

hochschule mannheim

Hochschule Mannheim
Institut für Robotik

Diplomarbeit

Erstellung eines Geländeprofils anhand von Stereobildern

vorgelegt von: Joel Bout
eingereicht am: 15. März 2008
Betreuer: Prof. Dr. Thomas Ihme
Zweitkorrektor: MSc Kai Wetzelsberger

Zusammenfassung

Diese Arbeit befasst sich mit dem automatisierten Erstellen eines Geländeprofils anhand von Stereobildern, um mobilen autonomen Robotern das Navigieren mittels eines Stereokamerasystems zu ermöglichen. Dazu werden in jeder Roboterposition, unter Verwendung des Scaling Invariant Feature Transform-Algorithmus (SIFT-Algorithmus), gemeinsame, markante Punkte zwischen dem linken und dem rechten Kamerabild gesucht. Die Kenntnis der relativen Kamerastellung erlaubt, die von diesen Punktpaaren abgebildeten Merkmale dreidimensional zu lokalisieren. Punkte, die in mehreren Positionen wiedererkannt werden, machen es möglich, mehrere Ansichten zueinander in Relation zu bringen. Mit den so erhaltenen dreidimensionalen Punkten als Stützwerte, kann durch Interpolation eine Höhenkarte der Umgebung erstellt werden.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden sind als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Bout, Joel

Inhaltsverzeichnis

1. Einleitung.....	1
1.1. Motivation.....	1
1.2. Zielsetzung.....	2
1.3. Gliederung.....	2
2. Grundlagen.....	3
2.1. Der Laufroboter.....	3
2.2. Die Simulationsumgebung.....	5
2.3. Das Kameramodell.....	6
2.4. Approximation von Kreuzungspunkten.....	7
2.5. Triangulation.....	8
2.6. Der RANSAC-Algorithmus.....	10
2.7. Gauß'sches Weichzeichnen.....	11
2.8. Interest-Operatoren.....	12
Corner Detection.....	12
Der SIFT-Algorithmus.....	13
3. Lösungskonzept.....	20
3.1. Dreidimensionale Lokalisierung.....	20
3.2. Terrainrekonstruktion.....	21
4. Implementierung.....	24
4.1. Merkmalsuche.....	24
4.2. Korrespondenzanalyse.....	25
4.3. Dreidimensionale Lokalisierung.....	26
4.4. Erstellen einer Höhenkarte.....	27
4.5. Filter.....	30
Probleme des RANSAC Filters.....	30
Alternative Filter.....	31
4.6. Kombination mehrerer Ansichten.....	31
Triangulation des vereinten Modells.....	34
5. Ergebnisse.....	36
5.1. Siftgpu.....	36
5.2. Blockweises Vergleichen.....	37
Gleichmäßig verteilte Punkte.....	37
Feste Blockgröße.....	39
5.3. Erstellen der Höhenkarte.....	40
5.4. Probleme.....	41
Ausreißer.....	41
6. Zusammenfassung und Ausblick.....	43
6.1. Erreichte Resultate.....	43
6.2. Mögliche Umsetzung.....	43
6.3. Geschwindigkeitsoptimierung.....	43
Parallelisierbarkeit.....	43
Korrespondenzanalyse zwischen Ansichten.....	44
6.4. Verbesserung der Resultate.....	44
Detailliertere Interpolation.....	45
Abbildungsverzeichnis.....	46
Literaturverzeichnis.....	47

1. Einleitung

1.1. Motivation

Seit Jahrhunderten träumen Menschen davon, dass Roboter ihnen die Arbeit abnehmen könnten. Das Wort Roboter wurde von dem tschechischen Schriftsteller K. Capek vom geprägt und stammt aus dem slawischen Wort „rabota“, das soviel wie Arbeit bedeutet.

Doch diese Vision wurde nur sehr eingeschränkt realisiert und Roboter werden noch vorwiegend stationär oder unter Laborbedingungen eingesetzt. Dies liegt unter anderem daran, dass Roboter ihre Umgebung kennen müssen, um darin arbeiten zu können. Im Fall von stationären Robotern ist es ein leichtes, Umgebungsdaten manuell vorzugeben, aber bei mobilen Robotern stößt man auf erhebliche Probleme. Selbst innerhalb von Gebäuden verändert sich das Umfeld ständig, und für einen Roboter, der unter freiem Himmel eingesetzt werden sollte, wäre dies gänzlich unmöglich. Außerdem muss ein mobiler Roboter in der Lage sein können, seine Position zu bestimmen, ansonsten ist selbst die detaillierteste Karte nutzlos.

Damit autonome mobile Roboter vielseitig eingesetzt werden können, müssen sie in der Lage sein, selbstständig Informationen über ihr Umfeld zu sammeln, und sich in diesem zurecht finden. Ähnlich wie bei dem Menschen, sind auch bei Robotern die vielseitigsten Sensoren zum Erfassen von Umgebungsinformationen die „Augen“, sprich die Kameras. Die Fülle an Informationen in Bildern, führen dazu das diese deutlich aufwendiger zu interpretieren sind. Eine der einfacheren Anwendungen von Mehrkamerasystemen, ist das Gewinnen von Tiefeninformationen.

Obwohl bereits einige Algorithmen hierzu untersucht worden sind, insbesondere „Corner Detection“-Algorithmen und der „Graph-Cut“-Algorithmus, liefern diese noch ungenaue Resultate oder benötigen zu viel Rechenzeit. Des Weiteren ist es schwierig, Bilder aus verschiedenen Ansichten miteinander in einen Kontext zu bringen, da durch die Bewegung des Roboters zwischen zwei Aufnahmen sowohl eine Rotation als auch eine Skalierung der dargestellten Objekte auftreten kann. Bisherigen Algorithmen war es deshalb noch nicht möglich zuverlässig Korrespondenzen zwischen diesen zu finden.

1.2. Zielsetzung

Ziel dieser Arbeit ist es, eine Vorgehensweise zu finden, mit Hilfe derer die Stereobilder des autonomen mobilen Roboters Lauron annähernd in Echtzeit, ausgewertet und in eine interne Abbildung umgewandelt werden können. Anhand dieser internen Abbildung oder Karte, muss Lauron in der Lage sein können, die jeweils nächste Teilstrecke zu planen.

Da die Arbeit an Lauron selbst sehr aufwändig ist, werden die Algorithmen zuerst innerhalb der Simulationsumgebung untersucht. Erst wenn sich die Lösung in der Simulationsumgebung bewährt hat, kann man dazu übergehen, die Algorithmen auf Lauron umzusetzen.

1.3. Gliederung

Diese Diplomarbeit ist wie folgt gegliedert:

Die Einleitung in Kapitel 1 beschreibt die Motivation, Zielsetzung, sowie die Gliederung dieser Aufgabe.

Die grundlegenden Techniken werden in Kapitel 2 erläutert. Diese schließen die mathematischen Modelle und Bildverarbeitungsalgorithmen ein, die im Rahmen der Diplomarbeit verwendet worden sind.

Das Konzept, wie man von Stereobildern zu einer Geländekarte gelangen kann, wird in Kapitel 3 behandelt. Hier werden die Lösungsetappen und die benötigten Zwischenschritte beschrieben.

Die Implementierung des Lösungskonzepts wird in Kapitel 4 vorgestellt. An dieser Stelle werden die Vor- und Nachteile des vorgestellten Lösungsweges aufgezeigt.

Das Ergebnis wird in Kapitel 5 präsentiert. Ein besonderes Augenmerk wird dabei auf die Laufzeiten zur Generierung der Geländekarten gelegt.

Möglichkeiten zur Fortführung zeigt das Kapitel 6. Hier werden die Erkenntnisse, die während dieser Diplomarbeit gewonnen wurden, zusammengefasst und Ansätze vorgestellt, wie diese in die Praxis umgesetzt und die Laufzeiten und Qualität der Karten weiterhin verbessert werden können.

2. Grundlagen

In diesem Kapitel werden die Rahmenbedingungen der Diplomarbeit, sowie die mathematischen Grundlagen und die verwendeten Algorithmen erläutert.

In dieser Diplomarbeit wird ein dreidimensionales Koordinatensystem verwendet, das der Darstellung am Monitor, mit dem Ursprung im oberen linken Eck, nachempfunden wurde. Die X-Achse verläuft von links nach rechts, die Y-Achse von oben nach unten, und die Z-Achse zeigt vom Betrachter weg.

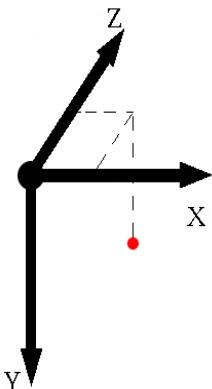


Abbildung 1:
Koordinatensystem

2.1. Der Laufroboter

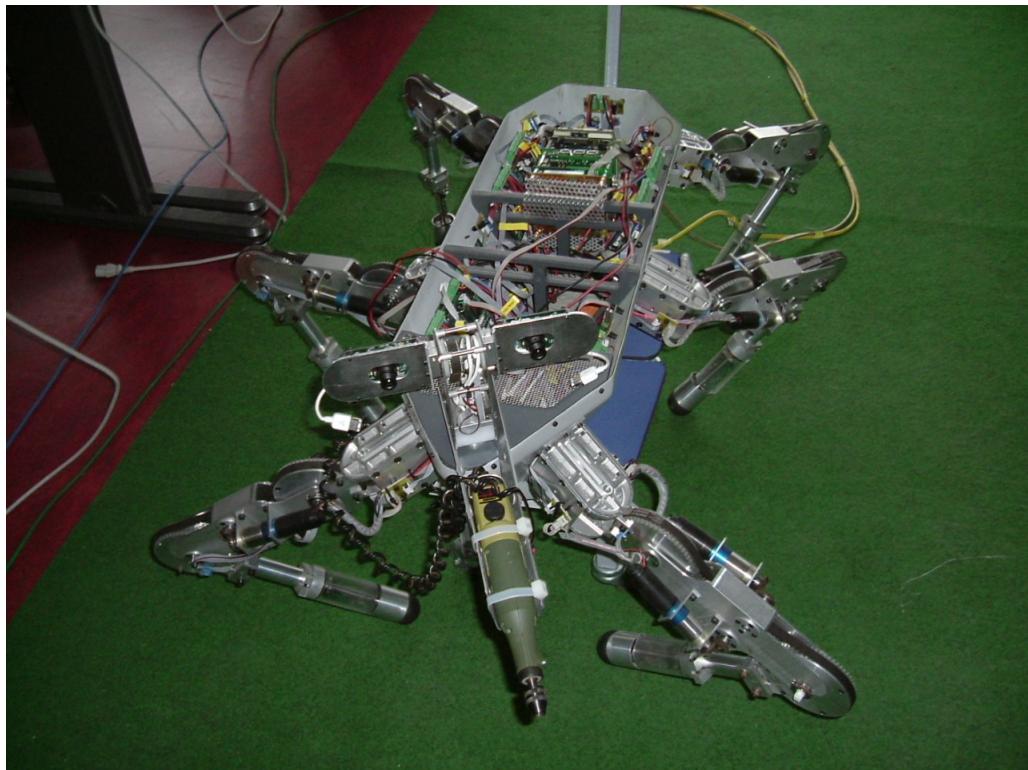


Abbildung 2: Lauron IV b

Lauron IV b, eine Weiterentwicklung des damals neuronal gesteuerten Lauron (**Laufrobotor neuronal gesteuert, bzw. Legged Autonomous Robot Neural Controlled**), ist ein sechsbeiniger, insektenartiger Laufroboter. Entwickelt wurde er von der Gruppe Interaktive Diagnose- und Servicesysteme (IDS) des Forschungszentrum Informatik Karlsruhe.

Lauron IV b ist der indischen Stabheuschrecke nachempfunden und besteht aus einem starren Hauptkörper, sechs identischen Beinen, die jeweils drei Freiheitsgrade besitzen und einen drehbaren Kopf. Seine Maximalgeschwindigkeit beträgt 0,5 Kilometer pro Stunde.

Der drehbare Kopf, auf dem zwei Kameras angebracht sind, kann sich neigen, ähnlich wie der menschliche Kopf. Die Kameras werden via IEEE 1394¹ an die Steuerung angeschlossen. Sie liefern Graustufenbilder mit einer Auflösung von 512 x 512 Pixeln. Diese bilden die Grundlage der Tiefeninformationsgewinnung.

Gesteuert wird Lauron von einem standardisierten PC-104 Board PC. Als Betriebssystem wird die Linux Distribution Debian mit einem Real-Time Kernel benutzt, um den Echtzeitanforderungen gerecht zu werden. Die aktuelle Steuerung baut auf dem Modular Controller Architecture 2 (MCA 2) Framework auf.

¹ auch bekannt unter dem Namen Firewire ©Apple Inc.

2.2. Die Simulationsumgebung

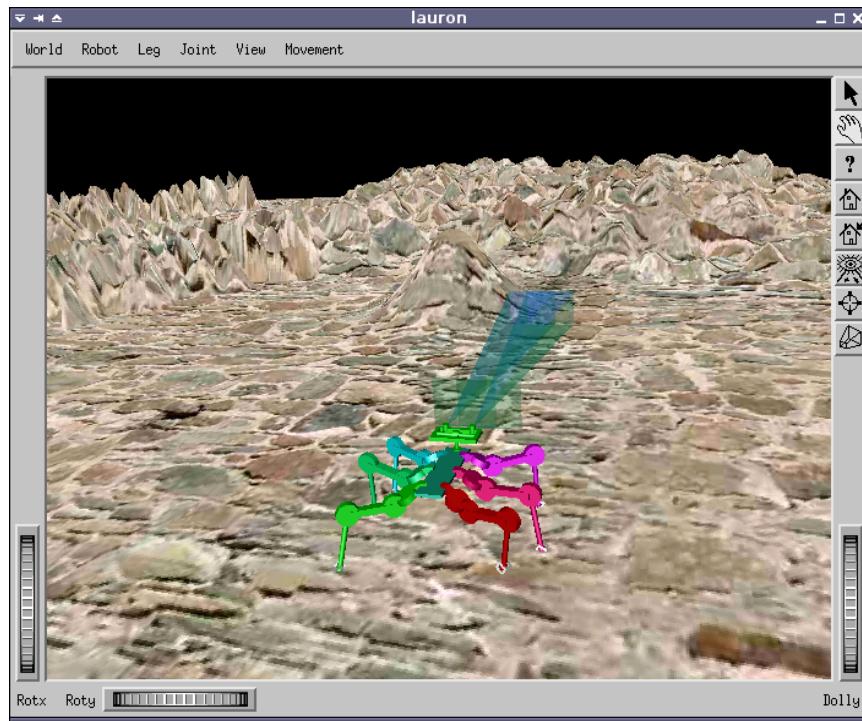


Abbildung 3: Simulationsumgebung für Lauron

Die Simulationsumgebung, die zurzeit im Robotikinstitut der HS Mannheim verwendet wird, wurde von Andre Herms entwickelt, um verschiedene Laufalgorithmen zu untersuchen. Sie wurde in C++ geschrieben und verwendet die Open Inventor Bibliothek zum Darstellen des Roboters und seines Umfeldes. Das Terrain wird als eine Höhenkarte dargestellt und kann aus einem Bild eingelesen werden. Ausgehend von einer Startposition und einem gewünschten Ziel, wird die Wahl der Strecke als ein Optimierungsproblem betrachtet. Ein openMosix Cluster wurde eingesetzt, um die Laufzeit zu reduzieren, und Random Sampling hat sich als das günstigste Verfahren für die Laufplanung herausgestellt. [1]

Die Betrachtungsweise der Aufgabe als ein mathematisches Problem, von der Anfangskonfiguration ausgehend, hin zu einer Endkonfiguration, setzt voraus, dass der Roboter die gesamte Karte kennt. Davon kann man aber in der Realität nur selten ausgehen, und selbst wenn der Roboter das gesamte Terrain im Voraus kennen würde, könnten kleine Ungenauigkeiten den Roboter schnell von seiner geplanter Bahn abweichen lassen, sodass ein neuer Bewegungsablauf für die gesamte verbleibende Strecke berechnet werden müsste.

