Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Christian Sandhagen

Methodenentwicklung zur Qualitätsverbesserung von Tiefenbildern

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Sandhagen

Methodenentwicklung zur Qualitätsverbesserung von Tiefenbildern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 25. September 2014

Christian Sandhagen

Thema der Arbeit
Methodenentwicklung zur Qualitätsverbesserung von Tiefenbildern

Stichworte
Kinect, Tiefenbild, Qualitätverbesserung, Fusion, Bildverarbeitung

Kurzzusammenfassung
In diesem Dokument werden Tiefenbilder untersucht, welche verschiedene Fehlereffekte aufweisen können. Hierfür werden Ansätze verfolgt, um diese Fehlereffekte in Tiefenbildern zu beheben,um dadurch eine höhere Qualität der Tiefenbilder zu erreichen.
Christian Sandhagen

Title of the paper
Development of methods to improve the quality of depth images

Keywords
Kinect, depth image, quality improvement, fusion, image processing

Abstract
In this document, depth images are studied which may have different error effects. In order to achieve a higher quality of depth images, attempts are being pursued to resolve error effects in the depth images.

Danksagung

An dieser Stelle möchte ich mich bei einigen Personen für Ihre Unterstützung bedanken.

Zu aller erst, möchte ich mich bei meinem betreuenden Prof. Dr. Florian Vogt, für die Unterstützung bei der Themenfindung und für die Betreuung bei dieser Arbeit bedanken.

Desweiteren danke ich, meinem Erstprüfenden Prof. Dr. Andreas Meisel und Zweitprüfenden Prof. Dr. Wolfgang Fohl .

Mein Dank gilt außerdem Tom Sander, Iris Sander, Kim Sander und Kim Bergander für Ihre Unterstützung bei der Korrektur dieser Arbeit.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Seit 5 Jahren sind verschiedene Tiefenkameratechnologien in Konsumerprodukten zu finden. Tiefenbilder besitzen zu den herkömmlichen 2D-Bildern, die wir alle aus unseren normalen Standard-Kameras kennen, eine 3. Dimension. Dies bedeutet, nicht nur die Höheninformation und Breiteninformation einer bestimmten Szene auf einem Bild zu besitzen, sondern ebenfalls die Entfernung der Kamera (Tiefeninformation). Diese Tiefenbilder findet man kombiniert mit 2D-Bildern beispielsweise beim Fernsehen, dem sogenannten 3DTV. Weiterhin kommen Tiefenbilder auch in der Industrie und in der Medizin zum Einsatz. In der Industrie beispielsweise bei Robotern mit Greifarmen. Hier werden die Tiefeninformationen dazu genutzt, die Entfernung zu berechnen, die der Greifarm zurücklegen muss, um das vorgesehene Objekt zu greifen. Tiefenbilder können auf verschiedene Weise generiert werden.Generierte Tiefenbilder besitzen jedoch nach ihrer Generierung verschiedene Problemfälle, sodass man sie nicht sofort nutzen kann. Dies ist zurückzuführen auf die technische Limitierung der Bildgenerierungsverfahren, die hierfür genutzt werden. Daher müssen die generierten Tiefenbilder noch bearbeitet werden, bevor sie weiterverwendet werden können.

1.2 Thema dieser Arbeit

Diese Arbeit beschäftigt sich mit der Untersuchung von Tiefenbildern, welche verschiedene Fehlereffekte aufweisen. Es werden mehrere Ansätze aufgezeigt, wie Tiefenbilder generiert werden können. Anhand eines Testaufbaus sollen die speziellen Fehlereffekte erkannt werden, die aus diesem resultieren. Um die Qualität der Tiefenbilder zu verbessern, sollen jene Fehlereffekte analysiert und Ansätze vorgestellt werden, mit denen besagte Fehlereffekte behoben werden können.

1.3 Aufbau

In Kapitel 2 werden einige Grundlagen dieser Arbeit ausgeführt und erläutert, sowie der Testaufbau erklärt. In Kapitel 3 werden Fehlereffekte und Problemstellungen analysiert und verdeutlicht. In Kapitel 4 findet man verschiedene Arbeiten, die sich mit denselben Fehlereffekten und deren Behebung befasst. Kapitel 5 befasst sich mit den Erläuterungen von Ideen, die zur Behebung der Fehlereffekte und zur Qualitätsverbesserung der Tiefenbilder untersucht wurden.Kapitel 6 beinhaltet eine Zusammenstellung der einzelnen Ergebnisse dieser Untersuchungen, die anhand von Bilder visualisiert wurden. Im weiteren Verlauf dieser Arbeit wird in Kapitel 7 eine Zusammenfassung dargestellt und ein Ausblick darauf geboten, wie man die untersuchten Methoden noch weiterentwickeln kann.

2 Grundlagen

In diesem Kapitel werden verschiedene Grundlagen besprochen, in denen es um Begriffs- und Verfahrenserklärungen geht.

2.1 Entstehung von Tiefenbildern

Tiefenbilder können auf verschiedene Weise erzeugt werden. Im folgenden Kapitel werden verschiedene Tiefenbilderzeugungen aufgezeigt.

2.1.1 Tiefenbildgenerierung durch Lasersensoren

Generierung von Tiefenbildern durch Lasersensoren basieren auf dem Prinzip der Lichtlaufzeitmessung. Hier wird von einer Laserdiode ein Laserimpuls ausgesendet. Dieser wird an der Oberfläche von Objekten reflektiert und zurückgestrahlt. Der reflektierte Strahl wird von einem optoelektronischen Empfänger empfangen und dabei wird vergangene Zeit zwischen dem (ver-)Senden und dem Empfangen des Strahls gespeichert. Mit dieser Zeit kann man über die Konstante der Lichtgeschwindigkeit die Entfernung zum jeweiligen Objekt berechnen.

$$s = t/2 * c$$

s = Entfernung des Objekts
t = Laufzeit des Lichtstrahls
c = Lichtgeschwindigkeit

Dieses Verfahren wird beispielsweise von Laserscannern verwendet, um Tiefenbilder zu generieren. [XPER]

2.1.2 Tiefenbildergenerierung durch Ultraschall

Wie bei den Lasersensoren wird hier das Verfahren der Laufzeitmessung genutzt. Im Gegensatz zum Lasersensor wird nicht die Lichtlaufzeit verwendet, sondern die Laufzeit des Schalls. Man nutzt die Ausbreitungsgeschwindigkeit von Schall in Räumen. Ein Ultraschallsensor funktioniert als Sender und als Empfänger. Der Sender strahlt in einer bestimmten Frequenz einen kegelförmigen Strahl aus. Wenn der Strahl auf ein Objekt trifft, reflektiert dieses die Schallwellen zurück auf den Empfänger. Durch die verstrichene Zeit zwischen Senden und Empfangen, kann man nun die Entfernung vom Ultraschallsensor zu dem Objekt, welches die Schallwellen reflektiert hat, berechnen.

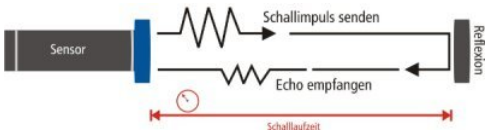


Abbildung 2.1: Funktionsprinzip der Ultraschallmessung[1]

Vorteile: Ultraschallwellen bieten eine zuverlässige Möglichkeit, Entfernungen unabhängig von Farbeigenschaften und Oberflächeneigenschaften des Objekts zu berechnen. [XPER]

2.1.3 Tiefenbildgenerierung mit Kameras

Bei einigen Varianten der Gewinnung von Tiefenbildern durch Kameras, wird das Verfahren der Stereoskopie angewendet. Die Stereoskopie ist vom menschlichen Sehen abgeleitet und beruht darauf, aus zwei unterschiedlichen Quellen die Tiefe eines Objektes bestimmen zu können. Weiterhin wird meist ein Verfahren verwendet welches als Triangulation bezeichnet wird. Hierbei bilden eine Lichtquelle und ein Sensor zusammen mit dem zu messenden Objekt ein Dreieck. Der Abstand zum Objekt, wird durch die festgelegten Abstände zwischen Sensor und Lichtquelle und dem festgelegten Winkel zwischen Sensor und Lichtquelle ermittelt. [FEMO]

2.1.3.1 Passive Tiefenbildgewinnung

Bei dem passiven Verfahren wird mittels Stereoskopie ein Tiefenbild berechnet. Hier werden mit einer oder mehreren Kameras, Bilder eines Objekts aus unterschiedlichen Ansichten erfasst. Um nun ein dreidimensionales Bild zu konstruieren, müssen sogenannte Korrespondenzen gefunden werden. Dies bedeutet, dass auf dem Objekt identische Bildpunkte

[1] Quelle: http://www.dietz-sensortechnik.de/info/11.html letzter Zugriff : 20.09.2014

gefunden werden müssen, um daraus Raumpunkte berechnen zu können. Wenn genügend Punkte gefunden werden, kann man daraus das Bild rekonstruieren. Hier wird keine gerichtete Lichtquelle verwendet sondern nur das Umgebungslicht als Beleuchtung eingesetzt, deshalb bezeichnet man dieses Verfahren auch als passives Verfahren. Die Tiefenwerte werden mittels Triangulation berechnet. [FEMO] [HEHA, September 2001]

2.1.3.2 Aktive Tiefenbildgewinnung

Bei der aktiven Tiefenbildgewinnung wird ebenfalls das Verfahren der Stereoskopie angewandt. Man wirft auf das Objekt zusätzlich ein Muster-Gitter, beispielsweise durch einen Laser. Man kann mit Hilfe des Musters korrespondierende Bildpunkte leichter finden und so ein Tiefenbild generieren. Dieses Verfahren bezeichnet man als aktives Verfahren, weil man eine aktive Lichtquelle zur Bestrahlung des Objektes verwendet (beispielsweise einen Laser). Auch hier wird das Verfahren der Triangulation zum berechnen der Tiefenwerte verwendet. [FEMO] [HEHA, September 2001]

2.2 Aufbau eines Digitalbildes

In der Bildverarbeitung wird ein einzelner Bildpunkt als Pixel bezeichnet. Ein digitales Bild besteht aus einer Anzahl von Bildpunkten. Ein Bild mit der Auflösung 640 x 480 Pixel besitzt beispielsweise 307200 Bildpunkte. Jeder Bildpunkt enthält eine bestimmte Information. [UNIM]

2.2.1 Pixel eines Grauwertbildes

Ein Pixel in einem Grauwertbild enthält in dieser Arbeit einen 8-Bit-Wert, der die Helligkeit des Pixels angibt. Durch verschiedene Helligkeiten entstehen unterschiedliche Details, die unser Auge auf diesem Bild wahrnimmt. Insgesamt kann jeder Pixel in einem Grauwertbild einen Wert von 0 bis 255 annehmen. Wenn der Grauwert den Wert 0 annimmt, ist der dargestellte Grauwert schwarz, bei einem Wert von 255 wird der angegebene Grauwert weiß dargestellt. Durch die dazwischenliegenden Abstufungen (0-255) können auf einem Grauwertbild 256 verschiedene Grauwerte dargestellt werden. Bild 2.2 verdeutlicht den Aufbau eines Grauwert-Bildpunktes und die Kombinationen die daraus entstehen können.

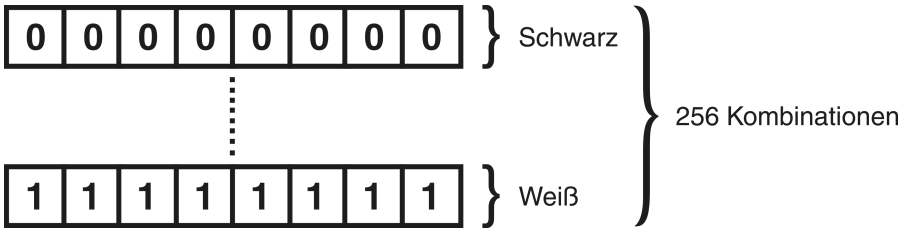


Abbildung 2.2: Grauwert Bildpunkt 8-Bit

2.2.2 Pixel eines RGB Bildes

Farbbilder sind etwas komplexer als ein Grauwert-Bild. Die Informationen, die ein Pixel in einem RGB-Bild enthält, sind drei 8-Bit-Werte, die gemischt eine bestimmte Farbe ergeben. Jeder dieser 8-Bit-Werte ergibt eine der Grundfarben rot, grün oder blau. Da drei Grundfarben existieren, ist ein Pixel in einem RGB-Bild 24 Bit groß, dies wurde in Bild 2.3 veranschaulicht. Jede Grundfarbe kann einen Wert zwischen 0 und 255 annehmen. Setzt man alle einzelnen Komponenten des 24-Bit-Pixels auf 0, erhält man die Farbe Schwarz. Wenn man alle einzelnen Farbanteile rot, grün oder blau auf 255 setzt, ergibt sich die Farbe weiß. Setzt man nur einen Farbanteil auf den Wert 255, erhält man den jeweiligen Farbton rot, grün oder blau (siehe Bild 2.4). Wenn man alle 3 Farbstufen miteinander multipliziert erhält man 16,7 Millionen verschiedene Farbabstufungen, die ein einzelner Pixel annehmen kann (siehe Bild 2.5). [VBPA]

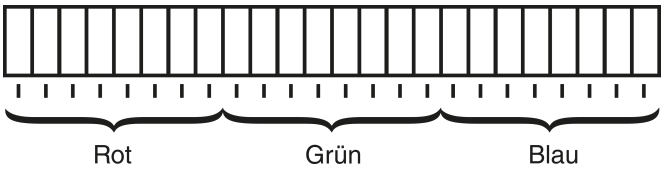


Abbildung 2.3: Aufbau eines RGB 24 Bit Bildpunktes

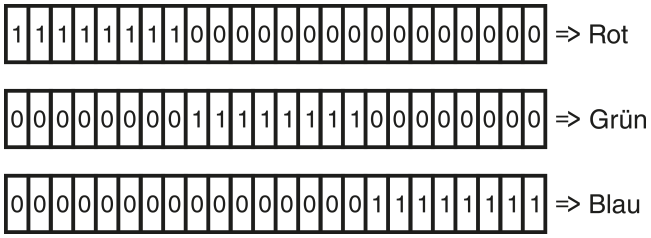


Abbildung 2.4: Einzelne Farbwerte im RGB 24 Bit Bildpunkt

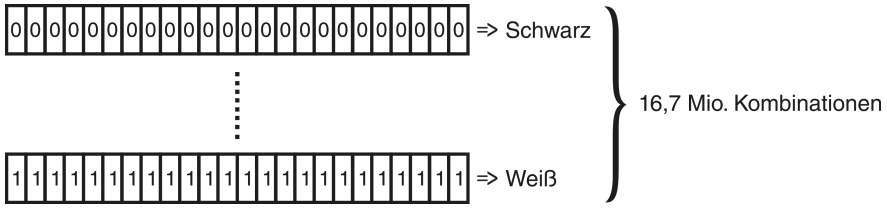


Abbildung 2.5: Mögliche Kombinationen die ein RGB-Bildpunkt annehmen kann

2.3 Tiefpassfilter zur Rauschunterdrückung

Filter sind ein Element der Bildverarbeitung. Mit ihnen kann man beispielsweise das Rauschen in Bildern unterdrücken oder Kanten hervorheben, oder diese wegschneiden.

Im Grundlegenden sind Filter Bildoperatoren die auf ein Bild angewendet werden. Im Eigentlichen besteht ein Filter aus einer sogenannten Filtermaske. Sie hat eine ungerade Anzahl an Zeilen und Spalten. Diese Filtermaske wird über die einzelnen Pixel eines Bildes gelegt. Die kleinste Filtermaske ist eine 3x3 Maske. [HUBE]

2.3.1 Median Filter

Der Median Filter, wird auch als Mittelwertfilter bezeichnet. Zuerst wird eine Maske über einen Bildpunkt im Bild zu gelegen. Anschließend werden alle Bildpunkte die in dieser Maske liegen der Größe nach sortiert. Den mittleren Wert dieser Sortierung nennt man Median. Der Bildpunkt über den die Maske gelegt wurde, wird durch den Medianwert ersetzt. Bild 2.3.1 verdeutlicht die Funktion an einem Beispiel.

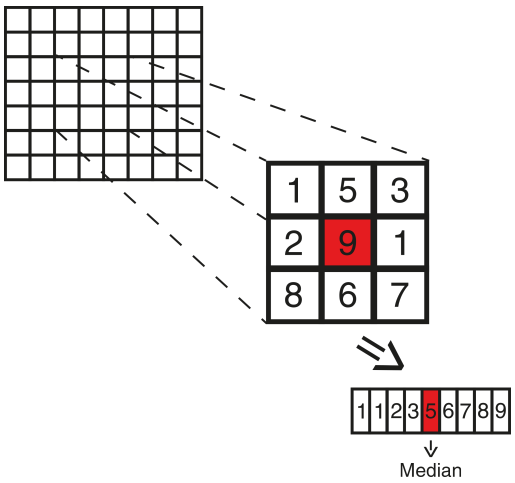


Abbildung 2.6: Median Filter Funktionsweise

2.4 Inpainting

Inpainting ist ein Verfahren der Bildverarbeitung. Zum einen können Bilder rekonstruiert werden. Wenn also Anteile von Bildern zerstört wurden oder fehlerhaft sind, können mit dieser

Methode diese Anteile wieder hergestellt werden. Zum anderen ist es möglich, Objekte aus Bildern zu entfernen. Ein Beispiel hierfür findet man in dem Paper von Criminisi A. , Perez P. und Toyama K. [CPT, 2003]. In ihm gezeigt, wie man mit Inpainting Objekte aus einem Bild entfernen kann.

2.5 Microsoft Kinect

Für den Testaufbau wurde in dieser Arbeit die Kinect der Spielekonsole XBOX 360 verwendet. Kinect ist ein Produkt von Microsoft in Kooperation mit Primesense. Die Kinect wurde für den Spielemarkt entwickelt, um Bewegungen zu erkennen und so mittels Gesten, Vidoespiele steuern zu können.

2.5.1 Aufbau

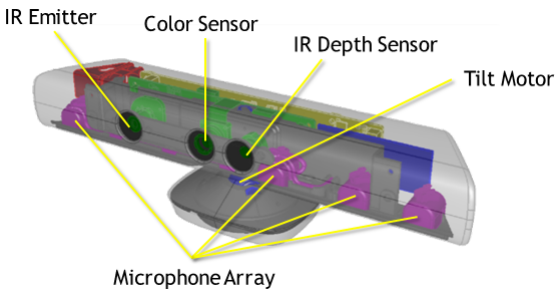


Abbildung 2.7: Aufbau der Kinect[2]

Die Microsoft Kinect ist für einen Bereich zwischen 0,6 m und ca. 6 m ausgelegt. In diesem Bereich können Objekte erkannt und aufgenommen werden.

Die Kinect besitzt einen Standard Color Sensor. Dieser ist eine normale RGB Farbkamera. Mit ihr kann die Kinect Farbbilder aufnehmen.

Weiterhin enthält die Kinect einen IR-Projektor und eine IR-Kamera. Der IR-Projektor sendet ein spezielles Infrarot-Muster in den Raum. Die IR-Kamera kann dieses Muster aufnehmen und weiterleiten. Aus diesen Daten werden intern in der Kinect die räumlichen Koordinaten, durch Triangulation berechnet. So entsteht ein Tiefenbild.

