

Objektmodellierung mit RGB-D Sensoren bei hoher Bildrate

DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
Dipl.-Ing. Dr.techn. J. Prankl

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von

Oliver Fischer, B.Sc.
Prinz Eugen-Straße 3/3/24
2000 Stockerau
Österreich

Stockerau, im Oktober 2016

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Ort, Datum

Oliver Fischer, B.Sc.

Vorwort

Ich möchte mich bei allen Betreuern, Kollegen und Familienmitglieder bedanken, die mich während dem Verfassen dieser Diplomarbeit unterstützt haben. Begonnen mit Markus Vincze, der es möglich gemacht hat die Arbeit in der Vision for Robotics Gruppe des Instituts für Automatisierung und Regelungstechnik zu verfassen. Gefolgt von Johann Prankl, der mich im Laufe der Erstellung des Frameworks und beim Schreiben der Arbeit zu jeder Zeit unterstützt hat und mir mit seiner Erfahrung und seinem Wissen zur Seite stand, wenn ich es benötigt habe. Erwähnen möchte ich außerdem Sergey Alexandrov, der mit seinem Tool zur Anzeige von Teilergebnissen wesentlich dabei geholfen hat das Framework zu entwickeln.

Außerdem möchte ich allen Personen danken, deren Open Source Software ich verwendet habe. Vor allem der Gemeinschaft, die hinter der PCL steht ohne deren Bibliothek die Entwicklung erheblich erschwert wäre.

Besonderer Dank gilt meinen Eltern, Gerhard und Manuela Fischer, die mich mein ganzes Leben immer unterstützt haben und mir bei weit mehr als meinem Studium geholfen haben.

Speziell bedanken möchte ich mich bei meiner Lebensgefährtin Doris Mayr. Sie begleitet mich seit Jahren und steht jederzeit unterstützend an meiner Seite. Zusätzlich möchte ich mich bei meinen baldigen Schwiegereltern, Leopold und Ulrike Mayr für die vielfältige Unterstützung bedanken.

Nicht zu vergessen ist mein Studienkollege, Patrick Frager, den ich im Laufe des Studiums kennen lernen durfte. Gemeinsam haben wir uns für annähernd alle Prüfungen vorbereitet und haben uns dabei immer wieder gegenseitig motiviert.

Abstract

There are many application areas for 3D models of real world objects. Such models are used in many different areas of engineering, in the area of architecture and in the gaming industry. They often make development processes faster and easier but typically used scanner systems are big and often used in combination with additional hardware such as controlled mechanical turntables. Therefore the flexibility of such systems is strongly restricted.

In the course of this thesis a flexible scanner system is developed which enables the user to create fully automated 3D models of real world objects. Instead of an expensive optical sensor a cost-efficient RGB-D camera is used. The model is represented by many surface elements which get updated with every new recording. All recordings need to be transformed into one common coordinate system and therefore the pose of the camera needs to be determined with respect to six degrees of freedom. This is done with a modern SLAM method. The difference between a Frame-To-Frame and a Frame-To-Keyframe camera tracking is investigated and the problem of loop closing is solved. In order to separate the object from its surroundings it is assumed that the object stands alone on a flat surface. Due to the successive expansion of the model it is possible to take care of outliers and other problems during the modelling process.

The accuracy of the system is analyzed by comparing a generated model to a laserscan. Additionally an optimisation problem is expressed which makes it possible to decide how many records should be used with respect to runtime, degree of detail and failure rate. Finalizing the advantages and disadvantages of the proposed framework are discussed and possible future work is suggested.

Kurzzusammenfassung

Für gescannte 3D-Modelle von Objekten der realen Welt existieren zahlreiche Anwendungsgebiete. Diese Modelle werden in etlichen Bereichen der Technik, der Architektur und im Bereich der Spieleindustrie benötigt und machen oftmals Entwicklungsprozesse einfacher und schneller. Viele Scanner-Systeme basieren auf großen optischen Scanner, die oftmals in Verbindung mit zusätzlicher Hardware, wie zum Beispiel einem geregelten Drehteller, verwendet werden. Die Flexibilität dieser Systeme ist stark eingeschränkt.

Im Zuge dieser Arbeit wird ein flexibles Scanner-System entwickelt, das das Scannen von Objekten und automatisierte Generieren von 3D-Modellen ermöglicht. Anstelle eines optischen Sensors wird eine kostengünstige RGB-D Kamera eingesetzt. Das Gesamtmodell wird aus einer Vielzahl von Oberflächenelementen repräsentiert, deren Informationen mit jeder neuen Aufnahme verfeinert werden.

Damit sich die neue Aufnahme und das Modell in einem Koordinatensystem befinden, muss die Pose der Kamera in allen sechs Freiheitsgraden bestimmt werden. Dazu wird ein moderner Kameraverfolgungsalgorithmus verwendet. Es wird der Unterschied zwischen einer Bild-zu-Bild und Bild-zu-Schlüsselbild Bahnverfolgung untersucht und das Problem des Schleifenschlusses gelöst. Damit das Objekt von der Umgebung getrennt werden kann, wird angenommen, dass es separiert auf einer ebenen Fläche vor der Kamera steht. Durch die sukzessive Erweiterung des Modells kann auf viele Probleme, wie z.B. Ausreißer, direkt während der Modellierung eingegangen werden.

Die Genauigkeit des entwickelten Systems wird untersucht indem ein erzeugtes Modell mit einem Laserscan verglichen wird. Ebenso wird ein Optimierungsproblem formuliert und gelöst, mit dem es möglich ist die Anzahl der zu verwendenden Aufnahmen für die Modellerstellung, unter Berücksichtigung der Laufzeit, des Detailgrads und der Fehlerrate zu berechnen. Abschließend werden die Vor- und Nachteile des vorgestellten Ansatzes diskutiert und auf mögliche nachfolgende Arbeiten hingewiesen.

Inhaltsverzeichnis

1	Einführung	1
1.1	Problemstellung	2
1.2	Lösungsansatz	2
2	Stand der Technik	4
2.1	Aufnahme der Rohdaten	4
2.2	Visuelle Odometrie	6
2.3	Frameworks zur Modellerstellung	7
3	Objektmodellierung mit RGB-D Sensor	11
3.1	Aufnahme der Grunddaten	11
3.2	Kameratracking	13
3.2.1	DVO - Dense Visual Odometry	14
3.2.2	Schlüsselbildbasiertes Kameratracking	18
3.2.3	Schleifenschluss	21
3.3	Erstellung des Modells	25
3.3.1	Modellrepräsentation	25
3.3.2	Modellerstellung	26
3.3.3	Auswahl der verwendeten Aufnahmen zur Modellerstellung	42
3.3.4	Parameter des Frameworks	44
4	Auswertung	46
4.1	Untersuchung der Genauigkeit	46
4.1.1	Untersuchung anhand einfacher Objekte	46
4.1.2	Vergleich mit einem Laserscan	49
4.2	Untersuchung zum Einfluss des Winkelabstands	52
4.2.1	Erstellung der Untersuchungsdaten	54
4.2.2	Formulierung des Optimierungsproblems	56
5	Zusammenfassung und Ausblick	61

Abbildungsverzeichnis

1.1	Fotografie eines Objekts, das zur Modellierung verwendet wurde.	2
1.2	Erzeugtes Modell des in Abbildung 1.1 dargestellten Objekts. . .	2
2.1	Einfaches Triangulationssystem, wodurch, mit Hilfe eines Laserstrahls und einer Kamera, der Abstand L bestimmt werden kann, vgl. [7].	5
2.2	Aufnahme eines vollständigen Modells durch Verwendung von drei unterschiedlichen Posen, vgl. [26].	8
2.3	(links) Darstellung des verwendeten Echtzeit 3D-Scannersystem. (rechts) Interaktiver Scan eines Schuhs, vgl. [27]	9
2.4	Schnelle und direkte Objektsegmentierung mit KinectFusion. (A) Oberflächennormalen der Szene, (B) Szene mit Textur, (C) Modell der Teekanne. (unten) Teekanne wird physikalisch bewegt um das Objekt zu segmentieren und anschließend zu modellieren, vgl. [28].	10
3.1	Abbildung eines Microsoft Kinect Sensors und ihrer Komponenten, vgl. [1].	11
3.2	Aufnahme mit einer Microsoft Kinect. Im markierten Bereich ist keine Tiefeninformation verfügbar.	12
3.3	Detailansicht einer Aufnahme. An Kanten ist die Tiefeninformation oftmals falsch.	12
3.4	Angenommene Bewegung der Kamera um das Objekt.	14
3.5	Grundidee des DVO-Algorithmus. Die Transformation g^* wird geschätzt indem der photometrische und der geometrische Fehler minimiert werden, vgl. [4].	15
3.6	Koordinatensystem der Microsoft Kinect, vgl. [36].	16
3.7	Frame-To-Frame Tracking einer statischen Szene. Aufgrund der akkumulierten Fehler ist ein Drift der Kameraposition zu beobachten.	17
3.8	Rotation des neuen Koordinatensystem zur Berechnung der Rotation um die \mathbf{x}_1 -Achse.	19
3.9	Frame-To-Keyframe Tracking einer statischen Szene. Es ist kein Drift zu beobachten.	20

3.10	Akkumulierter Fehler bei vollständiger Kreisbewegung unter Verwendung von Schlüsselbilder.	21
3.11	Bewegungsgraph zum Schleifenschluss. Knoten repräsentieren Schlüsselbilder, Kanten repräsentieren die Bewegung dazwischen. Die Werte der Tabelle zeigen die gewichteten Fehleranteile, vgl. [41].	22
3.12	Berechnung der Fehlertransformation ΔT	23
3.13	Schleifenschluss. Der Fehler wurde behoben, vgl. Abbildung 3.10. 24	
3.14	Darstellung eines Oberflächenelements. Dabei stellt \mathbf{p}_i die Position, \mathbf{n}_i den Vektor der Oberflächennormale, \mathbf{c}_i die Farbe und r_i den Radius des Oberflächenelements dar, vgl. [27].	26
3.15	Unbearbeitete Punktwolke wie sie von der Microsoft Kinect erzeugt wird. Im markierten Bereich erkennt man, dass falsche Punkte zum Objekt hinzugefügt wurden.	28
3.16	Ergebnis der Bearbeitung. Die falschen Punkte an den Kanten wurden entfernt.	28
3.17	Tiefenbild einer Aufnahme. Hellere Bereiche sind weiter von der Kamera entfernt als dunklere Bereiche.	29
3.18	Mit der Canny-Kantenerkennung wurden die Tiefensprünge erkannt.	29
3.19	Die gefundenen Kanten werden mittels anisotroper Filterung verbreitert.	29
3.20	Mittels RANSAC wurde der Boden entfernt.	29
3.21	Vorbereitete Punktwolke mit berechneten Oberflächennormalen. 29	
3.22	Ideenskizze zur Wahl des Radius der Oberflächenelemente.	31
3.23	Ideenskizze für die Suche der korrespondierenden Oberflächenelemente.	32
3.24	Skizze zur Idee des dreidimensionalen Suchgitters.	33
3.25	Skizze zur Berechnung des Radial- und Normalabstands zwischen zwei Oberflächenelementen.	34
3.26	Aktualisierung des Oberflächenelements \mathcal{S}_i mit der Information des Oberflächenelements \mathcal{S}_j . \mathcal{S}_i^* ist das Ergebnis der Aktualisierung. 36	
3.27	Verwendetes Kugelkoordinatensystem, um die Betrachtungsrichtung eines Oberflächenelements zu bestimmen, vgl. [27].	38
3.28	Durch Ungenauigkeiten beim Löschen des Bodens sind ein paar Oberflächenelemente ins Modell aufgenommen worden.	39
3.29	Durch den Vertrauenswert v_i können die Punkte vom Boden entfernt werden.	39
3.30	Darstellung des Suchzylinder \mathcal{Z}_1 zum Glätten des finalen Modells. 41	

3.31	Darstellung des Suchzylinder \mathcal{Z}_2 zum Filtern des finalen Modells. Das grüne Element \mathcal{S}_j wird für die weitere Generierung des finalen Modells ignoriert.	41
3.32	Berechnung des finalen Oberflächenelements $\mathcal{S}_{\text{final};i} \in \partial\mathcal{M}_{\text{final}}$. \mathcal{S}_i ist das Oberflächenelement des Modells $\partial\mathcal{M}$. $\mathcal{S}_{\text{mean}}$ ist durch Mittelwertbildung der Elemente im Zylinder \mathcal{Z}_1 berechnet worden.	41
3.33	Finales Modell nach der Erzeugung der Oberflächenelemente $\mathcal{S}_{\text{final};i}$	41
3.34	Darstellung der ausgewählten Aufnahmen zur Erstellung des Modells. Die Aufnahmen \mathcal{A}_k werden zu \mathcal{V}_s zusammengefasst. . .	43
4.1	Fotografie des Würfels, der zur Untersuchung der Genauigkeit verwendet wird.	47
4.2	Modell des Würfels aus Abbildung 4.1. Die gemessenen, charakteristischen Längen sind eingezeichnet.	47
4.3	Fotografie des Zylinders der zur Untersuchung, der Genauigkeit verwendet wird.	47
4.4	Modell des Zylinders aus Abbildung 4.3. Die gemessenen, charakteristischen Längen sind eingezeichnet.	47
4.5	Fotografie des sechseitigen Prismas, das zur Untersuchung der Genauigkeit verwendet wird.	47
4.6	Modell des sechseitigen Prismas aus Abbildung 4.5. Die gemessenen, charakteristischen Längen sind eingezeichnet.	47
4.7	Lasergescanntes Objekt.	50
4.8	Modell des Objekts.	50
4.9	Vergleich zwischen dem Laserscan und dem erzeugten Modell (Werte in mm).	51
4.10	Histogramm und approximative Normalverteilung des Modellierungsfehlers.	51
4.11	Auswirkung des Winkelabstands. (links) Modellierung erfolgte mit $\psi = 1^\circ$, (Mitte) Modellierung erfolgte mit $\psi = 5^\circ$, (rechts) Modellierung erfolgte mit $\psi = 15^\circ$	54
4.12	Darstellung der Laufzeit und der Rechenzeit in Abhängigkeit vom Winkelabstand ψ	56
4.13	Darstellung der erzeugten Punktanzahl und der Fehlerrate in Abhängigkeit vom Winkelabstand ψ	57
4.14	Darstellung der Laufzeit und der approximierenden Funktion in Abhängigkeit vom Winkelabstand ψ	58
4.15	Darstellung der erzeugten Punktanzahl und der Fehlerrate im Vergleich mit ihren Näherungen in Abhängigkeit vom Winkelabstand ψ	59

Tabellenverzeichnis

3.1	Zusammenfassung der Parameter des vorgestellten Frameworks. Die angegebenen Werte werden für die Modellierung der Modelle aus Kapitel 4 verwendet.	45
4.1	Vergleich der charakteristischen Längen zwischen den realen Objekten und den digitalen Modellen (Werte in mm).	49
4.2	Abbildungen der verwendeten Objekte und ihrer Modelle. Alle dargestellten Modelle wurden mit einem Zentriwinkel von $\psi = 1^\circ$ erstellt.	53
4.3	Berechnung des optimalen Zentriwinkels ψ_0 bei unterschiedlichen Gewichtungen.	60

1 Einführung

Die Entwicklung der Microsoft Kinect startete eine Revolution in der Spieleindustrie, vgl. [1]. Mit Hilfe der Kinect ist es möglich neue Arten der Interaktion zwischen Spielern und der virtuellen Welt umzusetzen, die ohne die Verwendung des klassischen Controllers funktioniert. Diese neue Art der Interaktion ist möglich, indem die Kinect zusätzlich zu einem Farbbild auch Tiefeninformation bereitstellt, wobei jedem Punkt des Farbbildes zusätzlich der Abstand zum Sensor hinzugefügt wird. Durch die Verfügbarkeit der Tiefeninformation ist es möglich Spieler von der Umgebung zu segmentieren und auf deren Bewegungen zu reagieren. Obwohl die Microsoft Kinect in erster Linie für die Spieleindustrie entwickelt wurde, eröffnete die schnelle, einfache und vor allem günstige Möglichkeit Tiefeninformation zu erhalten neue Möglichkeiten.