Deshalb wurde die ursprüngliche Version weiterentwickelt, und in der Diplomarbeit von Uli Ruffler an der HS Mannheim praxistauglicher gemacht. Indem der Roboter immer nur einige Schritte im Vorhinein berechnet, wird der Algorithmus fast echtzeittauglich. Obwohl die Folgeversion immer noch davon ausgeht, dass der Roboter das gesamte Terrain kennt, ist dies ein wichtiger Schritt, um einen Bewegungsablauf anhand einer unvollständigen Karte planen zu können. [2]

Die Simulationsumgebung wurde im Rahmen dieser Arbeit nicht verändert, sondern die von ihr generierten Stereobildansichten wurden genutzt, um anhand dieser die Geländekarte zu rekonstruieren.

2.3. Das Kameramodell

Der Versuch, eine dreidimensionale Szene auf eine zweidimensionale Fläche fotografisch abzubilden, geht bis in das 15. Jahrhundert zurück. Bis heute ist das Grundprinzip das gleiche, alle Raumpunkte werden über ein optisches Zentrum auf eine Bildebene abgebildet.

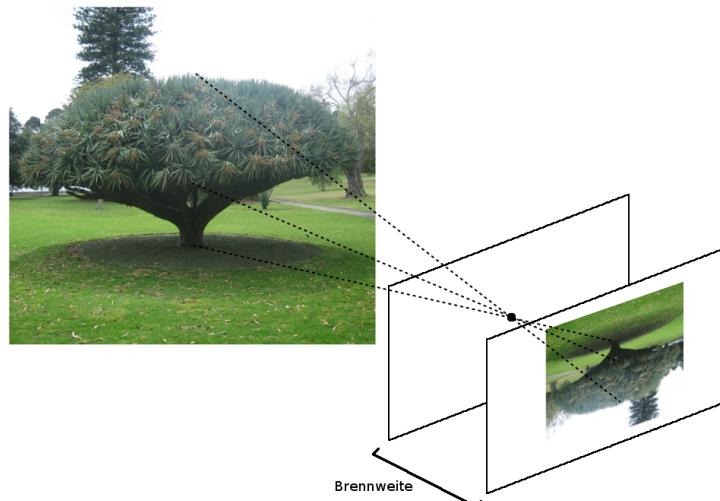


Abbildung 4: Prinzip einer Lochkamera

Bei modernen digitalen Kameras wird das Bild bereits in der Kamera gespiegelt erfasst. Deshalb kann das mathematische Modell vereinfacht werden, indem man sich die Bildebene vor dem Brennpunkt, der als Kamerazentrum gewählt wird, vorstellt. Mathematisch gesehen ist eine Kamera eine Abbildung, die einen dreidimensionalen Punkt X auf $P(x)$ abbildet. [3]

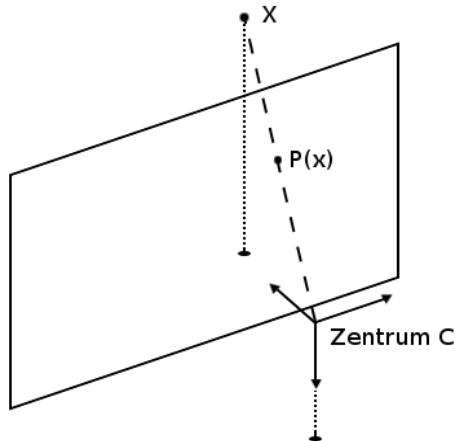


Abbildung 5: Mathematisches Kameramodell

Die Kamera in der Simulationsumgebung dieser Arbeit ist eine idealisierte Kamera, weshalb ihre Bilder nicht mehr bereinigt werden müssen, bevor die Bilderkennungsalgorithmen auf sie angewandt werden können. Bei realen Kameras kann es zu einer Scherung des Bildes, einer kissenförmigen Verzerrung durch die Linse, oder einem natürlichen Randlichtabfall durch das Objektiv kommen, welche die Resultate der Terrainerkennung beeinträchtigen würden. Deshalb wäre in diesem Fall eine Kalibrierung der Kamera, um das Bild vorbearbeiten zu können, notwendig.

2.4. Approximation von Kreuzungspunkten

Im Rahmen dieser Diplomarbeit werden Kreuzungspunkte von Linien im dreidimensionalen Raum gesucht, die sich aufgrund von Ungenauigkeiten leicht verfehlten können. Als Alternative zu dem realen Kreuzungspunkt der Linien kann der Punkt, der zu beiden Linien einen minimalen Abstand hat, als Approximation gewählt werden.

Dieser Punkt kann mit Hilfe eines mathematischen Handgriffs gefunden werden. Dazu definiert man die Linien, die bisher mit Hilfe von zwei Punkten, P_0 und P_1 definiert wurden, anhand eines Punktes P_0 und einem Vektor $\vec{u} = P_1 - P_0$.

$$L_0 = \{P \in \mathbb{R}^3 \mid \exists s \in \mathbb{R} \text{ mit } P = P_0 + s \cdot \vec{u}\}$$

$$L_1 = \{Q \in \mathbb{R}^3 \mid \exists t \in \mathbb{R} \text{ mit } Q = Q_0 + t \cdot \vec{v}\}$$

Gesucht werden dann s und t , so dass der Abstand zwischen P und Q minimal ist. Dieser Abstand kann mit Hilfe eines Vektors w dargestellt werden:

$$w(s, t) = P(s) - Q(t)$$

Dieser Vektor kann mit Hilfe der Differentialrechnung bestimmt werden [4]. Geometrisch lässt sich diese Aufgabe aber mit einem geringeren Rechenaufwand lösen. Vorausgesetzt, dass die Linien nicht parallel zueinander verlaufen – also der abgebildete Punkt sich nicht im Unendlichen befindet – ist der Vektor von P_s zu Q_t der einzige Vektor, der rechtwinklig zu \vec{u} und \vec{v} steht. [5] Dies ist gleichbedeutend mit:

$$\vec{u} \cdot \vec{w} = 0 \text{ und } \vec{v} \cdot \vec{w} = 0$$

Durch Einsetzen von \vec{w} in die obige Formel erhält man:

$$\begin{aligned} \vec{u} \cdot (P(s) - Q(t)) &= 0 \\ \equiv \vec{u} \cdot ((P_0 + s \cdot \vec{u}) - (Q_0 + t \cdot \vec{v})) &= 0 \\ \equiv \vec{u} \cdot (P_0 + s \cdot \vec{u}) - \vec{u} \cdot (Q_0 + t \cdot \vec{v}) &= 0 \\ \equiv \vec{u} \cdot P_0 + \vec{u} \cdot s \cdot \vec{u} - \vec{u} \cdot Q_0 - \vec{u} \cdot t \cdot \vec{v} &= 0 \\ \equiv \vec{u} \cdot s \cdot \vec{u} - \vec{u} \cdot t \cdot \vec{v} &= \vec{u} \cdot P_0 + \vec{u} \cdot Q_0 \\ \equiv s \cdot (\vec{u} \cdot \vec{u}) - t \cdot (\vec{u} \cdot \vec{v}) &= \vec{u} \cdot (P_0 + Q_0) \\ \equiv s \cdot (\vec{u} \cdot \vec{u}) - t \cdot (\vec{u} \cdot \vec{v}) &= -\vec{u} \cdot w_0 \end{aligned}$$

$$\text{analog dazu: } s \cdot (\vec{v} \cdot \vec{u}) - t \cdot (\vec{v} \cdot \vec{v}) = -\vec{v} \cdot w_0$$

Durch Substituieren von $a = \vec{u} \cdot \vec{u}$, $b = \vec{u} \cdot \vec{v}$, $c = \vec{v} \cdot \vec{v}$, $d = \vec{u} \cdot w_0$ und $e = \vec{v} \cdot w_0$ folgt:

$$s = \frac{be - cd}{ac - b^2} \quad t = \frac{ae - bd}{ac - b^2}$$

Nach Einsetzen von s und t erhält man die Punkte P_s und Q_t zwischen denen der approximierte Kreuzungspunkt liegt.

$$P_{\text{Kreuzung}} = P_s + \frac{P_s - Q_t}{2}$$

2.5. Triangulation

Die Delaunay-Triangulation ist ein Verfahren, mit dem eine Ebene E in Dreiecke zerlegt wird, die über eine Punktmenge P gespannt werden. Delaunay definiert eine günstige Zerlegung als eine Dreiecksmenge, für welche gilt, dass in den Kreisen, die

durch die drei Eckpunkte eines jeden Dreiecks definiert sind, keine Punkte enthalten sein dürfen, außer auf dem Umfang.

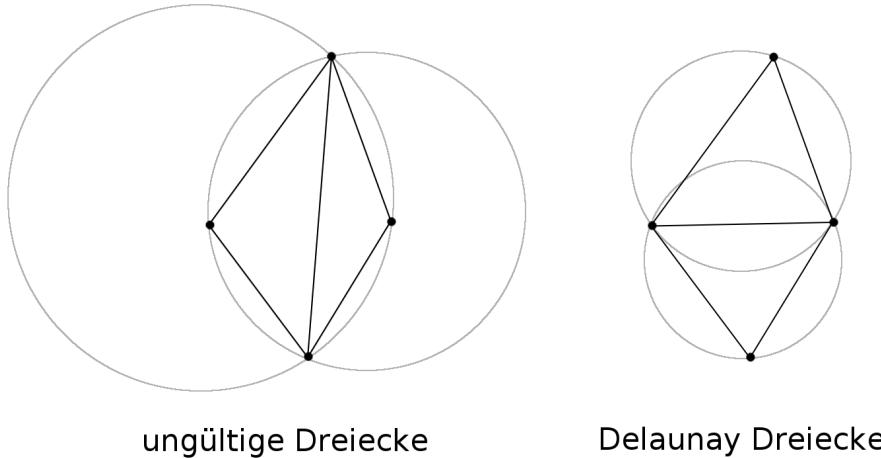


Abbildung 6: Delaunay-Kriterium

Der einfachste Algorithmus basiert auf der Eigenschaft, dass zwei Dreiecke, welche eine gemeinsame Kante haben und das Delaunay-Kriterium nicht erfüllen, durch den Wechsel der Trennkante (wie in Abbildung 6 gezeigt), in zwei neue Dreiecke geteilt werden, welche die Bedingungen erfüllen. Ein inkrementeller Algorithmus, um eine beliebige Punktmenge zu zerlegen besteht aus drei Schritten:

- 1) Drei beliebige Punkte zu einem Dreieck zusammenfassen
- 2) Ein beliebiger Punkt, der noch nicht Teil der Dreieckzerlegung ist, wird hinzugefügt und mit den benachbarten Punkten, die bereits im Diagramm enthalten sind, verbunden, so dass neue Dreiecke entstehen.
- 3) Untersuchen, ob die neuen Dreiecke das Delaunay-Kriterium erfüllen, und falls dies nicht der Fall ist, das Dreieckpaar, welches die Bedingungen verletzt, neu aufteilen. Die dadurch neu entstandenen Dreiecke müssen entlang der nicht gemeinsamen Kanten rekursiv geprüft werden und falls nötig, müssen diese Kanten wieder gewechselt werden.

Punkt 2 und 3 werden solange wiederholt, bis alle Punkte in die Dreieckszerlegung aufgenommen worden sind.

Da für jeden Punkt, der hinzugefügt wird, im schlimmsten Fall alle bisherigen Dreiecke neu zerlegt werden müssen, hat dieser Algorithmus eine Laufzeit von $O(n^2)$

Diese Laufzeit kann mit Hilfe des „Divide and Conquer“-Ansatzes deutlich verbessert werden. Hierzu wählt man geschickt eine Linie, entlang welcher man die Punktmenge teilt. Die Untermengen zerlegt man rekursiv und fügt die so entstandenen Triangulationen am Ende wieder zusammen. Mit diesem Algorithmus kann man die Laufzeit auf $O(n \log n)$ reduzieren.[6]

2.6. Der RANSAC-Algorithmus

Der „RANdom SAmple Consensus“-Algorithmus (RANSAC-Algorithmus) ist eine Methode, um mathematische Modelle in Datenmengen zu finden, die viele Ausreißer enthalten. RANSAC ist ein iterativer, nicht deterministischer Algorithmus und kann deshalb nicht immer garantieren, dass ein gutes Resultat gefunden wird. Deshalb ist er aber sehr robust gegenüber verrauschten Daten.

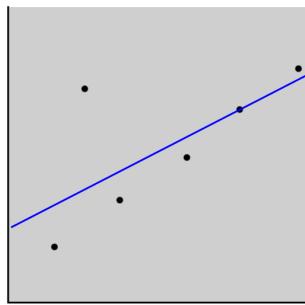


Abbildung 7:
Ausgleichungsrechnung

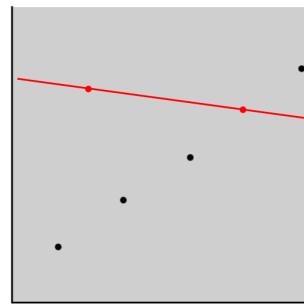


Abbildung 8: RANSAC mit
ungünstigen Kandidaten

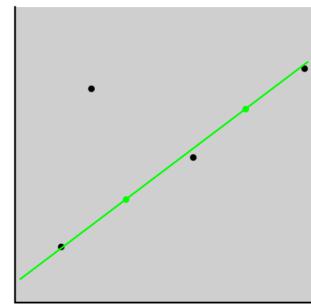


Abbildung 9: RANSAC mit
günstigen Kandidaten

Der traditionelle Ansatz, um aus einer Menge an Daten ein mathematisches Modell abzuleiten, ist die Ausgleichungsrechnung. Dieser sieht vor, für eine gegebene Funktion, die Parameter so anzupassen, dass die Abweichung der Funktion von den Messdaten minimal ist. Da Ausreißer aber nicht ohne Weiteres erkannt werden können, werden diese bei der Schätzung der Parameter genauso berücksichtigt wie die richtigen Messungen. Obwohl man somit immer eine Lösung bekommt, ist diese nicht die Genaueste.