Zwischen der RGB-Kamera und der IR-Kamera gibt es einen Versatz von ca. 2,5 cm. Der Abstand zwischen dem IR-Projektor und IR-Sensor beträgt 7,5 cm. Somit stimmen die Sichtweiten der Kameras nicht überein (Parallaxe) und müssen vor der Weiterverarbeitung noch vorverarbeiten werden. Dies wird im Kapitel "Kalibrierung"(5.1) näher erläutert.

[2]Quelle: http://msdn.microsoft.com/en-us/library/jj131033.aspx letzter Zugriff: 02.09.2014

Die Kinect besitzt weiterhin 4 Mikrofone, mit deren Hilfe man Audioinformationen aufnehmen kann. Diese sind am äußeren Rand angebracht und dienen dazu, Sprachbefehle aufzunehmen. In dieser Arbeit wird den Mikrofonen keine weitere Beachtung geschenkt.

Die Microsoft Kinect besitzt außerdem einen Motor. Dieser Motor ist dazu da die Sensorleiste neigen zu können. Die Neigung der Sensorleiste ist um $+28°$ und $-28°$ möglich. [FEMO]

2.5.2 Tiefenbild der Kinect

Das Tiefenbild wird mittels aktiver Stereoskopie erzeugt. Die Bildpunkte des Tiefenbildes sind als 11-Bit-Wert kodiert. Deshalb werden die Bildpunkte in einer Short-Maske abgespeichert. In dieser Arbeit werden die Tiefenwerte umgewandelt um verschiedene Aspekte und Problemstellungen mit Hilfe von Bildern verdeutlicht. Die Bildpunkte des Tiefenbildes werden somit in 8-Bit-Werte umgewandelt.

2.5.3 Farbbild der Kinect

Der Colorsensor der Kinect liefert ein Standard RGB-Bild. Dieses Bild wird in einer Byte-Maske ausgegeben. Um einen Bildpunkt zu bekommen, muss man immer 3 Byte-Werte aus der Maske auslesen. Die einzelnen Bytes stehen für die einzelnen Farbanteile eines Bildpunktes (rot,grün und blau). Das RGB-Bild ist, wie oben im Kapitel (2.2.2) beschrieben, aufgebaut.

3 Problemstellung

In diesem Kapitel geht es um die Problemstellung der Arbeit. In dieser Arbeit werden Tiefenbilder untersucht, um ihre Qualität zu verbessern. Um dies tun zu können, muss man sich die Probleme, welche entstehen können, anschauen und analysieren.
Hier werden die Probleme aufgezeigt und analysiert, die durch die Eigenschaften des Testaufbaus entstehen.

3.1 Fehlereffekte

1. Fehlerhafte Tiefenbild-Erzeugung unter Einfluss des Sonnenlichts
2. Schwarze Oberflächen von Objekten
3. Rauschen in Tiefenbildern
4. Fehlerhafte Bildpunkte
5. Reflektierende und durchsichtige Oberflächen
6. Ränder

3.2 Analyse der Fehlereffekte

3.2.1 Fehlerhafte Tiefenbild-Erzeugung unter Einfluss des Sonnenlichts

Im Testaufbau wird eine Kinect von Microsoft verwendet. Die Kinect von Microsoft erzeugt ihre Tiefenbilder mittels aktiver Stereoskopie. Hierbei wird ein Infrarot-Muster durch einen Projektor auf die Szene geworfen. Anschließend wird durch eine Infrarot-Kamera das Muster aufgefangen und so ein Tiefenbild errechnet. Das Problem beim Erzeugen von Tiefenbildern mittels der Kinect von Microsoft unter Einfluss von Sonnenlicht liegt an den Infrarotanteilen des Sonnenlichts. Das Sonnenlicht weist eigene Infrarotanteile auf, somit wird das Infrarot-Muster des Infrarot-Projektors überlagert. Die Infrarot-Kamera kann nicht mehr

zwischen dem Muster und den Infrarotanteilen des Sonnenlichts unterscheiden. Somit ist eine Berechnung der Tiefenwerte nicht mehr möglich.

3.2.2 Schwarze Oberflächen von Objekten

Für Microsofts Kinect-Kamera ist es sehr schwer, schwarze Objekte zu erkennen und hier die zugehörigen Tiefenwerte zu berechnen. Schwarze Objekte reflektieren das Infrarot-Muster des Infrarot-Porjektors nicht. Daher kann die Infrarot-Kamera der Micorsoft-Kinect keine Referenzpunkte bestimmen, mit deren Hilfe die Tiefe berechnet werden kann.

3.2.3 Rauschen in Tiefenbildern

Die Tiefenbilder der Microsoft-Kinect zeigen in den Bereichen, in denen entfernte Objekte dargestellt werden,verrausschte Bildpunkte. Dies kommt durch eine ungenaue Berechnung und durch die Entfernung des Objektes. Meist tritt das Rauschen an Kanten auf und erschwert somit die genaue Identifizierung von Kanten.

3.2.4 Fehlerhafte Bildpunkte

Fehlerhafte Bildpunkte in dem Tiefenbild der Kinect sind auf den Aufbau der Micorsoft-Kinect zurückzuführen. Da der Kinect-Infrarot-Projektor und die Infrarot-Kamera 7,5 cm auseinander liegen, haben sie eine unterschiedliche Sicht auf die Objekte, deren Tiefe berechnet werden soll. Durch die unterschiedlichen Blickwinkel entsteht ein Schattenbereich. Dadurch, dass die Infrarot-Kamera einen anderen Bereich aufnimmt, als von dem Infrarot-Projektor angestrahlt wird, kann es dazu kommen, dass von der Kamera Bereiche aufgenommen werden, die hinter dem angestrahlten Objekt liegen. Da der Projektor nicht hinter das Objekt strahlt, entsteht der Schattenbereich. Für diesen Schattenbereich können keine Tiefenwerte berechnet werden, weshalb die Kinect diese Schattenbereiche als fehlerhafte Bildpunkte erkennt und sie den Tiefenwert als "Value 0"definiert. Diese fehlerhaften Bildpunkte stellen ein Problem dar. Wenn man beispielsweise Kanten erkennen möchte und man am Rande einer Objektkante auf ein fehlerhaften Bildpunkt trifft, besitzt man keine Entfernungsdaten. Diese fehlerhaften Bildpunkte gilt es zu befüllen. Bild 3.1 verdeutlicht die Entstehung der fehlerhaften Bildpunkte.

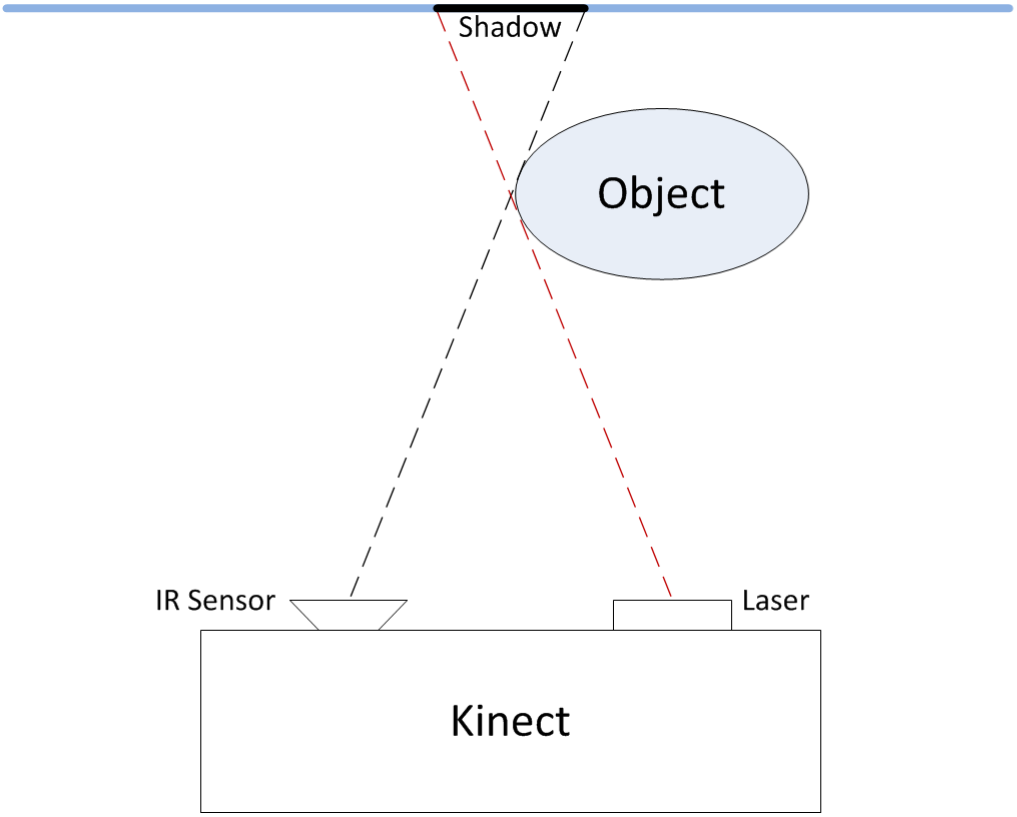


Abbildung 3.1: **Tiefenbild Kinect Schattenbildung [ATPA, 2012]**

3.2.5 Reflektierende und durchsichtige Oberflächen

Objekte, welche durchsichtig sind bzw. spiegeln, sind in Tiefenbildern schlecht erkennbar. Durchsichtige Objekte reflektieren das Infrarot-Muster nicht und daher sind sie nicht darstellbar. Reflektierende Oberflächen reflektieren das Infrarot-Muster in eine falsche Richtung. So wird das Muster über Umwege wahrgenommen.

3.2.6 Ränder

Die Tiefenbilder, die man mit dem Testaufbau erhält, weisen Ränder auf. Warum diese Ränder entstehen, also auf welches Verfahren diese zurückzuführen sind,wird im späteren Teil der Arbeit unter dem Kapitel (5.1) dargestellt. Diese Ränder enthalten keine Tiefenwerte und werden als "0 Value"definiert. Um die Tiefenbilder richtig betrachten zu können, müssen wir eine Unterscheidung zwischen einem Rand und den oben beschriebenen fehlerhaften

Bildpunkten treffen können . Es gilt also, die fehlerhaften Bildpunkte und die Ränder voneinander zu trennen.

3.3 Zusammenfassung

Die in der Problemstellung aufgeführten Fehlereffekte, welche zum Teil von sowohl der Kamera als auch von äußeren, unvermeidlichen Faktoren bedingt sind, werden in dieser Arbeit analysiert und anschließend ein an die Problemstellung knüpfender Lösungsansatz entwickelt. Hierbei wird der Fokus in dieser Arbeit auf die Fehlereffekte (Ränder, fehlerhafte Bildpunkte und Rauschen) gelegt.

4 Verwandte Arbeiten zur Qualitätsverbesserung von Tiefenbildern

4.1 Tiefenbild Erweiterung für die Kinect durch Nutzung von Bereichsvergrößerung und einem Bilateral Filter

In diesem Paper von Li Chen, Hui Lin und Shutao Li wird ein Ansatz untersucht, der es ermöglichen soll, Tiefenbilder der Kinect in drei Schritten in Echtzeit zu verarbeiten und die Qualität dieser zu verbessern. Man beschreibt hier, wie man die fehlerhaften Bildpunkte des Tiefenbildes füllen kann, die Kanten verfeinern und das Rauschen, welches in Tiefenbildern entsteht, zu reduzieren.
Im ersten Schritt werden falsche Pixelwerte entfernt, welche meist an Kanten entstehen. Man vergleicht hier die gefundene Kante im Tiefenbild und dieselbe Kante aus dem Farbbild der Szene. Die Bildpunkte zwischen den beiden Kanten werden als falsche Bildpunkte bezeichnet und entfernt. Eine detaillierte Erklärung hierfür erhält man in Kapitel 2.1 des Papers.
Im Kapitel 2.2 des Papers werden die fehlerhaften Bildpunkte auf Basis eines Schätzverfahrens für glatte Regionen verwendet. Hierfür werden die Bildpunkte in unmittelbarer Nähe mit einbezogen. Die restlichen Bildpunkte, die nicht zu glatten Regionen des Tiefenbildes gehören, werden mit Hilfe eines Bilateral Filters gefüllt.
In Kapitel 2.3 wird ein Verfahren aufgezeigt, mit dem das Rauschen im Tiefenbild unterdrückt werden soll, durch einen modifizierter Bilateral Filter. Nähere Erläuterung zu diesem modifizierten Filter werden im Paper dargestellt und erläutert.
Abschließend wird an Tests erläutert, welche Ergebnisse mit dieser Methode ermittelt wurden. Diese Tests tragen sehr gut dazu bei, Tiefenbilder qualitativ zu verbessern. [LHS, 2012]

4.2 Hochqualitative Tiefenkarten Schätzung durch Kinect upsampling und Lochfüllung durch die Benutzung des RGB-Bildes

In [NCSC, 2013] wird ein Verfahren beschrieben, welches eine Methode untersucht, die sich mit dem Befüllen fehlerhafter Bildpunkte in Tiefenbildern befasst. Dieses Befüllen wird mit Hilfe des RGB Bildes der Kinect vorgenommen.
Nidhi Chahal und Santanu Chaudhury haben einen eigenen Algorithmus geschrieben, der auf der Basis der RGB-Farbanteile agiert, um den fehlerhaften Bildpunkt im Tiefenbild zu füllen. Nähere Informationen zu diesem Algorithmus findet man in ihrem Paper in Kapitel 4. Abschließend verwenden sie noch einen Median Filter mit einer 9x9 Maske, um das gefüllte Tiefenbild zusätzlich zu verfeinern und qualitativ zu verbessern. [NCSC, 2013]

4.3 Texturunterstütztes Kinect Tiefen Inpainting

Ziel hierbei ist es, das Bild einer eigenen Inpainting Methode zu unterziehen, um Bereiche wiederherzustellen und das Bild zu entrauschen. Dan Miao, Jingjing Fu und Yan Lu haben ein Textur basiertes Inpaiting-Schema entwickelt, mit dem sie die Qualität von Tiefenbildern erhöhen wollen. Im ersten Schritt wird mittels eines Bilateral Filters das Tiefenbild entrauscht. Im weiteren Vorbereitungsschritt werden nun die Blöcke extrahiert, damit ein Inpainting möglich ist. Diese gefundenen Blöcke werden in 2 Blockarten unterteilt, dem glatten Bereich-Block und dem Rand-Bereich-Block. Diese beiden Blöcke werden nun unterschiedlich verarbeitet, um diese entsprechend zu befüllen. Eine ausführliche Erklärung zu diesen beiden Verfahren findet man im Paper unter Kapitel 4. [DJY, 2012]

4.4 Eine Methode zur Lochfüllung für die Tiefenkarte der Kinect mit sich bewegenden Objekterkennung

Xu Kang, Zhou Jun und Wang Zhen [KJZ, 2012] befassen sich in diesem Paper, mit dem Befüllen von Rändern in Körperbereichen. Hierzu wird eine 4-Nachbar-Pixel-Interpolation verwendet. Dies soll an sich bewegenden Objekten vorgenommen werden. Im einzelnen sollen diese, sich bewegenden Objekte, detektiert und mit Hilfe eines Algorithmus befüllt werden. Durch diesen Algorithmus können 3D Bilder qualitativ verbessert werden. Genauere Informationen zu dem Algorithmus findet man in dem Paper in Kapitel 3.

4.5 Kinect Rauschunterdrückung

Das Paper von Jingjing Fu, Shiqi Wang und Yan Lu befasst sich mit der Rauschunterdrückung in Tiefenbildern auf Basis der Kinect. Zunächst wird das Tiefenbild erklärt, deren Entstehung sowie deren räumliche und zeitliche Charakteristika. Im ersten Schritt werden die fehlerhaften Bildpunkte durch die räumlich benachbarten Tiefenwerte grob aufgefüllt. Hiernach wird ein räumlich-zeitlicher Rauschunterdrückungs-Algorithmus verwendet, um das Rauschverhalten in dem Tiefenbild zu reduzieren. Der räumlich-zeitliche Rauschunterdrückungs-Algorithmus ist der Hauptinhalt dieser Arbeit und wird ausführlich in Kapitel 3 des Papers erklärt. [JSY, 2012]

4.6 Hochauflösendes Tiefenkarten Schätzungssystem durch Stereo Vision

In diesem Paper von Shuai Zhang, Chong Wang und S.C. Chan wird ein Ansatz vorgestellt, bei dem eine Kinect-Kamera und eine hochauflösende Kamera gekoppelt werden und daraus ein qualitativ hochwertiges Tiefenbild erzeugt wird. In dem Paper wird der Versuchsaufbau und die Probleme beim Kombinieren dieser beiden Kameras erläutert. Weiterhin wird ein Ansatz dargestellt, mit dem qualitativ hohe Tiefenbilder erzeugt werden können. [SCSC, 2013]

5 Design der Qualitätsverbesserungsverfahren

In Bild 5.1 wird die Vorgehensweise als Pipeline dargestellt. In dieser Reihenfolge sollen in diesem Kapitel die einzelnen Schritte durchlaufen werden.

Abbildung 5.1: Pipelining in diesem Kapitel

5.1 Kalibrierung der Kinect 360

5.1.1 Problem des Testaufbaus

Um mit dem Tiefenbild, welches wir mit Hilfe der Kinect 360 gewonnen haben, weiter arbeiten zu können, ist ein weiterer Schritt nötig. Das RGB-Bild und das Tiefenbild müssen gemappt werden.

Dies bedeutet, dass das Tiefenbild und das RGB-Bild übereinander geschoben werden müssen, sodass sich die Pixel beider Bilder überdecken. Durch die Parallaxe zwischen den beiden Kameras, sind die Positionen der Pixel nicht übereinstimmend. Dies bedeutet, dass wenn man auf dem Tiefenbild einen Pixel über x-Koordinate und y-Koordinate auswählt, man den zugehörigen RGB Pixel über das RGB Bild nicht finden kann. Dies ist jedoch wichtig, da das RGB-Bild als Referenz zum Tiefenbild genutzt werden soll. Um dieses Problem zu lösen, wurden 2 Ansätze untersucht.

5.1.2 Berechnung der Farbpixel mit Hilfe von Tiefenpixel

Dieser Ansatz wird von Nicolas Burrus [NIBU] auf seiner Webseite beschrieben. Er stellt hier eine Reihe von Daten zur Verfügung, die nach einem Kalibrierungsansatz gewonnen wurden.

In diesem Teil wird kurz die Berechnung dieses Verfahrens erläutert. Wie diese Daten mit der Kamera gewonnen wurden, wird kurz auf seiner Webseite dargestellt. Um detaillierte Informationen zu der Gewinnung zu bekommen, verweist er auf eine Webseite, auf der eine spezifischere Darstellung des Problems erläutert wurde und auch ein Code-Beispiel zu finden ist, mit dem man diese Daten gewinnen kann.