Seit der Verfügbarkeit von günstigen und flexiblen RGB-D Sensoren steigt das Interesse 3D-Modelle von realen Objekten zu erzeugen, wobei ein typisches Einsatzgebiet die Robotik ist. Ein Service Roboter, der möglichst selbstständig Tätigkeiten im Haushalt oder im Büro erledigen soll, steht oft vor dem Problem in einem unbekanntem Raum, unbekanntem Objekten zu begegnen. Falls der Roboter die Möglichkeit hat neue Objekte, in Form von 3D Modellen, zu lernen, hilft diese Information später beim Erkennen, beim Lokalisieren und beim Greifen von Objekten, vgl. [2]. 3D-Modelle werden aber auch abseits der Robotik eingesetzt. Modelle können in der Filmindustrie verwendet werden, um reale Objekte besser in virtuellen Welten einzubetten. Sie werden im Bereich der Architektur verwendet, wenn entwickelte Prototypen digitalisiert werden sollen und sie können in Kombination mit einem 3D-Drucker dazu dienen Objekte zu vervielfältigen.

Die Einsatzgebiete für 3D-Modelle sind vielfältig, jedoch sind klassische Scannersysteme meist teuer und unflexibel. Die meisten verwendeten Laserscanner sind relativ groß, schwer und zusätzlich ist der Scanvorgang oftmals, aufgrund der geringen Scanrate, langsam.



Abbildung 1.1: Fotografie eines Objekts, das zur Modellierung verwendet wurde.



Abbildung 1.2: Erzeugtes Modell des in Abbildung 1.1 dargestellten Objekts.

1.1 Problemstellung

Im Zuge dieser Diplomarbeit wird ein Framework entwickelt, mit dem es möglich ist dreidimensionale Modelle von realen Objekten mit Hilfe einer kostengünstigen RGB-D Kamera vollautomatisch zu erzeugen. Dabei wird besonders auf die hohe Bildrate der RGB-D Sensoren eingegangen. Bei der Erstellung des Modells wird davon ausgegangen, dass die RGB-D Kamera in einer kreisförmigen Bewegung um das zu modellierende Objekt bewegt wird. Das Objekt steht dabei auf dem Boden und es sind keine weiteren Objekte in der Nähe. Es werden keine zusätzlichen Sensoren verwendet und so muss die Bewegung der Kamera alleine aus den erzeugten Aufnahmen geschätzt werden. Zusätzlich muss das Objekt von seiner Umgebung segmentiert werden, um abschließend zu einem Gesamtmodell zusammengefügt werden zu können.

1.2 Lösungsansatz

Damit alle Aufnahmen, die von der RGB-D Kamera geliefert werden für weitere Berechnungen verwendet werden können, wird ein Framegrabber entwickelt der alle Aufnahmen speichert. Diese sukzessiven Aufnahmen werden verwendet, um die Bewegung der Kamera mittels DVO-Algorithmus zu schätzen, vgl. [3], [4]. Zur Erhöhung der Genauigkeit der Bewegungsschätzung werden Schlüsselbilder verwendet und eine, auf das Problem angepasste, Lösung des Schleifenschlusses umgesetzt. Dabei vereinfacht die angenommene kreisförmige Bewegung die Lösung erheblich. Das Gesamtmodell wird durch Oberflächenelemente dargestellt und sukzessiv mit jeder verwendeten Aufnahme erweitert. Dabei wird jede Aufnahme, bevor sie in das Gesamtmodell integriert wird, vorbereitet um typischen Problemen der Microsoft Kinect entgegenzuwirken. Während der

Erweiterung des Modells werden Ausreißer gesucht und durch zwei unterschiedliche Ansätze gelöscht. Abschließend wird in einem Nachbearbeitungsschritt die Oberfläche geglättet und redundante Oberflächenelemente aus dem Modell gefiltert. Ein erstelltes Modell ist in Abbildung 1.2 und eine Fotografie des Objekts in Abbildung 1.1 dargestellt.

In Kapitel 2 wird der Stand der Technik dargestellt und bereits vorhandene Frameworks zur Modellierung von Objekten vorgestellt. Anschließend wird in Kapitel 3 die Umsetzung des Lösungsansatzes detailliert erklärt, wobei zuerst die Aufnahme der Daten erklärt wird, anschließend die Schätzung der Kamerabewegung erläutert wird und am Ende die Erstellung des Gesamtmodells vorgestellt wird. In Kapitel 4 wird die Genauigkeit des vorgestellten Frameworks untersucht und eine Möglichkeit entwickelt systematisch die Anzahl der verwendeten Aufnahme in Bezug auf die Rechenzeit, die Detailrate und die Fehlerrate auszuwählen. Abschließend wird in Kapitel 5 die Arbeit kurz zusammengefasst und ein Ausblick für zukünftige Arbeiten gegeben.

2 Stand der Technik

Der Weg vom realen Objekt zu einem 3D-Modell wird typischerweise in drei Schritte eingeteilt. Zu Beginn müssen die Rohdaten des Objektes aufgenommen werden, anschließend müssen diese Rohdaten in ein gemeinsames Koordinatensystem transformiert werden, um abschließend das Modell generieren zu können.

2.1 Aufnahme der Rohdaten

Für den ersten Schritt, die Aufnahme von Datenpunkten, gibt es unterschiedliche Techniken, wobei in der Objektmodellierung oftmals ein Laserscanner als Sensor verwendet wird.

Laserscanner können in Time-of-Flight (TOF) Sensoren und Triangulationssensoren unterschieden werden. TOF Sensoren emittieren einen Laserstrahl und messen die Zeit, die das Licht benötigt, um vom Objekt reflektiert und wieder zum Sensor zurückzugelangen. Diese gemessene Zeit wird anschließend umgerechnet, um den Abstand des Objekts zum Sensor zu bestimmen. Zusätzlich gibt es auch die Möglichkeit die Phasenverschiebung zwischen dem ausgestrahlten und den reflektierten Licht zu verwenden, um die Messgenauigkeit weiter zu erhöhen. Der erste TOF-Sensor wurde bereits 1983 erfunden, vgl. [5]. Für eine genaue Beschreibung von TOF-Sensoren sei auf [6] verwiesen.

Sensoren, die mit Triangulation den Abstand bestimmen, messen einen Winkel zwischen einem ausgesendeten und einem reflektierten Laserstrahl. Mittels der bekannten Geometrie zwischen Sender und Empfänger kann auf den Abstand eines Objekts geschlossen werden. Der einfachste Fall ist in Abbildung 2.1 dargestellt, vgl. [7]. Es wird ein Laserstrahl ausgesendet und dessen Reflexion wird von einer eindimensionalen Kamera beobachtet. Die Länge L lässt sich mittels

$$L = \frac{B}{\tan(\alpha - \gamma)} \quad (2.1)$$

bestimmen. Dabei ist B die Basisbreite, also der Abstand zwischen dem zentralen Punkt der Linse und dem Laserstrahl, α ist der Winkel zwischen der optischen Achse der Kamera und des reflektierten Laserstrahls und γ kann mit

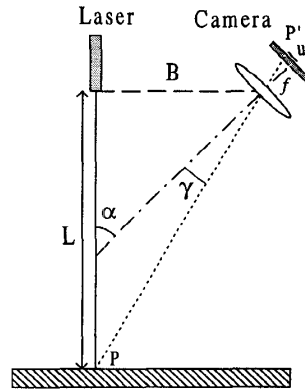


Abbildung 2.1: Einfaches Triangulationssystem, wodurch, mit Hilfe eines Laserstrahls und einer Kamera, der Abstand L bestimmt werden kann, vgl. [7].

der gemessenen Position u des Laserstrahls über

$$\gamma = \arctan\left(\frac{u}{f}\right), \quad (2.2)$$

mit der Brennweite f , berechnet werden.

Eine detaillierte Erklärung des Prinzips ist in [7] zu finden und unterschiedliche Umsetzungen des Prinzips wurden in [7]–[11] durchgeführt. In dieser Arbeit wird ein Konica Minolta VI-9i Laserscanner zum Erstellen eines Referenzobjekts verwendet, vgl. [12]. Dieser Scanner verwendet eine Laserlinie, mit der das Objekt durch eine vertikale Bewegung gescannt wird. Zusätzlich kann über einen RGB-Filter ein Farbbild des Objekts aufgenommen werden, das zur Bereitstellung von Textur verwendet werden kann.

Die Idee der Triangulation wird ebenso bei Stereokameras verwendet, um Tiefeninformation zu erzeugen. Eine klassische Stereokamera besteht aus mindestens zwei Kameras, wobei diese Kameras unterschiedliche Ansichten aufnehmen. Die Geometrie zwischen diesen Kameras ist bekannt. Bevor auf den Abstand zwischen Bildpunkt und Kamera geschlossen werden kann, muss die Korrespondenz zwischen Bildpunkten der unterschiedlichen Aufnahmen gefunden werden. Diese Korrespondenz wird oftmals unter Verwendung der Epipolargeometrie untersucht. Durch die Epipolargeometrie ist es möglich die Effizienz der Suche von korrespondierenden Punkten wesentlich zu erhöhen, indem der Suchbereich auf eine Linie reduziert wird, vgl. [13]. Typischerweise kommen bei Stereokameras zwei gleichwertige Kameras zum Einsatz, vgl. [14]. Jedoch hat die Kalibrierung der Kameras maßgeblichen Einfluss auf die Genauigkeit des Sensors. Alternativ dazu gibt es Entwicklungen die Stereoinformation mit nur

einer Kamera aus einer Sequenz von Bildern erzeugen können, vgl. [15], [16]. Neuere Entwicklungen kombinieren Stereokameras mit anderen Sensortypen um die jeweiligen Schwächen auszugleichen, vgl. [17].

Als Alternative zu den Laserscannern und Stereokameras sind in den letzten Jahren RGBD-Kameras verstärkt in den Fokus gerückt. Eine RGBD-Kamera erzeugt, im Vergleich zur einer klassischen Fotokamera, eine zweieinhalb dimensionale Punktwolke. Jeder Punkt dieser erzeugten Punktwolke enthält Farbinformation (=RGB) und den Abstand zwischen Kamera und Punkt (=D). Eine Möglichkeit den Abstand von einem Objekt zur Kamera zu bestimmen, ist durch die Verwendung von speziell strukturiertem Licht gegeben. Dabei wird ein bekanntes Muster aktiv ausgesendet und aufgrund der Reflexion an Objekten verzerrt. Diese Verzerrung wird verwendet, um, mittels trigonometrischen Überlegungen, auf den Abstand des Objektes zu schließen, vgl. [18], [19]. Es gibt eine Vielzahl sogenannter RGBD-Kameras, die dieses Prinzip verwenden. In dieser Arbeit wird eine Microsoft Kinect verwendet, um die Datenpunkte aufzunehmen. Die Microsoft Kinect besteht aus einer RGB-Kamera, einem IR-Projektor und einer IR-Kamera. Der Abstandssensor besteht aus dem IR-Projektor, der mit der Hilfe eines Beugungsgitters ein zufälliges Punktmuster erzeugt. Dieses zufällige Punktmuster wird an Objekten der Szene verzerrt und durch den Vergleich mit dem unverzerrten Muster ist es möglich die Tiefeninformation zu bestimmen. Da die Geometrie zwischen IR-Projektor, IR-Kamera und RGB-Kamera bekannt ist, kann die Tiefeninformation mit dem Farbbild der RGB-Kamera in Einklang gebracht werden und es entsteht eine zweieinhalb dimensionale Punktwolke, vgl. [1].

2.2 Visuelle Odometrie

Ein weiteres Problem, das gelöst werden muss, um ein 3D-Modell zu erzeugen, ist die Bestimmung der Transformation zwischen den einzelnen Aufnahmen, um sie in ein gemeinsames Koordinatensystem transformieren zu können. Die Bestimmung der Position und Orientierung der Kamera aus den aufgenommenen Bildern bezeichnet man als visuelle Odometrie, vgl. [20]. Es existieren viele unterschiedliche Algorithmen, um das Problem zu lösen. Eines der bekanntesten ist der Iterative-Closest-Point (ICP) Algorithmus. Ursprünglich wurde dabei der geometrische Fehler zwischen zwei Punktwolken minimiert und so auf die Transformation und Rotation geschlossen, vgl. [21]. In den Jahren sind viele unterschiedliche Varianten entwickelt worden, bei denen der geometrische Fehler zu einem verallgemeinerten Fehlermaß adaptiert wurde.

Die Unterschiede in den existierenden Umsetzungen liegen im Bereich

- der Auswahl der korrespondierenden Punkten,
- der Abstimmung zwischen korrespondierenden Punkten,
- der Gewichtung von korrespondierenden Punkten,
- des Ignorieren von korrespondierenden Punktpaaren,
- des Fehlermaßes das verwendet wird und
- der Minimierung des Fehlers.

Levoy u.a. vergleichen ausführlich die unterschiedlichen Umsetzungen, vgl. [22]. Ein anderer Ansatz das Problem der visuellen Odometrie zu lösen basiert auf der Berechnung von Features, die in den unterschiedlichen Aufnahmen eindeutig bestimmt werden können. Die Features beschreiben aussagekräftige Punkte in den unterschiedlichen Aufnahmen und machen es möglich gleiche Punkte in unterschiedlichen Aufnahmen zu finden. Dadurch ist es möglich die Transformation zwischen diesen Punkten, und somit die Transformation der Kamera, zu berechnen. Oftmals wird dieser Ansatz verwendet um Aufnahmen grob zu positionieren, bevor mit ICP die exakte Transformation bestimmt wird, vgl. [22]. Einige exemplarische Umsetzungen sind in [23]–[25] zu finden.