Der RANSAC-Algorithmus hingegen wählt sich aus den Messwerten zufällig eine Teilmenge aus, und ermittelt für diese die günstigsten Parameter. Dann werden die verbleibenden Punkte untersucht und Punkte, die signifikant von dieser Funktion abweichen, als Ausreißer behandelt. Falls nicht übermäßig viele Punkte eliminiert

worden sind, wird nun über die verbleibenden Punkte anhand der herkömmlichen Ausgleichungsrechnung ein Modell erzeugt und anhand von diesem der Fehler bestimmt.

Diese Schritte werden mehrfach wiederholt, und das beste Modell, sprich das Modell mit dem geringsten Fehler ohne Berücksichtigung der Ausreißer, wird als Ergebnis verwendet. [7]

2.7. Gauß'sches Weichzeichnen

Beim Gauß'schen Weichzeichnen handelt es sich um ein Verfahren, bei dem die Intensität jedes Pixels mit der seiner Nachbarn verrechnet wird. Dadurch entsteht der Effekt eines Verwischens und das Bild wird unschärfer. Dazu wird die Gauß'sche Funktion benutzt, welche den Pixeln, die sich in einer größeren Entfernung zu dem zentralen Pixel befinden, eine geringere Wertigkeit zuordnet.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}(\frac{x^2+y^2}{\sigma^2})}$$

Theoretisch müsste für jedes Pixel das gesamte Bild betrachtet werden. In der Praxis hat sich aber ein Fenster von 6σ als ausreichend herausgestellt. Wie Abbildung 10 zeigt, werden beim Gauß'schen Weichzeichnen mit $\sigma = 1,6$ die Gewichte außerhalb des 9x9-Fensters vernachlässigbar gering.

Abbildung 10: Auszug aus einer Gauß'schen Weichzeichnen Matrix mit $\sigma = 1,6$

2.8. Interest-Operatoren

Unter Interest-Operatoren versteht man Algorithmen, die markante Stellen in Bildern (englisch: „Points of interest“) suchen. Diese sind meist der erste Ansatzpunkt, um Bilder digital verarbeiten zu können. Die hier vorgestellten Algorithmen arbeiten alle auf Grauwertbildern.

Corner Detection

„Corner Detection“-Algorithmen sind bereits seit einigen Jahrzehnten im Einsatz. Obwohl ihr Name dies vermuten ließe, ist ihr eigentliches Ziel nicht die Suche nach Ecken, sondern markanten Regionen, die zwischen mehreren Aufnahmen gut wieder zu erkennen sind.

Einer der bekanntesten Algorithmen ist der 1977 vorgestellte Moravec-Operator. Dieser definiert eine markante Position, als einen Punkt, an dem in jede Richtung die Intensität stark variiert. Da dies vor allem an Ecken der Fall ist, hat es den Namen dieser Algorithmen geprägt. Ermittelt werden diese Punkte, indem ein Fenster mit einer Größe von drei, fünf oder sieben Pixeln jeweils in die acht prinzipielle Richtungen um ein Pixel verschoben und jeweils die Varianz berechnet wird:

$$V(x,y) = \sum_{\forall a,b \text{ im Fenster}} (I(x+u+a, y+v+b) - I(x+a, y+b))^2$$

mit $(u,v) \in \{(-1,-1), (0,-1), (1,-1), (-1,0), (1,0), (-1,1), (0,1), (1,1)\}$

Punkte deren kleinste Varianz über einem Schwellenwert liegen, werden als Ecken erkannt. Obwohl dieser Algorithmus sehr schnell ist, ist er auch sehr anfällig gegenüber Rauschen oder Rotation, weshalb er heute kaum noch verwendet wird. [8]

Verbessert wurde der Moravec-Operator bereits im Jahre 1988 von Harris und Stephens zum Plessy-Operator. Diese gingen dazu über Ableitungen in mehr als den acht prinzipiellen Richtungen zu betrachten. Außerdem werden, im Gegensatz zum Moravec-Operator, alle Varianzen berücksichtigt, und die interessanten Regionen in Ecken und Kanten aufgeteilt.

Das bewerten der Punkte, mit den Gewichten der auf dem jeweiligen betrachtetem Punkt zentrierter Gauß'schen Matrix, reduziert die Rauschanfälligkeit. Der Plessy-Operator ist rechenaufwändiger als der Moravec-Operator, aber deutlich robuster, weshalb er immer noch in der Bildverarbeitung eingesetzt wird. Da die Gradienten aber

anhand der prinzipiellen Richtungen approximiert werden ist er noch anfällig bezüglich der Rotation. [9]

Der SIFT-Algorithmus

Im Jahre 2004 entwickelte David G. Lowe der „University of British Columbia“ den SIFT-Algorithmus, der Schlüsselpunkte in Bildern identifizieren und beschreiben kann. Diese werden in Merkmale umgewandelt, die invariant gegenüber Skalierung und Rotation, und relativ robust gegenüber Veränderungen in der Perspektive, Belichtung oder affinen Transformation (Kameraverzerrung) sind, weshalb Merkmale unter wechselnden Betrachtungswinkeln und Entfernung wiedererkannt werden können. Der Name SIFT (Scaling Invariant Feature Transform) röhrt von dem Umwandeln der Bildinformation in diese Merkmale her. [10]

Diese Merkmale können benutzt werden, um Korrespondenzpunkte in zwei Bildern zu finden. Diese Technologie wird bereits zur Objekterkennung und zum Aneinanderfügen von einzelnen Aufnahmen zu Panoramabildern eingesetzt.

Der Algorithmus besteht aus vier Phasen:

- Das Entdecken von Extremwerten in den verschiedenen Auflösungsstufen, die als Kandidaten für gut wieder zu erkennende Merkmale in Frage kommen.
- Die subpixelgenaue Lokalisierung der gefundenen Punkte.
- Dem Punkt eine oder mehrere Orientierungen zuordnen.
- Das Umwandeln der Bildinformationen im Umfeld des Punktes zu Merkmalen.

Extremwerteerkennung

Ein naheliegender Ansatz, interessante Regionen in einem Bild zu identifizieren, ist die Suche nach Punkten, die ein lokales Maximum oder Minimum darstellen. Andrew P. Witkin der „Fairchild Laboratory for Artificial Intelligence Research“ hat 1983 in einem eindimensionalen Kontext gezeigt, dass um Merkmale in unterschiedlichen Maßstäben wieder zu erkennen, diese über verschiedene Skaliersstufen gesucht werden müssen [11]. Wenn in diesem Zusammenhang von einem Skalierungsraum (englisch: Scale-Space) gesprochen wird, ist damit die Menge der Abbildungen eines Ausgangsbildes in verschiedenen Detailgraden gemeint.

Auf den Roboter bezogen kann dies von einem sehr kleinen Merkmal, wie zum Beispiel einem Fleck auf dem Boden, bis hin zu einem sehr großen Merkmal, wie beispielsweise ein Berg in nächster Nähe reichen. Hierbei muss man sich vor Augen halten, dass zu diesem Zeitpunkt noch nicht unterschieden werden kann, ob etwas wegen seiner physischen Größe, oder wegen seiner Entfernung als „kleines“ Merkmal abgebildet wird.

Markante Merkmale, die in einer Skalierungsstufe auftauchen, verschwinden meistens mit dem Wechsel in eine andere Stufe. Deshalb ist es wichtig eine Vorgehensweise zu benutzen, die Merkmale übergangslos zwischen Skalierungsstufen finden kann. Obwohl nur das Weichzeichnen eine stufenlose Reduktion des Detailgrades erlaubt, ermöglicht die Kombination mit einer Reduktion der Auflösung deutliche Ersparnisse in der Rechenzeit.

Dazu teilt der SIFT-Algorithmus den Skalierungsraum in mehrere Oktaven auf, wobei jede Oktave die Bildauflösung jeweils auf die Hälfte reduziert. Innerhalb dieser Oktaven werden die Skalierungsstufen mit Hilfe des Gauß'schen Weichzeichnens mit Mehrfachen von σ berechnet. Um effizient stabile Merkmale zu entdecken, hat sich die Differenz zwischen zwei aufeinander folgenden Skalierungsstufen als geeignet herausgestellt.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

mit $I(x, y)$ entspricht dem analysiertem Bild.

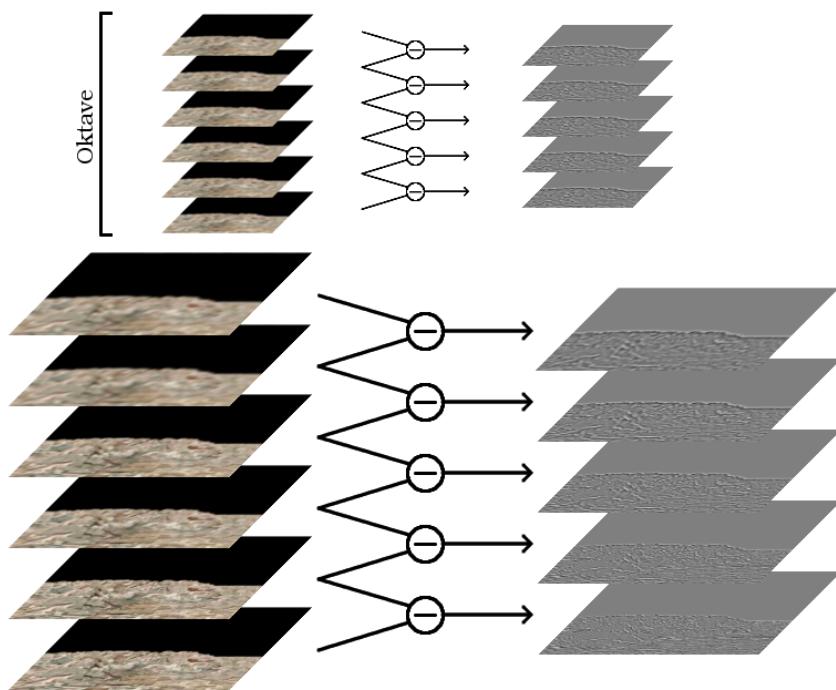


Abbildung 11: Differenz des Gauß'schen Weichzeichnens

Aus $k+1$ weichgezeichneten Bildern lassen sich k Differenzen errechnen. Da der Extremwert nicht nur in der Position des Bildes, sondern auch im Skalierungsraum gesucht wird, können keine Extremwerte in der ersten und der letzten Skalierung einer Oktave gefunden werden. Somit können nur in $k-2$ Skalierungsstufen gesucht werden. Ein Extremwert kommt als Kandidat in Frage, wenn er größer oder kleiner als alle acht benachbarten Pixel der gleichen Skalierungsstufe und neun Pixel der beiden angrenzenden Skalierungsstufen ist.

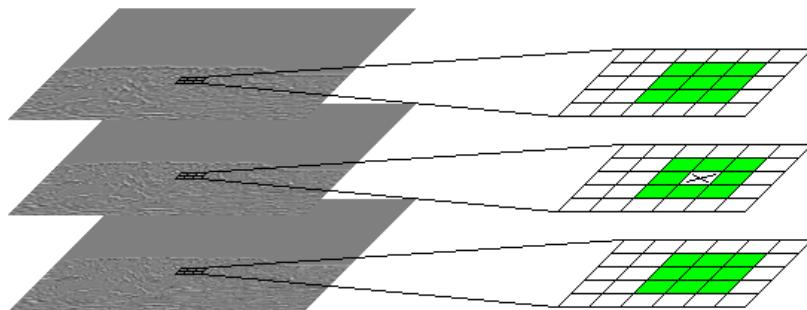


Abbildung 12: Betrachtete Nachbarn der Extremwertsuche

Experimentell hat sich die Suche über drei Skalierungsstufen pro Oktave und σ gleich 1,6 als am erfolgreichsten herausgestellt. Obwohl mehr Skalierungsstufen auch mehr Extremwerte liefern würden, würde die Qualität der Referenzpunkte abnehmen. Die Ergebnisse der vorliegenden Arbeit wurden mit diesen Parametern errechnet.

Subpixelgenaue Lokalisierung

Häufig reicht die pixelgenaue Lokalisierung der Merkmale nicht aus. Um eine genauere Lokalisierung der Punkte innerhalb des Bildes, aber auch innerhalb der Skala zu erhalten als dies anhand der Pixelkoordinaten möglich wäre, kann man über die Nachbarwerte interpolieren und den Scheitelpunkt bestimmen. Dies kann mit Hilfe einer Taylorreihe berechnet werden:

$$D(u) = D + \frac{\partial D^T}{\partial u} u + \frac{1}{2} u^T \frac{\partial^2 D}{\partial u^2} u \text{ mit } u = (x, y, \sigma)^T$$

Die erste Ableitung nach u ergibt somit die genauere Position des Scheitelwertes:

$$\hat{u} = \frac{-\partial^2 D^{-1}}{\partial u^2} \frac{\partial D}{\partial u}$$

Zur Reduzierung des Rechenaufwands werden die Ableitungen von D anhand der Differenz der Nachbarpixel approximiert. Falls \hat{u} mehr als 0,5 von dem ursprünglichen Punkt abweicht, bedeutet dies, dass der Nachbarpunkt näher an dem Extremwert liegt, und der Algorithmus wird für diesen wiederholt.

Zusätzlich zur subpixelgenauen Lokalisierung des Extremwertes wird die Ableitung von D an der Position \hat{x} genutzt, um den Kontrast des Merkmals zu bestimmen. Punkte, deren Kontrast unter einem Schwellenwert (in dieser Arbeit wurde der Wert von 0,03 von Lowe übernommen) liegen, werden als zu schwache Merkmale verworfen.

Kantenprüfung

Abgesehen von Punkten mit zu schwachem Kontrast, müssen auch Punkte, die entlang einer Kante liegen, ignoriert werden, da diese nur in eine Richtung genau lokalisierbar sind. Diese können erkannt werden daran, dass der Krümmungsgrad entlang einer Achse niedrig ist. Mit Hilfe der Hesse-Matrix kann dies nachgewiesen werden, da ihre Eigenwerte proportional zum Krümmungsgrad der Hauptachsen sind. Die Ableitungen werden auch hier anhand der Differenzen der Nachbarpunkte angenähert.

$$H = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix}$$

Wenn man nun r als maximales Verhältnis definiert, das zwischen dem kleinsten und dem größten Krümmungsgrad bestehen darf, muss die Hesse-Matrix folgende Bedingung erfüllen:

$$\frac{\text{Spur}(H)^2}{\text{Determinante}(H)} < \frac{(r+1)^2}{r}$$

Andernfalls wäre der Punkt zu leicht mit einem andern Punkt entlang derselben Kante zu verwechseln.