Man kann die Tiefenwerte vom Tiefenbild auslesen und dadurch die Position des jeweiligen Tiefenbildpunktes erhalten, so erhält man den jeweiligen Höhenwert und Breitenwert des Bildpunktes. Der Tiefenwert repräsentiert die Entfernung, die an diesem Punkt vorliegt. Die Entfernung wird hier nun in mm angegeben und anschließend in Metern umgerechnet. Nun kann man mit Hilfe der Position und dem Tiefenwert die Position des gewünschten Punktes im 3D-Raum berechnen. Hierzu verwendet man noch die jeweiligen Konstanten, die man aus dem Kalibrierungsansatz erhält, um den Versatz der beiden Kameras mit einzuberechnen.

$$\mathrm{P3D.x} = (\mathrm{x_d} - \mathrm{cx_d}) * \mathrm{depth}(\mathrm{x_d}, \mathrm{y_d})/\mathrm{fx_d}$$
$$\mathrm{P3D.y} = (\mathrm{y_d} - \mathrm{cy_d}) * \mathrm{depth}(\mathrm{x_d}, \mathrm{y_d})/\mathrm{fy_d}$$
$$\mathrm{P3D.z} = \mathrm{depth}(\mathrm{x_d}, \mathrm{y_d})$$

P3D repräsentiert hier den 3D-Punkt im Raum. Die Werte x_d, y_d und depth(x_d, y_d) stehen für den x-, y- und Tiefenwert, den man aus dem Tiefenbild erhält. Die Werte cx_d, cy_d, fx_d und fy_d stehen für die Konstanten, die aus dem vorherig erwähnten Kalibrierungsansatz heraus genommen wurden. Wenn man diese Rechnung durchgeführt hat, erhält man den gesamten 3D-Punkt im Raum.
Nachdem man den 3D-Punkt im Raum berechnet hat, multipliziert man diesen Punkt mit der Rotationsmatrix R und addieren die Transformationsmatrix T zu den daraus resultierenden Werten.

$$\mathrm{P3D'} = \mathrm{R.P3D} + \mathrm{T}$$

Mit diesem Punkt kann man nun den jeweiligen Farbwert, der zu dem Tiefenwert gehört, auf dem RGB-Bild errechnen.

$$\mathrm{P2D_rgb.x} = (\mathrm{P3D'.x} * \mathrm{fx_rgb}/\mathrm{P3D'.z}) + \mathrm{cx_rgb}$$
$$\mathrm{P2D_rgb.y} = (\mathrm{P3D'.y} * \mathrm{fy_rgb}/\mathrm{P3D'.z}) + \mathrm{cy_rgb}$$

Mit dieser Formel erhält man die Position auf dem RGB-Bild mit der entsprechenden Farbe, die zu diesem Tiefenwert gehört. Dies muss mit jedem einzelnen Punkt durchführen.

5.1.3 Mapping mithilfe einer Softwarelösung

Im Ansatz 2 wird eine Bibliothek verwendet, die sowohl im Microsofts Software Paket für die Kinect enthalten ist, als auch in der Open Source Version von OpenNi. Verwendet wird hier eine Klasse AlternativeViewpointCapability, die dazu dient, die Bildausgabe umzuwandeln, die durch eine simulierte Veränderung der Kameraposition hervorgerufen wird. Dabei werden zwei Kameras, die aus unterschiedlichen Positionen auf die Szene schauen, wie es bei der Kinect der Fall ist, verwendet um diese Simulation durchzuführen. Diese Bilddaten werden intern zu einer Bildausgabe zusammengefügt und somit mappt einem die Software automatisch das Tiefenbild mit dem RGB-Bild. Das Tiefenbild wird nun stark beschnitten, da durch den Versatz der Kameras nicht alle Tiefenwerte automatisch einen Wert im RGB-Bild besitzen.

5.1.4 Auswertung

Nachdem beide Verfahren verfolgt wurden, wurde festgestellt das bei dem Verfahren mit der Softwarelösung, die brauchbarsten Ergebnisse entstanden sind. Verfahren 5.1.2 lieferte unbrauchbare Ergebnisse. Aus Zeitgründen wurde dieser Ansatz nicht weiter verfolgt. Für den weiteren Verlauf dieser Arbeit werden die Ergebnisse des Verfahrens 5.1.3 der Softwarelösung genutzt.

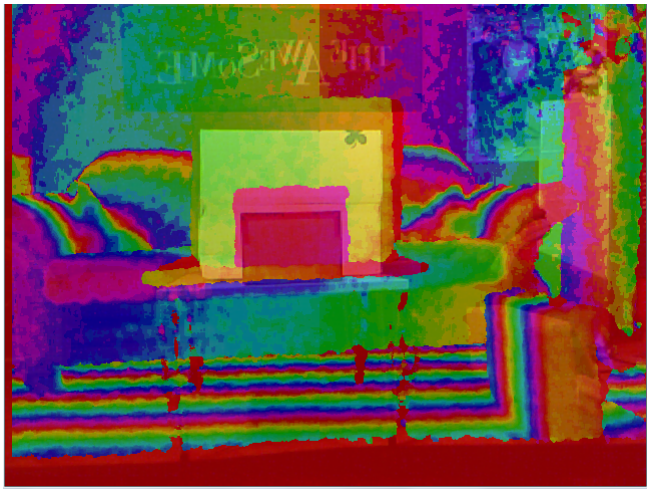


Abbildung 5.2: Unkalibriertes Bild der Kinect

Bild 5.2 zeigt eine Szene eines RGB-Bildes und eines Tiefenbildes. Das Tiefenbild wurde hier farbig dargestellt, um die Unterschiede zu verdeutlichen. Wenn man sich die Mitte des Bildes

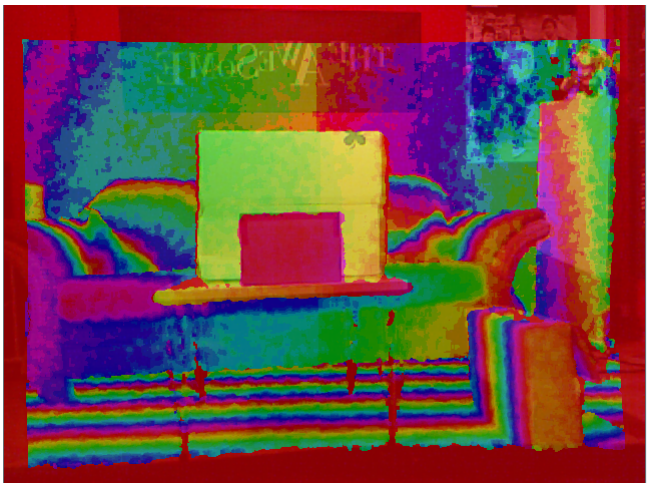


Abbildung 5.3: Kalibriertes Bild der Kinect

anschaut, sieht man zwei Objekte, die hintereinander stehen. Das vordere Objekt ist kleiner als das hintere. Man kann in dem Bild 5.2 gut sehen, dass Tiefenwerte des kleinen Objekts nicht zu den Kanten des Objekts auf dem RGB-Bild passen. Im Tiefenbild ist das kleinere Objekt höher dargestellt als im RGB-Bild. In Bild 5.3 ist das Bild zu sehen, nachdem man mit der Softwarelösung die Bilder bearbeitet hat. Hier ist deutlich zu erkennen, dass nun die Kanten der beiden Bilder übereinstimmen und man so die richtigen Tiefenwerte an den richtigen Positionen vorfindet.

5.2 Ansätze zur Problembehandlung von Rändern

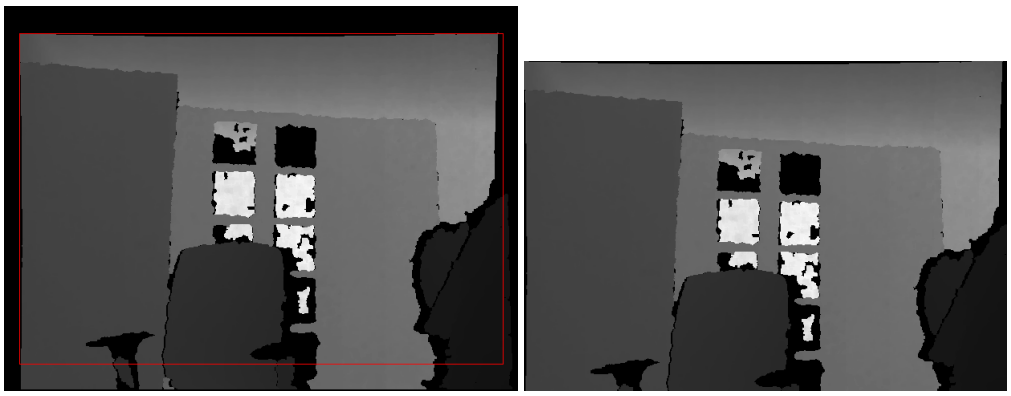
In dem Testaufbau, der hier verwendet wurde, zeigt sich sofort ein Problem. Das Bild besitzt Ränder, in denen der Tiefenwert 0 ist. Diese Ränder erhöhen den Rechenaufwand, wenn man das Bild weiterverarbeiten möchte. Somit wurde sich hier kurz mit 4 Varianten auseinandergesetzt, die den Rand wegschneiden. Man kann in den Tiefenbildern erkennen das die Ränder nicht gerade sind, was die Beschneidung erschwert. Dies bedeutet, dass man sich einen Ansatz überlegen muss, der zum einen die Informationen wegschneidet die nicht relevant sind und zum anderen die Informationen erhält, die sich im inneren des Bildes befinden.

Dies soll an dem folgendem Tiefenbild-Beispiel verdeutlicht werden.

Abbildung 5.4: Tiefenbild-Beispiel nach der Kalibrierung und vor der Verarbeitung

5.2.1 Erster Versuch zur Randbehandlung

Der erste Versuch ist der einfachste. Hier geht man davon aus, dass die Ränder um das Bild gleich groß sind. Der Ansatz sucht den ersten Bildpunkt mit einem Tiefenwert der größer als 0 ist. Wenn man den Bildpunkt gefunden hat, erhält man durch seine Position den jeweiligen Höhen- und Breitenwert. Man verwendet nun die Koordinaten als Startwerte beim Iterieren über das Bild, als Startwert für den Höhenwert und als Startwert für den Breitenwert. Als Abbruchbedingung in den Schleifen werden die Max Werte des Bildes (Höhenwert und Breitenwert) noch vorverarbeitet. Man zieht vom Max-Höhenwert den Höhenwert des Bildpunktes ab, den man gefunden hat und vom Max-Breitenwert den Breitenwert des Pixels. So erhält man einen gleichmäßigen Schnitt der Ränder im Bild. In der Algorithmus Abbildung [1] wird der Ansatz als Pseudocode dargestellt.



(a) Tiefenbild mit Beschneidungsrahmen (b) Tiefenbild beschnitten

Abbildung 5.5: Erster Versuch zur Randbehandlung

Algorithm 1 Erster Versuch zur Randbehandlung

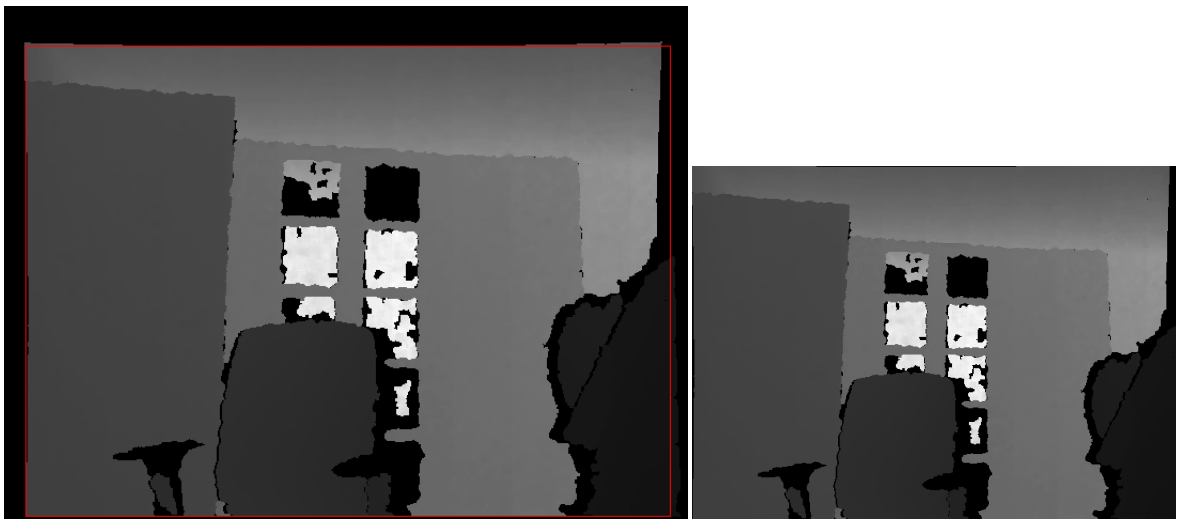
```
1: XAnfang = 0 // Anfangs Koordinate
2: YAnfang = 0
3: XEnd = 0
4: YEnd = 0
5: foundEdge = false // um ersten Punkt zu finden
6: for Y : alleZeilen do
7:    if foundEdge then
8:       BREAK
9:    end if
10:   for X : alleSpalten do
11:      if Tiefenwert(X, Y)! = 0 ∧ X < Imagewidth/4 then
12:         XAnfang = X
13:         YAnfang = Y
14:         foundEdge = true
15:         BREAK
16:      end if
17:   end for
18: end for
19: XEnd = Imagewidth-XAnfang
20: YEnd = ImageHeight-YAnfang
```

5.2.2 Zweiter Versuch zur Randbehandlung

Im zweiten Versuch zur Randbehandlung, werden die vier Eckpunkte im Bild gesucht, in denen kein Rand zu erkennen ist. Die Suche nach diesen Punkten erfolgt Zeilenweise. Es werden nur Zeilen verwendet, in denen die Anzahl der Bildpunkte der Hälfte der gesamten Punkte dieser Zeile entsprechen. Hat man die erste Zeile gefunden, wird nun noch das Ende dieser Zeile gesucht. Dies bedeutet, dass man einen Höhenwert und zwei Breitenwerte (Startpunkt und Endpunkt) erhält. Hat man die oberste Zeile gefunden, beginnt die selbes Suche vom unteren Rand Richtung oberen Rand des Bildes, um die letzte Zeile zu finden. Die Suchkriterien sind exakt dieselben wie beim Suchen der obersten Zeile. Somit erhält man am Ende dieses Ansatzes zwei Höhenwerte (Anfang des Bildes und Ende des Bildes) und vier Breitenwerte (Anfang und Ende der ersten Zeile, sowie Anfang und Ende der zweiten Zeile). Es folgt ein weiteres Problem bei dieser Betrachtung. Die Anfänge können variieren. Beispielsweise könnte die erste Zeile bei Pixel Nummer 5 anfangen und die letzte Zeile des Bildes bei Pixel Nummer 10. Das selbe Problem kann auch am Ende der jeweiligen Zeile entstehen. Um dem Abhilfe zu schaffen, addiert man die beiden Werte des Anfangswertes und

dividiert ihn durch 2. Anschließend runden man das Ergebnis. Somit erhält man den gemittelten Startpunkt und Endpunkt einer Zeile.

Um nun das Bild zu beschneiden, nutzt man bei der Höheniterartionsschleife als Startwert den Höhenwert der ersten gefundenen Zeile und als Abbruchbedingung den Höhenwert der letzten gefundenen Zeile. Um über die Breite zu iterieren, verwendet man den ermittelten Startwert als Startwert der Iteration und den gemittelten Endwert der beiden Zeilen als Abbruchbedingung der Iteration. In der Algorithmus Abbildung [2] wird der Ansatz als Pseudocode dargestellt.



(a) Tiefenbild mit Beschneidungsrahmen (b) Tiefenbild beschnitten

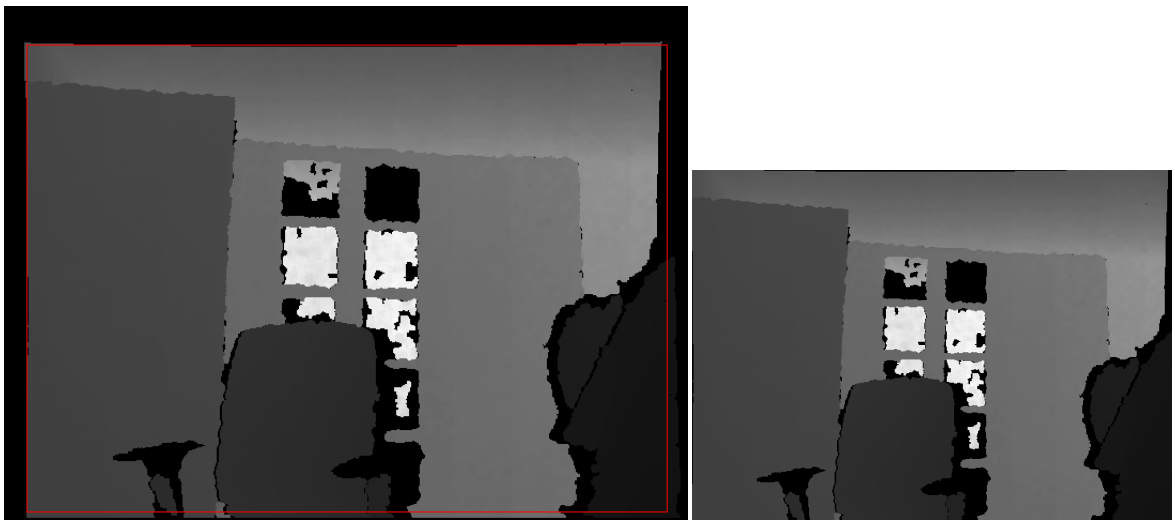
Abbildung 5.6: Zweiter Versuch zur Randbehandlung

Algorithm 2 Zweiter Versuch zur Randbehandlung

```
1: XAnfang = 0 // Anfangs Koordinate
2: YAnfang = 0
3: XEnd = 0
4: AnzPixel = 0 // Anzahl der Pixel die einen Tiefenwert besitzen in einer Zeile
5: YFirstLine = 0
6: YEndLine = 0
7: XLastStartLine = 0
8: XLastEndLine = 0
9: foundFirstLine = false
10: searchFirstX = true // in der Zeile das erste X suchen
11: for Y: Alle Zeile vom oberen Rand do
12:     if foundFirstLine then
13:         BREAK
14:     end if
15:     for X: Alle Spalten do
16:         if Tiefenwert(X, Y)! = 0 then
17:             if searchFirstX then
18:                 XFirstStartLine = X
19:                 searchFirstX = false
20:             end if
21:             AnzPixel++
22:         end if
23:     end for
24:     if AnzPixel > ImageWidth/2∧!foundFirstLine then
25:         YAnfang = Y
26:         XFirstEndLine = findEndLine()
27:         AnzPixel = 0
28:         searchFirstX = true
29:         foundFirstLine = true
30:     else
31:         AnzPixel = 0
32:         searchPixelX = true
33:     end if
34: end for
35: Äquivalent hierzu das selbe mit vom unteren Rand des Bildes für die letzte Zeile für YEnd
36: if XFirstStartLine != XLastStartLine then
37:     XAnfang = (XFirstLineStart + XLastStartLine)/2
38: else
39:     XAnfang = XFirstStartLine
40: end if
41: Äquivalent hierzu die Berechnung für die untere Zeile um XEnd zu berechnen
```

5.2.3 Dritter Versuch zur Randbehandlung

Der dritte Versuch der Problembehandlung der Ränder in Tiefenbildern, verfährt ähnlich wie der zweite Versuch. Hier werden jedoch nicht die erste und die letzte Zeile des Bildes gesucht, in denen Tiefenwerte enthalten sind, sondern man iteriert über alle Zeilen des Tiefenbildes. Wie schon erwähnt, wird in diesem Ansatz über das gesamte Tiefenbild iteriert. Dies geschieht diesmal Zeilenweise und Spaltenweise. Es werden nur die Zeilen bzw. Spalten verwendet, in denen der Anteil von Bildpunkten, die einen Tiefenwert besitzen, größer sind als die Hälfte der gesamten Breite bzw. Höhe des Tiefenbildes. Es wird von jeder Zeile bzw. Spalte, die gefunden wurde, die Anfangs- und Endposition ermittelt und zwischengespeichert. Nun werden alle Breitenanfangswerte gemittelt und durch die Anzahl der gefundenen Breitenanfangswerte dividiert. So erhält man den Durchschnittswert für alle Breitenanfangswerte. Dasselbe wird nun noch mit den Breitenendwerten vollzogen, sowie mit den Zeilenanfangs- und Zeilenendpositionen. Nun erhält man vier gemittelte Werte, die man als Startbedingung der Iterationen verwenden kann (Zeilen- und Spaltenanfangswert) und die Abbruchbedingungen dieser Iterationen durch die Werte der Zeilen- und Spaltenendwert gesetzt. In der Algorithmus Abbildung [3] wird der Ansatz als Pseudocode dargestellt.