In dieser Arbeit wird das Problem der visuellen Odometrie mit dem DVO-Algorithmus, [3], [4] gelöst. Im Gegensatz zum ICP-Algorithmus, bei dem ein geometrischer Fehler minimiert wird, und dem Ansatz über Features, bei dem nur wenige Punkte verwendet werden, wird beim DVO-Algorithmus der photometrische Fehler minimiert. Das Verfahren ist sehr schnell, obwohl alle Datenpunkte der RGB-D Aufnahmen zur Bestimmung der Kamerabewegung verwendet werden. In Kombination mit einem Schlüsselbildalgorithmus können die gesuchten Transformationen sehr genau bestimmt werden. Eine genauere Beschreibung des Verfahrens ist in Abschnitt 3.2 zu finden.

2.3 Frameworks zur Modellerstellung

Der letzte Schritt auf dem Weg vom Objekt zum 3D-Modell liegt in der Verarbeitung der aufgenommenen Rohdaten.

Prankl u.a. [26] entwickelten eine Modellierungspipeline mit der es möglich ist, vollständige Modelle, durch Zusammensetzen von Teilmodellen, zu erzeugen. Dazu wird eine auf Features basierte Kameraverfolgung, erweitert um



Abbildung 2.2: Aufnahme eines vollständigen Modells durch Verwendung von drei unterschiedlichen Posen, vgl. [26].

einen Schlüsselbildalgorithmus mit Schleifenschlusserkennung, verwendet. Ein Nachbearbeitungsschritt ist notwendig um Rauschen und andere Artefakte zu reduzieren. Durch die Aufnahme des Objektes aus unterschiedlichen Posen ist es möglich ein vollständiges, wasserdichtes 3D Modell vollautomatisch zu erzeugen. Abbildung 2.2 zeigt die Aufnahme eines Modells durch unterschiedliche Posen des Objekts. Die Aufnahmen können vollautomatisch zu einem Modell vereint werden.

Weise u.a. [27] verwenden einen anderen Ansatz. Durch den, von ihnen vorgestellten, Online-Schleifenschlussalgorithmus ist es möglich das finale Modell direkt während der Aufnahme dem Benutzer sichtbar zu machen. Der akkumulierte Fehler, der während der Schätzung der Kameraposition passiert, wird direkt während der Aufnahme korrigiert, indem das Modell mit Hilfe starrer Transformationen verändert wird. Es wird ein ICP Algorithmus verwendet, um die Bewegung des Objektes zu verfolgen. Das Modell wird durch Oberflächenelementen dargestellt, die mit jeder neuen Aufnahme verfeinert und erweitert werden. Durch diesen Ansatz ist es möglich Rauschen und unerwünschte Artefakte bereits während der Aufnahme zu erkennen und zu reduzieren. Mit diesen Neuerungen ist ein Nachbearbeitungsschritt nicht mehr notwendig und der Benutzer sieht bereits während der Aufnahme das endgültige Modell. Zum Aufnehmen eines Modells muss der Benutzer das Objekt vor dem Scannersystem drehen, siehe Abbildung 2.3.

KinectFusion [28], [29] macht es möglich ganze Räume, mit der kostengünstigen Microsoft Kinect, zu modellieren. Zur Verfolgung der Kamera wird nur die Tiefeninformation genutzt und die Szene wird Aufnahme für Aufnahme erwei-

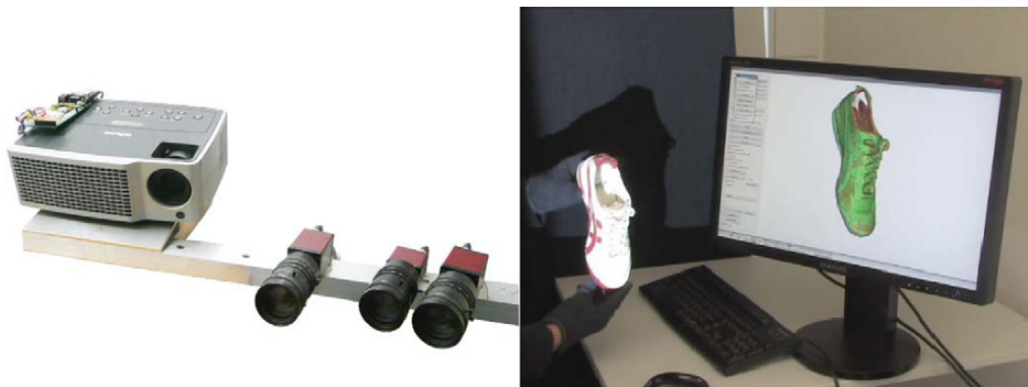


Abbildung 2.3: (links) Darstellung des verwendeten Echtzeit 3D-Scannersystem. (rechts) Interaktiver Scan eines Schuhs, vgl. [27]

tert. Um einzelne Objekte modellieren zu können, benötigt man als Benutzer zwei Schritte. Zuerst muss die komplette Szene aufgenommen werden und anschließend muss das Objekt physikalisch in der Szene bewegt werden, damit es von KinectFusion segmentiert und separat modelliert wird. In Abbildung 2.4 ist eine Szene dargestellt in der die dargestellte Teekanne modelliert wird. Zu Beginn wird die statische Szene aufgenommen und bei Bewegung des Objekts, wird es von seiner Umgebung segmentiert. Dadurch ist es möglich einzelne Objekte zu modellieren. Durch die Einführung einer neuen GPU Pipeline erfolgen die Berechnungen in Echtzeit und es ist möglich Interaktion mit dem Benutzer umzusetzen. Zusätzlich zum Modellieren von Szenen und Objekten kann Augmented-Reality geschaffen werden, bei der echte Geometrie und physikalisch korrekte Interaktionen möglich sind.

Ein andere Ansatz der unter dem Namen Structure from Motion (SfM) bekannt ist, verwendet klassische RGB-Kameras um Objekte zu modellieren. Oftmals werden Features verwendet um die Bewegung der Kamera relativ zum Objekt zu berechnen. Somit ist es nur möglich Objekte, die über genügend Textur verfügen, zu modellieren. Es gibt einige frei zugängliche Umsetzungen, wobei als Beispiel 123D Catch von Autodesk genannt sei, vgl. [30]. Der Vorteil von 123D Catch liegt vor allem in der einfachen Bedienung und der sehr geringen Anforderung an die verwendete Hardware. Zum Erstellen von 3D-Modellen ist eine einfache Kamera, wie sie heutzutage in vielen Smartphones zu finden ist, ausreichend. Der Benutzer erstellt lediglich die Aufnahmen und die restliche Modellierung funktioniert vollautomatisch, vgl. [30]. Ein Nachteil von typischen SfM Modellen ist, dass vollständige Umgebungen modelliert werden und keine automatische Segmentierung eines Objekts erfolgt. Wenn ein einzelnes Objekt modelliert werden soll, muss es nach der Berechnung manuell aus der Aufnahme

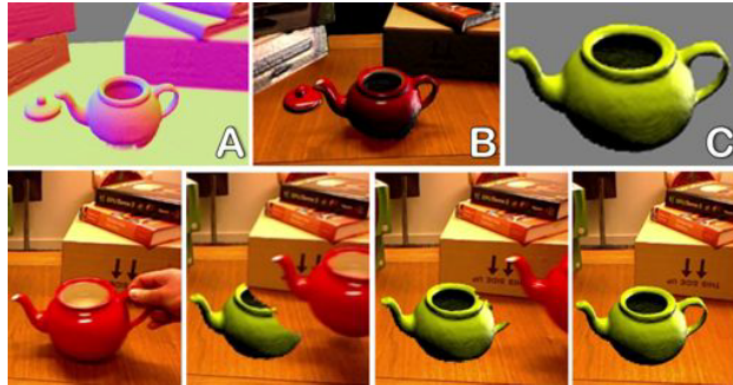


Abbildung 2.4: Schnelle und direkte Objektsegmentierung mit KinectFusion. (A) Oberflächennormalen der Szene, (B) Szene mit Textur, (C) Modell der Teekanne. (unten) Teekanne wird physikalisch bewegt um das Objekt zu segmentieren und anschließend zu modellieren, vgl. [28].

segmentiert werden. SfM findet auch Einsatz im Bereich der Robotik. Collet u.a. entwickelten ein Framework, das es möglich macht aus mehreren einzelnen Aufnahmen ein Modell zu erstellen und verbindet dieses Framework mit dem Einsatzgebiet eines Roboters im Haushalt, vgl. [31]. Die gewonnenen Modelle ermöglichen es Objekte exakter zu lokalisieren und es wird somit ein Problem beim Greifen von Objekten gelöst.

3 Objektmodellierung mit RGB-D Sensor

In diesem Kapitel wird das umgesetzte Framework zum Modellieren von Objekten erklärt. Es wird die verwendete Kamera erklärt, der verwendete Kameratracking Algorithmus näher erläutert und die Schritte vorgestellt, die unternommen werden, um ein Modell aus vielen unterschiedlichen Aufnahmen zu generieren.

3.1 Aufnahme der Grunddaten

Für die Aufnahme der Grunddaten wurde eine Microsoft Kinect verwendet. Mit Hilfe der Kinect können Spieler mit der Spielekonsole kommunizieren ohne den klassischen Controller verwenden zu müssen. Das einzigartige an der Kinect ist, dass sie dem Computer direkt die dritte Dimension - die Tiefeninformation - zusätzlich zu einem Farbbild bereitstellt. Diese Eigenschaft hat das Einsatzgebiet der Microsoft Kinect weit über die Spieleindustrie hinaus interessant gemacht. Durch die große erzeugte Stückzahl stellt sie eine kostengünstige Alternative für Abstandssensoren in vielen Anwendungen dar, vgl. [1].



Abbildung 3.1: Abbildung eines Microsoft Kinect Sensors und ihrer Komponenten, vgl. [1].

In Abbildung 3.1 ist die Microsoft Kinect dargestellt. Sie besteht im wesentlichen aus drei einzelnen Sensoren - einem Abstandssensor, einer Farbkamera

und einem Mikrofonarray, bestehend aus vier Mikrofonen. Da für diese Arbeit nur die optischen Sensoren verwendet werden, wird auf das Mikrofonarray nicht weiter eingegangen und für weitere Informationen auf [32] verwiesen. Die Lizenz für die Abstandserkennung unterlag der Firma PrimeSense, die 2013 von Apple gekauft wurde. Der exakte Algorithmus zur Bestimmung der Tiefeninformation ist nicht zugänglich, jedoch ist sicher, dass er auf dem Prinzip des strukturierten Lichts basiert. Die Tiefeninformation wird mittels eines Infrarotprojektors und einer Infrarotkamera gewonnen. Der IR-Projektor besteht aus einem IR-Laser und einem Beugungsgitter, die gemeinsam ein nahezu zufälliges Punktmuster projizieren. Da das projizierte Muster und die Geometrie zwischen dem IR-Projektor und der IR-Kamera bekannt ist, kann der Abstand eines Punktes berechnet werden, vgl. [1]. Der berechnete Abstand jedes Bildpunktes wird mit dem Bild der Farbkamera verknüpft und es entsteht ein 2.5 dimensionales Bild, das Punktwolke genannt wird.

Der Algorithmus, um Tiefeninformation zu berechnen, ist sehr schnell und so ist die Kinect fähig Punktwolken in einer Rate von 30 Bilder pro Sekunde aufzunehmen. Der Nachteil des Kinect Sensors ist, dass die gewonnenen Daten wesentlich stärkeres Rauschen aufweisen verglichen mit Daten eines Laserscanners. Außerdem ist die Tiefeninformation an Kanten oftmals falsch oder nicht vorhanden. Eine Aufnahme einer Microsoft Kinect ist in Abbildung 3.2 dargestellt. Man erkennt in den markierten Bereichen, dass die Tiefeninformation nicht vorhanden ist. In Abbildung 3.3 ist zu erkennen, dass bei Kanten die Tiefeninformation nicht exakt mit der Farbinformation übereinstimmt. Diese Fehler können im Idealfall mit einer perfekten Kalibrierung vermieden werden. In dieser Arbeit wird dieses Problem jedoch anders gelöst, siehe Abschnitt 3.3.2.



Abbildung 3.2: Aufnahme mit einer Microsoft Kinect. Im markierten Bereich ist keine Tiefeninformation verfügbar.



Abbildung 3.3: Detailansicht einer Aufnahme. An Kanten ist die Tiefeninformation oftmals falsch.

Da das Ziel dieser Arbeit ist möglichst alle, von dem Sensor bereitgestellten Punktwolken zum Erstellen des Modells zur Verfügung zu haben, wurde ein Framegrabber basierend auf der Implementierung der OpenNI Schnittstelle der

Point Cloud Library (PCL) erstellt. OpenNI ist ein Interface, das es möglich macht Daten von PrimeSense basierten Sensoren zu verwenden. Das Interface unterstützt neben den PrimeSense Sensoren, die Microsoft Kinect und die Asus Xtion, vgl. [33]. PCL ist eine Bibliothek die viele aktuelle Algorithmen zur Bearbeitung von Punktwolken und 3D-Daten im Allgemeinen bereitgestellt. Die PCL beinhaltet Algorithmen im Bereich der Filterung, der Feature Berechnung, der Oberflächenrekonstruktion, der Registrierung, der Modelleinpassung und der Segmentierung, vgl. [34]. Damit es möglich ist die bereitgestellten 30 Punktwolken pro Sekunde zu speichern, werden alle Punktwolken in eine Warteschlange im Arbeitsspeicher zwischengespeichert und parallel auf die Festplatte geschrieben. Dies ist notwendig, da handelsübliche Festplatten zu geringe Schreibgeschwindigkeiten aufweisen um die Datenrate der Kinect direkt zu speichern. Für die weitere Verwendung der Aufnahme werden die abgespeicherten Punktwolken von der Festplatte geladen. Es wird somit nicht direkt mit dem Datenstream der Kamera gearbeitet. Das hat zwei wesentliche Vorteile. Erstens müssen alle weiteren Berechnung, wie Kameratracking und Modellerzeugung, nicht in Echtzeit geschehen. Zweitens kann immer mit den gleichen Daten getestet werden, ohne immerzu das Modell aufzunehmen.