Orientierung

Im Gegensatz zu anderen Algorithmen verzichtet der SIFT-Algorithmus auf eine rotations-unabhängige Beschreibung der Merkmale, weil dadurch wichtige Informationen verloren gehen. Um dennoch rotierte Merkmale wiedererkennen zu können, wird jedem Merkmal eine Orientierung zugeordnet.

Dazu werden für jeden Extremwert die Gradienten von jeweils 36 verschiedenen Winkeln berechnet. Im Regelfall wird anschließend der ausgeprägteste Gradient als Orientierung gewählt, wobei über die benachbarten Gradienten wiederum interpoliert wird, um ein genauereres Resultat zu erzielen.

Falls mehrere Gradienten eine ähnlich große Differenz aufweisen, könnte dem gleichen Merkmal unter leicht anderen Bedingungen (wie zum Beispiel Beleuchtung, Betrachtungswinkel oder Rauschen) eine andere Orientierung zugeordnet werden. Da dies unter Umständen dazu führen könnte, dass das Merkmal nicht wiedererkannt wird, werden den Merkmalen für alle Gradienten, die eine fast so große Differenz wie der dominierende Gradient aufweisen, eine zusätzliche Orientierung zugeordnet.

Transformation in Merkmale

Nun, da die Lokalisierung des Merkmals, die Skalierung und die Orientierung bestimmt worden sind, müssen lediglich die Bildinformationen noch auf eine Art und Weise erfasst werden, die gegenüber einem verändertem Betrachtungspunkt, Belichtung sowie Rauschen robust ist.

Dazu nutzt der SIFT-Algorithmus Beobachtungen, die bezüglich des biologischen Sehens gemacht wurden. Die Neuronen im visuellen Cortex (der Sehrinde) reagieren auf periodisch angeordnete Gradienten, wobei die genaue Lokalisierung der Gradienten weniger ausschlaggebend ist. Dieses erleichtert dem Menschen, Gegenstände im dreidimensionalem Raum wieder zu erkennen. [12]

Der SIFT-Algorithmus berechnet den Deskriptor für die interessante Region aus den Gradienten in der Skalierungsstufe, in welcher der jeweilige Extremwert gefunden worden ist. Um Rechenzeit zu sparen werden die Gradienten für jede Skalierungsstufe im Voraus berechnet.

Für den Deskriptor werden um den Schlüsselpunkt die Gradienten in einer 16x16-Matrix betrachtet. Da die Gradienten in unmittelbarer Nähe wichtiger sind als die am Rand, werden diese mit den Gewichten der Gauß'schen Weichzeichner Matrix, auf den Extremwert zentriert, bewertet.

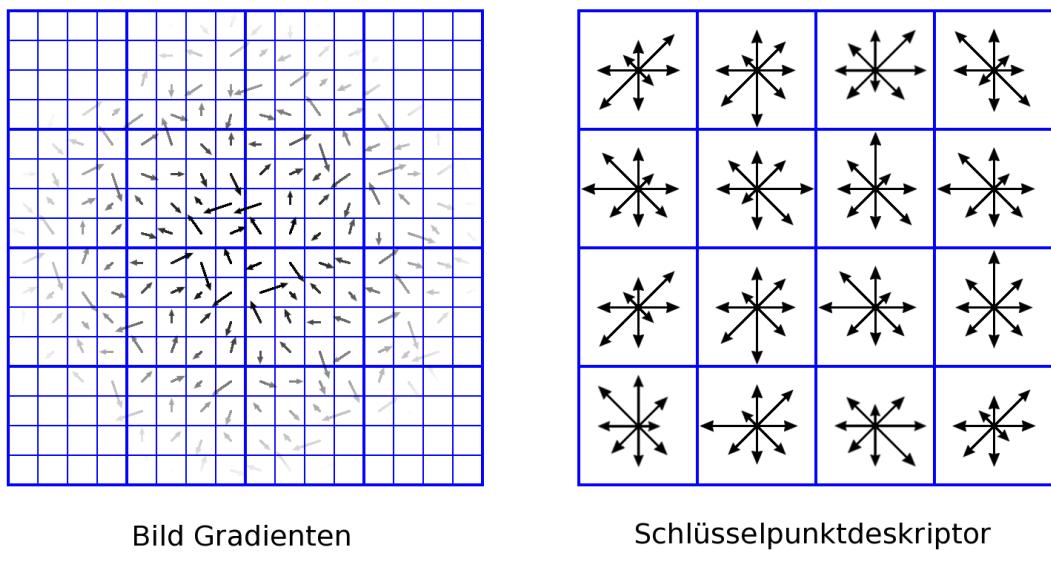


Abbildung 13: Berechnung eines Schlüsselpunktdeskriptors

Diese Gradienten werden dann entsprechend der ermittelten Orientierungen rotiert, so dass für jede Orientierung ein Deskriptor entsteht. Anschließend werden die Gradienten innerhalb einer 4x4-Region jeweils zu einem Vektordiagramm zusammengefasst. Die Länge der Vektoren entsprechen hierbei der Summe der Gradienten, die ungefähr in die jeweilige Richtung zeigen. Um Sprünge zu vermeiden, die entstehen könnten, wenn ein Gradient auf der Grenze zwischen zwei Kästen liegt, werden die Gradienten innerhalb

eines Kastens umgekehrt proportional zu ihrem Abstand zum Zentrum des Kastens gewichtet.

Die Vektoren des Deskriptors werden dann normalisiert, um den Einfluss von Veränderungen der Beleuchtung zu verringern. Der so entstandene Deskriptor hat sich als der günstigste Kompromiss, bezüglich Laufzeit, Anzahl an wiedererkannten Merkmalen, sowie der Verlässlichkeit, dass ähnliche Deskriptoren das gleiche Merkmal beschreiben, herausgestellt.

3. Lösungskonzept

In diesem Kapitel wird die prinzipielle Vorgehensweise beschrieben, um von den Stereobildaufnahmen zu einer Höhenkarte zu gelangen.

In der Praxis wird der Roboter die Bilder anhand seiner beiden Graustufenkameras aufnehmen, und dann entweder auf dem PC-104 Board PC verarbeiten, oder an einen zweiten Rechner zur Auswertung übertragen. Wie dies am besten funktionieren könnte, wird in dieser Arbeit nicht behandelt. Stattdessen wird davon ausgegangen, dass die Bilder bereits in digitaler Form vorliegen.

In einem ersten Schritt werden unabhängig in beiden Bildern markante Merkmale mit Hilfe des SIFT-Algorithmus gesucht. Diese Merkmale werden dann miteinander verglichen und im Falle einer Übereinstimmung werden diese Punkte zu einem Korrespondenzpaar zusammengefasst.

3.1. Dreidimensionale Lokalisierung

Nachdem diese Korrespondenzpaare gefunden wurden, kann man geometrisch das Merkmal dreidimensional lokalisieren. Dies ist möglich da die Kameraparameter bekannt sind. Somit kann jedem Pixel des Bildes ein Punkt im dreidimensionalem Raum zugeordnet werden.

$$P_i = \text{KameraMatrix}_i \times \begin{pmatrix} \text{Pixel}_x \\ \text{Pixel}_y \\ \text{Brennweite} \\ 1 \end{pmatrix}$$

Mit Hilfe dieses Punktes und des Kamerazentrums kann eine Linie gezogen werden, entlang welcher sich das auf dem Bild abgebildete Objekt befinden muss.

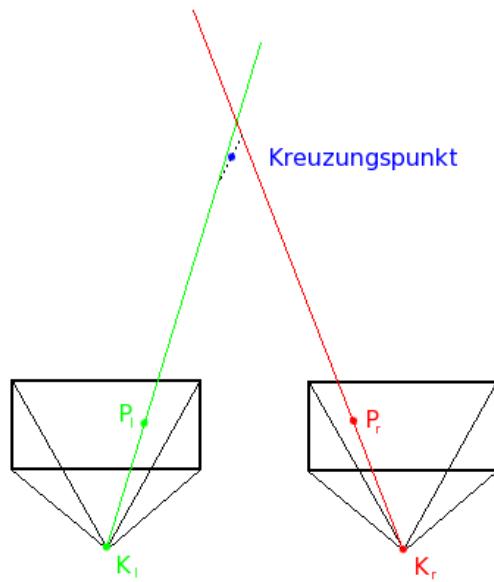


Abbildung 14: Annäherung des Kreuzungspunktes

Obwohl im Modell sich die Linien der beiden Kameras theoretisch treffen würden, und der Kreuzungspunkt durch das Lösen des Gleichungssystems gefunden werden könnte, so führen Rauschen und Rundungsfehler dazu, dass sich in der Realität diese Linien nur selten treffen. Als Alternative zu dem realen Kreuzungspunkt der Linien wird der Punkt als Approximation gewählt, der zu beiden Linien einen minimalen Abstand hat.

3.2. Terrainrekonstruktion

Die dreidimensionale Punktwolke, die aus den korrespondierenden Merkmalen gewonnen wurde, gibt einen ersten Eindruck von dem Umfeld, ist jedoch zur Suche von Aufsetzpunkten für den Laufroboter noch ungeeignet. Deshalb muss diese Darstellung in eine für den Laufroboter geeignete Form umgewandelt werden.

Unter der Voraussetzung, dass die Welt in einer Höhenkarte vorliegt, also alle Objekte Hindernisse darstellen unter denen sich der Roboter nicht hindurch bewegen kann, kann man davon ausgehen das alle gefundenen Punkte, Punkte auf dem Boden repräsentieren.

Um eine Höhenkarte zu erstellen, benötigt man für jeden beliebigen Punkt eine Höheninformation. Dadurch das alle Referenzpunkte Bodenpunkte repräsentieren, kann man dazu für jeden beliebigen Punkt die Referenzpunkte suchen die sich am Nächsten befinden und anhand dieser interpolieren. Die einfachste Art der Interpolation ist hierbei die lineare Interpolation, die eine dreidimensionale Ebene anhand von drei

Referenzpunkten aufspannt, und den Funktionswert an der gesuchten Position als interpolierte Höhe übernimmt.

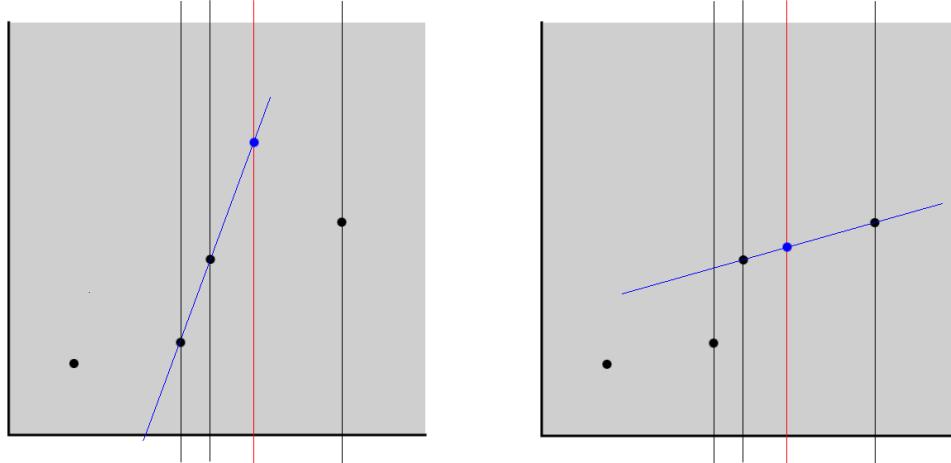


Abbildung 15: Interpolation anhand von nächsten Punkten *Abbildung 16: Interpolation anhand von umschließenden Punkten*

Wie Abbildung 15 und 16 zeigen, werden bei der Interpolation zuverlässigere Resultate erzielt wenn sie anhand von Referenzpunkten, die den gesuchten Punkt umschließen, durchgeführt wird. Deshalb sollte für jeden der Punkt dessen Höhe interpoliert wird, die verwendeten Referenzpunkte ein Dreieck bilden in welchem sich dieser Punkt befindet.

Da eine Höhenkarte aber keine Lücken aufweisen kann, müssen alle Punkte einen Wert haben. Für die Punkte, welche außerhalb der konvexen Hülle der Referenzpunkte liegen, funktioniert die vorgestellte Interpolation nicht, weshalb für diese keine zuverlässige Höhenapproximation möglich ist. Unabhängig davon welchen Wert man den Punkten zuordnen würde, für die keine Information vorhanden sind, könnte nicht zwischen dem Fall das keine Informationen, und dem Fall das eine Höhe die dem gewählten Wert entspricht vorliegt, unterschieden werden.

Obwohl es möglich wäre einen beliebigen Wert der Höhenkarte zu reservieren, um Punkte darzustellen für die keine Höheninformation vorliegen, würde dies die bisherige Implementierung der Höhenkarte in der Simulationsumgebung verletzen. Außerdem wäre diese binäre Lösung, recht unflexibel gegenüber von Punkten die nur eine geringe Güte haben.

Alternativ dazu kann man parallel zu der Höhenkarte eine Qualitätskarte anlegen, welche für jeden Punkt der Höhenkarte dessen Zuverlässigkeit angibt. Diese kann in

Abhängigkeit von der Entfernung zu den Referenzpunkten bestimmt werden, so dass Punkten die eng zwischen Referenzen liegen eine höhere Qualität zugeordnet werden, als Punkten die zwischen weit entfernten Referenzpunkten liegen. Außerdem kann eine solche Karte den Roboter bei der Entscheidung unterstützen, welche Informationen ihm noch fehlen.

Anhand dieser Qualitätskarte und der Höhenkarte, wäre der Roboter in der Lage die nächsten Schritte zu planen.

4. Implementierung

Als Programmiersprache für die Testimplementierung wurde C# gewählt, weil sie zur schnellen Entwicklung geeignet ist und eine große Sammlung von Bibliotheken zur Verfügung steht. Obwohl die Portierbarkeit nicht die höchste Priorität hat, ermöglicht Mono als plattformübergreifende C# Umgebung eine höhere Wiederverwertbarkeit.

Die Benutzeroberfläche wurde mit dem GIMP² ToolKit (GTK) erstellt, da dieses sehr gut in Monodevelop eingebunden ist und eine Vielzahl an Bildoperationen bietet.

Das gesamte Programm hat in etwa 5000 Zeilen Code.

4.1. Merkmalsuche

Der erste Schritt, um Tiefeninformationen aus einem Stereobildpaar gewinnen zu können, ist die Suche nach Korrespondenzpaaren zwischen den Aufnahmen.