(a) Tiefenbild mit Beschneidungsrahmen (b) Tiefenbild beschnitten

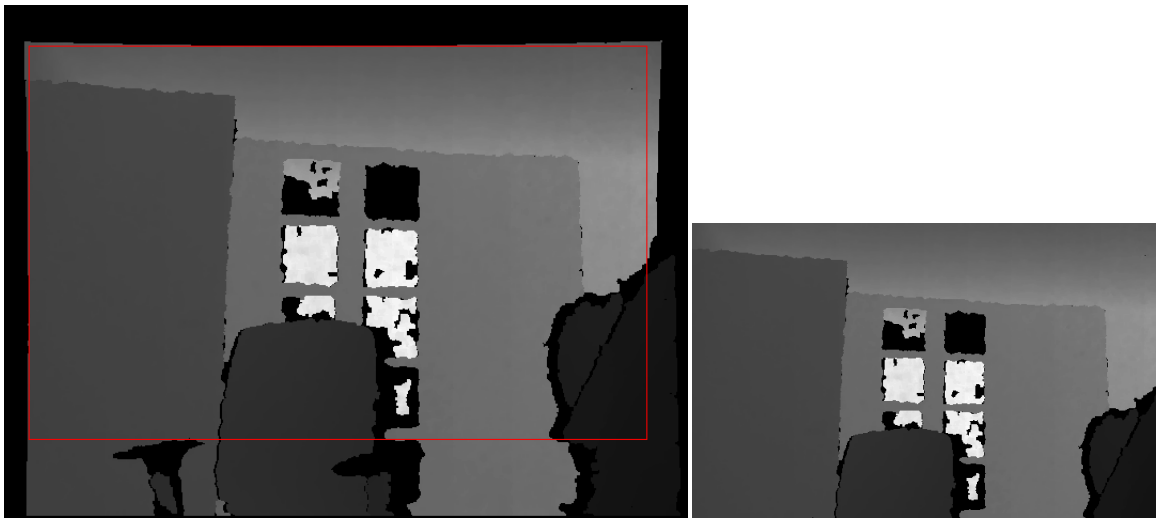
Abbildung 5.7: Dritter Versuch zur Randbehandlung

Algorithm 3 Dritter Versuch zur Randbehandlung

```
1: XAnfang = 0 // Anfangs Koordinate
2: YAnfang = 0
3: XEnd = 0
4: YEnd = 0
5: coordsX [ImageHeight][2] = findAllX() // [][0] => XAnfang [][1] => XEnd
6: coordsY [ImageWidth][2] = findAllY()// [][0] => YAnfang [][1] => YEnd
7: countAnfang = 0
8: countEnd = 0
9: count = 0
10: for i über coordsX[i][] do
11:    if coordsX[i][0]! = 0 ∧ coordsX[i][1]! = 0 then
12:       countAnfang = countAnfang + coordsX[i][0]
13:       countEnd = countEnd + coordsX[i][1]
14:       count++
15:    end if
16: end for
17: XAnfang = countAnfang/count
18: XEnd = countEnd/ count
19: äquivalent hierzu dasselbe mit den Koordinaten aus coordsY um YAnfang und YEnd zu
    berechnen
```

5.2.4 Vierter Versuch zur Randbehandlung

In diesem Versuch wird ebenfalls jede Zeile und Spalte, welche unseren Voraussetzungen entsprechen, gesucht. Man speichert hier ebenfalls die Zeilenanfangskoordinate und die Zeilenabschlusskoordinate. Wenn man dies für alle Zeilen getan hat, nimmt man jede Seite des Bildes und vergleicht die Koordinaten miteinander, also die linke, rechte, obere und untere Seite des Bildes. Man vergleicht die jeweiligen Werte der Seite miteinander und sucht den Punkt, der am weitesten weg ist vom Bildrand. Nun haben wir 4 Punkte, je ein Punkt der jeweiligen Seite, der am weitesten vom Rand entfernt ist. Diese Punkte nutzt man als Start- und Abbruchbedingungen unserer Iterationen und beschneidet somit das Bild. In der Algorithmus Abbildung [4] wird der Ansatz als Pseudocode dargestellt.



(a) Tiefenbild mit Beschneidungsrahmen (b) Tiefenbild beschnitten

Abbildung 5.8: Vierter Versuch zur Randbehandlung

Algorithm 4 Vierter Versuch zur Randbehandlung

```
1:  XAnfang = 0 // Anfangs Koordinate
2:  YAnfang = 0
3:  XEnd = 0
4:  YEnd = 0
5:  coordsX [ImageHeight][2] = findAllX() // [][0] => XAnfang [][1] => XEnd
6:  coordsY [ImageWidth][2] = findAllY()// [][0] => YAnfang [][1] => YEnd
7:  for i über coordsX[i][] do
8:      if XAnfang < coordsX[i][0] then
9:          XAnfang = coordsX[i][0]
10:     end if
11:     if XEnd < coordsX[i][1] then
12:         XEnd = coordsX[i][1]
13:     end if
14: end for
15: for i über coordsY[i][] do
16:     if YAnfang < coordsY[i][0] then
17:         YAnfang = coordsY[i][0]
18:     end if
19:     if YEnd < coordsY[i][1] then
20:         YEnd = coordsY[i][1]
21:     end if
22: end for
```

5.3 Filling

In den Bildern unseres Versuchsaufbaus kann man deutlich einige Stellen erkennen, die keinen exakten Wert enthalten. Sie besitzen den Tiefenwert 0 und sind als schwarze Punkte in dem Tiefenbild zu erkennen. Diese fehlerhaften Bildpunkte sind auf den Versuchsaufbau zurückzuführen. Da der IR-Projektor und der IR-Sensor 7,5cm voneinander entfernt sind, kommt es zu einer Schattenbildung. Für diese Schatten kann kein direkter Tiefenwert errechnet werden. Daher werden sie als 0 Value dargestellt und bilden Lücken im Tiefenbild.

Zum Thema Filling werden zwei Ideen vorgestellt, in den versucht wird, diese Lücken zu schließen. Die Suche nach fehlerhaften Bildpunkten und der Aufruf der Filling Methoden wird in Algorithmus Abbildung [5] dargestellt.

Algorithm 5 Filling Start Pseudocode

```
1: for Y:Alle Zeilen do
2:     for X: Alle Spalten do
3:         if Tiefenwert(X,Y) == 0 then
4:             DepthFilling(X,Y)
5:             GrayFilling(X,Y)
6:         end if
7:     end for
8: end for
```

5.3.1 Depth Filling

Diese Variante des Filling beschäftigt sich damit, die einzelnen fehlerhaften Bildpunkte, die in unserem Tiefenbild existieren, zu befüllen. Hierbei werden nur die Informationen aus dem Tiefenbild genutzt. Die Idee dahinter ist, diese Bildpunkte anhand eines Durchschnittswertes zu befüllen. Der Durchschnittswert soll aus Punkten bestehen die in einer Zeile zu dem gefundenen fehlerhaften Bildpunkten bestehen. Hier kann man nun sehr variieren. Einerseits schaut man sich an, wie sich das Bild verändert, wenn man nur eine kleine Anzahl an Bildpunkten der Zeile nutzt. Weiterhin wird untersucht, wie die Werte sich verändern, wenn man eine große Anzahl von Bildpunkten nutzt, um die fehlerhaften Bildpunkte zu befüllen.

Algorithm 6 Depth-Filling Pseudocode

```
1:  AnzPixel = 5 // Anzahl der Pixel die zur Durchschnittsberechnung verwendet werden
2:  Value = 0
3:  count = 0
4:  Ergebnis = 0
5:  for X: von X aus bis X - AnzPixel do
6:      if X > 0 then
7:          if TiefenImage(X, Y)! = 0 then
8:              Value = Value + TiefenImage(X,Y)
9:              count++
10:         end if
11:     end if
12: end for
13: for X: von X aus bis X + AnzPixel do
14:     if X < TiefenImageBreite then
15:         if TiefenImage(X, Y)! = 0 then
16:             Value = Value + TiefenImage(X,Y)
17:             count++
18:         end if
19:     end if
20: end for
21: if count > 0 then
22:     Ergebniss = Value/count
23: end if
```

5.3.2 Grauwert Filling

Im Grauwert-Filling wird untersucht, wie die fehlerhaften Bildpunkte des Tiefenbildes mit Hilfe des Grauwertbildes befüllt werden können. Die Idee ist hierbei, sich fehlerhaften Bildpunkte aus dem Tiefenbild zu suchen. Sobald einer gefunden wird überprüft man, ob es ein einzelner fehlerhafter Bildpunkt ist oder ob es mehrere aufeinanderfolgende fehlerhafte Bildpunkte sind. Sollten es mehrere aufeinanderfolgende Bildpunkte sein, muss man das Ende dieser fehlerhaften aneinandergereiten Bildpunkte bestimmen. Somit erhält man eine Menge von fehlerhaften Bildpunkten.

Nun folgt der Teil, weswegen das aufgenommene Tiefenbild und das aufgenommene RGB-Bild des Versuchsaufbaus gemappt wurden. Somit besitzt man eine Referenz zu den gefundenen fehlerhaften Bildpunkten, da die Koordinaten der beiden Bilder übereinstimmen.

Man besitzt nun einen oder eine Menge von fehlerhaften Bildpunkten und die Referenz in Form des Grauwertbildes, welches aus dem RGB-Bild gewonnen wurde.

Der nächste Schritt ist, die gefundenen fehlerhaften Bildpunkt-Koordinaten zu nehmen und die Grauwerte, die man durch diese erhält, zu vergleichen.

In diesem Versuch wird eine einfache Methode angewendet, um die fehlerhaften Bildpunkte zu befüllen. Da die Koordinaten der fehlerhaften Bildpunkte bekannt sind, wird der Fokus auf den vorderen und hinteren Bereich des fehlerhaften Bildpunktes bzw. der Menge an fehlerhaften Bildpunkten gelegt. Dazu speichert man eine bestimmte Anzahl der Graubild-Bildpunkte in einem Zwischenspeicher. Dies wird für einen bestimmten Bereich vor dem ersten fehlerhaften Bildpunkt und für den selben Bereich hinter dem letzten fehlerhaften Bildpunkt gemacht. Als nächstes schaut man sich die Grauwerte der fehlerhaften Bildpunkte an und vergleicht diese mit den vorderen und hinteren gespeicherten Bildpunkten. Liegt eine Übereinstimmung mit einem passenden Wert vor, wird der fehlerhafte Bildpunkt durch den Tiefenwert des jeweiligen Bereiches ersetzt. Weitere Details hierzu und zu einer Auswertung folgen im nächsten Kapitel Ergebnisse [6].

Algorithm 7 Gray-Filling Pseudocode

```
1: m = 10 // Menge der Punkte die Untersucht werden sollen vor und hinter den fehlerhaften Bildpunkten
2: count = 0
3: startDepth = 0
4: endDepth = 0
5: GrauwertVorPunkt[m] // Grauwerte vor dem fehlerhaften Bildpunkt
6: GrauwertNachPunkt[m]// Grauwerte nach dem fehlerhaften Bildpunkt
7: AnzPixel = findeLochLänge(X,Y) // hier wird die Menge an Fehlerhaften Bildpunkten bestimmt
8: if (X − m) > 0 ∧ (X + m + AnzPixel) < ImageBreite then
9:    startDepth = TiefenImage(X-1,Y)
10:   endDepth = TiefenImage((X + AnzPixel + 1), Y)
11:   while count < m do
12:      GrauwertVorPunkt[count] = ImageGray(X - count,Y)
13:      GrauwertNachPunkt[count] = ImageGray(x + AnzPixel + count)
14:      count++
15:   end while
16: end if
17: for X: Iteriere über X + AnzPixel do
18:   count = 0
19:   while count < m do
20:      if ImageGray(X, Y) == GrauwertVorPunkt[count] then
21:         TiefenImage(X,Y) = startDepth
22:         BREAK
23:      end if
24:      if ImageGray(X, Y) == GrauwertNachPunkt[count] then
25:         TiefenImage(X,Y) = endDepth
26:         BREAK
27:      end ifcount++
28:   end while
29: end for
```

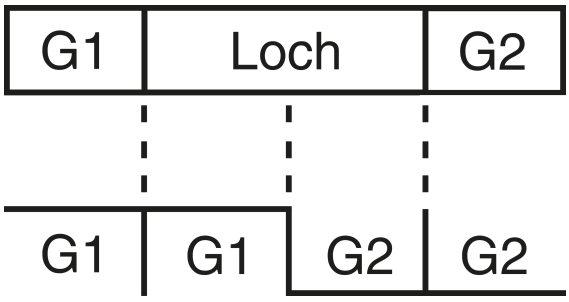


Abbildung 5.9: Skizze zur Idee des Gray-Filling

In Bild 5.9 soll skizziert, die Idee hinter dem Gray-Filling veranschaulicht werden. Dargestellt ist hier ein Ausschnitt aus einer Zeile, in dem Grauwertbild. Hier kann man auf der einen Seite den Grauwert G1 sehen und auf der anderen Seite den Grauwert G2. Zwischen beiden Grauwerten soll eine Menge an fehlerhaften Bildpunkten dargestellt sein. In der Mitte der Menge von fehlerhaften Bildpunkten soll eine Kante existieren. Die Idee hinter diesem Ansatz ist es, die fehlerhaften Bildpunkte mit den Tiefenwerten des jeweiligen Grauwerts bis zu der Kante, in der Menge der fehlerhaften Bildpunkte aufzufüllen.

6 Ergebnisse

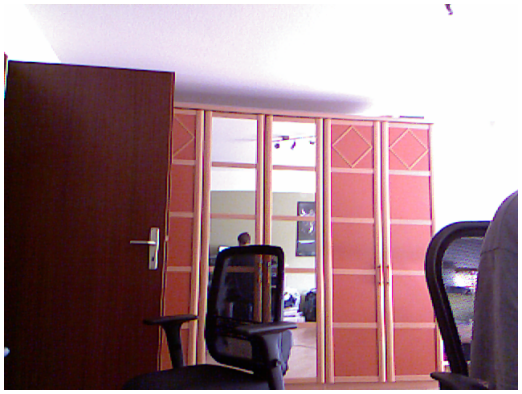
In diesem Kapitel werden die in Kapitel 5 beschriebenen Verfahren, hinsichtlich Ihrer Eigenschaften untersucht und miteinander verglichen.

Die Verfahren werden an drei verschiedenen Beispielen dargestellt. Jedes dieser Beispiele besitzt Besonderheiten, die eine bessere Darstellung der Problemeffekte ermÃ¶glichen.

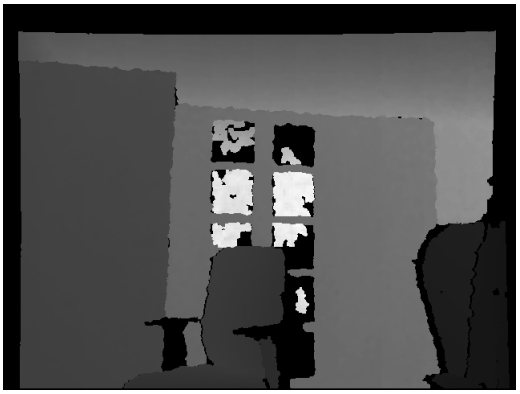
(a) Beispiel 1 Farbbild (b) Beispiel 1 Tiefenbild

Abbildung 6.1: Beispiel 1

Bild 6.1(a) und Bild 6.1(b) zeigen das erste Beispiel, welches in den jeweiligen Tests verwendet wurde. Hier sind zwei einfache Objekte dargestellt. Diese beiden Objekte stehen hintereinander und weisen an ihren Rändern fehlerhafte Bildpunkte auf.



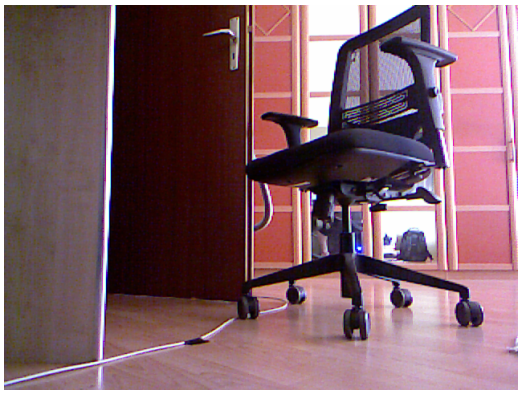
(a) Beispiel 2 Farbbild



(b) Beispiel 2 Tiefenbild

Abbildung 6.2: Beispiel 2

Bild 6.2(a) und Bild 6.2(b) zeigen das zweite Beispiel. Hier findet man verschiedene Fehlereffekte. Zum einen zeigen sich hier große Kanten, wie die Tür und der Schrank. Desweiteren findet man Spiegel, also reflektierende Oberflächen und einen Stuhl, als Objekt in diesem Bild. Darüber hinaus sind verrauschte Kanten in dem Bild zu erkennen.



(a) Beispiel 3 Farbbild

(b) Beispiel 3 Tiefenbild

Abbildung 6.3: Beispiel 3

Bild 6.3(a) und Bild 6.3(b) zeigen das dritte Beispiel. Hier findet man eine Vielzahl von Kanten, sowie einen Stuhl. Der Hauptgrund für die Verwendung dieses Beispiels ist, die hohe Anzahl an fehlerhaften Bildpunkten in diesem Bild.

6.1 Ergebnisse der einzelnen Ränder Ansätze

Man schaut sich nun die Ergebnisse der einzelnen Ansätze an und vergleicht diese unter dem hauptsächlichen Aspekt, wie viele Bildpunkte weggeschnitten wurden. Das Ziel dieser Ansätze ist es, die Ränder der kalibrierten Bilder wegzuschneiden, wobei so wenig wie möglich Bildpunkte vom eigentlichen Tiefenbild weggeschnitten werden sollen. Durch diese Versuche sollen die fehlerhaften Bildpunkte aus dem Tiefenbild von den Rändern getrennt werden.

Um vergleiche daraus ziehen zu können, betrachtet man zunächst drei Tiefenbilder und deren Beschneidung.