3.2 Kameratracking

Nachdem die einzelnen Aufnahmen auf der Festplatte gespeichert wurden, ist die nächste zu lösende Aufgabe die visuelle Odometrie. Das Ziel der visuellen Odometrie ist die Bewegung der Kamera ausschließlich aus der Bildinformation zu schätzen. Dabei werden meistens zwei aufeinanderfolgende Aufnahmen des Videostreams miteinander verglichen und eine starre Transformation gefunden die eine Fehlerfunktion minimiert, vgl. [20]. Nachdem die Transformation zwischen den Aufnahmen gefunden wurde, können alle Aufnahmen in ein gemeinsames Koordinatensystem transformiert werden. Dieser Schritt ist notwendig um den Datenpunkten einen örtlichen Zusammenhang zu geben, der notwendig ist um ein Modell aus den Aufnahmen erzeugen zu können.

In dieser Arbeit wird vereinfachend angenommen, dass sich die Kamera in einer kreisförmigen Bewegung um das zu modellierende Objekt bewegt - siehe Abbildung 3.4.

Diese Einschränkung unterscheidet den entwickelten Algorithmus von allgemeinen SLAM (Simultaneous Localization and Mapping) Anwendungen, bei denen eine Lokalisierung der Kamera im unbekanntem Raum versucht wird, ohne die Bewegung einzuschränken, vgl. [35]. Durch die getroffene Einschränkung können spezielle Lösungen für die Schlüsselbildauswahl, siehe Abschnitt 3.2.2 und die Suche des Schleifenschlusses, siehe Abschnitt 3.2.3 umgesetzt werden.

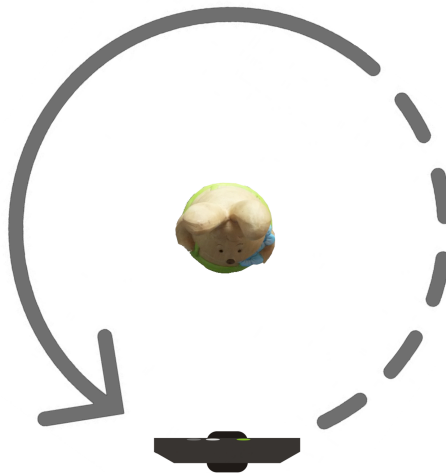


Abbildung 3.4: Angenommene Bewegung der Kamera um das Objekt.

3.2.1 DVO - Dense Visual Odometry

Zur Lösung der visuellen Odometrie wird die Umsetzung des, von Christian Kerl u.a. entwickelten, DVO-Algorithmus zum Schätzen der Transformation zwischen zwei RGB-D Aufnahmen verwendet, vgl. [3], [4]. Zu jedem Zeitpunkt t liefert die Microsoft Kinect eine Punktwolke. Für den DVO Algorithmus wird das Farbbild zu einem Grauwertbild \mathcal{I}_t umgerechnet und die Tiefeninformation gesondert in einem zweiten Grauwertbild \mathcal{Z}_t gespeichert.

Abbildung 3.5 zeigt die Grundidee des Algorithmus. Für den 3D Punkt \mathbf{p} sind die Pixelkoordinaten \mathbf{x} im Bild \mathcal{I}_1 bekannt. Mittels der optimalen Transformation g^* können die Pixelkoordinaten \mathbf{x}' im Bild \mathcal{I}_2 berechnet werden. Im Idealfall gilt für die Intensitätswerte der berechneten Positionen $\mathcal{I}_1(\mathbf{x}) = \mathcal{I}_2(\mathbf{x}')$. Die Gleichheit der Intensitätswerte ist bekannt als *photo-consistency*. Diese Gleichheit kann für jeden Punkt \mathbf{p} eintreten, falls der Sensor rauschfrei ist, die Szene sich zwischen t_1 und t_2 nicht ändert und die Beleuchtung konstant ist. Für reale Sensoren und Aufnahmen existiert keine Transformation g^* , die die *photo-consistency* Bedingung für alle Punkte erfüllt. Es ist aber möglich eine Approximation g für g^* zu finden indem man die *photo-consistency* maximiert. Dies ist gleichbedeutend damit, dass man den photometrischen Fehler $r_{\mathcal{I}}$ minimiert. Für die Tiefeninformation \mathcal{Z}_t lassen sich ähnliche Überlegungen durchführen und führen zum geometrischen Fehler $r_{\mathcal{Z}}$. Die Herleitung beider Fehler sei nun zusammengefasst und für nähere Details sei auf [3], [4] verwiesen. Für alle weiteren Betrachtungen wird ein rechtsdrehendes orthonormales Koordinatensystem, das von der Kinect vorgegeben wird, verwendet. Dabei verläuft

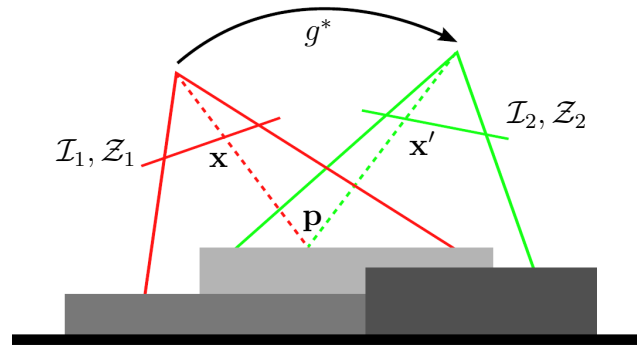


Abbildung 3.5: Grundidee des DVO-Algorithmus. Die Transformation g^* wird geschätzt indem der photometrische und der geometrische Fehler minimiert werden, vgl. [4].

die x -Achse entlang der Sensorleiste, die y -Achse von den Kameras in Richtung Standbein und die z - Achse in Ausrichtung der Kameras, siehe Abbildung 3.6. Ein 3D-Punkt \mathbf{p} kann mittels homogenen Koordinaten als

$$\mathbf{p} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.1)$$

dargestellt werden. Dabei bezeichnen X , Y , Z die Position des Punktes im Ortskoordinatensystem. Der Punkt \mathbf{p} kann mit der Verwendung des Pinhole-Models und dem Punkt $\mathbf{x} = (x, y)^T$ einer Aufnahme, wobei x und y die Position in Pixel angeben, durch

$$\mathbf{p} = \pi^{-1}(\mathbf{x}, Z) = \begin{bmatrix} \frac{x-o_x}{f_x} Z \\ \frac{y-o_y}{f_y} Z \\ Z \\ 1 \end{bmatrix} \quad (3.2)$$

berechnet werden.

Dabei bezeichnet f_x die Brennweite in x -Richtung, f_y die Brennweite in y -Richtung, o_x die Koordinate des Kamerazentrums in x -Richtung und o_y die Koordinate des Kamerazentrums in y -Richtung. Die inverse Projektionsfunktion wurde mit π^{-1} bezeichnet. Dementsprechend kann die Position eines 3D-Punktes in der Bildebene mittels der Projektionsfunktion

$$\mathbf{x} = \pi(\mathbf{p}) = \begin{bmatrix} \frac{X f_x}{Z} + o_x \\ \frac{Y f_y}{Z} + o_y \end{bmatrix} \quad (3.3)$$

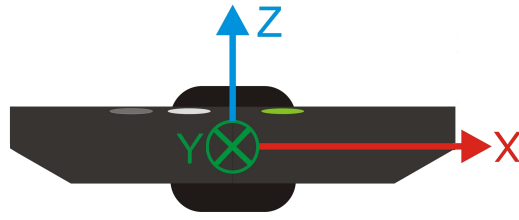


Abbildung 3.6: Koordinatensystem der Microsoft Kinect, vgl. [36].

berechnet werden. Eine starre Transformation \mathbf{T} kann in homogenen Koordinaten als

$$\mathbf{T}_{4 \times 4} = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{t}_{3 \times 1} \\ \mathbf{0} & 1 \end{bmatrix} \quad (3.4)$$

dargestellt werden. Dabei beschreibt $\mathbf{R}_{3 \times 3} \in \mathbb{R}^{3 \times 3}$ eine Rotationmatrix im dreidimensionalen Raum und $\mathbf{t}_{3 \times 1} \in \mathbb{R}^3$ einen Translationsvektor. Die gesuchte Transformation g kann nun als $g(\mathbf{p}) = \mathbf{T}\mathbf{p}$ geschrieben werden.

Durch die Kombination der Projektionsfunktion π und der Transformation \mathbf{T} lässt sich eine Funktion τ definieren, die aus einem Punkt des ersten Bildes \mathbf{x} die Position im zweiten Bild \mathbf{x}' berechnet. Es gilt

$$\mathbf{x}' = \tau(\mathbf{x}, \mathbf{T}) = \pi\left(\mathbf{T}\pi^{-1}\left(\mathbf{x}, \mathcal{Z}_1(\mathbf{x})\right)\right). \quad (3.5)$$

Somit lässt sich der photometrische Fehler als

$$r_{\mathcal{I}} = \mathcal{I}_2\left(\tau(\mathbf{x}, \mathbf{T})\right) - \mathcal{I}_1(\mathbf{x}) \quad (3.6)$$

und der geometrische Fehler als

$$r_{\mathcal{Z}} = \mathcal{Z}_2\left(\tau(\mathbf{x}, \mathbf{T})\right) - \left[\mathbf{T}\pi^{-1}\left(\mathbf{x}, \mathcal{Z}_1(x)\right)\right]_{\mathcal{Z}} \quad (3.7)$$

darstellen. Dabei beschreibt $[\cdot]_{\mathcal{Z}}$ die Z Komponente des Vektors. Mit Hilfe dieser Fehlerfunktionen wird ein Optimierungsproblem erstellt und gelöst, [4]. Das Ergebnis ist eine Transformationsmatrix \mathbf{T} , die angibt, wie sich die Kamera zwischen den Zeitpunkten τ_1 und τ_2 bewegt hat.

Mit dem erklärten Algorithmus ist es nun möglich ein sogenanntes Frame-To-Frame Kameratracking zu implementieren. Dabei wird jeweils die Transformation zwischen der aktuellen Aufnahme zum Zeitpunkt τ_t und der Aufnahme zum Zeitpunkt τ_{t-1} berechnet. Eine Gesamtbewegung wird anschließend durch viele Teilbewegungen zusammengesetzt. Das Problem an diesem Ansatz ist, dass die Transformationen zwischen den einzelnen Aufnahmen mit einem kleinen Fehler

behaftet sind. Dieser Fehler wird durch Sensorrauschen, Beleuchtungsänderungen, Ungenauigkeiten im Rauschmodell und weiteren Nichtidealitäten ausgelöst. Die Einzelfehler sind meistens sehr gering, jedoch akkumulieren sich diese Fehler bei dem Frame-To-Frame Ansatz zu einem nicht vernachlässigbaren Gesamtfehler. Dieses Problem wird oftmals als Drift bezeichnet, denn, wenn man die Kamera still hält und eine Frame-To-Frame Tracking verwendet, wird sich die berechnete Pose langsam von dem Ausgangspunkt wegbewegen, obwohl die Kamera sich nicht bewegt hat, siehe Abbildung 3.7. Für die Aufnahme von Abbildung 3.7 wurde eine statische Szene für 4 Sekunden aufgenommen und die Kamerabewegung mittels Frame-To-Frame Tracking unter Verwendung des DVO Algorithmus geschätzt. Der akkumulierte Fehler ist deutlich zu erkennen. Obwohl sich die Kamera bei der Aufnahme nicht bewegt hat, wurde eine Translation von 1.52 cm, eine Rotation um die x -Achse von 0.1° , eine Rotation um die y -Achse von 0.39° und eine Rotation um die z -Achse von 0.08° geschätzt. Dieses Problem kann durch die Einführung von Schlüsselbilder wesentlich verringert werden.

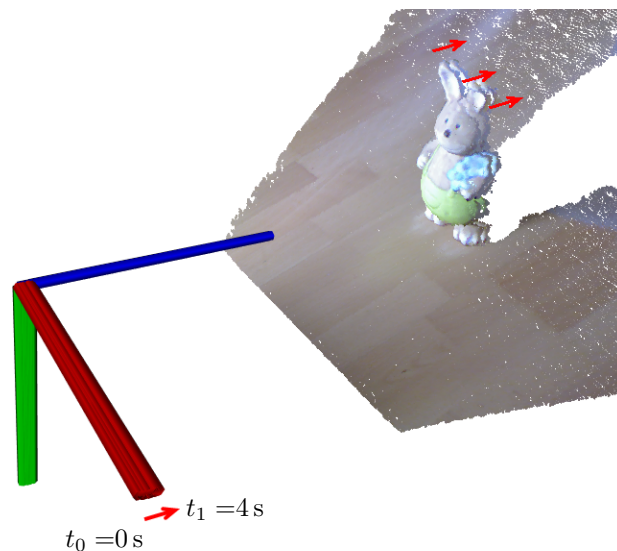


Abbildung 3.7: Frame-To-Frame Tracking einer statischen Szene. Aufgrund der akkumulierten Fehler ist ein Drift der Kameraposition zu beobachten.

3.2.2 Schlüsselbildbasiertes Kameratracking

Die Idee hinter einem Frame-To-Keyframe Kameratracking ist, dass man die Transformation der Kamera zwischen der aktuellen Aufnahme und einem Schlüsselbild schätzt. Das Schlüsselbild kann dabei zeitlich weiter als einen Zeitschritt zurückliegen. Der Vorteil liegt darin, dass, solange die Kamera in der Nähe eines Schlüsselbildes ist, der Fehler nicht akkumuliert wird. Somit entsteht bei einem Stillhalten der Kamera kein Drift, da alle Aufnahmen mit dem gleichen Schlüsselbild verglichen werden. Zusätzlich reduziert sich der Gesamtfehler bei größeren Bewegungen erheblich.

Die Aufgabe bei einem Frame-To-Keyframe Kameratracking liegt in der Auswahl der Schlüsselbilder. Es sollten bei der Auswahl möglichst wenig Schlüsselbilder ausgewählt werden, denn umso weniger Schlüsselbilder im Bewegungsgraphen vorhanden sind, desto weniger Fehler werden akkumuliert. Jedoch dürfen die Schlüsselbilder nicht zu unterschiedlich sein, da sonst der Kameratrackingalgorithmus Schwierigkeiten hat eine verlässliche Transformation, aufgrund der geänderten Beleuchtung und der dadurch nicht mehr gültigen Fotokonsistenz, zu berechnen.

Der einfachste Ansatz bei der Auswahl von Schlüsselbildern ist, dass nach einer fixen Anzahl von Aufnahmen ein neues Schlüsselbild ausgewählt wird. Dieser Ansatz verhindert jedoch das Abdriften bei einem Stillstand der Kamera nicht vollständig - er verlangsamt es nur. Eine andere Möglichkeit ist die Anzahl von gemeinsamen Features zwischen der aktuellen Aufnahme und dem Schlüsselbild zu zählen. Wenn die Anzahl unter eine definierte Schwelle fällt, wird ein neues Schlüsselbild ausgewählt, vgl. [37]. Kerl u.a. entwickelten eine Auswahl für Schlüsselbilder die, abhängig von der Entropie, die während der Berechnung mit dem DVO Algorithmus benötigt wird, neue Schlüsselbilder auswählt, vgl. [4].