Im Rahmen des Programms, das dieser Diplomarbeit zugrunde liegt, wurde die Implementierung des SIFT-Algorithmus in C# von Sebastian Nowozin [13] benutzt, um gut wiedererkennbare Regionen in den Bildern zu identifizieren. Jedes Bild wird in fünf Oktaven geteilt, wobei in jeder Oktave sechs Skalierungen anhand des Gauß'schen Weichzeichnens berechnet werden. Von den fünf Differenzen dieser Skalierungen, können dann drei Differenzen nach Extremwerten durchsucht werden. Diese Differenzen wurden in den Abbildungen 17 bis 21 farblich voneinander getrennt.

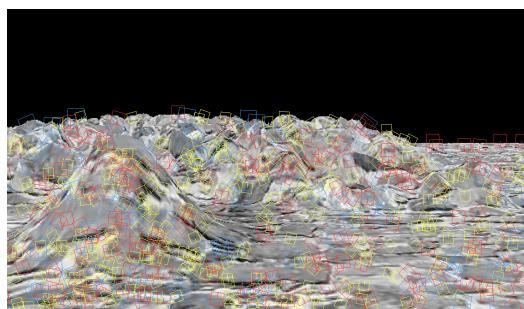


Abbildung 17: SIFT Oktave 0

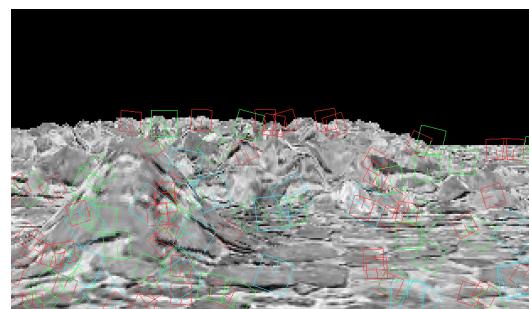


Abbildung 18: SIFT Oktave 1

² GNU Image Manipulation Program

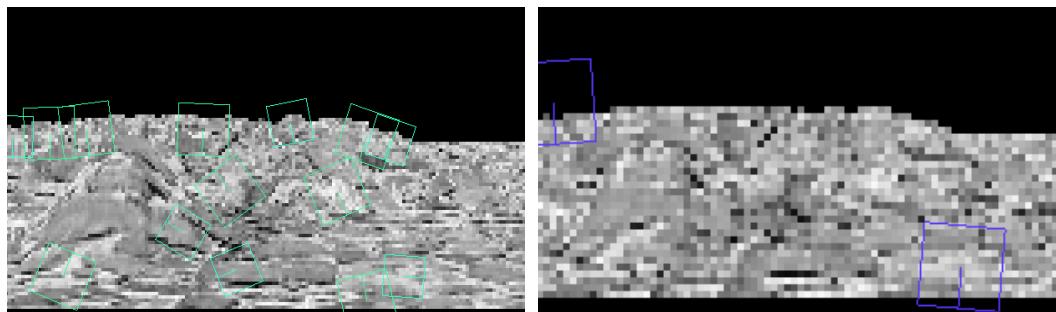


Abbildung 19: SIFT Oktave 2

Abbildung 20: SIFT Oktave 3

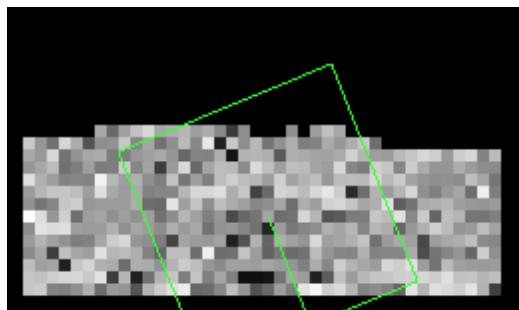


Abbildung 21: SIFT Oktave 4

Obwohl der SIFT-Algorithmus ebenfalls auf der Graphics Processing Unit (GPU) ausgeführt werden kann, existiert hierfür bisher nur eine Implementierung in C++, die nicht ohne Weiteres in ein C#-Programm eingebunden werden kann. Die möglichen Ersparnisse werden in Kapitel 5.1 behandelt.

4.2. Korrespondenzanalyse

Nachdem die Merkmale in beiden Bildern gefunden wurden, muss nun jedes Merkmal aus dem linken Bild mit jedem Merkmal aus dem rechten Bild verglichen werden. Die Laufzeit beträgt deshalb:

$$O(n^2)$$

mit n = Anzahl der Merkmale pro Bild³.

Da die Kameras parallel zueinander montiert sind, kann man davon ausgehen, dass die Korrespondenzpaare nur leicht vertikal voneinander abweichen. Deshalb ist es möglich, die Bildpunkte in vertikale Blöcke aufzuteilen, und diese paarweise miteinander zu vergleichen.

³ Unter der Annahme, dass beide Bilder in etwa die gleiche Anzahl an Merkmalen besitzen

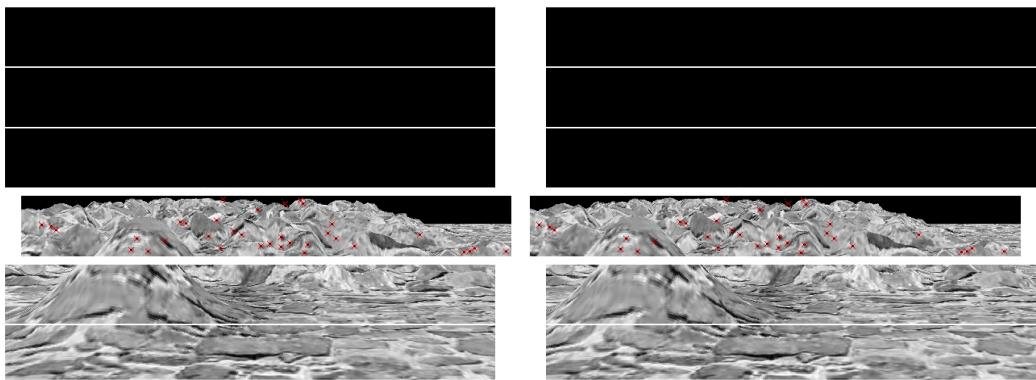


Abbildung 22: Blockweises Vergleichen mit 6 Blöcken

Dadurch kann die Anzahl der Vergleiche auf

$$b \cdot \left(\frac{n}{b}\right)^2 \quad \text{mit } b = \text{Anzahl der Blöcke}$$

reduziert werden. Der optimale Wert für b muss experimentell ermittelt werden, da es von der Strategie der Blockgrößenwahl, der Verteilung der Merkmale sowie der Anzahl von Punkten, die verloren gehen, weil sie in unterschiedliche Blockpaare aufgeteilt sind, bzw. der Verschlechterung der Höhenkarte, abhängt.

4.3. Dreidimensionale Lokalisierung

Für jedes Korrespondenzpaar wird nun die Position des abgebildeten Raumpunktes ermittelt. Daraus entsteht eine Wolke von Punkten (in Abbildung 23 dargestellt als Voxel⁴), die den Objekten der abgebildeten Welt entsprechen.

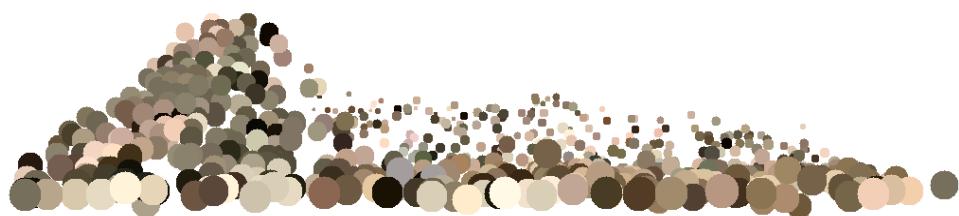


Abbildung 23: Voxeldarstellung der gefundenen Punkte

⁴ Volumen Pixel

4.4. Erstellen einer Höhenkarte

Die dreidimensionale Punktwolke, die in den bisherigen Schritten erstellt wurde, kann von der Simulationsumgebung noch nicht genutzt werden. Die Simulationsumgebung hat bisher das Terrain als eine Höhenkarte dargestellt, welche aus den Helligkeitswerten des grünen Farbkanals eines beliebigen Bildes geladen werden kann.

Bevor nun eine Höhenkarte erstellt werden kann, ist es notwendig, dass für jede zweidimensionale Koordinate eine Höheninformation vorliegt. Wie im Kapitel 3 erklärt wurde, müssen hierzu für jeden Punkt Referenzpunkte gefunden werden, die ein günstiges Dreieck um diesen Punkt herum bilden, und anhand derer die Höhe interpoliert werden kann.

Da es sehr aufwändig wäre, für jeden Punkt ein dazu gehörendes Dreieck zu bilden, wurde in dieser Arbeit ein Top-Down-Ansatz gewählt, bei dem die Dreiecke im Voraus gebildet werden. Dazu ist es aber erforderlich die Daten der Punktwolke etwas anders zu interpretieren. Indem man die Punkte der Wolke senkrecht auf die Ebene, die durch die X-Achse und Z-Achse definiert wird, abbildet, erhält man eine zweidimensionale Punktmenge, die in etwa der Draufsicht der Punktwolke entspricht.



Abbildung 24: Draufsicht auf die Voxelwolke

Diese Ebene kann nun anhand von Delaunay Dreieckszerlegung aufgeteilt werden. Dazu wurde der „Divide and Conquer“-Algorithmus angewendet.

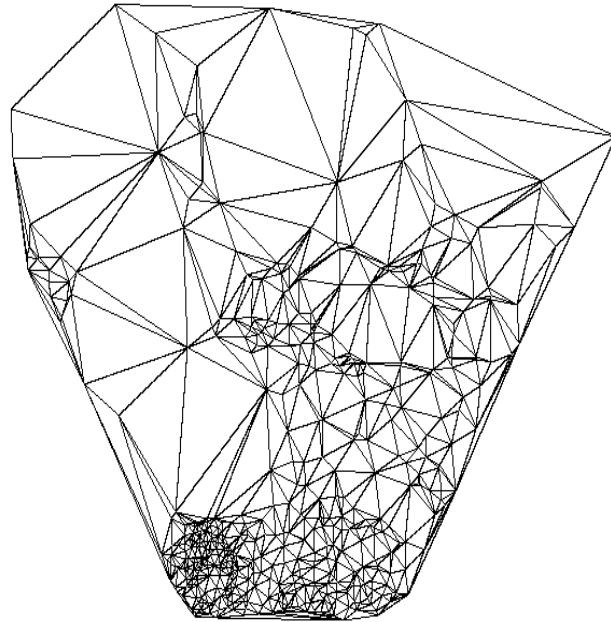


Abbildung 25: Delaunay-Dreieckszerlegung

Innerhalb dieser Dreiecksmenge ist es nun möglich, über jedes Dreieck eine Ebene zu spannen, anhand welcher die Höhe der enthaltenen Punkte interpoliert wird. Diese Ebene wird definiert mit Hilfe der Funktion:

$$f(x, y) = a \cdot x + b \cdot y + c$$

Die jeweiligen Parameter werden hierbei durch Einsetzen der Referenzpunkte mit folgendem linearen Gleichungssystem ermittelt:

$$\begin{aligned} f(P1_x, P1_y) &= P1_z \\ f(P2_x, P2_y) &= P2_z \\ f(P3_x, P3_y) &= P3_z \end{aligned}$$

Durch Eintragen eines jeden Dreiecks in die Höhenkarte, erhält man eine vollständige Höhenkarte.

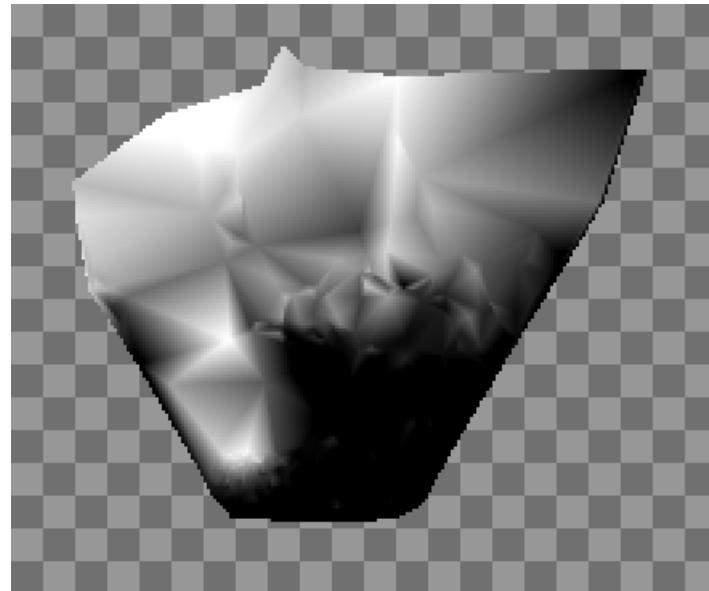


Abbildung 26: Beispiel einer Höhenkarte

Zur Veranschaulichung wurden in Abbildung 26 die Bereiche, für die keine Höheninformationen vorlagen mit einem Schachbrettmuster ausgemalt. Anhand der Qualitätskarte kann man gut erkennen wo sich die Referenzpunkte der Höhenkarte befanden.

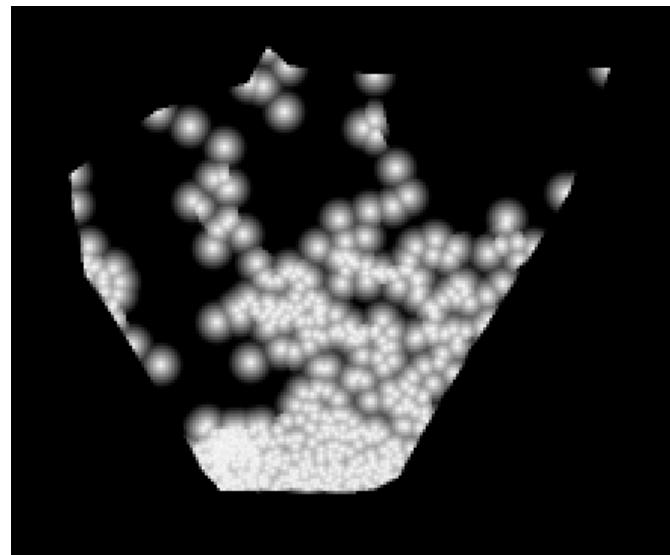


Abbildung 27: Beispiel einer Qualitätskarte

4.5. Filter

Leider sind Ausreißer, sowie kleine Ungenauigkeiten unvermeidbar, und müssen manchmal zu Gunsten der Geschwindigkeit in Kauf genommen werden. Extreme Ausreißer können aber das Resultat so stark verfälschen, dass die Ergebnisse nicht mehr nutzbar sind. Deshalb muss ein gutes Gleichgewicht zwischen Rechenzeit und Qualität der Filter gefunden werden. Die Qualität eines Filters kann daran gemessen werden, wie viel Prozent der unerwünschten Punkte entfernt wurden und wie viele der „guten“ Punkte erhalten geblieben sind.