6.1.1 Ränder Beispiel 1

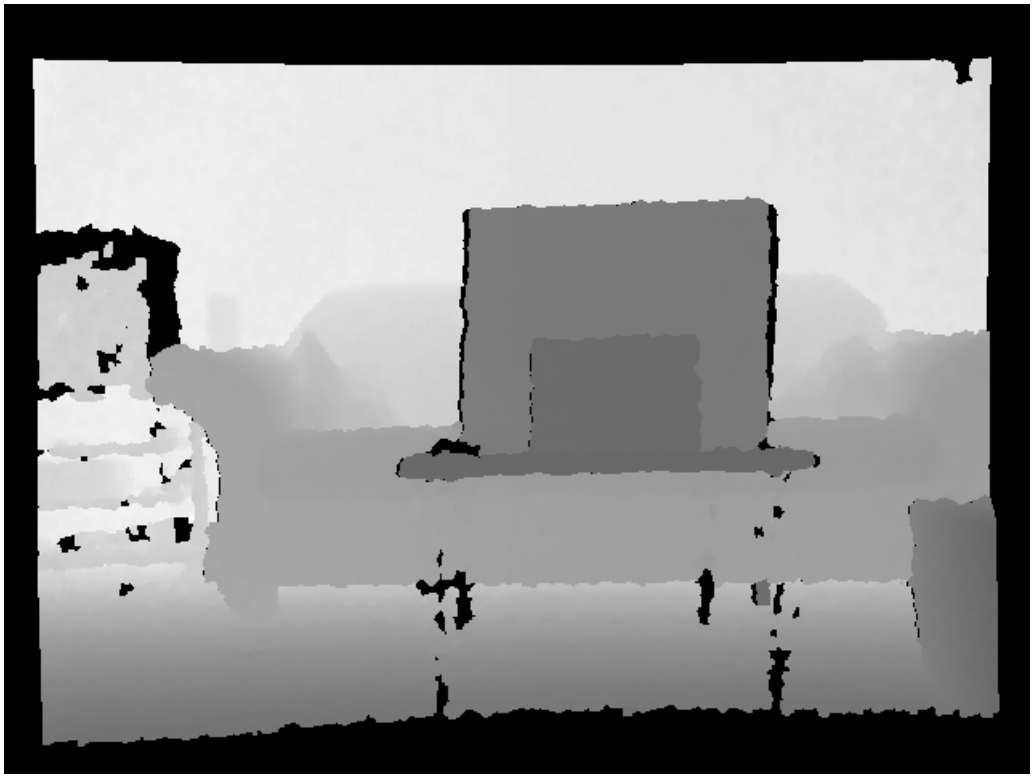
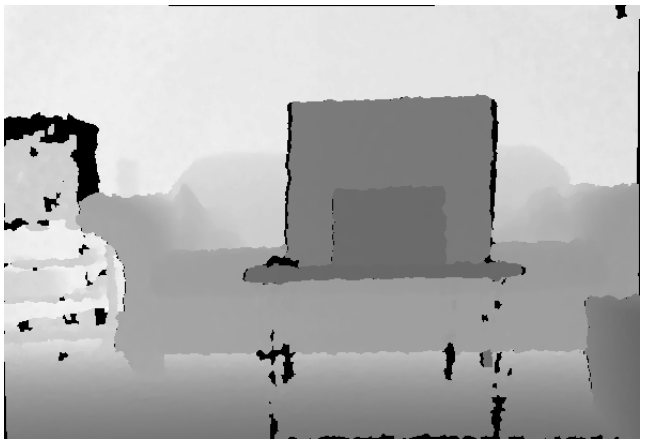
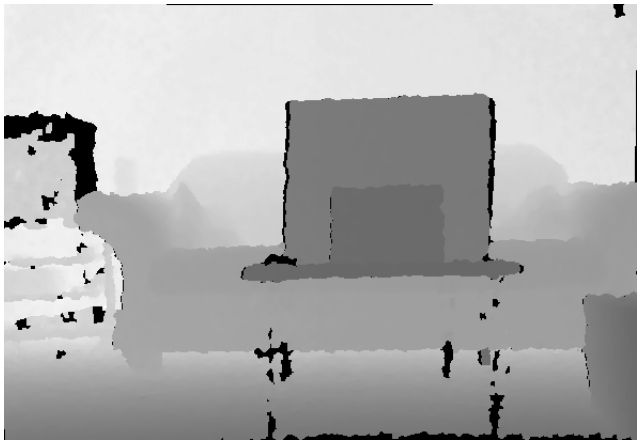


Abbildung 6.4: Beispiel 1 mit der Auflösung 640x480

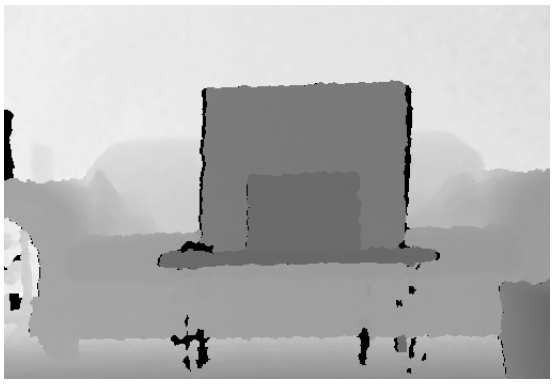
(a) Ansatz 1 (5.2.1) mit der Auflösung 604x412



(b) Ansatz 2 (5.2.2) mit der Auflösung 595x405



(c) Ansatz 3 (5.2.3) mit der Auflösung 592x408



(d) Ansatz 4 (5.2.4) mit der Auflösung 511x350

Abbildung 6.5: Beispiel 1 Analyse der 4 Ansätze

Bild 6.4 ist ein aufgenommenes Tiefenbild mit der Kinect 360. Die Auflösung dieses Bildes ist 640x480. Auf dieses Bild werden die 4 verschiedenen Ansätze angewandt, die vorher in Kapitel 5.2 beschrieben wurden.

Bild 6.5(a) zeigt den beschriebenen Ansatz 1 aus 5.2.1. Die Auflösung dieses Bildes beträgt nach dem Anwenden dieser Methode 604 x 412. Der Verschnitt dieses Bildes beträgt 58352 Bildpunkte. Beim betrachten des Bildes sind Teile von Rändern zu erkennen, die nicht weggeschnitten wurden.

In Bild 6.5(b) findet der Ansatz 2 aus 5.2.2 seine Verwendung. Nachdem dieser Ansatz auf das Bild angewendet wurde, beträgt die Auflösung des Bildes 595x405. Mit diesem Ansatz wurden insgesamt 66225 Bildpunkte vom Rand entfernt. Im Gegensatz zu Bild 6.5(a) befinden sich hier weniger noch verbleibende Ränder.

Bild 6.5(c) zeigt Ansatz 3 aus Kapitel 5.2.3. Hier wurden insgesamt 65664 Bildpunkte weggeschnitten, die Auflösung beträgt noch 592x408. Es wurden im Gegensatz zu Bild 6.5(b) weniger Bildpunkte weggeschnitten, wobei am unteren Rand des Bildes im Detail mehr Rand zusehen ist als im vorhergehendem Bild.

In Bild 6.5(d) wurde der letzte Ansatz aus Kapitel 5.2.4 angewendet. Die Auflösung des Bildes beträgt nur noch 511x350 und nachdem 178850 Bildpunkte weggeschnitten wurden. Dieser Verschnitt und der einhergehende Verlust an Bildpunkten, die man noch hätte nutzen können, ist sehr hoch. Diese Methode zeigt einen sehr aggressive Verschnitt. Durch den keine Ränder mehr sichtbar sind, sondern nur noch die fehlerhaften Bildpunkte des Tiefenbildes.

Zusammenfassend kann man bei diesem Beispiel sagen, dass die Ansätze aus Kapitel 5.2.2 und 5.2.3 sich am meisten bewährt haben. Man kann deutlich sehen, dass der Verschnitt in Bild 6.5(b) und 6.5(c) am wenigsten Rand stehen lässt, während in Bild 6.5(a) noch sehr viel Rand zu sehen ist und in Bild 6.5(d) der gesamte Rand zwar weggeschnitten wurde, somit aber auch die meisten Bildpunkte verloren worden.

6.1.2 Ränder Beispiel 2

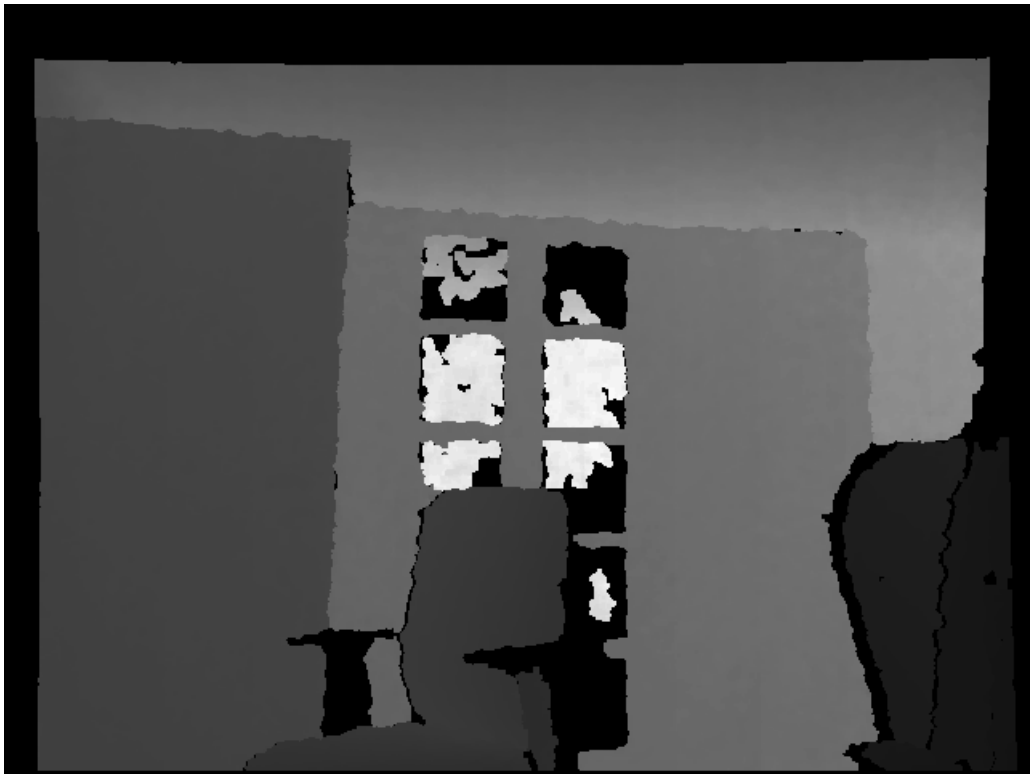
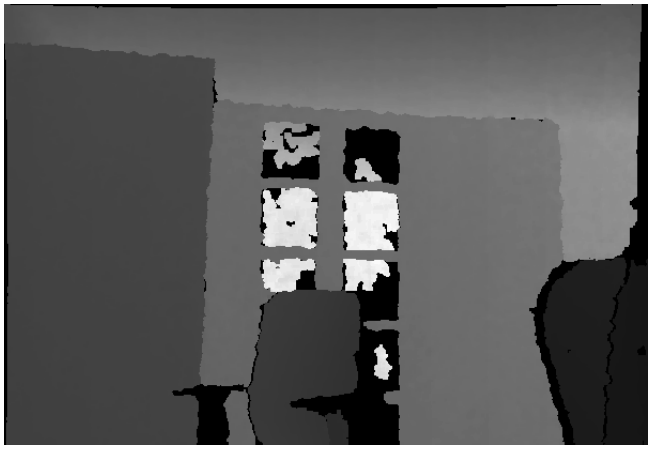
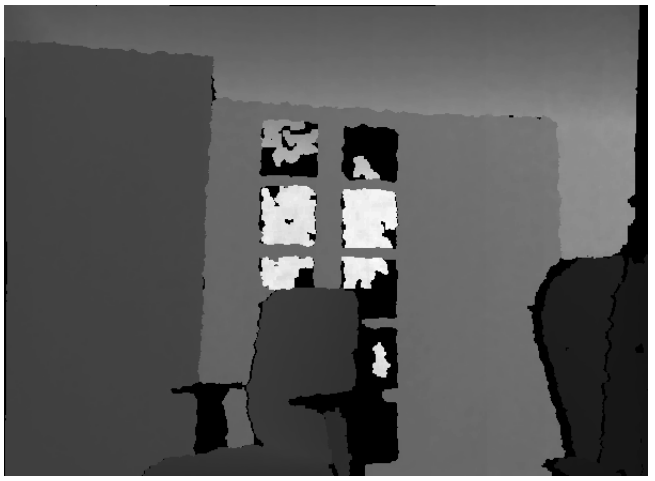


Abbildung 6.6: Beispiel 2 mit der Auflösung 640x480

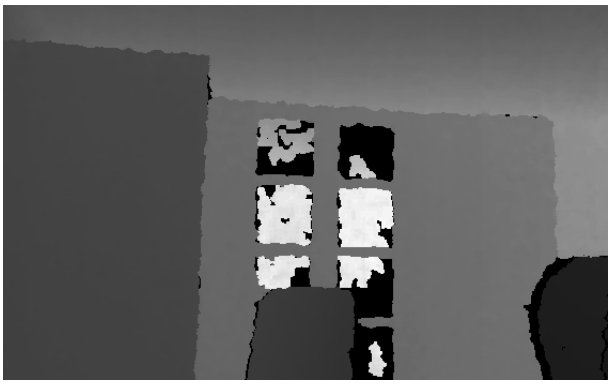


(a) Ansatz 1 (5.2.1) mit der Auflösung 602x412



(b) Ansatz 2 (5.2.2) mit der Auflösung 603x440

(c) Ansatz 3 (5.2.3) mit der Auflösung 596x432



(d) Ansatz 4 (5.2.4) mit der Auflösung 566x350

Abbildung 6.7: Beispiel 2 Analyse der 4 Ansätze

Schaut man sich nun das zweite Beispiel Bild 6.6 an, sieht man die Besonderheit im Gegensatz zum ersten Beispiel, dass nämlich im unteren Bereich des Bildes kein Rand existiert.

Bild 6.7(a) besitzt nach der Beschneidung eine Auflösung von 602x412. Damit wurden in diesem Ansatz 59176 Bildpunkte des Bildes weggeschnitten. Auch hier sind wie im Beispiel 1 6.4 noch teile der Ränder sichtbar. Auch wurden durch die Art des Verfahrens im unteren Bildbereich Bildpunkte weggeschnitten obwohl, hier kein Rand zu erkennen war.

In Bild 6.7(b) hat man nach dem Anwenden des Ansatzes eine Auflösung von 603x440. Somit wurden 41880 Bildpunkte aus dem Bild entfernt. Auch hier ist auf der rechten Seite des Bildes noch ein Rand zu sehen, wobei der Verschnitt wesentlich mehr Punkte erhalten hat als in Bild 6.7(a).

Bild 6.7(c) zeigt ein Bild mit der Auflösung 595x432. Der rechte Rand ist mehr beschnitten als der Rand des Bildes 6.7(b),man sieht jedoch wieder mehr Rand am oberen Rand des Bildes als im vorangegangenem Bild. Es liegt ein Verschnitt von 61392 Bildpunkten vor.

Im letzten Bild 6.7(d) dieses Versuches ist wie in Beispiel 6.5(d) der Verschnitt sehr hoch. Die Auflösung des Bildes beträgt nach dem Anwenden des Ansatzes 566x350. Somit hat man in diesem Bild einen Verlust von 109100 Bildpunkte,also wie im ersten Beispiel den höchsten Verschnitt aller Ansätze. Hier sind ebenfalls keine Ränder mehr zu sehen.

Zusammenfassend zeigen auch hier die Ansätze 5.2.2 und 5.2.3 die besten Ergebnisse. In Bild 6.7(a) wurde zu viel vom unteren Bildrand weggeschnitten und Ansatz 5.2.4 in Bild 6.7(d) ist der Verschnitt sehr hoch.

6.1.3 Ränder Beispiel 3

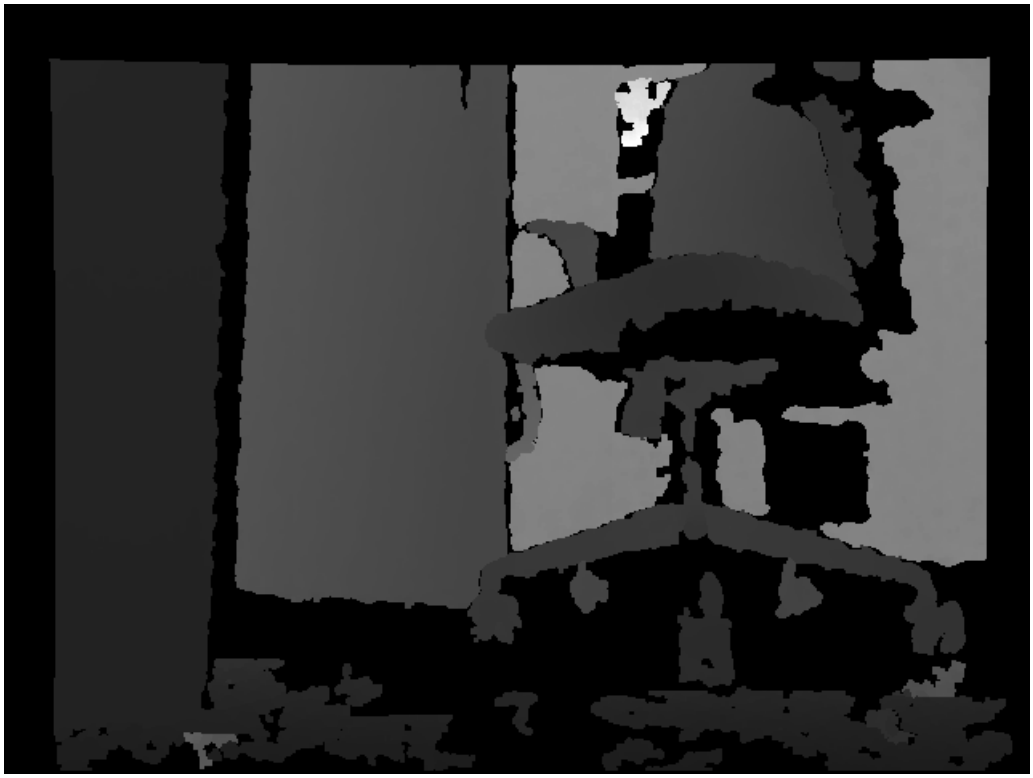
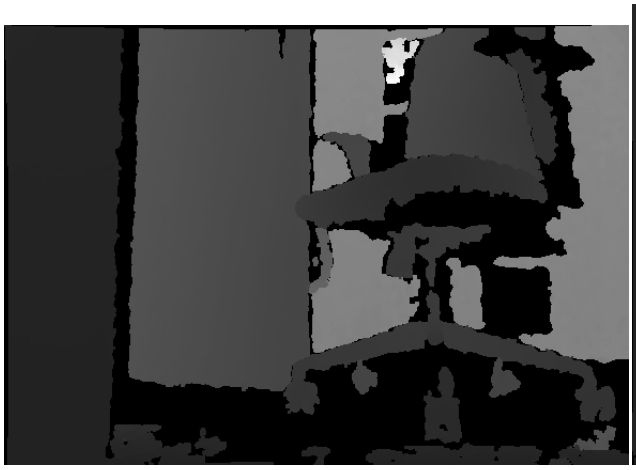


Abbildung 6.8: Beispiel 3 mit der Auflösung 640x480



(a) Ansatz 1 (5.2.1) mit der Auflösung 584x412

(b) Ansatz 2 (5.2.2) mit der Auflösung 587x431

(c) Ansatz 3 (5.2.3) mit der Auflösung 579x404

(d) Ansatz 4 (5.2.4) mit der Auflösung 535x307

Abbildung 6.9: Beispiel 3 Analyse der 4 Ansätze

Schaut man sich das letzte Beispiel der Versuche an 6.8,sind in diesem Bild viele undefinierte Punkte, die den Tiefenwert 0 aufweisen. Daher wurde dieses Beispiel gewählt, um herauszufinden wie der Verschnitt aussieht.