In dieser Arbeit wird ein neues Schlüsselbild ausgewählt wenn die neue Pose der Kamera eine gewisse Rotation oder Translation von dem alten Schlüsselbild entfernt ist. Dabei kann die Translation direkt aus der Transformationsmatrix bestimmt werden - siehe $\mathbf{t}_{3 \times 1}$ in Gleichung 3.4. Als Maß für die Entfernung wird die euklidische Norm

$$t = |\mathbf{t}_{3 \times 1}| = \sqrt{x^2 + y^2 + z^2} \quad (3.8)$$

verwendet. Die Rotation pro Achse kann nicht direkt aus der Transformationsmatrix abgelesen werden, da die Winkel der Rotationen in nichtlinearen Gleichungen verwendet werden. Die Matrizen

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \quad (3.9)$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (3.10)$$

und

$$\mathbf{R}_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

werden elementare Rotationsmatrizen genannt. \mathbf{R}_x entspricht einer Rotation um die x -Achse um den Winkel α , \mathbf{R}_y entspricht einer Rotation um die y -Achse um den Winkel β und \mathbf{R}_z entspricht einer Rotation um die z -Achse um den Winkel γ . Dabei ist das zugrundeliegende Koordinatensystem wie es in Abschnitt 3.2.1 definiert wurde. Die Rotationsmatrix $\mathbf{R}_{3 \times 3}$ aus Gleichung 3.4 kann nun als

$$\mathbf{R}_{3 \times 3} = \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x \quad (3.12)$$

dargestellt werden. Die nichtlinearen Gleichungen 3.12 sind schwierig zu lösen. Deswegen wurde ein andere Weg gewählt um die Rotation um die Hauptachsen zu bestimmen. Die Achsen des ersten Koordinatensystem werden \mathbf{x}_1 , \mathbf{y}_1 und \mathbf{z}_1 bezeichnet. Zur Bestimmung der Rotation um die \mathbf{x}_1 -Achse wird ein rotiertes Koordinatensystem berechnet, wobei für die Hauptachsen des neuen Koordinatensystems

$$\mathbf{x}_2 = \mathbf{R}_{3 \times 3} \mathbf{x}_1 \quad (3.13)$$

$$\mathbf{y}_2 = \mathbf{R}_{3 \times 3} \mathbf{y}_1 \quad (3.14)$$

$$\mathbf{z}_2 = \mathbf{R}_{3 \times 3} \mathbf{z}_1 \quad (3.15)$$

gilt, siehe Abbildung 3.8.

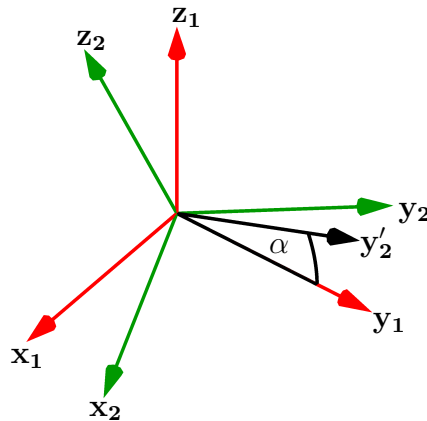


Abbildung 3.8: Rotation des neuen Koordinatensystem zur Berechnung der Rotation um die \mathbf{x}_1 -Achse.

Die Rotation um die \mathbf{x}_1 -Achse wird als α bezeichnet und es gilt

$$\alpha = \arccos(\mathbf{y}'_2 \cdot \mathbf{y}_1). \quad (3.16)$$

Dabei ist \mathbf{y}'_2 die Projektion der \mathbf{y}_2 -Achse in die Ebene die durch \mathbf{y}_1 und \mathbf{z}_1 aufgespannt wird. Es gilt

$$\mathbf{y}'_2 = (\mathbf{y}_2 \cdot \mathbf{y}_1)\mathbf{y}_1 + (\mathbf{y}_2 \cdot \mathbf{z}_1)\mathbf{z}_1. \quad (3.17)$$

Die Berechnung der Rotation um die \mathbf{y}_1 und \mathbf{z}_1 -Achse erfolgt analog. Somit ist es möglich die Rotationen zu berechnen ohne die nichtlinearen Gleichungen 3.12 lösen zu müssen. Ein neues Schlüsselbild kann abhängig von der Translation t aus Gleichung 3.8 und den Rotationen um die Hauptachsen - repräsentativ α aus Gleichung 3.16 - ausgewählt werden.

Durch die Verwendung des vorgestellten Schlüsselbildalgorithmus ist es möglich das Problem der Drift, wie es in Abschnitt 3.2.1 vorgestellt wurde, zu lösen. Abbildung 3.9 zeigt dieselbe statische Szene wie sie für die Aufnahme der Abbildung 3.7 verwendet wurde. Zum Schätzen der Kamerabewegung wurde der Schlüsselbildalgorithmus verwendet und es ist kein Drift zu beobachten. Nach 4 Sekunden wurde ein Fehler der Translation von 0.14 mm und praktisch kein Fehler der Rotation geschätzt.

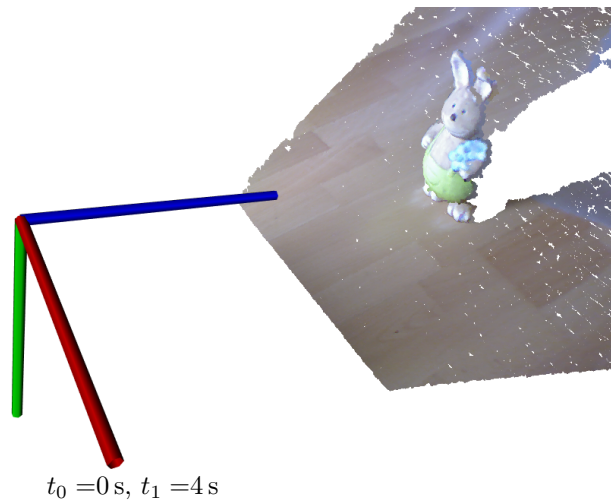


Abbildung 3.9: Frame-To-Keyframe Tracking einer statischen Szene. Es ist kein Drift zu beobachten.

Mit der Verwendung des Frame-To-Keyframe Trackings ist es möglich größere Szenen aufzunehmen. Für die Aufnahme von Abbildung 3.10 wurde die Kamera kreisförmig um das zu modellierende Objekt bewegt. Es wurden die Aufnahmen mit den berechneten Transformationen in ein gemeinsames Koordinatensystem

transformiert und einige Positionen der Kamera durch das jeweilige Koordinatensystem dargestellt. Obwohl der akkumulierte Fehler durch die Verwendung von Schlüsselbilder stark reduziert wird, kann er trotzdem nicht vollständig verhindert werden. Man erkennt den Fehler darin, dass die geschätzte Kameraposition nicht komplett zum Startpunkt zurückgekehrt ist, obwohl die Kamera einen vollständigen Kreis um das Objekt herum bewegt wurde. Dieser Fehler kann reduziert werden indem die Idee des Schleifenschlusses verwendet wird.

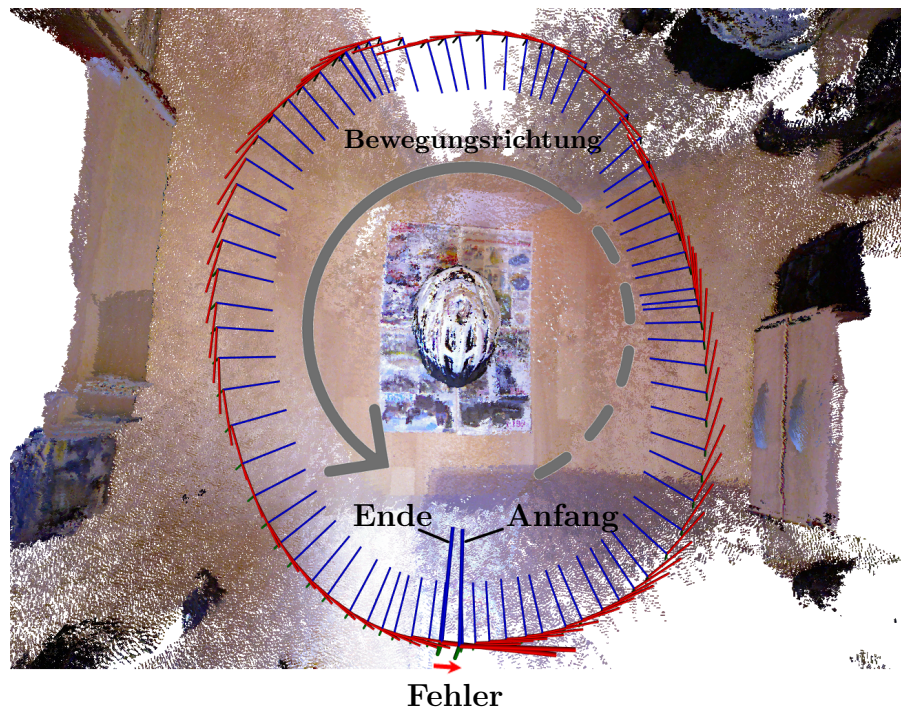


Abbildung 3.10: Akkumulierter Fehler bei vollständiger Kreisbewegung unter Verwendung von Schlüsselbilder.

3.2.3 Schleifenschluss

Im Kontext von allgemeinen SLAM Anwendungen bezeichnet der Schleifenschluss das Erkennen ob ein Fahrzeug zu einem zuvor besuchten Gebiet zurückgekehrt ist, vgl. [38]. Durch das Wissen, dass dieses Gebiet bereits erkundet wurde, kann der akkumulierte Fehler, der in der Zwischenzeit entstanden ist, behoben werden. In Bezug auf diese Arbeit lässt sich die Idee des Schleifenschlusses verstehen indem die Bewegung der Kamera als Graph interpretiert wird. In diesem Graph repräsentieren Knoten die Schlüsselbilder und Kanten

die Bewegung zwischen diesen Schlüsselbildern. Wenn die Kamera in die Nähe eines bereits beobachteten Knoten kommt, kann eine zusätzliche Kante in den Graph eingetragen werden. Die Abweichung der Position kann als Fehlerwert der neuen Kante verstanden werden und mit Hilfe von Graphenoptimierung könnte der entstandene Fehler minimiert werden. Ein Open-Source Framework, mit dem eine solche Graphenoptimierung möglich wäre, ist *g²o: A General Framework for Graph Optimization*, vgl. [39].

In dieser Arbeit wird jedoch die *Explicit Loop Closing Heuristic* (ELCH) verwendet um den akkumulierten Fehler mittels Schleifenschluss zu reduzieren, vgl. [40], [41]. Die Problemlösung ist direkter und durch die Implementierung in die PCL direkt verwendbar. Durch das Einfügen der neuen Kante ist es möglich die Transformation zwischen dem letzten Schlüsselbild und der aktuellen Aufnahme zu berechnen und zusätzlich zwischen der aktuellen Aufnahme und einem Schlüsselbild, das schon einige Zeit zuvor aufgenommen wurde. Durch diese zusätzliche Information ist es möglich den akkumulierten Fehler herauszufinden indem die Differenz zwischen den beiden Transformationen berechnet wird.

Als Beispiel sei in Abbildung 3.11 ein Bewegungsgraph dargestellt. Der Knoten *A* repräsentiert die erste Aufnahme und der Knoten *E* repräsentiert die Aufnahme, die zum Schleifenschluss geführt hat. Wenn kein Fehler akkumuliert wird, sind die Knoten *A* und *E* ident. Die Schlangenlinie repräsentiert demnach den Fehler den es zu beheben gilt.

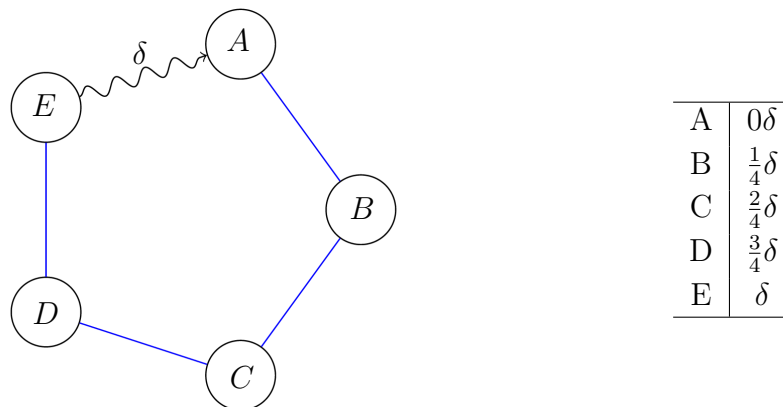


Abbildung 3.11: Bewegungsgraph zum Schleifenschluss. Knoten repräsentieren Schlüsselbilder, Kanten repräsentieren die Bewegung dazwischen. Die Werte der Tabelle zeigen die gewichteten Fehleranteile, vgl. [41].

Im ersten Schritt ist es ausreichend den Fehler δ als Skalar zu interpretieren. Damit der Fehler behoben wird muss der Knoten *E* um δ und der Knoten *A* um 0δ verschoben werden. Wenn keine Information darüber besteht wo welcher

Anteil des Fehlers gemacht wurde, ist es sinnvoll den Fehler gleichmäßig zu verteilen. Die Werte für dieses Beispiel sind in der Tabelle von Abbildung 3.11 dargestellt und repräsentativ für die Aufteilung des Fehler zu verstehen. Für eine detaillierte Beschreibung des Verfahrens sei auf [41] verwiesen.

Im tatsächlichen Bewegungsgraphen ist der Fehler nicht skalar sondern eine affine Transformation die es zu berechnen gilt. Abbildung 3.12 zeigt die Überlegung um die Fehlertransformation ΔT zu berechnen. T_1 ist die Transformation von der aktuell geschätzten Position P_1 zur Startposition P_0 . Diese Transformation ist mit dem akkumulierten Fehler behaftet und wurde durch die Bewegung der Kamera um das Objekt erzeugt. Die Transformation T_2 ist die Transformation von der ersten Position zur korrekten Position P_k . Diese Transformation definiert den Schleifenschluss und gibt die korrekte Position der Kamera an. Die Fehlertransformation lässt sich demnach als zusammengesetzte Transformation von T_1 und T_2 darstellen. Es gilt

$$\Delta T = T_2 T_1. \quad (3.18)$$

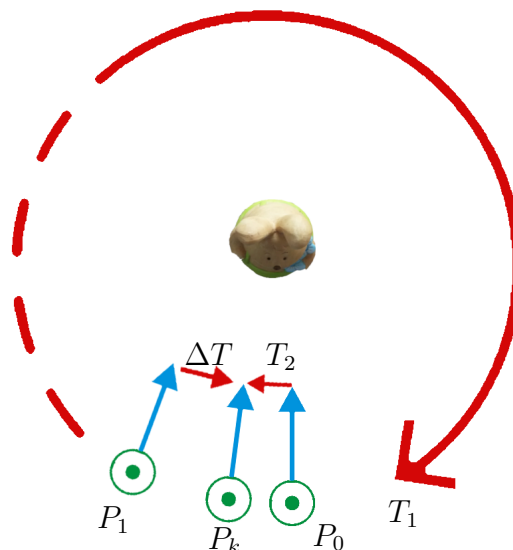


Abbildung 3.12: Berechnung der Fehlertransformation ΔT .