Probleme des RANSAC Filters

Der RANSAC Filter ist ein nicht deterministischer Filter, der standardmäßig von den SIFT-Bibliotheken genutzt wird. Er versucht anhand von zufällig ausgewählten Kandidaten ein Modell zu finden, das zu der Disparität der meisten Punktpaare passt. Alle Punkte, die diesem Modell nicht entsprechen, werden als Ausreißer ausgeschlossen.

Durch das Aufteilen in mehrere Blöcke erhält man immer kleinere Gruppen von Korrespondenzpunkten. Dies kann dazu führen, dass ein Modell ausgewählt wird, das zwar für einige Punkte optimal ist, aber eine Vielzahl anderer Punkte ausschließt, weil diese einem anderen Modell entsprechen. Abbildung 28 zeigt, wie z.B. bei einer Optimierung mit drei Blöcken 180 Punkte ausgeschlossen werden, da sie nicht mit der gleichen Funktion erfasst werden können wie der Hügel im Vordergrund.

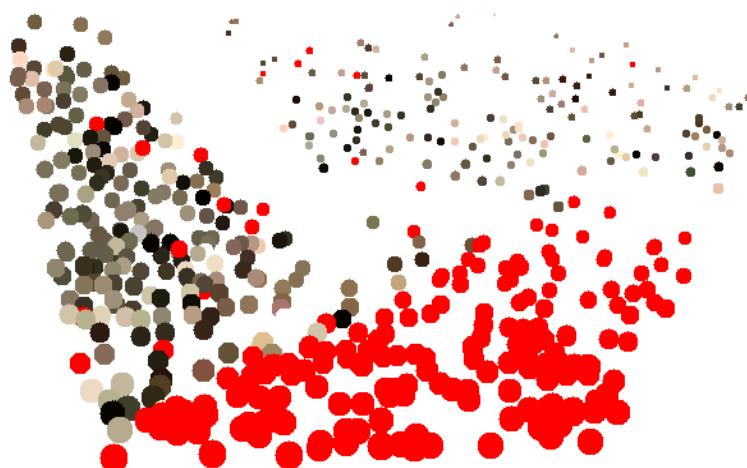


Abbildung 28: Durch RANSAC entfernte Punkte

Alternative Filter

Da der RANSAC Filter ungeeignet ist, aber einzelne Ausreißer das Resultat zu sehr verzerren können, müssen diese durch alternative Filter erfasst werden. Es gibt zwei Ebenen, auf denen dies möglich ist:

- Im **Voxelmodell** kann man Punkte eliminieren, die zu weit von der Kamera entfernt sind, da die Genauigkeit der Berechnung mit der Distanz abnimmt. Außerdem können Punkte entfernt werden, die „alleine“ stehen, also deren Distanz zu dem nächsten Referenzpunkt einen Grenzwert übersteigt.
- In der **Triangulation** können die Dreiecke entfernt werden, welche von der Kamera weg zeigen, da diese Dreiecke, vorausgesetzt sie sind nicht sehr flach, von der Kamera nicht erfasst werden konnten, und damit unzuverlässig sind. Neben den nicht sichtbaren Dreiecken, können auch Dreiecke, die eine zu große Fläche haben oder zu lang gezogen sind ausgeschlossen werden, da ansonsten der Abstand zu den Referenzpunkten, die zur Interpolation dienen, zu groß wird um zuverlässige Werte zu erhalten.

4.6. Kombination mehrerer Ansichten

Um die Terrain-Informationen aus vorherigen Schritten weiterhin nutzen zu können, müssen die verschiedenen Ansichten zu einer gemeinsamen Karte zusammengefasst werden. Das Wissen über die Position des Roboters innerhalb der bisher erstellten Karte und die Möglichkeit die Karte, die aus der aktuellen Ansicht entstanden ist, in die bisherige Umgebungskarte einzufügen, gehen dabei Hand in Hand.

Am Besten geeignet, um die Informationen der verschiedenen Ansichten zusammen zu fügen, ist die Punktfolge. Dadurch, dass die neuen Punkte eingefügt werden, bevor der erste Filter angewandt wird, können auch diejenigen Punkte berücksichtigt werden, die bisher wegen ihrer isolierten Position entfernt worden wären, aber nun durch die neuen Punkte gestützt werden. Obwohl das Zusammenführen der Informationen auch anhand der Höhenkarten möglich wäre, würde als Resultat wegen der Ungenauigkeiten, die durch die Interpolation in die Höhenkarten eingeflossen sind, eine ungenauere und kontrastärmere Karte entstehen.

Um die Punktfolge, die aus dem letzten Stereobildpaar gewonnen wurde, mit der Punktfolge, die aus den bisherigen Ansichten vorliegt, zu kombinieren, gibt es mehrere

Möglichkeiten. Falls die relative Position der Aufnahme innerhalb der Karte bekannt ist, kann anhand dieser Information die neue Punktewolke ausgerichtet und dem Modell hinzugefügt werden. Obwohl die Robotersteuerung in der Lage ist, die zurückgelegte Strecke zu berechnen, würden bereits kleine Ungenauigkeiten zu einer sich ständig verschlechternden Karte führen.

Die Alternative, die in dieser Implementierung gewählt wurde, sucht nach Referenzpunkten zwischen den verschiedenen Ansichten. Bisher wurde aus jedem Bild die SIFT-Merkmale extrahiert und die Merkmale der linken Kamera mit denen der rechten Kamera verglichen. In einem zweiten Schritt kann man nun auch die Merkmale, die in einer Ansicht gefunden wurden, mit den Merkmalen der nächsten Ansicht vergleichen. Dabei kann man sich auf die Merkmale beschränken, die in beiden Bildern einer Ansicht gefunden wurden.

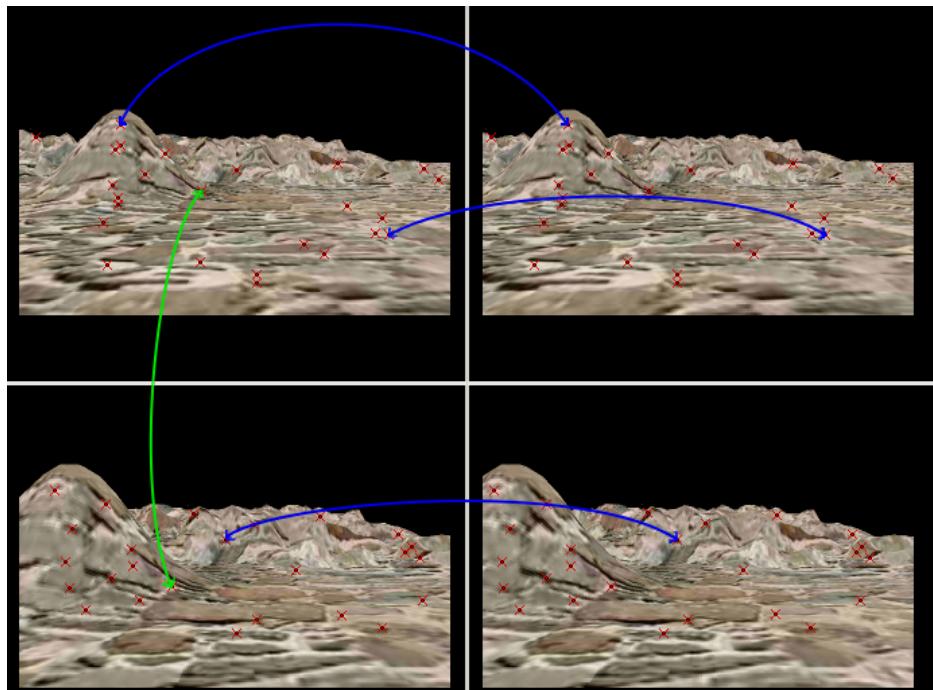


Abbildung 29: Zuordnen von Ansichten

Da diese Merkmale ebenfalls in beiden Bildern einer Ansicht gefunden wurden, und daraus jeweils ein Punkt im dreidimensionalem Raum berechnet wurde, setzen die so gefundenen Übereinstimmungen jeweils einen Punkt der neuen Punktewolke mit einem Punkt des bisherigen Modells in Verbindung. Falls ausreichend Punkte gefunden wurden, kann das neue Modell zuerst gedreht werden, bis alle Vektoren die zwischen den Punkten liegen parallel zueinander sind, und dann verschoben werden, bis sich die

Punkte möglichst genau übereinander befinden. Beide Schritte sind mit RANSAC optimierbar, um Ausreißer zu eliminieren.

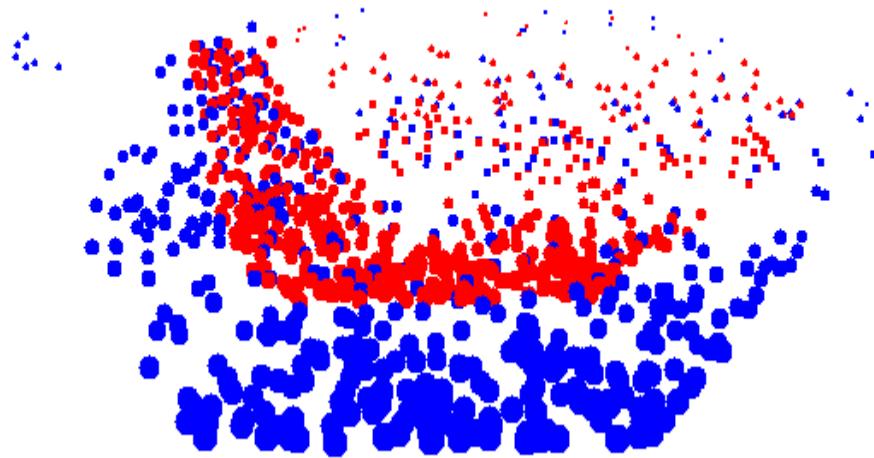


Abbildung 30: Kombination von zwei Voxelwolken

Falls jede neue Punktwolke nur anhand der vorherigen Ansicht ausgerichtet werden würde, würde sich bereits ein sehr kleiner Fehler in einer frühen Ansicht auf alle Folgenden gravierend auswirken.

Deshalb wurde im Rahmen dieser Arbeit jedes Element der Punktwolke mit dem Deskriptor des Merkmals verknüpft, aus dem dieses gewonnen worden ist, ohne das dazu die Bilder der Ansichten gespeichert werden müssen. Die Merkmale der Punkte der neue Punktwolke werden also nicht lediglich mit der vorherigen Ansicht verglichen, sondern mit den Merkmalen der gesamten Karte.

Die Anzahl der hierzu benötigten Vergleiche entspricht:

$$n \cdot m$$

mit n = Anzahl von neuen Punkten, und m = Anzahl der Punkte im bisherigen Modell

Da die Vergleiche rechenintensiv sind, und m umso größer wird, je länger der Roboter unterwegs ist, könnte dieser Schritt zum Flaschenhals der Terraingenerierung werden.

Triangulation des vereinten Modells

Die Punktmenge, die aus der Vereinigung des bisherigen Modells und der Punkte die aus der letzten Ansicht gewonnen wurden, muss erneut trianguliert werden. Dazu kann man den inkrementellen Ansatz nutzen und die bisherige Triangulation um die neu hinzugekommenen Punkte zu erweitern oder über die gesamte Punktmenge eine neue Zerlegung mit dem „Divide and Conquer“-Algorithmus erstellen.

Der inkrementelle Algorithmus hat den Vorteil, dass beim Hinzufügen neuer Punkte, die bisherige Triangulation erweitert werden kann. Gesetzt den Fall, dass in einer Triangulation über n Punkte, m Punkte hinzugefügt werden müssen, so müssen für jeden dieser m Punkte im ungünstigsten Fall in etwa $n+m$ Dreiecke geprüft werden. Die Laufzeit wäre also:

$$O(m \cdot (n+m))$$

Der „Divide and Conquer“-Algorithmus könnte für den gleichen Fall nicht auf die bisherige Triangulation zurückgreifen und müsste eine neue Triangulation für $n+m$ Punkte erstellen. Dadurch entsteht die Laufzeit von

$$O(\log(n+m) \cdot (n+m))$$

Daraus wird ersichtlich, dass sich der inkrementelle Algorithmus nur lohnt, wenn m kleiner als $\log(n+m)$ ist. Da m deutlich kleiner sein muss als n , kann man vereinfacht schreiben:

$$m < \log(n)$$

Da mit jedem Bildpaar etwa 200 Punkte hinzukommen, ist der „Divide and Conquer“-Algorithmus weitaus schneller, weshalb es sich vorerst nicht lohnt eine Logik einzubauen, die zur Laufzeit den geeigneteren Algorithmus auswählt.

Die Vorgehensweise bei der Interpolation des Terrains von mehreren Ansichten ist identisch mit der Vorgehensweise im Fall einer einzigen Ansicht.

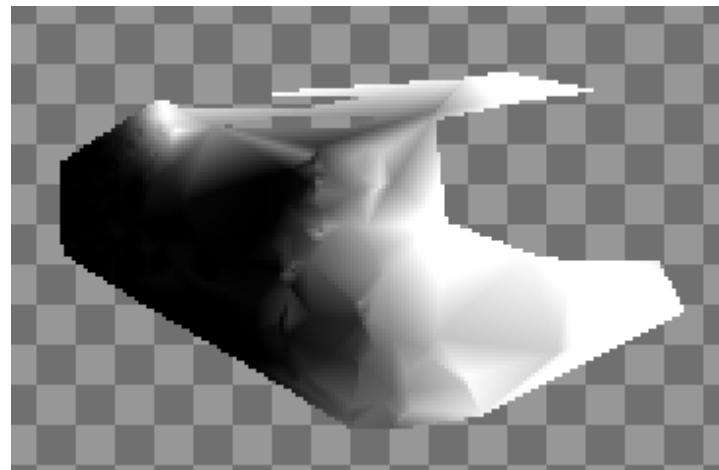


Abbildung 31: Karte über die vereinte Punktmenge

5. Ergebnisse

In Anbetracht der Tatsache, dass die Berechnungen auf dem Roboter ausgeführt werden, ist die Rechenleistung, die zur Bildevaluierung eingesetzt werden kann, beschränkt. Je kleiner die Zeit ist, die für eine Analyse benötigt wird, desto mehr Bilder können pro zurückgelegter Strecke gemacht werden. Da der Roboter sich mit einer Maximalgeschwindigkeit von 0,5 Kilometer pro Stunde (~13,9 Zentimeter pro Sekunde) bewegt, würde zum Beispiel eine Verarbeitungszeit von 2 Sekunden dem Roboter erlauben, alle 30 Zentimeter ein Bild auszuwerten.