Hier bietet der erste Ansatz 5.2.1 ein Bild 6.9(a) mit der Auflösung von 584x412. Somit hat man einen Verschnitt von 66592 Bldpunkte. Die Ränder sind noch vorhanden.

Der nächste Ansatz 5.2.2 zeigt ein Bild 6.9(b) mit der Auflösung 587x431. Die rechte Seite weist, wie bei den anderen beiden Beispielen, ebenfalls einen Rand auf. Hier liegt der Verschnitt bei 54203 Bildpunkten.

Im Bild 6.9(c) sieht man im Gegensatz zu Bild 6.9(b) keinen rechten Rand mehr. Nach Anwendung dieses Ansatzes hat man eine Auflösung von 579x404. Der Bildverschnitt dieses Bildes beträgt 73284 Bildpunkte.

Im vierten Ansatz 5.2.4 bekommt man eine Bild mit der Auflösung 535x307. Der Verschnitt in diesem Ansatz ist, wie in dem beiden vorhergehenden Beispielen, sehr hoch und beträgt 142955 Bildpunkte. Alle Ränder wurden erfolgreich entfernt.

Auch mit bei einem Bild mit einer hohen Anzahl an fehlerhaften Bildpunkte wird gezeigt, dass die Ansätze 5.2.2 und 5.2.3 die besten Ergebnisse liefern.

6.1.4 Zusammenfassung und Erkenntnisse

Ansatz	Beispiel 1 (6.4)		Beispiel 2 (6.6)		Beispiel 3 (6.8)	
	Auflösung	Verschnitt	Auflösung	Verschnitt	Auflösung	Verschnitt
Ansatz 1 (5.2.1)	604 x 412	58352	602 x 412	59176	584 x 412	66592
Ansatz 2 (5.2.2)	595 x 405	66225	603 x 440	41880	587 x 431	54203
Ansatz 3 (5.2.3)	582 x 408	65664	596 x 432	61392	579 x 404	73284
Ansatz 4 (5.2.4)	511 x 350	178850	566 x 350	109100	535 x 307	142955

Tabelle 6.1: Zusammenfassende Tabelle aller 3 Versuche zur Randerkennung

In Tabelle 6.1.4 sieht man eine Zusammenfassung aller 3 Versuche die im vorherigen Teil durchgeführt wurden.

Hier ist zu sehen, dass Ansatz 5.2.4 den höchsten Verschnitt mit über 100000 Bildpunkten aufweist. In den einzelnen Versuchen hat man gesehen, dass in einer Hinsicht alle Ränder gut weggeschnitten wurden, doch ist auch zu sehen, dass viele Bildpunkte des eigentlichen Bildes verloren gehen. Auch sind die Auflösungen der Bilder geringer als bei allen anderen Ansätzen. Die Idee des Ansatzes verdeutlicht dieses Problem ebenfalls. Wenn beispielsweise eine Spalte bzw. eine Zeile in dem Bild einen Rand aufweist, der in fehlerhafte Bildpunkte im Bild übergeht und somit die Zeile oder Spalte sehr verkürzt, dann unterscheidet dieser Ansatz nicht zwischen Rand und fehlerhafte Bildpunkte.So kann es passieren, dass das Bild stark beschnitten wird, da dieser Ansatz die Bildpunkt vom Rand sucht, der am weitesten Entfernt ist.

Ansatz 3 5.2.3 besitzt den zweitgrößten Verschnitt der Versuche. Anhand der Bilder ist jedoch zu sehen, dass dieser Ansatz die Ränder gut weggeschnitten hat und dass nur ein geringer Teil der eigentlich wichtigen Bilddaten weggeschnitten wurde.

Der zweite Ansatz 5.2.2 lieferte bei Beispiel 2 und Beispiel 3 den geringsten Verschnitt und bei Beispiel 1 den zweitgeringsten Verschnitt. Meist war auf der rechten Seite noch etwas Rand zu sehen, die restlichen Ränder wurden jedoch gut weggeschnitten.

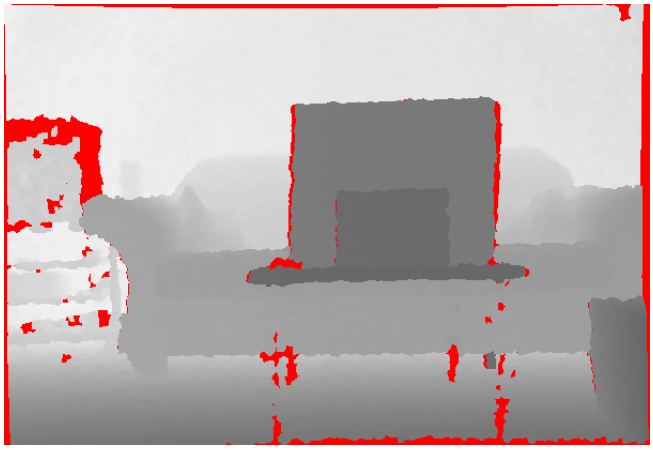
Der erste Ansatz 5.2.1 lieferte bei Beispiel 1 den geringsten Verschnitt. Doch wurden in allen 3 Beispielen die Ränder nur schlecht weggeschnitten und somit der Rand immer noch zu einem gewissen Anteil existierte. Weiterhin habt man bei Beispiel 2 6.6 gesehen, dass das Bild keinen unteren Rand besitzt. Dieser Ansatz 5.2.1 schneidet aber auch am unteren Rand, obwohl kein Rand besteht, einen Teil weg, was nicht gewollt ist.

Abschließend kann man sagen, dass Ansatz 2 5.2.2 und Ansatz 3 5.2.3 die optisch besten Ergebnisse liefern. Der Verschnitt von Ansatz 3 ist zwar der dritthöchste, aber man sieht anhand der Bilder auch, dass der Unterschied zwischen diesen beiden Ansätzen sehr gering ist. Ansatz 4 5.2.4 weist immer zu viel Verschnitt auf und in Ansatz 1 5.2.1 befinden sich noch zuviele Randpixel im Bild.

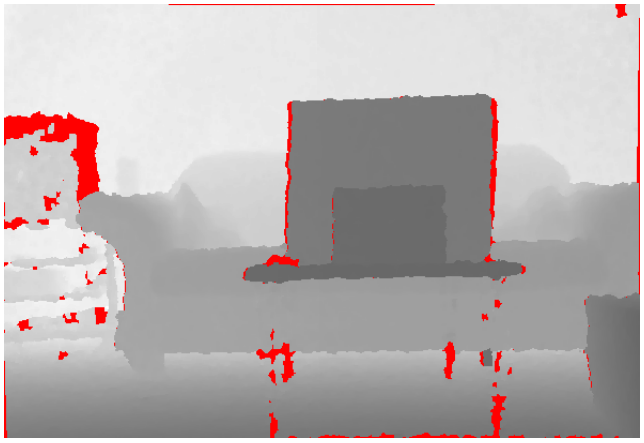
6.2 Filling Tests

Um näher auf diesen Versuch eingehen zu können, soll zunÃ¤chst gezeigt werden, um welchen Fehlereffekt es sich in diesem Versuch handelt. Wie bei dem vorangegangenen Test zu sehen war, weisen Tiefenbilder Ränder und fehlerhafte Bildpunkte auf. In dem Test wurden die 4 Ansätze untersucht, mit denen die Ränder des Bildes weggeschnitten wurden. Ziel war es, dass die fehlerhaften Bildpunkte im eigentlichen Bild erhalten bleiben.
Um nun den Unterschied noch einmal darzustellen, wurden in den folgenden Bildern die Werte hervorgehoben, die einen Tiefenwert Wert von 0 aufweisen. Hierfür wird das erste Beispiel 6.4 aus dem Ränderversuch verwendet.

Abbildung 6.10: Fehlerhafte Bildpunkte

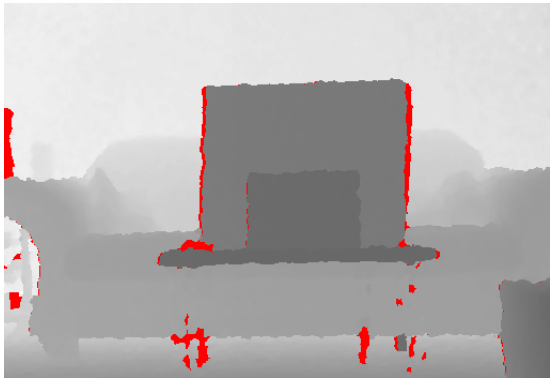


(a) Ansatz 1 (5.2.1) fehlerhafte Bildpunkte



(b) Ansatz 2 (5.2.2) fehlerhafte Bildpunkte

(c) Ansatz 3 (5.2.3) fehlerhafte Bildpunkte



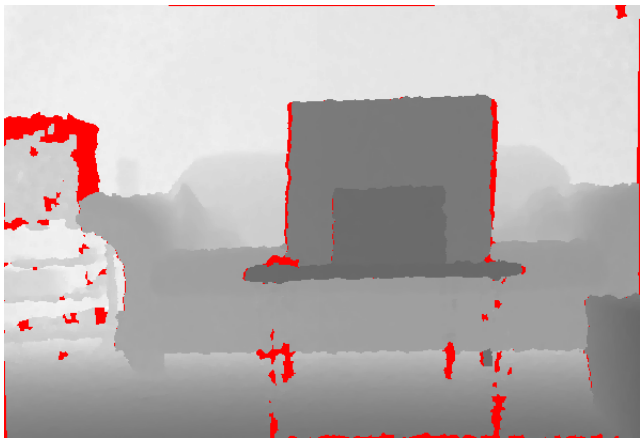
(d) Ansatz 4 (5.2.4) fehlerhafte Bildpunkte

Abbildung 6.11: Darstellung des Problems der fehlerhaften Bildpunkte im Bild

Das Bild 6.10 zeigt das aufgenommene Tiefenbild mit Hilfe der Kinect-Kamera. In dem Tiefenbild wurden alle Werte, die den Wert 0 haben rot hervorgehoben. Hier sieht man deutlich die Ränder und die Anteile der fehlerhaften Bildpunkte im Bild. Nachdem die Bilder beschnitten wurden, sind in den Bildern 6.11(a) , 6.11(b) , 6.11(c) und 6.11(d) die fehlerhaften Bildpunkte markiert worden, die nach dem Beschnitt des Bildes noch vorhanden sind. Die Punkte, die hier zu sehen sind, enthalten alle keinen Tiefenwert. Diese fehlerhaften Bildpunkte gilt es nun zu befüllen. Hierzu schaut man sich die beiden beschriebenen Methoden 5.3.1 und 5.3.2 an.

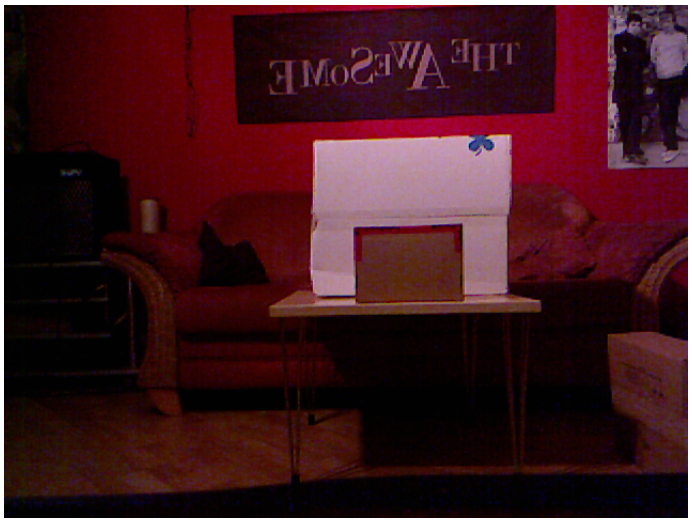
6.2.1 Tiefenbild durch die Depth Filling Methode befüllen

Man kann nun mit verschiedenen Mengen an Werten arbeiten, aus denen am Ende der Mittelwert für den zu füllenden Bildpunkt berechnet wird. Da nach einigen Test mit höheren Bereichen das Bild sehr unsauber und auch sehr dunkel wurde, so das man keine Details mehr erkennen konnte, wird hier nur mit niedrigen Mengen an Bildpunkten gearbeitet.



(a) Bild vor dem Filling

(b) Bild nach Depth Filling



(c) RGB-Bild

Abbildung 6.12: Beispiel 1 Depth Filling

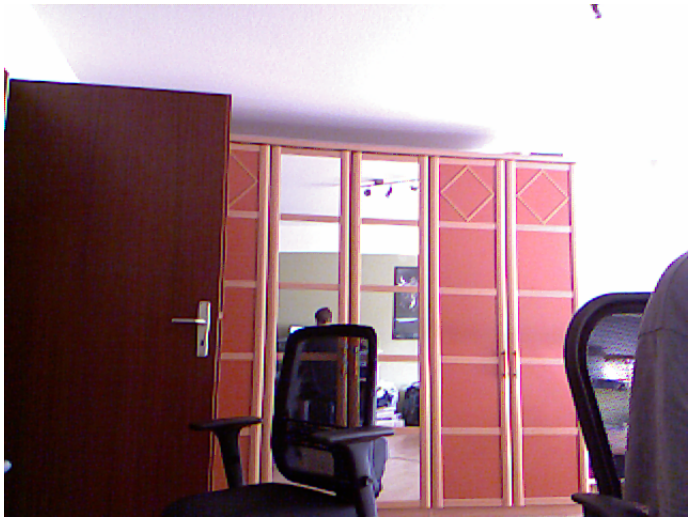
Im Bild 6.12(b) wird der zweite Ansatz 5.2.2 zum Entfernen der Ränder des Bildes verwendet. In Bild 6.12(a) sind die fehlerhaften Bildpunkte, die befüllt werden sollen, rot markiert. In

diesem Versuch wird eine Pixelbreite von 5 Pixel rechts und links des fehlerhaften Bildpunktes genutzt, aus denen man den Mittelwert errechnet, um den fehlerhaften Bildpunkt zu befüllen.

Bild 6.12(b) zeigt das befüllte Bild. Hier wurden ebenfalls alle Tiefenwerte, die den Wert 0 haben, markiert. Wie man auf dem Bild erkennen kann, ist die Anzahl dieser fehlerhaften Bildpunkte stark reduziert worden. Bild 6.12(c) zeigt das zugehörige Farbbild. Auf dem Farbbild sind zwei Kartons zu erkennen, welche auf einem ein Tisch stehen. Die Tischbeine sind nur schwer in dem Bild 6.12(a) zu erkennen, da der Raum auch sehr dunkel ist. Die Beine werden hier durch fehlerhafte Bildpunkte dargestellt. Die Ränder der Kartons weisen ebenfalls fehlerhaften Bildpunkte auf. In Bild 6.12(b) ist gut zu sehen, dass die fehlerhaften Bildpunkte an den Rändern der Kartons befüllt wurden und die Kartons nun in ihren Randeigenschaften gut zu sehen sind. Die Beine des Tisches sind hingegen komplett verschwunden und nicht mehr zu erkennen.

(a) Bild vor dem Filling

(b) Bild nach Depth Filling



(c) RGB-Bild

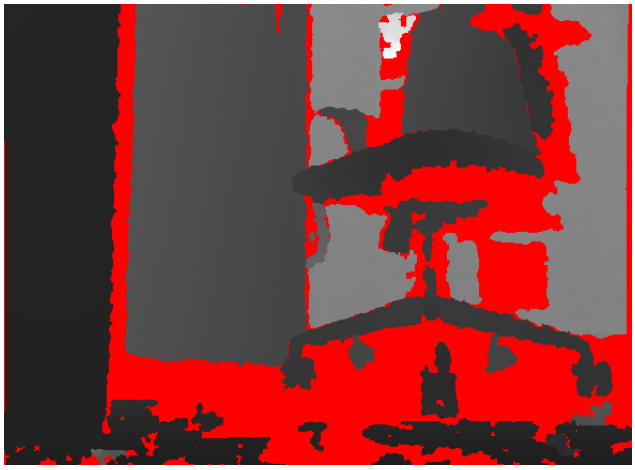
Abbildung 6.13: Beispiel 2 Depth Filling

Bei diesem Beispiel wurde ebenfalls mit einer Pixelbreite von 5 Pixeln vor und hinter dem fehlerhaften Bildpunkte, der Mittelwert berechnet. Auch hier wird der zweiten Ansatz zum Beschneiden des Randes verwendet.

In dem zugehörigen Farbbild 6.13(c) sind verschiedene Objekte im Bild zu sehen. Beispielsweise ist eine Tür zu erkennen, sowie ein Schrank im Hintergrund, der in der Mitte Spiegel aufweist. Weiterhin ist in der Mitte des Bildes ein Stuhl zu erkennen. Das Tiefenbild

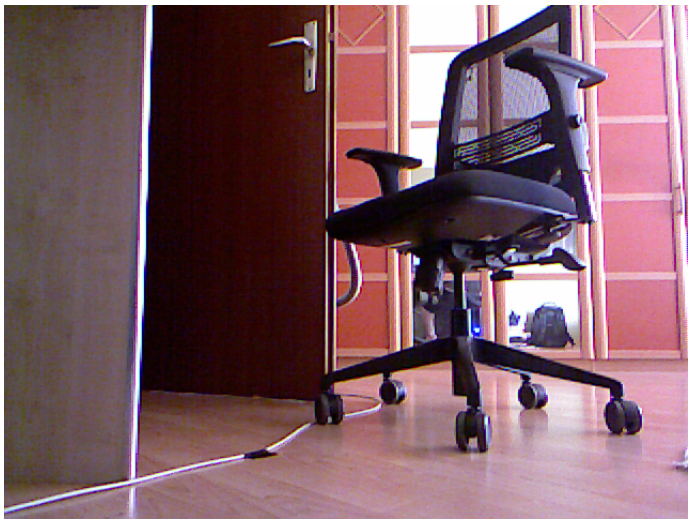
6.13(a) weist an der Tür, sowie am Stuhl einige fehlerhaften Bildpunkte auf, die rot markiert wurden. Eine Armlehne des Stuhls im Bild ist im Tiefenbild fast komplett unerkannt. Die Spiegel des Schranks weisen ebenfalls einige fehlerhafte Bildpunkte auf. Nach dem Filling 6.13(b) sind im Tiefenbild keine fehlerhaften Bildpunkte mehr zu erkennen. Die Tür ist nun beispielsweise im oberen Teil gut zu erkennen, während der Stuhl nun sehr unsauber dargestellt wird. Die Spiegelflächen sind zwar auch befüllt, die Ränder sind jedoch nicht mehr sauber dargestellt und die einzelnen Spiegelsegmente entsprechen auch nicht mehr ihren vorherigen Größen. Weiterhin ist an der Stuhllehne des Bildes zu erkennen, dass diese sehr verzerrt wird und in die Fläche der Spielgel hineingezogen wurde. Auch die Lehne des Stuhls ist nicht mehr richtig erkennbar. Sie ist nun Teil der Tür geworden.

Das Ergebnis dieses Beispiels kann man nun auf zwei unterschiedliche Weisen sehen. Wenn man es von dem Standpunkt aus betrachtet, dass man alle fehlerhaften Bildpunkte zu befüllen hat, funktioniert diese Variante sehr gut. Man findet keine fehlerhaften Bildpunkte mehr im Bild. Was weniger gut funktioniert hat ist, dass die Kanten gut erhalten geblieben sind. Die Tür und der Schrank, welches große Objekte in dem Bild darstellen, sind immer noch sehr gut zu erkennen sowie die Unterschiede in den Tiefen. Wenn man sich aber die kleineren Objekte wie beispielsweise den Stuhl anschaut, dann wirkt das Bild sehr verzerrt. Die Tiefenwerte verschwimmen mit anderen Tiefenwerten auf dem Bild und es gibt keine genauen Abhebungen zu den anderen Objekten.



