cluster.h
DiscreteSurveillanceArea.h
Environment.h
Facet.h
Inclusiontest_brute_force.h
Inclusiontest_coherent.h
Inclusiontest_konvexObjects.h
Inclusiontest.h
main.cpp
midaco.f
Object.h
ObjectiveFunction.cpp
ObjectiveFunction.h
Occlusiontest_bruteforce.h
Occlusiontest.h
Output.txt
SightAnalysis.h
Tetraeder.stl
Tetraeder36.stl
Tetraeder63.stl
Tetraeder115.stl
Tetraeder151.stl
Tetraeder511.stl
VoxelCuboid.h

d **Testlauf**

In diesem Ordner befinden sich alle Input- und Output-Dateien, die während der Testphase des Programms erstellt worden sind. Jeder Unterordner des Ordners „Testlauf“ist ein Verzeichnis, in dem genau eine Untersuchung enthalten ist. Das Shell-Script „0mittelwert.sh“ hat der Automatisierung der Mittelwertbildung des Zeitbedarfs einzelner Komponenten gedient. Neben den Ordnern für die Gruppen von Testläufen enthält dieser auch die Auswertungen der Untersuchung im ods-Format.
Jeder Ordner innerhalb eines Untersuchungs-Ordners enthält genau eine Gruppe von Testläufen. Der Inhalt der Verzeichnisse der Gruppen von Testläufen ist Folgender: Die beiden Shell-Scripte und die „*_input.txt“-Datei haben das Testen einer Gruppe automatisiert. Die „*.out“-Datei ist der Standardoutput des letzten Testlaufs der Gruppe. Die Hilfsdatei „EVAL_*_mem.txt“ hat der Abspeicherung des virtuellen Speicherbedarfs alle fünf Sekunden gedient. In „EVAL_*.csv“und „F_EVAL_*.csv“ sind letztlich die Testergebnisse dieser Gruppe dokumentiert. Dieser Inhalt des Ordners wird im Folgenden nicht mehr aufgelistet.

0mittelwert.sh
d Aufloesung
- *d* Aufloesung_1
- *d* Aufloesung_2
- *d* Aufloesung_3
- *d* Aufloesung_4
- *d* Aufloesung_5
- Testlauf_Aufloesung__F.csv
- Testlauf_Aufloesung__F.ods
- Testlauf_Aufloesung.ods

d Fac_dyn
- *d* Fac_dyn_1
- *d* Fac_dyn_2
- *d* Fac_dyn_3
- *d* Fac_dyn_4
- *d* Fac_dyn_5
- Testlauf_Fac_dyn_F.csv
- Testlauf_Fac_dyn_F.ods
- Testlauf_Fac_dyn.ods

d Fac_stat
- *d* Fac_stat_1
- *d* Fac_stat_2
- *d* Fac_stat_3
- *d* Fac_stat_4
- *d* Fac_stat_5
- Testlauf_Fac_stat_F.csv
- Testlauf_Fac_stat_F.ods
- Testlauf_Fac_stat.ods

d Fac_unkn
- *d* Fac_unkn_1
- *d* Fac_unkn_2
- *d* Fac_unkn_3
- *d* Fac_unkn_4
- *d* Fac_unkn_5
- Testlauf_Fac_unkn_F.csv
- Testlauf_Fac_unkn_F.ods
- Testlauf_Fac_unkn.ods

d Kam_Anzahl
- *d* Kam_3
- *d* Kam_4
- *d* Kam_5
- *d* Kam_6
- *d* Kam_7
- *d* Kam_8
- *d* Kam_9
- Testlauf_Kam_Anzahl_F.csv
- Testlauf_Kam_Anzahl_F.ods
- Testlauf_Kam_Anzahl.ods

d Kam_Standort
- *d* Kam_Def_ganzeDecke
- *d* Kam_Def_oberesViertel
- *d* Kam_Neb_ganzeDecke
- *d* Kam_Neb_oberesViertel
- Testlauf_Kam_Standort_F.csv
- Testlauf_Kam_Standort_F.ods
- Testlauf_Kam_Standort.ods

d Konf_dyn
- *d* Konf_dyn_1
- *d* Konf_dyn_2
- *d* Konf_dyn_3
- *d* Konf_dyn_4
- *d* Konf_dyn_5

Testlauf_Konf_dyn_F.csv
Testlauf_Konf_dyn_F.ods
Testlauf_Konf_dyn.ods

d Konf_unkn

d Konf_unkn_1
d Konf_unkn_2
d Konf_unkn_3
d Konf_unkn_4
d Konf_unkn_5
Testlauf_Konf_unkn_F.csv
Testlauf_Konf_unkn_F.ods
Testlauf_Konf_unkn.ods

d Obj_dyn

d Obj_dyn_0
d Obj_dyn_1
d Obj_dyn_2
d Obj_dyn_3
d Obj_dyn_4
d Obj_dyn_5
Testlauf_Obj_dyn_F.csv
Testlauf_Obj_dyn_F.ods
Testlauf_Objekt_dyn.ods

d Obj_stat

d Obj_stat_1
d Obj_stat_2
d Obj_stat_3
d Obj_stat_4
d Obj_stat_5
Testlauf_Obj_stat_F.csv
Testlauf_Obj_stat_F.ods
Testlauf_Objekt_stat.ods

d Obj_unkn

d Obj_unkn_0
d Obj_unkn_1
d Obj_unkn_2
d Obj_unkn_3
d Obj_unkn_4
d Obj_unkn_5
Testlauf_Obj_unkn_F.csv
Testlauf_Obj_unkn_F.ods
Testlauf_Objekt_unkn.ods