Zur Referenz wurde ein Laptop mit einem AMD Athlon64 4000+ Hauptprozessor (CPU), ein Gigabyte Arbeitsspeicher und einer ATI Mobility Radeon X700 Grafikkarte verwendet. Als Betriebssystem kam die Linux-Distribution Gentoo zum Einsatz. Dieser Rechner ist, von der Rechenleistung her betrachtet, nach dem heutigen Stand (2008) im Mittelfeld anzusiedeln und wäre ohne Probleme auf dem Roboter unterzubringen.

Die untersuchten Bilder hatten alle eine Auflösung von 640x480 Pixeln, und sollten aufgrund der in etwa gleichen Fläche wie die 512x512 Pixel Bilder der Roboterkameras, vergleichbare Laufzeiten haben.

Die Laufzeiten der unoptimierten Terraingenerierung liegen bei:

Linkes Bild	4 036 ms
Rechtes Bild	3 999 ms
Vergleichen der Merkmale	8 924 ms
Kartengenerierung	218 ms
Total	17 177 ms

5.1. Siftgpu

Eine Implementierung des SIFT-Algorithmus von Changchang Wu, an der „University of North Carolina at Chapel Hill“, verlegt die Berechnungen von dem Hauptprozessor auf die GPU. Da die Merkmalerkennung zu den grafischen Algorithmen gehört, für welche die modernen Grafikkarten optimiert wurden, können hiermit die Laufzeit deutlich reduziert werden. Bei einer Auflösung von 640x480 Pixeln auf einer nVidia 8800 GTX mit 768 Megabyte Speicher kann der Algorithmus nach Angaben des Entwicklers bis zu 13 Bilder in der Sekunde verarbeiten. [14]

Konstruktion des Skalierungsraumes	8,7 ms
Lokalisierung der Extremwerte	7,4 ms
Merkmale in den Texturenspeicher laden	21 ms
Ermittlung der Orientierung	10 ms
Erstellen des Deskriptor	27 ms
Total	73,1 ms

Tabelle 1: Siftgpu Laufzeiten nach Chanchan Wu

Leider waren diese erstaunlichen Resultate nicht reproduzierbar. Versuche auf der Mobility Radeon X700 hatten mit 1100 ms eine erheblich längere Laufzeit. Dies bedeutet aber dennoch ein Reduzieren von 75% der Rechenzeit, die der Algorithmus auf der CPU benötigte. Dieses wird wahrscheinlich von dem Unterschied der Hardware herrühren, da die GeForce 8800 GTX deutlich schneller als die Radeon X700 ist.⁵

5.2. Blockweises Vergleichen

Wie in den Grundlagen gezeigt wurde, kann das Vergleichen der Merkmale aus Stereobildern optimiert werden, indem man die Bilder horizontal in Blöcke aufteilt. Bei der Auswahl der optimalen Blockgröße gibt es drei Werte die wichtig sind um die Qualität zu bewerten:

- Anzahl der gefundenen Bildpaare
- Zeit zum Finden der Korrespondenzpaare
- Zeit zum Aufteilen in Blöcke

Im Rahmen dieser Arbeit wurden zwei verschiedene Aufteilungsalgorithmen analysiert. Die von dem SIFT-Algorithmus erkannten Merkmale können entweder in gleich hohe Blöcke, oder in Blöcke mit möglichst gleich vielen Punkten aufgeteilt werden.

Gleichmäßig verteilte Punkte

Bei dem Ansatz der gleichmäßig verteilten Punkte werden die Blöcke so gewählt, dass jeder Block in etwa die gleiche Anzahl an Punkten (Anzahl Punkte / Anzahl Blöcke) enthält. Dazu wird jeder Block solange um jeweils ein Pixel vergrößert, bis er wenigstens die berechnete Anzahl von Punkten enthält.

⁵ Computerfachzeitschrift bescheinigen der GeForce 8800 GTX eine 10- bis 12-fach höhere Bildwiederholfrequenz in modernen Computerspielen als der Mobility Radeon X700.

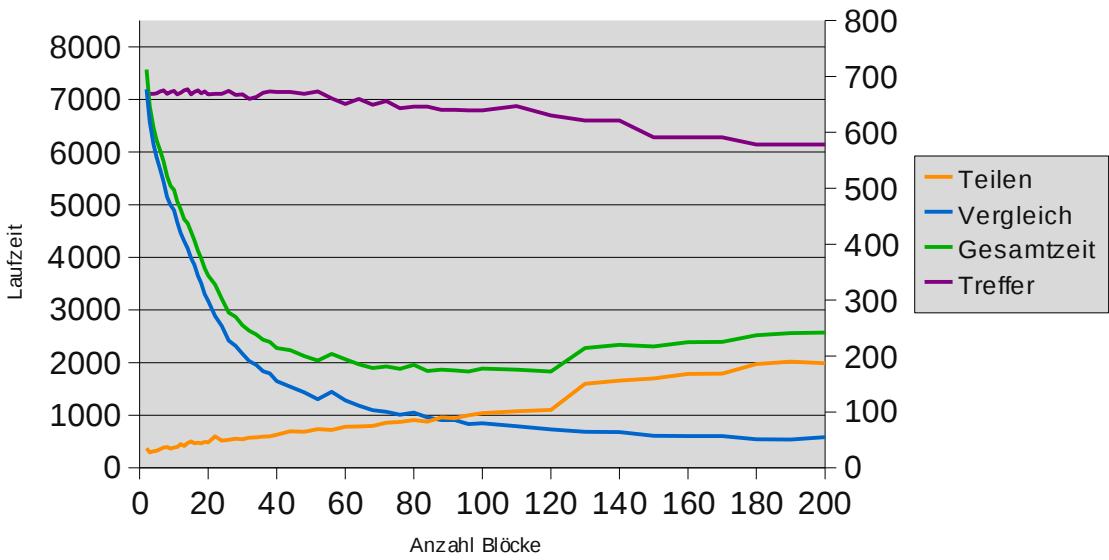


Diagramm 1: Bild aus der Simulationsumgebung, mit gleichmäßiger Punkteverteilung

Die Bilder aus der Simulationsumgebung sind sehr robust gegenüber dem Aufteilen in Blöcke, was man daran erkennen kann, dass die Anzahl an Treffern mit steigender Anzahl an Blöcken kaum zurück geht. Dies ist zu erwarten, da diese mit theoretischen Kamerassen gemacht wurden, die perfekt zueinander ausgerichtet sind.

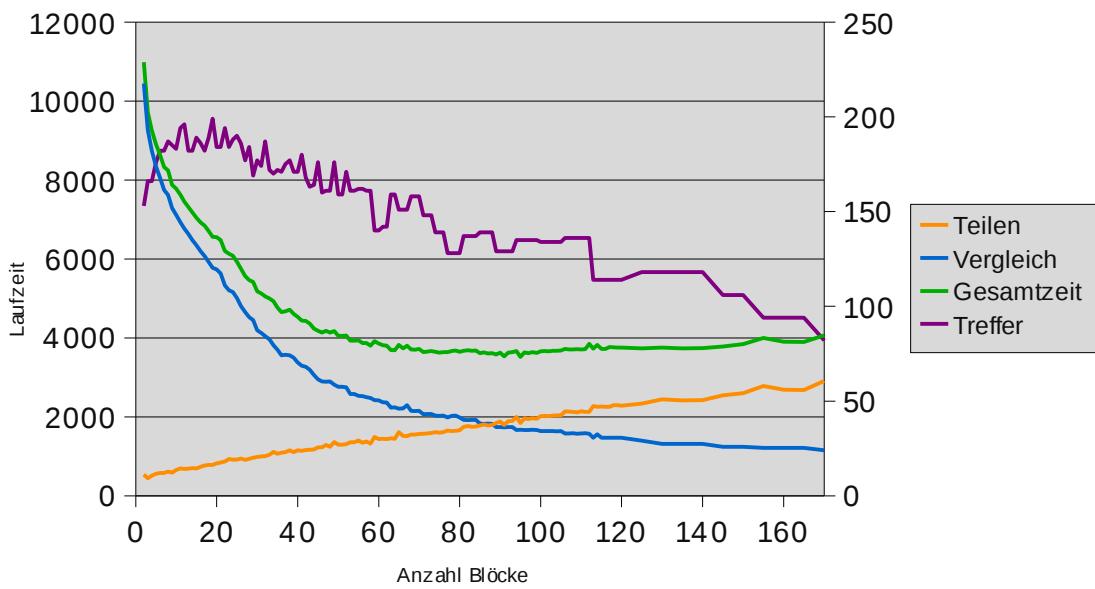


Diagramm 2: Reales Bild, mit gleichmäßigen Blöcken

In beiden Szenarien liegt das Optimum zwischen 60 und 120 Blöcken. In diesem Bereich nimmt der zusätzliche Aufwand, um die Punkte gleichmäßig zu verteilen so stark zu, dass er den benötigten Aufwand zum Vergleichen der Merkmale übersteigt. Im

Gegensatz zu der Anzahl an gefundenen Paaren in dem Simulationsbild, fällt die Anzahl der gefundenen Punktpaare bei Realbildern ab 60 Blöcken bereits unter 80% der ansonsten gefundenen Paare.

Feste Blockgröße

Bei dem Ansatz der festen Blockgröße, werden die Punkte in jeweils gleichgroße Blöcke aufgeteilt. Obwohl dadurch Blöcke entstehen können die keine, oder sehr viele Elemente enthalten, so ist das Aufteilen sehr einfach, und jeder Punkt kann anhand von nur einer Division einem Block zugeordnet werden. Deshalb fällt die Laufzeit für das Teilen auf nur fünf bis sechs Millisekunden, unabhängig davon in wie viele Blöcke aufgeteilt wird.

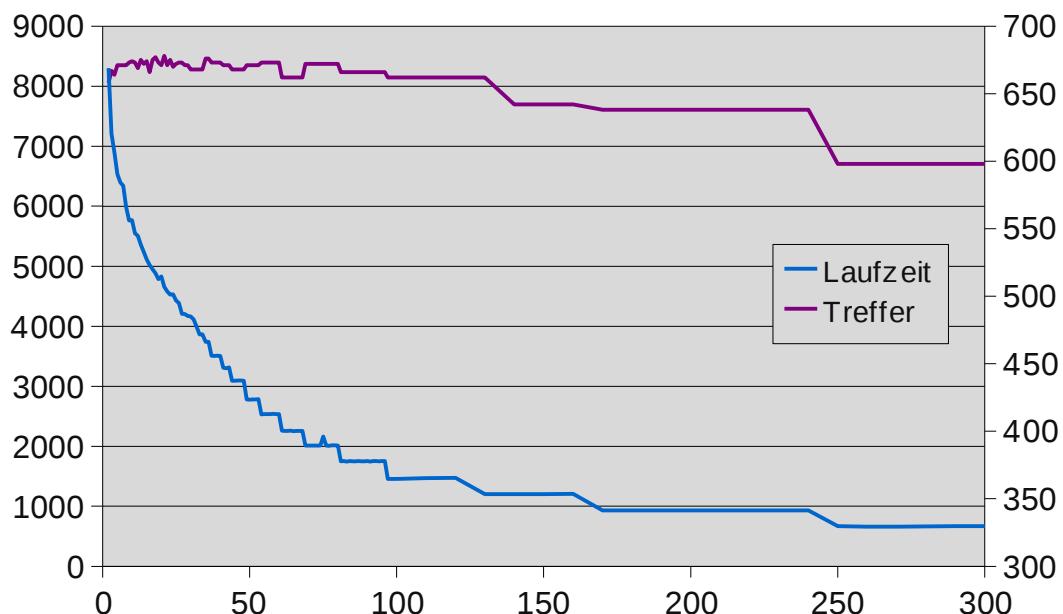


Diagramm 3: Bild aus der Simulationsumgebung, mit gleich großen Blöcken

In Diagramm 3 kann man Stufen erkennen, wie zum Beispiel für 160 oder 240. Diese stammen daher, dass das Bild nicht kontinuierlich in beliebig viele gleich große Blöcke aufgeteilt werden kann. Und so spiegelt beispielsweise der Sprung bei 240 Blöcken, den Sprung von drei Pixel zu zwei Pixel hohen Blöcken wieder.

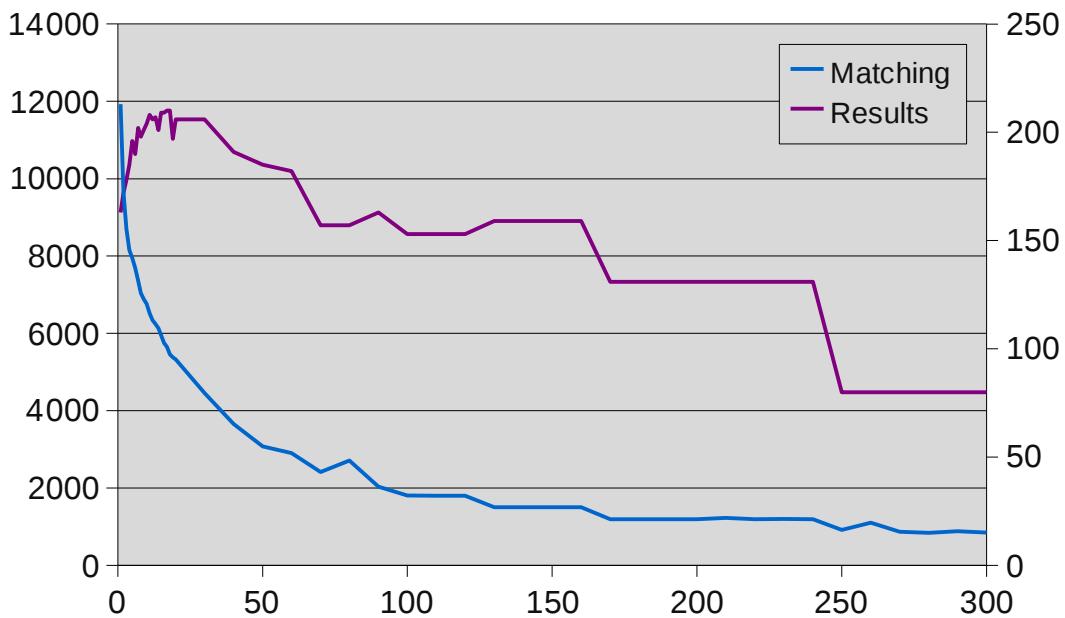


Diagramm 4: Reales Bild, mit gleich großen Blöcken

Das Aufteilen der Bilder in gleichgroße Blöcke hat sich somit bewährt, da bei gleicher Anzahl von gefundenen Korrespondenzpaaren, die Rechenzeiten sehr viel kürzer sind.