Bei den bisherigen Überlegungen wurde vorausgesetzt, dass der Schleifenschluss bereits erkannt wurde. Im Allgemeinen stellt das Erkennen eine schwierige und rechenintensive Aufgabe dar, denn es muss die aktuelle Aufnahme mit allen Schlüsselbildern des Bewegungsgraphen verglichen werden um Schleifen zu erkennen. Eine Möglichkeit ist es Features zu verwenden um bereits besuchte Stellen zu beschreiben, vgl. [38]. Das Erkennen des Schleifenschlusses ist

in dieser Arbeit durch die Annahme einer kreisförmigen Bewegung wesentlich vereinfacht worden, siehe Abbildung 3.4. Im Bewegungsgraphen kann nur eine Schleife entstehen und diese kann nur am Ende des Kreises erfolgen. Somit ist es ausreichend zu erkennen wenn sich die Kamera wieder in der Nähe des Startpunktes befindet. Zu diesem Zweck wird die Rotation um die y -Achse beobachtet während sich die Kamera um das Objekt bewegt. Wenn die Rotation in der Nähe von 360° ist, wird nach der Suche des tatsächlichen Schleifenschlusses begonnen. Dazu wird die Transformation zwischen der ersten und der aktuellen Aufnahme mit dem DVO Algorithmus berechnet. Es ist jedoch nicht sichergestellt, dass der Algorithmus eine sinnvolle Transformation berechnet, weil die aktuelle Aufnahme sich noch zu sehr von der ersten Aufnahme unterscheiden kann. Deswegen wird ein Schleifenschluss erst erkannt, wenn sich die berechneten Transformationen der letzten drei Aufnahmen in Translation und Rotation ähnlich sind.

Zur Implementierung wurde die Umsetzung der ELCH aus der PCL verwendet und das Ergebnis eines Schleifenschlusses ist in Abbildung 3.13 dargestellt. Im Vergleich mit Abbildung 3.10 erkennt man, dass der Fehler behoben wurde und somit die Kreisbewegung korrigiert werden konnte.

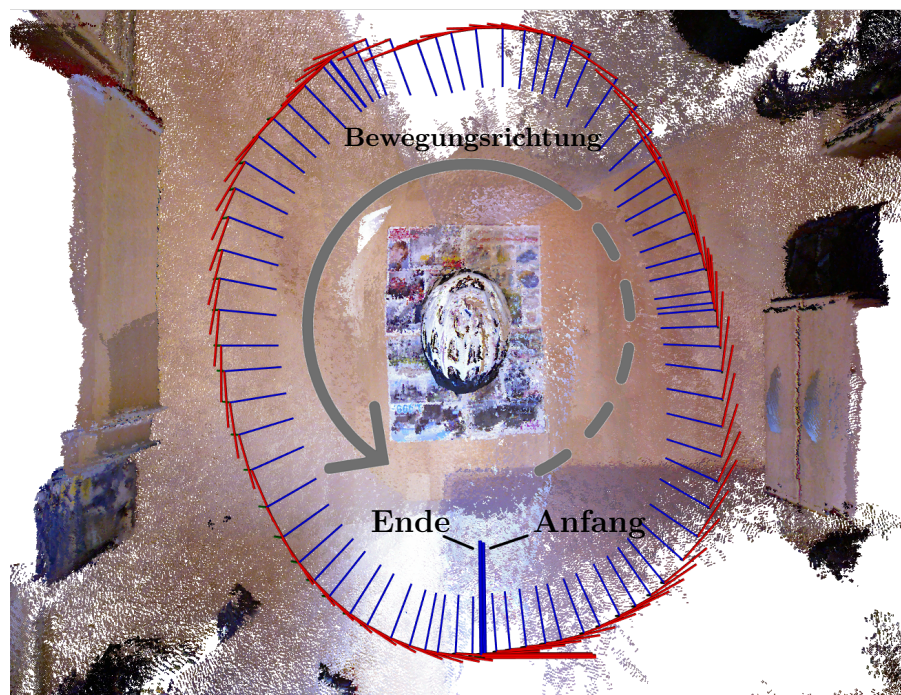


Abbildung 3.13: Schleifenschluss. Der Fehler wurde behoben, vgl. Abbildung 3.10.

3.3 Erstellung des Modells

In Abschnitt 3.2 wurde erklärt wie die einzelnen Aufnahmen in ein gemeinsames Koordinatensystem transformiert werden. Als Nächstes wird aus der großen Anzahl an Aufnahmen ein Modell generiert. Dazu muss zu Beginn geklärt werden wie das Modell repräsentiert wird, um anschließend klären zu können wie die einzelnen Elemente des Modells berechnet werden.

3.3.1 Modellrepräsentation

Zur Repräsentation eines 3D-Modells gibt es unterschiedliche Möglichkeiten. Eine mögliche Darstellungsform der Information des Modells ist die Punktwolke und wurde bereits in Abschnitt 3.1 vorgestellt. Eine Alternative, die in der Computergrafik weitverbreitet ist, ist die Darstellung der Information mittels Vertices und deren Verbindungen. Ein Vertex stellt eine Ecke eines Polygons dar. Zwei Vertices definieren somit eine Linie. Drei Vertices definieren ein Dreieck. Oftmals werden diese Dreiecke verwendet um Oberflächen von Objekten approximierend zu beschreiben. In dieser Arbeit wird das Modell, ähnlich wie in [27], als eine Menge von Oberflächenelementen (=Surfels, Surfaceelement) dargestellt. Dazu sind im Folgenden einige Definitionen angeführt.

Das zu modellierende Objekt \mathcal{M} hat die Oberfläche $\partial\mathcal{M}$, die als Menge von Oberflächenelementen \mathcal{S}_i approximiert wird. Es gilt

$$\partial\mathcal{M} \approx \{\mathcal{S}_i \mid 0 \leq i \leq N, i \in \mathbb{N}\}, \quad (3.19)$$

wobei $N \in \mathbb{N}$ die Anzahl der Oberflächenelemente darstellt. Ein Oberflächenelement \mathcal{S}_i speichert einen Positionsvektor $\mathbf{p}_i \in \mathbb{R}^3$, einen Vektor der Oberflächennormale $\mathbf{n}_i \in \mathbb{R}^3$, die Farbe $\mathbf{c}_i \in \mathbb{N}^3$ und einen Radius $r_i \in \mathbb{R}$. Die Einträge des Positionsvektors \mathbf{p}_i sind die Koordinaten des Punktes im verwendeten Koordinatensystem. Der Vektor der Oberflächennormale \mathbf{n}_i hat die Länge eins und seine Einträge geben die Richtung der Oberflächennormale an. Die Farbe des Oberflächenelements \mathbf{c}_i wird im RGB-Farbraum gespeichert und für die Einträge des Vektors gilt

$$\mathbf{c}_i = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (3.20)$$

mit

$$\begin{aligned} &\{R \in \mathbb{N} \mid 0 \leq R \leq 255\}, \\ &\{G \in \mathbb{N} \mid 0 \leq G \leq 255\}, \\ &\{B \in \mathbb{N} \mid 0 \leq B \leq 255\}. \end{aligned} \quad (3.21)$$

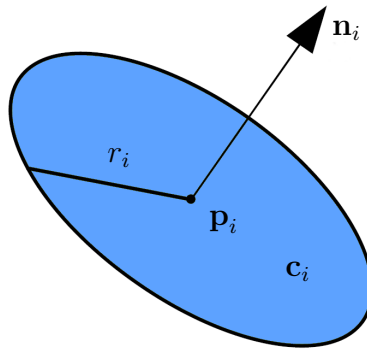


Abbildung 3.14: Darstellung eines Oberflächenelements. Dabei stellt \mathbf{p}_i die Position, \mathbf{n}_i den Vektor der Oberflächennormale, \mathbf{c}_i die Farbe und r_i den Radius des Oberflächenelements dar, vgl. [27].

Der Radius r_i definiert die Größe des runden Oberflächenelements. Mit diesen Definitionen lässt sich ein Oberflächenelement als orientierte (\mathbf{n}_i), eingefärbte (\mathbf{c}_i) Scheibe mit Radius r_i an der Position \mathbf{p}_i interpretieren, siehe Abbildung 3.14. Die Oberfläche des Objektes $\partial\mathcal{M}$ wird demnach aus einer Vielzahl von aneinandergereihten Scheiben approximiert. Die Verwendung von Oberflächenelementen bietet einige Vorteile. Einerseits ist es, im Vergleich zu Vertices und den daraus resultierenden Dreiecken, einfach die unstrukturierten Oberflächenelemente in sich konsistent zu halten. Andererseits speichern Oberflächenelemente mehr Information als reine Punkte einer Punktwolke. Somit ist es möglich die Flexibilität von einer Punktwolke mit zusätzlicher Information zu kombinieren. Der Vorteil der zusätzlichen Information liegt in der Möglichkeit diese Information durch die unterschiedlichen Aufnahmen zu mitteln und somit Rauschen und andere unerwünschte Effekte zu reduzieren.

3.3.2 Modellerstellung

Bei der Modellerstellung werden die Oberflächenelemente \mathcal{S}_i gesucht, die die Oberfläche des Objekts approximieren und somit das Modell beschreiben. In dieser Arbeit wird ein iterativer Ansatz verwendet, um das Modell aufzubauen. Das bedeutet, dass, begonnen mit der ersten Aufnahme, das Modell mit jeder verwendeten Aufnahme sukzessive erweitert wird. Dafür wird die Aufnahme vorbereitet, die Punkte in Oberflächenelemente umgewandelt und anschließend das Modell mit der neuen Aufnahme erweitert. Nachdem alle Aufnahmen in das Modell eingearbeitet wurden, wird das Modell im letzten Schritt verfeinert und eventuell redundante Oberflächenelemente gefiltert.

Für alle weiteren Betrachtungen wird angenommen, dass aus der Menge aller gespeicherten Aufnahmen \mathcal{V} eine Menge an Aufnahmen $\mathcal{V}_s = \{\mathcal{A}_\tau \mid 0 \leq \tau \leq N_s, \tau \in \mathbb{N}\}$ selektiert wird. Die Menge \mathcal{V}_s wird zur iterativen Erweiterung des Modells genutzt und die Aufnahmen \mathcal{A}_τ werden mit τ indiziert. Die Anzahl der verwendeten Aufnahmen wird mit $N_s \in \mathbb{N}$ bezeichnet. Die Auswahl der Aufnahmen der Menge \mathcal{V}_s wird in Abschnitt 3.3.3 im Detail beschrieben. Zusätzlich wird angenommen, dass bereits ein Modell $\partial\mathcal{M}_{\tau-1}$ bis zur Aufnahme $\mathcal{A}_{\tau-1}$ iterativ erzeugt wurde und im Folgenden werden die Schritte, die zum Berücksichtigen der Aufnahme \mathcal{A}_τ im Modell unternommen werden, erklärt.

Vorbereitung der Aufnahme

Der erste Schritt dient zur Vorbereitung der Aufnahme \mathcal{A}_τ . Dabei werden offensichtliche und potentiell falsche Punkte gelöscht und anschließend Oberflächennormalen berechnet, die zur Generierung von Oberflächenelementen benötigt werden. Wie bereits in Abschnitt 3.1 gezeigt wurde, kann es bei Aufnahmen, die von einer Microsoft Kinect stammen, vorkommen, dass die Tiefeninformation an Kanten oftmals falsch oder nicht vorhanden sind. Punkte in der Nähe von Sprüngen in der Tiefeninformation sind nicht vertrauenswürdig. Da in dieser Arbeit viele Aufnahmen zur Erstellung des Modells verwendet werden, ist es nicht notwendig die komplette Information einer einzelnen Aufnahme zu verwenden. Viele Punkte, die in einer Aufnahme falsch detektiert werden, sind korrekt, wenn sie von einer unterschiedlichen Richtung aufgenommen werden. Deswegen werden alle Punkte, die im Bereich eines Sprungs der Tiefeninformation liegen, gelöscht.

Zum Löschen der Punkte müssen Sprünge in der Tiefeninformation gefunden werden. Dazu wird aus der Punktwolke das Tiefenbild erstellt. Ein Beispiel einer verwendeten Punktwolke ist in Abbildung 3.15 dargestellt und das dazugehörige Tiefenbild ist in Abbildung 3.17 zu sehen. Die Kanten des Tiefenbildes sind die Sprünge der Tiefeninformation und können mit einer Canny-Kantenerkennung gefunden werden, siehe Abbildung 3.18. Es wird die Umsetzung des Canny-Algorithmus aus dem OpenCV Framework verwendet, vgl. [42], [43]. OpenCV ist eine Open Source Computer Vision Bibliothek die als Ziel hat eine effiziente, einfach zu verwendende Umgebung zu schaffen, um Computer Vision Anwendungen schnell umsetzen zu können. Da die Kanten der Canny-Kantenerkennung sehr dünn sind, werden sie mittels einer anisotropen Diffusion verbreitert, vgl. [44]. Das Ergebnis der anisotropen Diffusion ist in Abbildung 3.19 dargestellt. Die gefundenen Punkte werden anschließend aus der Punktwolke gelöscht. Im Vergleich von der aufgenommenen Punktwolke aus Abbildung 3.15 und der bearbeiteten Punktwolke, dargestellt in Abbildung 3.16, erkennt man, dass die falschen Punkte an den Kanten entfernt wurden.



Abbildung 3.15: Unbearbeitete Punktwolke wie sie von der Microsoft Kinect erzeugt wird. Im markierten Bereich erkennt man, dass falsche Punkte zum Objekt hinzugefügt wurden.



Abbildung 3.16: Ergebnis der Bearbeitung. Die falschen Punkte an den Kanten wurden entfernt.