(a) Bild vor dem Filling

(b) Bild nach Depth Filling



(c) RGB-Bild

Abbildung 6.14: Beispiel 3 Depth Filling

Im letzten Beispiel ist ein Bild 6.14(a) zusehen, welches viele fehlerhaften Bildpunkte aufweist. Hier wurden wie in den vorangegangenen Beispielen, ebenfalls die 5 Bildpunkte vor und hinter dem fehlerhaften Bildpunkt, zur Berechnung des Mittelwertes verwendet. Gleichfalls wurde der zweite Ansatz 5.2.2 zum Beschneiden des Bildes verwendet.

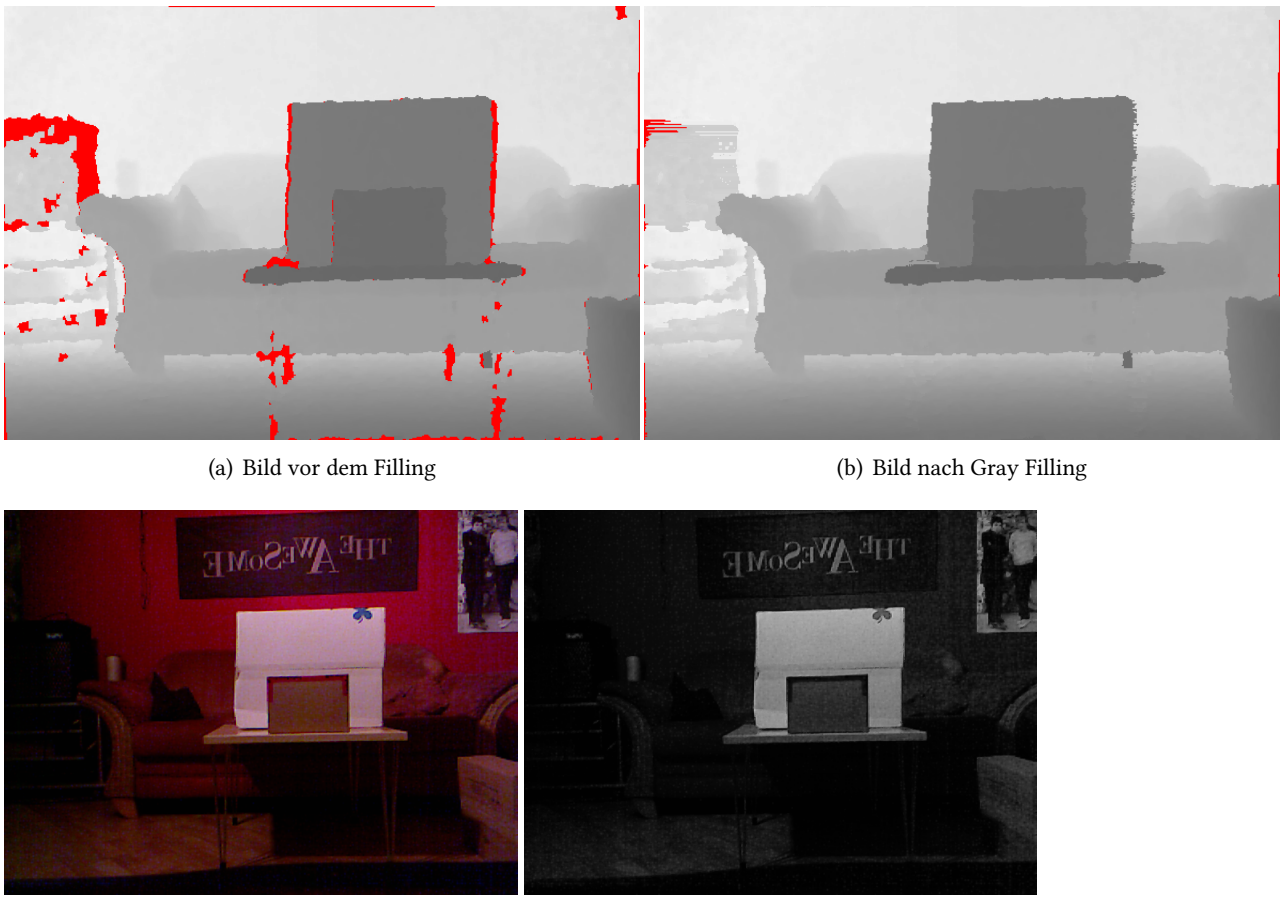
Im Farbbild 6.14(c) dieses Beispiels sieht man eine Tischkante im rechten vorderen Teil des Bildes, sowie die Tür und den Stuhl im Vordergrund. Weiterhin ist ein Boden zu sehen, der

aber zu nah an der Kamera ist. Nach der Befüllung des Tiefenbildes ist das Ergebnis in Bild 6.14(b) zu sehen. Wenn man sich das aufgenommene Tiefenbild hierzu anschaut 6.14(a) und alle Tiefenwerte hervorhebt, die einen Tiefenwert von 0 besitzen, dann sind diese Anteile in diesem Bild sehr hoch. Man kann an der Kante des Tisches fehlerhafte Bildpunkte sehen, sowie an der Tür und auch am Stuhl. Weiterhin enthält der Boden des Bildes ebenfalls viele fehlerhafte Bildpunkte. Schaut man sich nun das Tiefenbild nach der Befüllung an (Bild 6.14(b)) sind keine fehlerhaften Bildpunkte mehr in diesen Bild zu erkennen. Es werden keine Kanten mehr rot markiert. Die Kanten des Tisches und der Tür sind gut zu erkennen und befüllt. Doch wenn man im unteren Bereich des Bildes schaut, dann vermischen sich die Kanten der Tür und des Bodens miteinander, sodass man am unteren Rand keinen Unterschied mehr sieht. Die Ränder des Stuhls sind ebenfalls alle befüllt und man kann den Stuhl noch sehr gut erkennen, wobei manche Details nur schemenhaft dargestellt werden. Im unteren Teil des Bildes ist zwar noch zu erkennen, wo der Stuhl aufhört, aber es werden keinerlei Details dargestellt.

Zusammenfassend zu diesem letzten Versuch kann man sagen, dass zwar alle fehlerhaften Bildpunkte aus dem Bild befüllt sind, doch einige Details nicht mehr vorhanden. Im oberen und mittleren Bereich des Bildes sind die Kanten klar erkennbar, doch im unteren Teil des Bildes gehen alle Details ineinander über und man kann keine klaren Aussagen, über Ränder und einzelne Objekte treffen.

6.2.2 Filling von Tiefenbildern mit dem Grauwertbild

In diesem Kapitel soll das Filling mit Unterstützung des Grauwertbildes vorgenommen werden. Die einbezogene Pixelbreite, die zum Vergleich genutzt wurde, beträgt 10 Pixel vor und hinter den fehlerhaften Bildpunkten. Hier werden die selben 3 Tiefenbilder verwendet wie im vorherigen Beispiel.



(a) Bild vor dem Filling

(b) Bild nach Gray Filling

(c) RGB-Bild

(d) GRAY-Bild

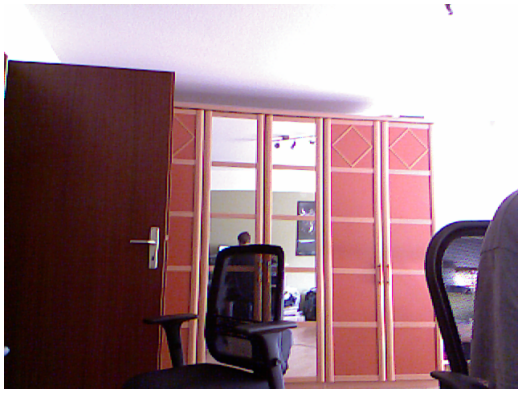
Abbildung 6.15: Beispiel 1 des Gray-Filling

Man sieht in Beispiel 1 die selbe Szene mit den beiden Kartons. Bild 6.15(a) zeigt das aufgenommene Tiefenbild, Bild 6.15(b) zeigt das Bild nach dem Befüllen der fehlerhaften Bildpunkte. Als Referenz hierzu zeigt Bild 6.15(c) das Farbbild der Szene und Bild 6.15(d) das Grauwertbild, welches als Referenz genutzt wurde.

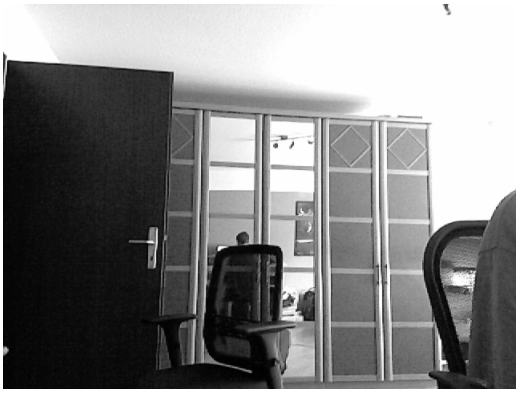
Man sieht an dem befüllten Bild 6.15(b) im rechten Bereiche des Bildes, einen kleinen Rand, die einen 0-Value aufweisen. Bei diesem Filling-Ansatz kann man deutlich erkennen, dass sehr viele fehlerhafte Bildpunkte befüllt wurden. Die Ränder der beiden Objekte sind komplett befüllt. Man kann deutlich erkennen das die Ränder der Objekte unsauber aussehen. In diesem Bild sind ebenfalls Details durch das Filling verloren gegangen sind.

(a) Bild vor dem Filling

(b) Bild nach Gray Filling



(c) RGB-Bild



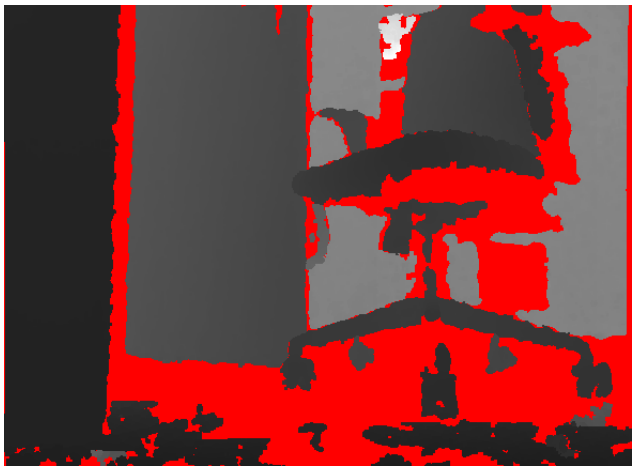
(d) GRAY-Bild

Abbildung 6.16: Beispiel 2 des Gray-Filling

Beispiel 2 unserer Versuchsreihe zeigt das aufgenommene Tiefenbild mit den fehlerhaften Bildpunkten [Bild 6.16(a)], sowie das Tiefenbild nachdem das Grauwert-Filling auf das Tiefenbild angewendet wurde [Bild 6.16(b)]. Bild 6.16(c) und Bild 6.16(d) zeigen das RGB-Bild und das Grauwertbild der Szene.

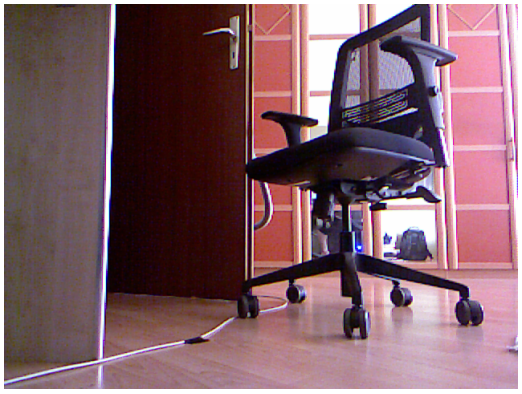
Bild 6.16(b) in Beispiel 2 wurde ebenfalls befüllt. Man kann in dem befüllten Bild noch einen kleinen Restrand sehen. In dem Bild werden die Kanten noch sehr gut dargestellt und sind eindeutig erkennbar. An den reflektierenden Flächen wurden alle fehlerhaften Bildpunkte befüllt und man kann sehen das die Flächen die viele fehlerhaften Bildpunkte aufgewiesen haben sehr stark befüllt werden. Wenn man sich den Stuhl anschaut dann fallen hier die

meisten Veränderungen auf. Die Konturen des Stuhls sind ungenau. Die Flächen des Stuhls und des Schranks verschwimmen ineinander und sind auch nicht mehr klar trennbar. Die Ränder an dem Stuhl sind sehr unsauber.



(a) Bild vor dem Filling

(b) Bild nach Gray Filling



(c) RGB-Bild

(d) GRAY-Bild

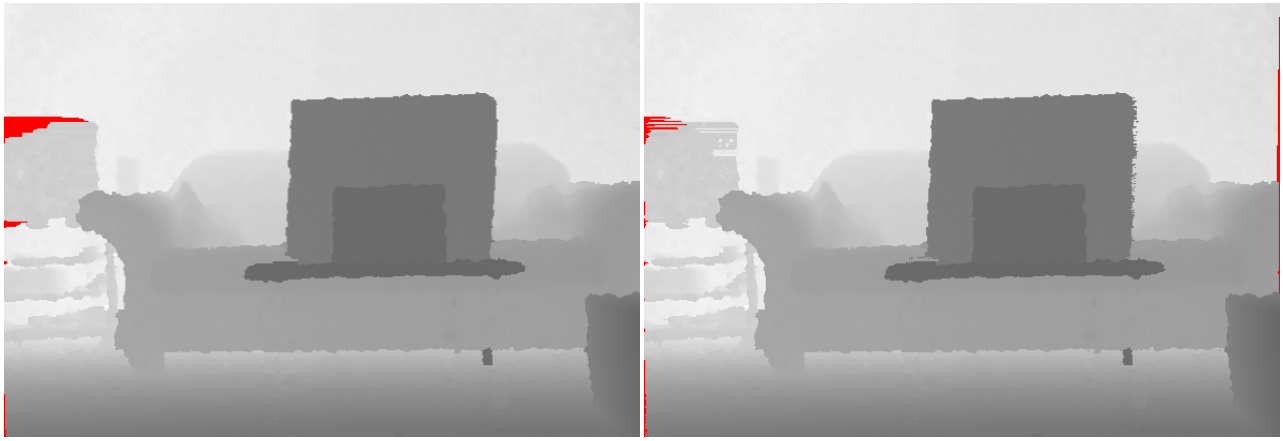
Abbildung 6.17: Beispiel 3 des Gray-Filling

Beispiel 3 des vorherigen Versuchs, weist viele fehlerhafte Bildpunkte auf. Bild 6.17(b) zeigt wieder das aufgenommene Tiefenbild und die fehlerhaften Bildpunkte rot hervorgehoben. Bild 6.17(b) ist das Bild nach dem Grauwert-Filling, während Bild 6.17(c) und Bild 6.17(d) das RGB-Bild und das Grauwertbild der Szene zeigt.

Beispiel 3 6.17(b) weist nach dem Befüllen nur noch wenige fehlerhafte Bildpunkte auf. Die Ränder in dem Bild sind befüllt und auch sehr unsauber. Beim Stuhl kann man deutliche

Unsauberkeiten erkennen. Im unteren Bereich des Bildes, verschwimmt der Boden komplett. Man kann keine Details mehr erkennen.

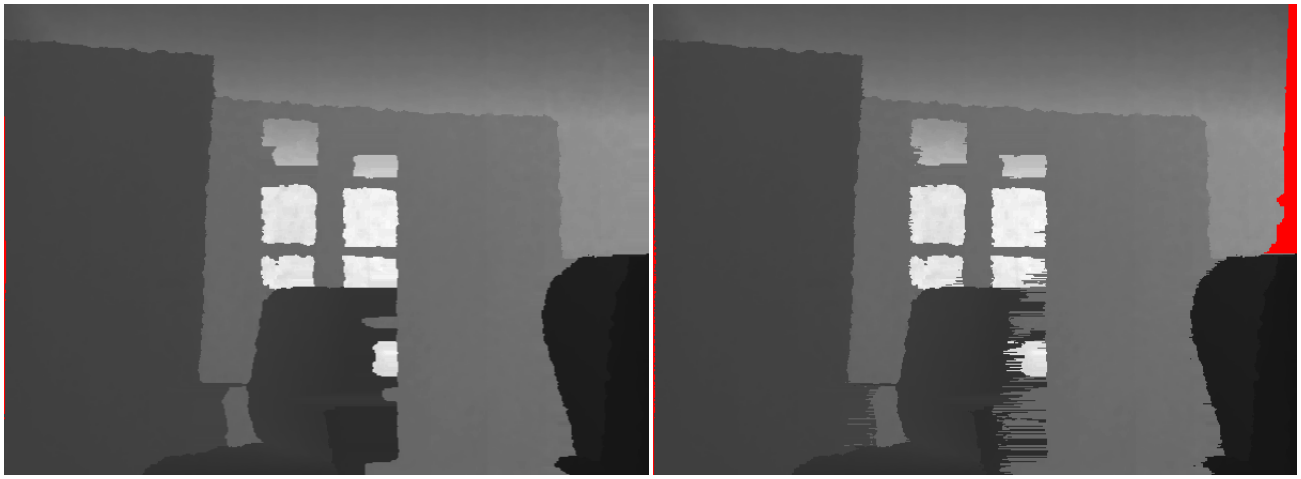
6.2.3 Auswertung der Filling-Ansätze



(a) Depth-Filling Beispiel 1 (b) Grauwert-Filling Beispiel 1

Abbildung 6.18: Auswertung Beispiel 1 der Filling-Methoden

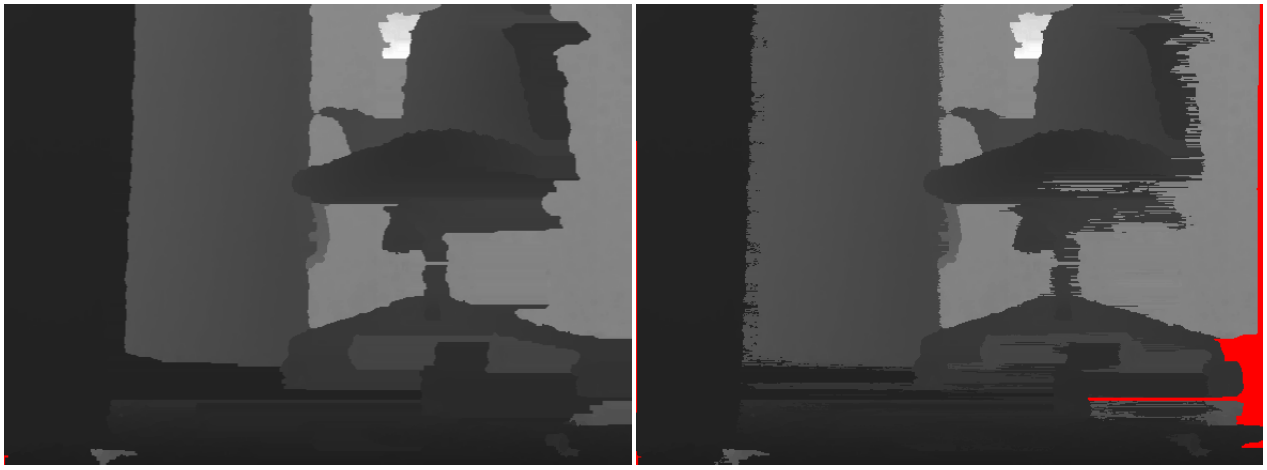
Bild 6.18(a) und Bild 6.18(b) zeigen die erste Szene der beiden Filling-Methoden. Hier ist deutlich zu erkennen, dass die fehlerhaften Bildpunkte bei beiden Ansätzen sehr gut befüllt wurden. In beiden Bildern ist zu erkennen das Details des Bildes verloren gegangen sind, doch die Kanten der Objekte sind in beiden Bildern befüllt worden. Beim Grauwert-Filling sind die Kanten unsauber, aber auch hier gut zu erkennen.