5.3. Erstellen der Höhenkarte

Verglichen mit dem Rechenaufwand, der durch das Suchen und Vergleichen der Merkmale entsteht, fallen die Laufzeiten der verbleibenden Operationen kaum noch ins Gewicht. Diese Laufzeiten wurden auf der gleichen Hardware realisiert wie die vorherigen Schritte, und beziehen sich auf 600 gefundene Korrespondenzpaare.

Das Erstellen einer Punktwolke aus den gefundenen Punktpaaren benötigt lediglich 62 Millisekunden.

Die zweidimensionale Triangulation über die X- und Z-Koordinaten der Punktwolke benötigt 109 Millisekunden.

Das Zeichnen der Höhenkarte ist linear abhängig von der gewünschten Auflösung. Eine einfache quadratische Karte mit einer Größe von 256x256 Pixeln wird in 47 Millisekunden erstellt. Supersampling – das Verfahren des Zeichnens bei doppelter Auflösung und des anschließenden Runterskalierens auf die gewünschte Auflösung – benötigt mit 147 Millisekunden, in etwa viermal solange. Höhere Auflösungen, wie 1024x1024 mit Supersampling, haben mit 1305 Millisekunden, eine deutlich zu lange

Laufzeit, ohne nennenswerte Vorteile zu bieten, da so feine Unterschiede im Gelände sowieso nicht erfasst werden würden.

5.4. Probleme

Obwohl die Vorgehensweise bereits optimiert wurde, gibt es immer noch Probleme:

Die Laufzeiten mit vier Sekunden sind noch zu lange, um von Echtzeit reden zu können. Außerdem ist der SIFT-Algorithmus sehr speicherintensiv, weshalb in etwa ein Gigabyte an Hauptspeicher zur Verfügung stehen sollte. Durch das Aufteilen der Korrespondenzsuche konnte dies aber zumindest für die Stereobildanalyse reduziert werden.

Objekte, die sich sehr nah an den Kameras befinden, weisen zwischen den beiden Kamerabildern eine starke Drehung auf, weshalb die Gefahr bestehen könnte, dass sie nicht mehr erkannt werden. Da es aber besonders wichtig ist, diese zu erkennen, sollte der Kameraabstand möglichst klein gehalten werden, auch wenn dadurch an Präzision bei der Entfernungsbestimmung verloren geht.

Für den SIFT-Algorithmus hat „The University of British Columbia“ ein Softwarepatent in den Vereinigten Staaten von Amerika angemeldet. Deshalb ist eine Verwendung der Algorithmen innerhalb der Vereinigten Staaten nur zu akademischen Zwecken erlaubt. Dieses Patent bezieht sich nicht auf eine Implementierung, sondern auf den prinzipiellen Lösungsansatz, weshalb alle Bibliotheken davon betroffen sind.

Obwohl diese außerhalb der Vereinigten Staaten nicht anerkannt werden, verzichten viele Software Hersteller auf den Einsatz solcher Algorithmen, besonders da die Europäische Union bereits das Einführen dieser Patente in Betracht gezogen hat. Dies muss bei der Betrachtung von möglichen Einsatzgebieten berücksichtigt werden.

Ausreißer

In einigen Fällen konnten Ausreißer in den Laufzeiten beobachtet werden. Für einzelne Fälle innerhalb einer Testreihe, traten manchmal Laufzeiten auf, die größer als erwartet waren, und nicht reproduziert werden konnten. Für das Ergebniskapitel wurden Durchgänge ohne zu starke Ausreißer ausgewählt.

Anzahl Blöcke	Aufteilung in ms	Vergleichen in ms
2	371	664
3	322	669
4	336	628
5	330	635
6	354	663
7	416	631
8	368	609
9	357	558
10	370	599
11	384	570
12	401	589
13	399	647
14	437	603
15	459	593
16	450	535
17	443	609
18	832	608
19	473	595
20	472	618
22	598	584
24	516	637

Tabelle 2: Beispiel eines Ausreißers in den Laufzeiten

Diese Ausreißer in der Laufzeit können eine Differenz von einigen Millisekunden bis hin zu mehreren Sekunden darstellen. Diese röhren aller Wahrscheinlichkeit nach von der C#-Philosophie her, ähnlich wie Java, dem Benutzer die Verwaltung der maschinennahen Ressourcen, insbesondere des Arbeitsspeichers, abzunehmen.

Da der Benutzer keine Kontrolle über die Speicherverwaltung hat, kann es passieren, dass während des Berechnens plötzlich kein Speicher mehr zur Verfügung steht und der Speicher entweder umorganisiert werden muss, oder Speicherblöcke auf die Auslagerungspartition verschoben werden müssen. Dies ist für ein Echtzeitsystem nicht vertretbar.

6. Zusammenfassung und Ausblick

6.1. Erreichte Resultate

Die Beispielimplementierung hat gezeigt, dass es möglich ist schnell die Bilder der Stereokameras zu interpretieren, und in eine für die Robotersteuerung nutzbare Repräsentation zu bringen. Mit vier Sekunden Laufzeit sind allerdings noch einige Verbesserungen nötig, bis von einer Echtzeitlösung gesprochen werden kann.

6.2. Mögliche Umsetzung

Die Ausreißer in der Beispielimplementierung haben gezeigt das nur eine maschinennahe Implementierung den Echtzeitanforderungen gerecht werden kann. Deshalb müsste eine Lösung die praktisch eingesetzt werden kann, entweder in C oder C++ entwickelt werden. Dies könnte zusätzlich auch noch einwenig Rechenleistung sparen. Da die Simulationsumgebung zurzeit noch nicht in der Lage ist eine Route zu planen, wäre eine Einbindung in diese aber nur bedingt nützlich.

Um sich nicht zu sehr auf eine Umgebung festzulegen, wäre es sinnvoll eine Framework zu benutzen welches sich bereits etabliert hat. Unter anderem wäre in diesem Kontext die MCA2⁶ denkbar, da diese bereits auf Lauron eingesetzt wird.

6.3. Geschwindigkeitsoptimierung

Eine offensichtliche Lösung um den Algorithmus zu beschleunigen wäre, eine schnellere Hardware zu benutzen, insbesondere eine leistungsstärkere Grafikkarte.

Parallelisierbarkeit

Um von modernen Multiprozessorsystemen profitieren zu können, ist insbesondere die Parallelisierbarkeit von Bedeutung. Viele Algorithmen die zur Rekonstruktion des Terrains benötigt werden sind parallelisierbar.

Da der SIFT-Algorithmus auf beiden Bildern getrennt nach markanten Merkmalen sucht, ist es naheliegend, dass dies parallel auf zwei verschiedenen Grafikkarten laufen könnte. Aber selbst innerhalb eines Bildes wäre es möglich, mehrere Oktaven parallel nach Extrema zu durchsuchen. Auch das Vergleichen der gefundenen Merkmale kann

⁶ Modular Controller Architecture Version 2

auf fast beliebig viele Threads aufgeteilt werden, unabhängig davon ob man sie unoptimiert, oder blockweise vergleicht.

Die Anwendung des Strahlensatzes zur Bestimmung der dreidimensionalen Punkte und die Interpolation der Informationen anhand der Dreiecke kann ebenfalls aufgeteilt werden, wobei voraussichtlich wegen der geringen Laufzeiten, der Mehraufwand für das Aufteilen der Prozesse die Einsparung der Rechenzeit übersteigt.

Dadurch, dass die Triangulation mit einem „Divide and Conquer“-Algorithmus realisiert wurde, ist das Parallelisieren sehr einfach, denn diese Algorithmen eignen sich sehr gut, um in Multiprozessorsystemen ausgeführt zu werden. Die inkrementelle Triangulation hingegen ist prinzipiell nicht parallelisierbar, weil jeder hinzugefügter Punkt potenziell die gesamte bisherige Triangulation verändern könnte.

Korrespondenzanalyse zwischen Ansichten

Mit der wachsenden Karte, müssen beim Einbinden einer neuen Ansicht in die bisherige Punktwolke, immer mehr Merkmale mit den neuen Punkten verglichen werden.

Aus diesem Grund muss ein Weg gefunden werden, wie die Anzahl dieser Vergleiche eingeschränkt werden kann. Eine Möglichkeit wäre eine Abbruchbedingung, die die Vergleiche unterbricht, sobald genügend Punkte vorliegen um die neue Punktwolke auszurichten. Darunter könnte jedoch die Genauigkeit der Kombination leiden.

Eine elegantere Lösung wäre die von der Robotersteuerung berechnete Position zu benutzen, um den Ausschnitt aus dem bisherigen Modell auszuwählen, der im Sichtbereich des Roboters liegen sollte. Dann müssten nur noch die Merkmale der Punkte dieses Ausschnittes mit den Merkmalen der neuen Punkte verglichen werden. Der Nutzen dieses eingeschränkten Suchraumes hängt von der Genauigkeit ab, mit der der Roboter in der Lage ist seine Position zu bestimmen, und kann erst nach Implementierung auf dem Laufroboter ermittelt werden.

6.4. Verbesserung der Resultate

Bei der Verbesserung der Resultate muss man bedenken, dass dies meistens zu längeren Laufzeiten führen kann.

Erste Schritte um bessere Resultate trotz Ausreißern oder Feldern mit schwachem Informationsgehalt zu erhalten, wurden bereits unter dem Kapitel 4.5 „Filter“ behandelt.

Bisher wurden diese einfach entfernt, jedoch könnte man alternativ dazu eine detailliertere Qualitätskarte erstellen, in der jedem Referenzpunkt eine Güte zugeordnet wird, die beispielsweise abhängig von seiner Entfernung zur Kamera, oder der Genauigkeit mit der die dazu gehörigen Sichtlinien sich kreuzen, sein könnte. Anhand dieser kann man dann später für die gesamte Karte eine detaillierte Gütekarte erstellen, so dass auch Regionen, die bisher wegen zu unsicheren Informationen ganz entfernt worden sind, eingebunden werden können, wenn dies auch nur mit schwacher Güte.

Detailliertere Interpolation

Bisher wurde innerhalb der Dreiecke, die mit Hilfe der Dreieckszerlegung gefunden wurden, linear interpoliert. Wenn man die angrenzenden Dreiecke mitberücksichtigen würde, könnte man entlang den Kanten eines Dreiecks die Steigung bestimmen und anhand dieser eine dreidimensionale Spline-Interpolation durchführen, die zu genaueren Resultaten führen würde.

Abbildungsverzeichnis

Abbildung 1: Koordinatensystem.....	3
Abbildung 2: Lauron IV b.....	3
Abbildung 3: Simulationsumgebung für Lauron.....	5
Abbildung 4: Prinzip einer Lochkamera.....	6
Abbildung 5: Mathematisches Kameramodell.....	7
Abbildung 6: Delaunay-Kriterium.....	9
Abbildung 7: Ausgleichungsrechnung.....	10
Abbildung 8: RANSAC mit ungünstigen Kandidaten.....	10
Abbildung 9: RANSAC mit günstigen Kandidaten.....	10
Abbildung 10: Auszug aus einer Gauß'schen Weichzeichen Matrix mit $\sigma = 1,6$	11
Abbildung 11: Differenz des Gauß'schen Weichzeichnens.....	15
Abbildung 12: Betrachtete Nachbarn der Extremwertsuche.....	15
Abbildung 13: Berechnung eines Schlüsselpunktdeskriptors.....	18
Abbildung 14: Annäherung des Kreuzungspunktes.....	21
Abbildung 15: Interpolation anhand von nächsten Punkten.....	22
Abbildung 16: Interpolation anhand von umschließenden Punkten.....	22
Abbildung 17: SIFT Oktave 0.....	24
Abbildung 18: SIFT Oktave 1.....	24
Abbildung 19: SIFT Oktave 2.....	25
Abbildung 20: SIFT Oktave 3.....	25
Abbildung 21: SIFT Oktave 4.....	25
Abbildung 22: Blockweises Vergleichen mit 6 Blöcken.....	26
Abbildung 23: Voxeldarstellung der gefundenen Punkte.....	26
Abbildung 24: Draufsicht auf die Voxelwolke.....	27
Abbildung 25: Delaunay-Dreieckszerlegung.....	28
Abbildung 26: Beispiel einer Höhenkarte.....	29
Abbildung 27: Beispiel einer Qualitätskarte.....	29
Abbildung 28: Durch RANSAC entfernte Punkte.....	30
Abbildung 29: Zuordnen von Ansichten.....	32
Abbildung 30: Kombination von zwei Voxelwolken.....	33
Abbildung 31: Karte über die vereinte Punktmenge.....	35

Literaturverzeichnis

- 1: **Herms, A.:** *Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren*, Diplomarbeit 2004, Otto-von-Guericke Universität
- 2: **Ruffler, U.:** *Laufplannung basierend auf realitätsnahen Umgebungsdaten für einen sechsbeinigen Laufroboter*, Diplomarbeit 2006, Hochschule Mannheim
- 3: **Shreer, O.:** *Stereoanalyse und Bildsynthese*, Springer Verlag, 2005, ISBN 3-540-23439-X
- 4: **Eberly, D.:** *3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics*, Morgan Kaufmann, 2001, ISBN 978-1558605932
- 5: **Sunday, D.:** *Distance between Lines and Segments with their Closest Point of Approach*,
http://www.softsurfer.com/Archive/algorithm_0106/algorithm_0106.htm, Letzte Einsicht am 10. März 2008
- 6: **Burke, P.:** *Efficient Triangulation Algorithm Suitable for Terrain Modelling*, Presented at Pan Pacific Computer Conference, Beijing, China., 1989
- 7: **Fischler A.; Bolles R. C.:** *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, Communications of the ACM, 1981
- 8: **Moravec H. P.:** *Towards automatic visual obstacle avoidance*, 5th International Join Conference on Artificial Intelligence, 1977
- 9: **Harris C.; Stephens M.:** *A combined corner and edge detector*, Fourth Alvey Vision Conference, 1988
- 10: **Lowe, D. G.:** *Distinctive Image Features from Scale-Invariant Keypoints*, International Journal of Computer Vision, 2004
- 11: **Witkin, A. P.:** *Scale-Space Filtering*, International Joint Conference On Artificial Inteligence, 1983
- 12: **Edelman, S.; Intrator, N.; Poggio, T.:** *Complex cells and object recognition*, <http://kybele.psych.cornell.edu/~edelman/nips97.pdf>, Letzte Einsicht am 10. März 2008
- 13: **Sebastian Nowozin:** *libsift - Scale-Invariant Feature Transform implementation*, <http://user.cs.tu-berlin.de/~nowozin/libsift/>, Letzte Einsicht am 10. März 2008
- 14: **Changchang Wu:** *Sift on GPU*, <http://cs.unc.edu/~ccwu/siftgpu/>, Letzte Einsicht am 10. März 2008