Nachdem sichergestellt ist, dass die verbleibenden Punkte mit hoher Wahrscheinlichkeit korrekt sind, kann das Objekt von seiner Umgebung getrennt werden. Dabei wird angenommen, dass das Objekt auf dem Boden vor der Kamera steht und sich in der näheren Umgebung kein weiteres Objekt befindet. Somit ist es möglich das Objekt von seiner Umgebung zu segmentieren indem der Boden gefunden wird und die Punkte des Bodens aus der Aufnahme gelöscht werden. Zum Auffinden des Bodens wird der Random Sample Consensus Algorithmus (RANSAC) verwendet, vgl. [45]. Bei RANSAC wird ein Modell eines gesuchten geometrischen Objekts (z.B. Linie, Kreis, Ebene,...) erzeugt und die notwendigen Punkte für das Modell zufällig aus den verfügbaren Punkten ausgewählt. Mit den ausgewählten Punkten wird das geometrische Objekt erzeugt und gezählt wie viele Punkte dieses Objekt unterstützen, das heißt nahe genug beim Modell liegen. Wenn viele Punkte das Modell unterstützen, werden alle passenden Punkte verwendet, um über ein Optimierungsverfahren (z.B. kleinste Fehlerquadrate) das endgültig gefundene Objekt zu definieren. Für die Suche des Bodens wird typischerweise eine möglichst große Ebene gesucht, denn der Boden ist mit großer Wahrscheinlichkeit die größte Ebene in einem geschlossenen Raum. Für die Umsetzung wird die Implementierung von RANSAC aus der PCL verwendet. Eine Punktwolke, aus der die unerwünschten Punkte an Tiefensprüngen und vom Boden gelöscht wurden, ist in Abbildung 3.20 dargestellt. Man erkennt, dass das Objekt bereits grob segmentiert wurde. Damit

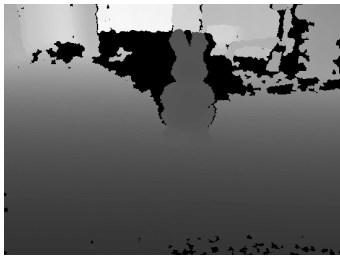


Abbildung 3.17: Tiefenbild einer Aufnahme. Hellere Bereiche sind weiter von der Kamera entfernt als dunklere Bereiche.



Abbildung 3.18: Mit der Canny-Kantenerkennung wurden die Tiefsprünge erkannt.



Abbildung 3.19: Die gefundenen Kanten werden mittels anisotroper Filterung verbreitert.



Abbildung 3.20: Mittels RANSAC wurde der Boden entfernt.

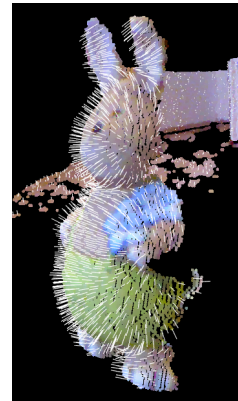


Abbildung 3.21: Vorbereitete Punktwolke mit berechneten Oberflächennormalen.

die restlichen Punkte der Umgebung nicht bei der Modellierung berücksichtigt werden, wird ein kugelförmiger ROI (Region of Interest = interessanter Bereich) definiert. Alle Punkte innerhalb des ROI werden zur Modellierung verwendet. Alle Punkte außerhalb des ROI werden verworfen. Zur Definition des ROI wird ein Mittelpunkt und ein Radius benötigt. Die Suche des Mittelpunkts wird mit der allerersten Aufnahme durchgeführt und es wird der Mittelpunkt des Tiefenbildes als Mittelpunkt der Kugel definiert. Falls dieser Punkt zu weit von der Kamera entfernt ist oder keine gültige Tiefeninformation hat, wird der Punkt in der Zeile darunter untersucht. Falls dieser ebenfalls nicht gültig ist, wird in der nächsten Zeile weitergesucht und so weiter. Somit ist sichergestellt, dass ein gültiger Punkt ausgewählt wird der nicht zum Hintergrund gehört. Der Radius des ROI wird mit einem halben Meter als konstant angenommen. Mit dieser Auswahl wurden zwei Einschränkungen in Kauf genommen. Erstens muss das Objekt in der ersten Aufnahme zentriert vor der Kamera positioniert

sein und zweitens ist die maximale Größe eines Objekts mit einem Meter im Durchmesser begrenzt. Die Punkte im ROI werden in der Menge der Punkt \mathcal{A}_{ROI} gespeichert und die Anzahl der Punkt im ROI wird mit N_{ROI} bezeichnet. Nachdem das Objekt von der Umgebung segmentiert wurde, ist werden die Punkte \mathcal{A}_{ROI} zu Oberflächenelementen umgewandelt. Dazu werden Oberflächennormalen benötigt, wobei die Berechnung mittels der Umsetzung der Normalenschätzung aus der PCL erfolgt. Dazu werden die Punkte des ROI in einen k-dimensionalen Suchbaum organisiert und über die Nachbarschaften zu anderen Punkten der Vektor der Oberflächennormale geschätzt. Das Ergebnis der Berechnung für die verwendete Beispielpunktwolke ist in Abbildung 3.21 dargestellt.

Modellerweiterung

Es gibt unterschiedliche Ansätze wie mehrere Aufnahmen in ein Modell kombiniert werden können. Prankl u.a. [26] verwenden einen Octtree der die maximale Auflösung definiert. Jeder Punkt der in einem Ast des Baumes zu liegen kommt wird gemittelt und so wird das Modell erzeugt. Weise u.a. [27] verwenden Oberflächenelemente zur iterativen Erweiterung des Modells. Bei der Integration einer neuen Aufnahme wird ein virtuelles Tiefenbild des Modells mittels Projektion auf eine Ebene, unter Verwendung der intrinsischen Parameter des Aufnahmesystems, erzeugt. Anschließend wird jedes Oberflächenelement mit dem korrespondierenden Oberflächenelement der Aufnahme verglichen und bearbeitet. Dabei wird jedes Oberflächenelement höchstens mit einem anderen Oberflächenelement verglichen.

In dieser Arbeit wird stärker berücksichtigt, dass Oberflächenelemente die Oberfläche eines Objektes approximieren. Ein Oberflächenelement einer neuen Aufnahme verändert nicht nur ein Oberflächenelement des Modells, sondern verformt alle Oberflächenelemente die im Bereich des Radius liegen. Somit verändert ein Oberflächenelement der Aufnahme nicht genau ein Element des Modells, sondern verformt viel mehr die gesamte Oberfläche in einem kleinen Bereich. Die Umsetzung der Idee ist im Folgenden detailliert erklärt.

In Abschnitt 3.3.2 wurde beschrieben wie die Aufnahme vorbereitet und die Oberflächennormalen berechnet werden. Es ist nun möglich aus jedem Punkt von \mathcal{A}_{ROI} ein Oberflächenelement zu erzeugen und die generierten Oberflächenelemente werden in der Menge

$$\partial\mathcal{M}'_{\text{ROI}} = \{\mathcal{S}'_j \mid 0 \leq j \leq N_{\text{ROI}}, j \in \mathbb{N}\} \quad (3.22)$$

gespeichert. Zur Auswahl des Radius der Oberflächenelemente wurde angenommen, dass sich das Objekt in einem Mindestabstand von 0.7m vor der Kamera befindet. Die RGB Kamera der Microsoft Kinect hat eine Auflösung

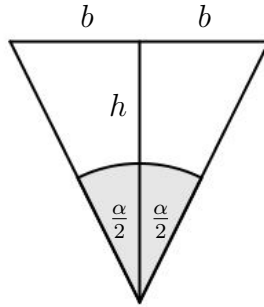


Abbildung 3.22: Ideenskizze zur Wahl des Radius der Oberflächenelemente.

von 640×480 Pixel und ein Field of View von $57^\circ \times 43^\circ$, vgl. [46]. Somit stehen ungefähr 11×11 Pixel pro 1° Öffnungswinkel zur Verfügung. Der räumliche Abstand der Bildpunkte ist abhängig vom Abstand des Objekts zur Kamera und kann mit Hilfe der Skizze in Abbildung 3.22 berechnet werden. Bei einem Öffnungswinkel α und einem Abstand h stehen für 11 Pixel ein Abstand von

$$2b = 2h \tan\left(\frac{\alpha}{2}\right) \quad (3.23)$$

zur Verfügung. Somit beträgt der Abstand zweier Bildpunkte bei Minimalabstand

$$c = \frac{2b}{11} = \frac{2 \times 0.7 \text{ m} \times \tan(0.5^\circ)}{11} = 1.11 \text{ mm}. \quad (3.24)$$

Durch die Wahl des Radius der Oberflächenelemente auf 1 mm wird sichergestellt, dass die Überlappung möglichst groß ist ohne Mittelpunkte in andere Oberflächenelemente einzuschließen.

Die neue Aufnahme \mathcal{A}_τ ist aus der Position der Kamera zum Zeitpunkt τ aufgenommen worden. Deswegen befinden sich die aus \mathcal{A}_{ROI} generierten Oberflächenelemente in dem, von der Kamera definierten, Koordinatensystem. Damit die Elemente in das Modell integriert werden können, muss die Aufnahme in das Koordinatensystem des Modells transformiert werden. Dazu werden die homogenen Transformationen

$$\mathbf{T}_\tau = \begin{bmatrix} \mathbf{R}_\tau & \mathbf{t}_\tau \\ \mathbf{0} & 1 \end{bmatrix} \quad (3.25)$$

aus Abschnitt 3.2 verwendet und die neue Position des transformierten Oberflächenelements \mathcal{S}_j lautet

$$\mathbf{p}_j = \mathbf{R}_\tau \mathbf{p}'_j + \mathbf{t}_\tau. \quad (3.26)$$

Dabei bezeichnet \mathbf{p}'_j die Position des Oberflächenelements \mathcal{S}'_j . Für die Transformation der Oberflächennormalen

$$\mathbf{n}_j = \mathbf{R}_\tau \mathbf{n}'_j \quad (3.27)$$

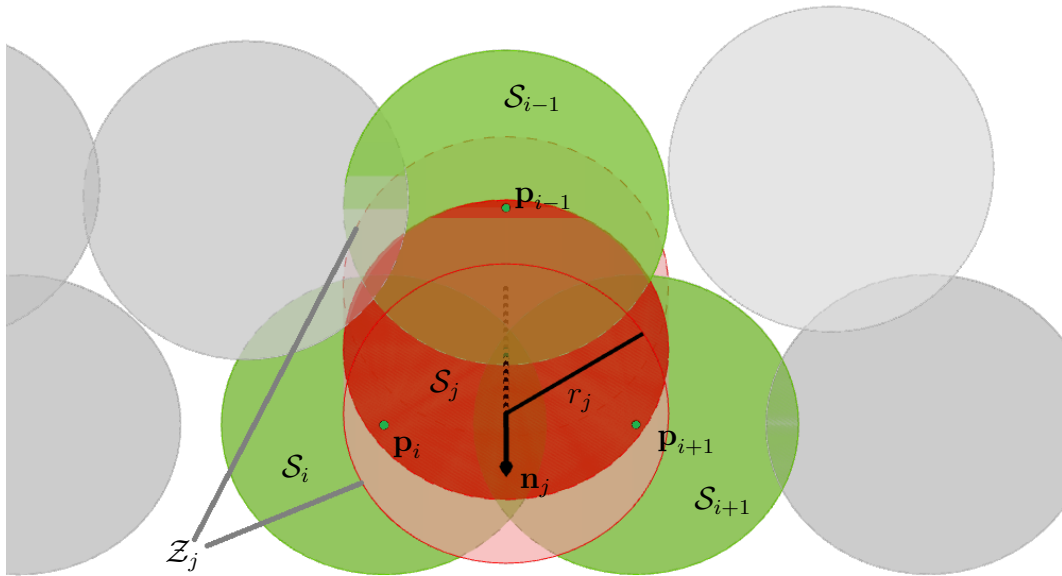


Abbildung 3.23: Ideenskizze für die Suche der korrespondierenden Oberflächenelemente.

ist es ausreichend die Rotation zu berücksichtigen. In Gleichung 3.27 bezeichnet \mathbf{n}'_j den Vektor der Oberflächennormale des Oberflächenelementes \mathcal{S}'_j . Die neue Position und die rotierte Normale werden mit allen zusätzlichen Informationen in dem transformierten Oberflächenelement \mathcal{S}_j gespeichert und alle transformierten Oberflächenelemente werden in der Menge

$$\partial\mathcal{M}_{\text{ROI}} = \{\mathcal{S}_j \mid 0 \leq j \leq N_{\text{ROI}}, j \in \mathbb{N}\} \quad (3.28)$$

zusammengefasst. Für jedes Element \mathcal{S}_j aus $\partial\mathcal{M}_{\text{ROI}}$ muss entschieden werden ob ein neues Element in das Modell aufgenommen wird, oder ob es korrespondierende Oberflächenelemente des Modells gibt, die durch die neue Information verfeinert werden sollen. Im Folgenden wird die Suche nach passenden Elementen für ein Oberflächenelement \mathcal{S}_j beschrieben, wobei die folgenden Berechnungen für alle Elemente von $\partial\mathcal{M}_{\text{ROI}}$ durchgeführt werden müssen. Ein Element $\mathcal{S}_i \in \mathcal{M}_{\tau-1}$ des aktuellen Modells wird als korrespondierend zum Oberflächenelement \mathcal{S}_j angesehen, wenn der Mittelpunkt des Elements \mathcal{S}_i in einem Zylinder, der durch die Oberflächennormale des Oberflächenelements \mathcal{S}_j , dem Radius von \mathcal{S}_j und einer konstanten Höhe definiert ist, zu liegen kommt. Es wird also ausgehend von dem Element der Aufnahme \mathcal{S}_j ein Zylinder \mathcal{Z}_j aufgespannt. Dieser Zylinder hat eine definierte Höhe, den Radius des Elements und ist ausgerichtet entlang der Oberflächennormale. Wenn sich der Mittel-

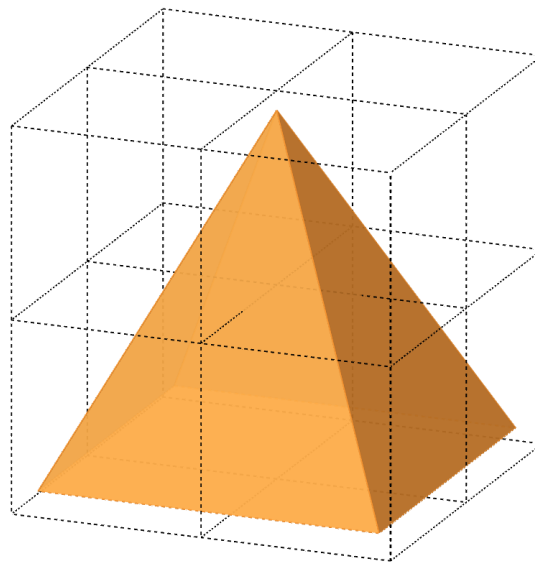


Abbildung 3.24: Skizze zur Idee des dreidimensionalen Suchgitters.