(a) Depth-Filling Beispiel 2 (b) Grauwert-Filling Beispiel 2

Abbildung 6.19: Auswertung Beispiel 2 der Filling-Methoden

Bild 6.19(a) und Bild 6.19(b) zeigen die zweite Szene der beiden Filling-Methoden. Aus dem Bild vom Grauwert-Filling sieht man, dass noch einige fehlerhafte Bildpunkte an den Rändern erhalten geblieben sind. Im Bild sind die fehlerhaften Bildpunkte befüllt worden. Die großen Objektkanten im Bild sind gut zu erkennen. Die spiegelnden Flächen im hinteren Bereich des Bildes, sind relativ gleich befüllt, jedoch sind die Flächen in beiden Bildern sehr unregelmäßig. Der Stuhl der in beiden Szenen zu sehen ist, wirkt auf der linken Seite sehr gut befüllt und seine Konturen sind gut zu erkennen. Die rechte Seite des Stuhls ist in beiden Ansätzen sehr ungenau und verschwimmt mit dem Objekt im hinteren Bereich. Hier kann keine klare Abtrennung aufgezeigt werden.



(a) Depth-Filling Beispiel 3 (b) Grauwert-Filling Beispiel 3

Abbildung 6.20: Auswertung Beispiel 3 der Filling-Methoden

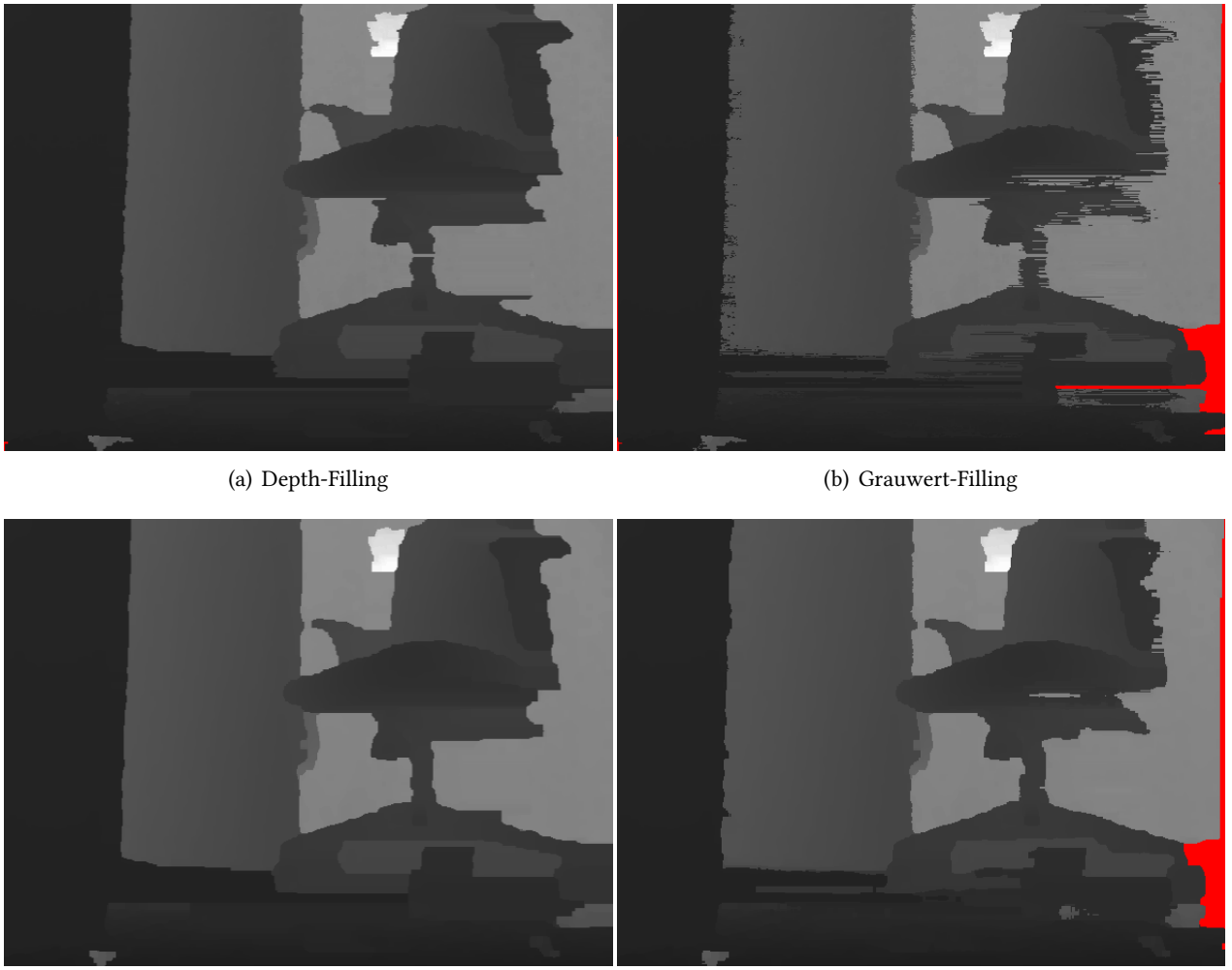
Bild 6.20(a) und Bild 6.20(b) zeigen die dritte Szene der beiden Filling-Methoden. Dieses Beispiel weist eine hohe Dichte an fehlerhaften Bildpunkten auf. In beiden Filling-Ansätzen wurden die fehlerhaften Bildpunkte gut befüllt. Nur im Grauwert-Filling sind noch am rechten Rand fehlerhafte Bildpunkte zu erkennen. Im Depth-Filling Beispiel sind die Kanten des Tisches und der Tür sehr gut zu erkennen. Im Grauwert-Filling sind die Kanten zwar befüllt worden, doch die Ränder sind sehr unsauber und verschwimmen an manchen Stellen mit ihrer Umgebung. Die Konturen des Sthuls sind im Depth-Filling Ansatz besser zu erkennen. Im Grauwert-Filling ist der Stuhl auf der rechten Seite sehr verschwommen und nicht deutlich abgrenzbar.

Zusammenfassend kann man sagen, das beide Filling-Ansätze funktionieren. Der Depth-Filling Ansatz und der Grauwert-Filling Ansatz befüllen beide den Großteil der fehlerhaften Bildpunkte. Im ersten Beispiel sind in beiden Fällen die Objektkanten gut gefüllt. Das Filling in den anderen beiden Beispielen ist sehr sehr ähnlich. In beiden Fällen sind die Objekte zu erkennen, doch verschiedene Details verschwimmen in den Bildern. Auch ist deutlich zu erkennen das die Kanten im Grauwert-Filling sehr unsauber sind. Man kann nicht immer genau sehen wo die Objektgrenzen anfangen und enden. Beide Ansätze befüllen die Kanten der Objekte, jedoch sind sie sehr ungenau. Abschließend zu den beiden Ansätzen kann man sagen, dass beide Methoden noch verfeinert werden müssen. Ideen zum Weiterentwickeln dieser Ansätze findet man unter Kapitel 7.1.

6.3 Tiefpassfilterung der befüllten Tiefenbilder

In Kapitel (4) Verwandte Arbeiten zur Qualitätsverbesserung von Tiefenbildern, werden einige Ansätze vorgestellt, die sich ebenfalls mit der Qualitätsverbesserung von Tiefenbildern befassen. Es wird der Median Filter erwähnt, der zur Rauschunterdrückung genutzt wird. Der Median Filter ist ein einfacher Mittelwert-Filter (2.3.1), der zum Verfeinern der einzelnen Tiefenbilder angewendet wird. Hier soll kurz dargestellt werden, wie sich der Filter auf das Tiefenbild auswirkt. Dies wird mit einer 5x5 Median-Filter-Maske veranschaulicht.

An Beispiel 3 soll die Bearbeitung durch den Median-Filter veranschaulicht werden, da hier unterschiedliche Kanten im Bild existieren und man so die Ergebnisse einfacher darstellen kann.



(a) Depth-Filling

(b) Grauwert-Filling

(c) Depth-Filling mit Median-Filter 5x5

(d) Grauwert-Filling mit Median-Filter 5x5

Abbildung 6.21: Beispiel Median-Filter

Bild 6.21(a) und Bild 6.21(b) zeigen die Bilder nach den Filling-Ansätzen. Die Bilder 6.21(c) und 6.21(d) zeigen den auf die Tiefenbilder angewendeten Median Filter. Man kann in den Bildern 6.21(c) und 6.21(d) sehr genau sehen, dass das Rauschen deutlich verringert wurde. Auch sieht man, dass die Kanten besser erkennbar sind. Der Median-Filter hat durch seine Funktionsweise in Bild 6.21(d) auch einige fehlerhafte Bildpunkte weggeschnitten. Man kann ebenfalls anhand des Bildes 6.21(d) erkennen, dass die Konturen des Stuhls besser sichtbar und das die Ränder der großen Objekte deutlicher geworden sind, als noch im Bild 6.21(b). Der

Median-Filter zeigt eine deutliche Qualitätsverbesserung der Bilder und ist als einfacher Filter leicht zu nutzen und zu implementieren.

7 Zusammenfassung

In dieser Arbeit wurden verschiedene, eigene Ansätze zur Qualitätsverbesserung von Tiefenbildern untersucht, weiterhin wurden auch einige Ansätze gezeigt, die aus anderen Arbeiten hervorgebracht wurden. Zusammenfassend kann man sagen, dass es verschiedene Ansätze gibt, Tiefenbilder vorzuverarbeiten und deren Qualität zu verbessern.

Man darf hierbei auch nicht vergessen, dass die Qualität von Tiefenbildern sehr vom Verfahren abhängt, mit welchen diese generiert wurden. Weiterhin spielt die Auflösung der Kameras eine große Rolle. Man wird immer bessere Methoden finden, um Tiefenbilder qualitativ zu verbessern und diese immer genauer werden zu lassen. Die Technik in diesem Bereich entwickelt sich sehr schnell. In dieser Arbeit wurde ein altes Modell der Kinect-Kamera verwendet. Zum jetzigen Zeitpunkt wurde bereits eine neue Kinect-Kamera entwickelt, die eine wesentlich effektivere Methode verwendet um Tiefenbilder zu generieren. Hier entstehen nicht dieselben Probleme wie bei der Microsoft Kinect 360 aus dem Testaufbau.

Man darf sehr gespannt darauf sein, wie sich in Zukunft Tiefenbilder verändern werden und wie genau sie in unseren Alltag mit einfließen werden.

7.1 Ideen zum weiterführen der besprochenen Ansätze

Aus den Ergebnissen aus Kapitel 6 lassen sich verschiedene Weiterentwicklungen der Ansätze ableiten. Die Filling-Ansätze sind noch sehr ungenau und könnten auf verschiedene Weise weiterentwickelt werden. Beispielsweise könnte man vor dem Filling eine Objekterkennung durchführen, um die verschiedenen Objekte einzeln zu klassifizieren und hierdurch zu bestimmen welche Punkte für die einzelnen Filling-Methoden verwendet werden. Weiterhin ist der Grauwert-Filling Ansatz sehr einfach. Er beruht darauf das zwei Objekte die eine Menge von fehlerhaften Bildpunkten aufweisen, eine klare Kante besitzen die sie voneinander trennt (siehe Bild 7.1). Hier wird nicht berücksichtigt, dass es in der Menge der fehlerhaften Bildpunkte mehrere Kanten geben kann. Dadurch könnte es sein das unterschiedliche Grauwerte in der Menge der fehlerhaften Bildpunkte existieren (siehe Bild 7.2). Hier muss eine Idee gefunden werden wie man diese Bereiche sinnvoll füllt.

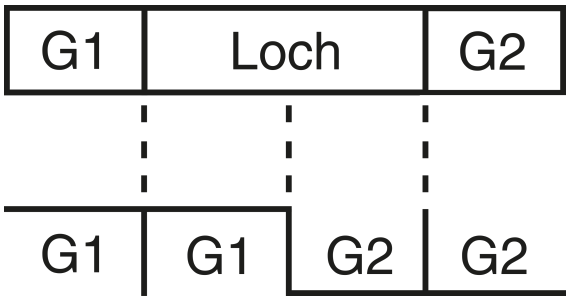


Abbildung 7.1: Idee des Gray-Fillings Ansatzes

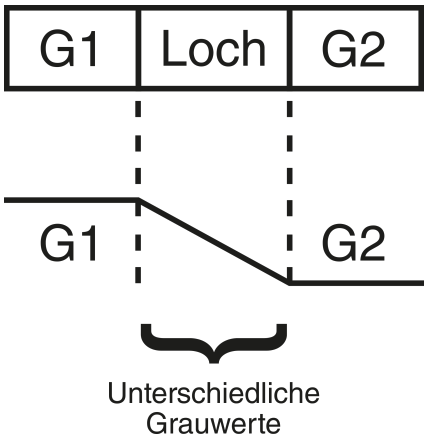


Abbildung 7.2: Problemstellung erweitert

8 Anhang

Im Anhang sollen noch einige Sachen erläutert werden, die nicht in den Text gepasst haben.

8.1 Verwendete Tools

In dieser Arbeit wurde ein Tool verwendet, welches sich ImageJ[1] nennt. Es ist ein auf Java basierendes Bildverarbeitungsporgramm. Man kann mit diesem Programm einfach Bilder laden und bearbeiten. Es bietet verschiedene Optionen und Plugins, die in dieser Arbeit Verwendung fanden. Beispielsweise wurden mit diesem Programm die Tiefenwerte mit einem 0-Value hervorgehoben. Außerdem bietet dieses Tool auch einige Filter der Bildverarbeitung, die man auf die Bilder anwenden kann.

Die Ansätze dieser Arbeit wurden in Java getestet. Hierfür wurde Java 7 verwendet.

Um mit der Kinect zu kommunizieren, wurde das Softwarepaket von OpenNi verwendet. Um unter Java mit der Kinect zu kommunizieren, wurde eine älteres Softwarepaket von OpenNi verwendet. Die Treiber wurden nach einer Installationsanleitung[2] installiert. Das System funktioniert sehr gut. Als Referenz zur Programmierung wurde das Buch von Andrew Davison verwendet. [AD, 1. Juni 2012]

In dieser Arbeit wurde eine Bildverarbeitungsbibliothek namens Opencv[3] verwendet. Um diese Bibliothek in Java verwenden zu können, wurde in dieser Arbeit ein Wrapper verwendet, Javacv[4]

[1] Quelle :http://imagej.nih.gov/ij/ Letzter Zugriffe : 17.09.2014
[2] Quelle : http://fivedots.coe.psu.ac.th/~ad/kinect/installation.html Letzter Zugriff : 17.09.2014
[3] Quelle : http://opencv.org/ Letzter Zugriff : 17.09.2014
[4] Quelle : http://www.cs.dartmouth.edu/~cs10/install/javacv-win/ Letzter Zugriff : 17.09.2014

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Literaturverzeichnis

[AD 1. Juni 2012] Davison, Andrew: *Kinect Open Source Programming Secrets [hacking the Kinect with OpenNi, NITE und Java].* Entrepreneur Media, 1. Juni 2012

[ATPA 2012] Andersen, M.R. ; Jensen, T. ; Lisouski, P. ; Mortensen, A.K. ; Hansen, M.K. ; Gregersen, T. ; Ahrendt, P.: Kinect Depth Sensor Evaluation for Computer Vision Applications / Department of Engineering – Electrical and Computer Engineering, Aarhus University. 2012. – Forschungsbericht

[CPT 2003] Criminisi, A ; Perez, P. ; Toyama, K.: Object removal by exemplar-based inpainting. In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on* Bd. 2, June 2003, S. II–721–II–728 vol.2. – ISSN 1063-6919

[DJY 2012] Miao, Dan ; Fu, Jingjing ; Lu, Yan ; Li, Shipeng ; Chen, Chang W.: Texture-assisted Kinect depth inpainting. In: *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, May 2012, S. 604–607. – ISSN 0271-4302

[FEMO] Siegen, Universität: *Fertigungsautomatisierung und Montage [letzter Zugriff : 06.09.2014].* – URL https://wiki.zimt.uni-siegen.de/fertigungsautomatisierung/index.php/Hauptseite

[HEHA September 2001] Hamfeld, Helmut: *Aktive Stereoskopie , Neue Verfahren zur dreidimensionalen Vermessung von Objekten [Dissertation von Hemlut Hamfeld Universität Kaiserslautern September 2001]*, Universität Kaiserslautern, Dissertation, September 2001

[HUBE] Hu-Berlin: *Verfahren der Digitalen Bildverarbeitung [letzter Zugriff : 06.09.2014].* – URL http://www2.informatik.hu-berlin.de/~bien/vddb.html#Filter

[JSY 2012] Fu, Jingjing ; Wang, Shiqi ; Lu, Yan ; Li, Shipeng ; Zeng, Wenjun: Kinect-like depth denoising. In: *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, May 2012, S. 512–515. – ISSN 0271-4302

[KJZ 2012] Xu, Kang ; Zhou, Jun ; Wang, Zhen: A method of hole-filling for the depth map generated by Kinect with moving objects detection [letzter Zugriff : 06.09.2014]. In: *Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on*, June 2012, S. 1–5. – ISSN 2155-504

[LHS 2012] Chen, Li ; Lin, Hui ; Li, Shutao: Depth image enhancement for Kinect using region growing and bilateral filter [letzter Zugriff : 06.09.2014]. In: *Pattern Recognition (ICPR), 2012 21st International Conference on*, Nov 2012, S. 3070–3073. – ISSN 1051-4651

[NCSC 2013] Chahal, N. ; Chaudhury, S.: High quality depth map estimation by kinect upsampling and hole filling using RGB features and mutual information [letzter Zugriff : 06.09.2014]. In: *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), 2013 Fourth National Conference on*, Dec 2013, S. 1–4

[NIBU] Burrus, Nicolas: *Kinect Calibration [letzter Zugriff : 06.09.2014]*. – URL http://nicolas.burrus.name/index.php/Research/KinectCalibration

[SCSC 2013] Zhang, Shuai ; Wang, Chong ; Chan, S.C.: A new high resolution depth map estimation system using stereo vision and depth sensing device [letzter Zugriff : 06.09.2014]. In: *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*, March 2013, S. 49–53

[UNIM] Uni-Muenster: *Bildgewinnung und Bilddarstellung[letzter Zugriff : 06.09.2014]*. – URL https://www.uni-muenster.de/ZIV/Lehre/MM_HWK/V001S03.htm

[VBPA] Berlin-Pankow, Volkshochschule: *Farbmodelle [letzter Zugriff : 06.09.2014]*. – URL http://www.vhs-seminar.de/farbmodelle.html

[XPER] Xpergate: *Wissen für Fabrikautomation [letzter Zugriff : 06.09.2014]*. – URL http://www.xpertgate.de/

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25. September 2014 Christian Sandhagen