Literaturverzeichnis

[1] Bodor, R. ; Drenner, A. ; Schrater, P. ; Papanikolopoulos, N.: Optimal Camera Placement for Automated Surveillance Tasks. In: *J Intell Robot Syst* Band 50 (2007), S. 257 bis 295

[2] Bronstein, I.N. ; Semendjajew, K.A. ; Musiol, G. ; Mhling, H.: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2008

[3] Ercan, A.O. ; Gamal, A.E. ; Guibas, L.J.: Camera Network Node Selection for Target Localization in the Presence of Occlusions / Department of Electrical Engineering and Department of Computer Science, Stanford University. 2006. – Forschungsbericht

[4] Ercan, A.O. ; Yang, D.B. ; Gamal, A.E. ; Guibas, L.J.: Optimal Placement and Selection of Camera Network Nodes for Target Localization. In: *Lecture notes in computer science*. Springer Verlag Berlin Heidelberg, 2006, S. 389–404

[5] Erdem, U.M. ; Sclaroff, S.: Optimal Placement of Cameras in Floorplans to Satisfy Task Requirements and Cost Constraints, 2004

[6] Fiore, L. ; Fehr, D. ; Bodor, R. ; Drenner, A. ; Somasundaram, G. ; Papanikolopoulos, N.: Multi-Camera Human Activity Monitoring. In: *J Intell Robot Syst* 52 (2007), Nr. 1, S. 5–43

[7] Fiore, L. ; Somasundaram, G. ; Drenner, A. ; Papanikolopoulos, N.: Optimal Camera Placement with Adaptation to Dynamic Scenes, 2008

[8] Foley, J.D. ; Dam, A. van ; Feiner, S.K. ; Hughes, J.F.: *Computer Graphics: Principles and Practice in C*. Pearson, 1995

[9] Hartley, R.I.: Estimation of Relative Camera Positions for Uncalibrated Cameras, 1992

[10] Holt, R.J. ; Hong, M. ; Martini, R. ; Mukherjee, I. ; Netravali, R. ; Wang, J.: Summary of Results on optimal camera placement for boundary monitoring, 2007

[11] Kuhn, S. ; Henrich, D.: Multi-View Reconstruction of Unknown Objects in the Presence of Known Occlusions / Universität Bayreuth, Angewandte Informatik III (Robotik und Eingebettete Systeme). 2009. – Forschungsbericht

[12] Kuhn, S. ; Henrich, D.: Multi-View Reconstruction in-between Known Environments / Lehrstuhl für Angewandte Informatik III (Robotik und Eingebettete Systeme) , Universität Bayreuth. 2010. – Forschungsbericht

[13] Mittal, A.: *Generalized Multi-Sensor Planning.* 2006

[14] Mittal, A. ; Davis, L.S.: *Visibility Analysis and Sensor Planning in Dynamic Environments.* 2004

[15] Mittal, A. ; Davis, L.S.: A General Method for Sensor Planning in Multi-Sensor Systems: Extension to Random Occlusion. In: *Int J Comput Vis* 76 (2006), S. 31–52

[16] Olague, G.: Design and simulation of photogrammetric networks using genetic algorithm / Departamento de Ciencias de la Computacin, Division de Fisica Aplicada, Centro de Investigacin Cientifica y de Educacin Superior de Ensenada. – Forschungsbericht

[17] Olague, G. ; Dunn, E.: Developement of a practical Photogrammetric Network Design Using Evolutionary Computing / Centre for Scientific Research and Higher Education of Ensada. 2007. – Forschungsbericht

[18] Olague, G. ; Mohr, R.: Optimal 3D Sensors Placement to Obtain Accurate 3D Points Positions / Movi-Gravir, Zirst, Montbonnot Saint Martin. 1998. – Forschungsbericht

[19] Olague, G. ; Mohr, R.: Otimal Camera Placement for Accurate Reconstruction / Institut National de recherche en informatique et en automatique. 1998. – Forschungsbericht

[20] Schlüter, M. ; Egea, J.A. ; Antelo, L.T. ; Alonso, A.A. ; Banga, J.R.: An extended ant colony optimizaion algorithm for integrated process and control system design. In: *Ind. Eng. Chem.* 48(14) (2009), S. 6723–6738

[21] Schlüter, M. ; Egea, J.A. ; Banga, J.R.: Extended Ant Colony Optimization for non-convex Mixed Integer Nonlinear Programming. In: *Comput. Oper. Res* 36(7) (2009), S. 2217–2229

[22] Schlüter, M. ; Gerdts, M.: The oracle penalty method. In: *Springer Science+Budiness Media, LLC* 30 (2008)

[23] Yang, D.B. ; Shin, J. ; Guibas, L.J. ; Ercan, A.O.: Sensor Tasking for Occupancy Reasoning in a Network of Cameras / Stanford University. 2004. – Forschungsbericht

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Bayreuth, den 26. April 2010 .....

Maria Hänel