punkt eines Oberflächenelements des Modells in diesem Zylinder befindet, wird eine Korrespondenz gefunden. Die Idee ist in Abbildung 3.23 dargestellt. Dabei stellt das rote Element das Oberflächenelement \mathcal{S}_j der aktuellen Aufnahme dar und die grünen Elemente sind die korrespondierenden Elemente des Modells. Für den gezeigten Fall werden also drei Oberflächenelemente des Modells durch ein Oberflächenelement der Aufnahme verändert. Für die Umsetzung müssen zwei Probleme gelöst werden. Erstens muss der Zylinder aufgespannt und der Radial- bzw. Normalabstand berechnet werden. Zweitens ist es aufgrund von Laufzeitproblemen nicht sinnvoll jedes Oberflächenelement der Aufnahme mit jedem Oberflächenelement des Modells zu vergleichen. Zur Lösung des zweiten Problems, also die Reduktion der Laufzeit, wird der gesamte ROI in ein dreidimensionales Suchgitter unterteilt, siehe Abbildung 3.24. Jedes Oberflächenelement ist einem Bereich dieses Suchgitters zugewiesen. Durch diese globale örtliche Zuweisung des Oberflächenelemente ist es möglich einen definierten Bereich anzugeben in dem die korrespondierenden Elemente gesucht werden. In der tatsächlichen Umsetzung sind die Bereiche des Gitters wesentlich kleiner als die Skizze vermuten lässt. Ein Bereich hat eine Länge von $\frac{1}{150}\text{m} = 6.66\text{ mm}$ und der ROI wird dadurch in viele 295.41 mm^3 große Würfel eingeteilt. Es ist nicht mehr notwendig jedes Oberflächenelement des Modells zu berücksichtigen, wodurch dieser Schritt die benötigte Zeit zum Suchen der korrespondierenden Elemente enorm verkürzt. Der Nachteil der Umsetzung ist ein erhöhter Speicherbedarf während der Modellerzeugung. Zur Berechnung des

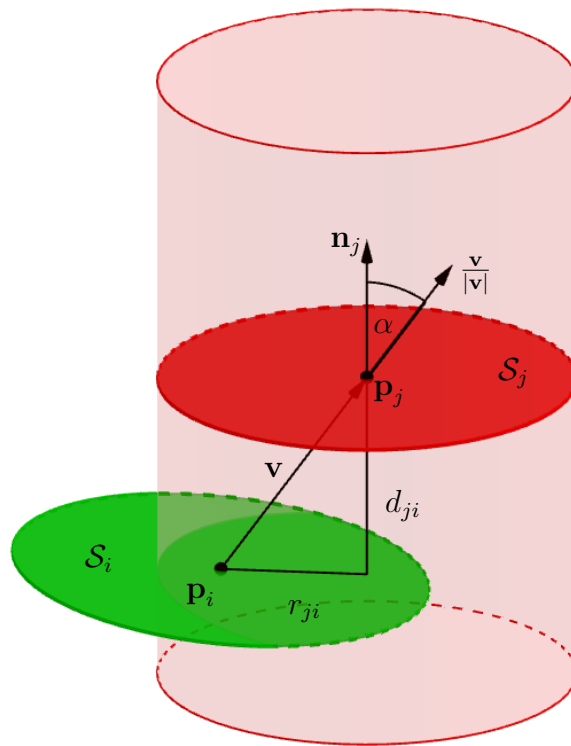


Abbildung 3.25: Skizze zur Berechnung des Radial- und Normalabstands zwischen zwei Oberflächelementen.

radialen und normalen Abstands betrachte man Abbildung 3.25. Dargestellt ist ein Oberflächelement \mathcal{S}_j der Aufnahme mit der Position \mathbf{p}_j und der Oberflächennormale \mathbf{n}_j . Das zweite Oberflächelement \mathcal{S}_i hat die Position \mathbf{p}_i und ist dem Modell zugehörig. Der Vektor

$$\mathbf{v} = \mathbf{p}_j - \mathbf{p}_i \quad (3.29)$$

gibt die Richtung und den Abstand zwischen den Positionen beider Oberflächelemente an. Der von der Oberflächennormale \mathbf{n}_j und dem Richtungsvektor \mathbf{v} eingeschlossene Winkel α kann über

$$\alpha = \arccos\left(\mathbf{n}_j \cdot \frac{\mathbf{v}}{|\mathbf{v}|}\right) \quad (3.30)$$

berechnet werden. Über das dargestellte ähnliche Dreieck lässt sich der Radialabstand

$$r_{ji} = |\sin(\alpha)| |\mathbf{v}| \quad (3.31)$$

und der Normalabstand

$$d_{ji} = | \cos(\alpha) | \mathbf{v} | \quad (3.32)$$

berechnen. Zwei Oberflächenelemente sind korrespondierend, wenn der Radialabstand r_{ji} kleiner als der Radius der Oberflächenelemente und der Normalabstand d_{ji} kleiner als eine einstellbare Schranke ist. Falls für ein Oberflächenelement der Aufnahme kein korrespondierendes Oberflächenelement des Modells gefunden werden kann, liegt es daran, dass das Modell $\delta\mathcal{M}_{\tau-1}$ diesen Bereich der Aufnahme \mathcal{A}_{ROI} nicht abbildet und es wird das Oberflächenelement in das Modell aufgenommen. Somit wächst die Anzahl der Oberflächenelemente im Modell, falls keine korrespondierenden Elemente gefunden werden. Für den Fall, dass korrespondierende Oberflächenelemente gefunden werden, können diese mit der Information der neuen Aufnahme verbessert werden. Dazu wird zuerst die aktualisierte Oberflächennormale

$$\hat{\mathbf{n}}_i^* = \frac{u_i \mathbf{n}_i + \mathbf{n}_j}{u_i + 1} \quad (3.33)$$

durch laufende Mittelwertberechnung der Oberflächennormalen bestimmt und anschließend mittels

$$\mathbf{n}_i^* = \frac{\hat{\mathbf{n}}_i^*}{|\hat{\mathbf{n}}_i^*|} \quad (3.34)$$

normiert. Dabei bezeichnet u_i die Anzahl der Aktualisierungen des Oberflächenelements \mathcal{S}_i . Die neue Position p_i^* wird durch die Projektion des Verschiebungsvektors auf die Oberflächennormale gebildet. In Abbildung 3.26 ist die Idee dargestellt. Zur Berechnung der neuen Position wird der Verschiebungsvektor

$$\mathbf{v} = \mathbf{p}_j - \mathbf{p}_i \quad (3.35)$$

und die Länge des Verschiebungsvektors, projiziert auf die neue Richtung der Oberflächennormale,

$$v_p = \mathbf{v} \cdot \mathbf{n}_i^* \quad (3.36)$$

berechnet. Die Veränderung der Position

$$\Delta \mathbf{p}_i = \frac{v_p \mathbf{n}_i^*}{u_i + 1} \quad (3.37)$$

wird verwendet um die neue Position

$$\mathbf{p}_i^* = \mathbf{p}_i + \Delta \mathbf{p}_i \quad (3.38)$$

zu berechnen. Die Verschiebung $\Delta \mathbf{p}_i$ ist ein Vektor in Richtung der neuen Oberflächennormale mit einer Länge die abhängig von der Anzahl der Aktualisierung ist. Die Verschiebung wird geringer je öfter die Position durch andere Aufnahmen bestätigt wurde. Die Reduktion der Verschiebungslänge ist ein Punkt, um die Erstellung des Modells robust gegen Fehler einzelner Aufnahmen zu machen.

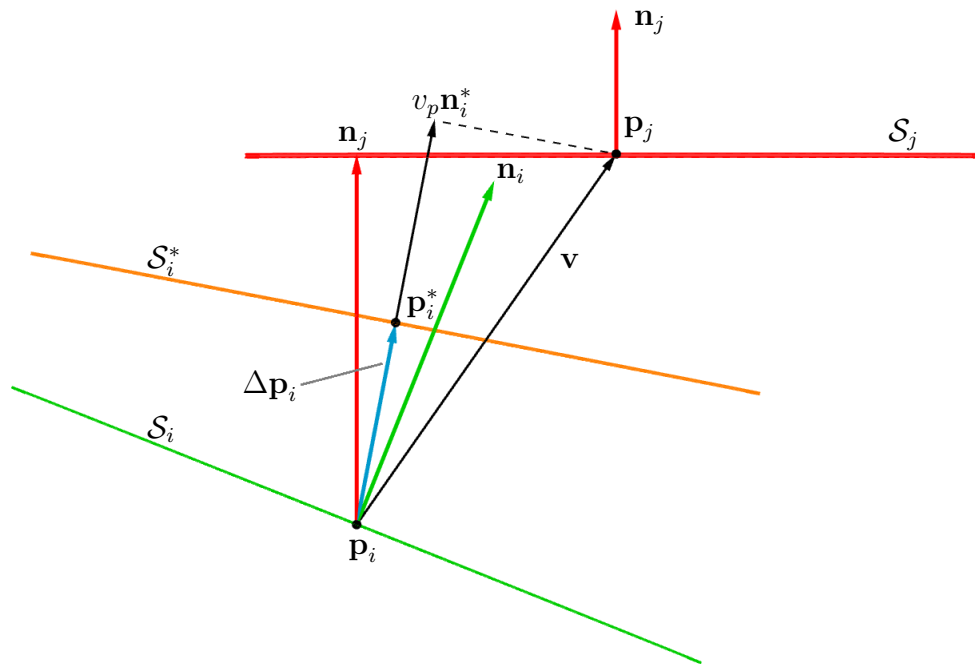


Abbildung 3.26: Aktualisierung des Flächenelements \mathcal{S}_i mit der Information des Flächenelements \mathcal{S}_j . \mathcal{S}_i^* ist das Ergebnis der Aktualisierung.

Behandlung von Ausreißern

Ein weiteres Problem, das bei der Modellierung von Objekten behandelt werden muss, ist das Erkennen und Löschen von Ausreißern. Ein Ausreißer ist ein Element eines Modells das nicht korrekt ist. Diese Ausreißer werden fälschlicherweise in das Objekt eingefügt und müssen erkannt werden um die Qualität des Modells zu erhöhen. In dieser Arbeit werden zwei unterschiedliche Vertrauenswerte verwendet, um die Vertrauenswürdigkeit eines Flächenelement zu bestimmen.

Der erste Vertrauenswert u_i wurde bereits in Gleichung 3.33 eingeführt. Er gibt an wie oft ein Flächenelement aktualisiert wurde. Je öfter ein Flächenelement mit einem anderen Flächenelement korrespondiert, desto wahrscheinlicher ist es, dass es korrekt ist. Zur Erkennung, ob ein Flächenelement ein potentieller Ausreißer ist, speichert jedes Element zusätzlich zu der Anzahl der Aktualisierungen u_i noch seine eigene Lebensdauer u_{alive} , die mit der Anzahl der möglichen Aktualisierungen seit seiner Erstellung gleich ist. Der Vertrauenswert u_i und die Lebensdauer u_{alive} werden während der Modellierung des Objekts mitgezählt. Nach einer einstellbaren Lebenszeit t_{max} muss ein Flächenelement eine Mindestanzahl an Aktualisierungen erlebt

haben, damit es als vertrauenswürdig gilt und sicher im Modell aufgenommen ist. Andernfalls wird das Oberflächenelement als Ausreißer markiert und bereits während der Erstellung des Modells gelöscht.

Für die Berechnung des zweiten Vertrauenswerts wird die Idee von [27] übernommen. Jedem Oberflächenelement wird ein Vertrauenswert $v_i \in \mathbb{N}$ zugewiesen. Je höher dieser Wert ist, desto wahrscheinlicher ist es, dass das Oberflächenelement korrekt ist. Ein Oberflächenelement wird als vertrauenswürdig eingestuft, wenn es von unterschiedlichen Richtungen beobachtet wurde. Für Ausreißer ist dies unwahrscheinlich. Zur Bestimmung aus wie vielen Richtungen ein Oberflächenelement beobachtet wurde, wird bei der Erstellung eines Oberflächenelements ein Kugelkoordinatensystem erzeugt. Der Ursprung des Koordinatensystems \mathcal{O} liegt im Mittelpunkt der Scheibe und als erster Basisvektor wird die Oberflächennormale \mathbf{n}_i verwendet. Der zweite Basisvektor \mathbf{b}_2 wird in der Ebene des Oberflächenelements gewählt und für weitere Berechnungen wird ein dritter Vektor \mathbf{b}_3 definiert, der die ersten zwei gewählten Vektoren zu einem rechtsdrehenden Koordinatensystem ergänzt. Der zweite Basisvektor wird so gewählt, dass er normal auf den Vektor der Oberflächennormale steht. Es soll also

$$\mathbf{n}_i \cdot \mathbf{b}_2 = \begin{bmatrix} n_{i1} \\ n_{i2} \\ n_{i3} \end{bmatrix} \cdot \begin{bmatrix} b_{21} \\ b_{22} \\ b_{23} \end{bmatrix} = n_{i1}b_{21} + n_{i2}b_{22} + n_{i3}b_{23} = 0 \quad (3.39)$$

gelten. Diese Bedingung für den Vektor \mathbf{b}_2 ist die einzige zu erfüllende Bedingung. Somit können zwei Einträge des Vektors frei gewählt werden. Wenn zum Beispiel $b_{22} = b_{23} = 1$ gewählt wird, muss

$$b_{21} = -\frac{n_{i2}b_{22} + n_{i3}b_{23}}{n_{i1}} \quad (3.40)$$

gelten, um einen Vektor in der Ebene des Oberflächenelements zu finden. Für den Fall, dass $n_{i1} = 0$ gilt, kann man andere Elemente von \mathbf{b}_2 wählen, um die Division durch 0 zu vermeiden. Nachdem der gefundene Vektor \mathbf{b}_2 normiert wurde, wird der ausstehende Vektor

$$\mathbf{b}_3 = \mathbf{n}_i \times \mathbf{b}_2 \quad (3.41)$$

berechnet. Somit sind die Basisvektoren des Kugelkoordinatensystems bestimmt. Die Richtungen, aus denen das Oberflächenelement beobachtet wird, können nun über den Polarwinkel θ und Azimutwinkel ϕ bestimmt werden. Das beschriebene Koordinatensystem ist in Abbildung 3.27 dargestellt, wobei \mathbf{v} den normierten Richtungsvektor zum Kamerazentrum darstellt und somit die Betrachtungsrichtung bestimmt.

Damit die Richtungen gezählt werden können, wird die Oberfläche der Einheitskugel mittels binärem Histogramm approximiert. Die Klassen des Histogramms

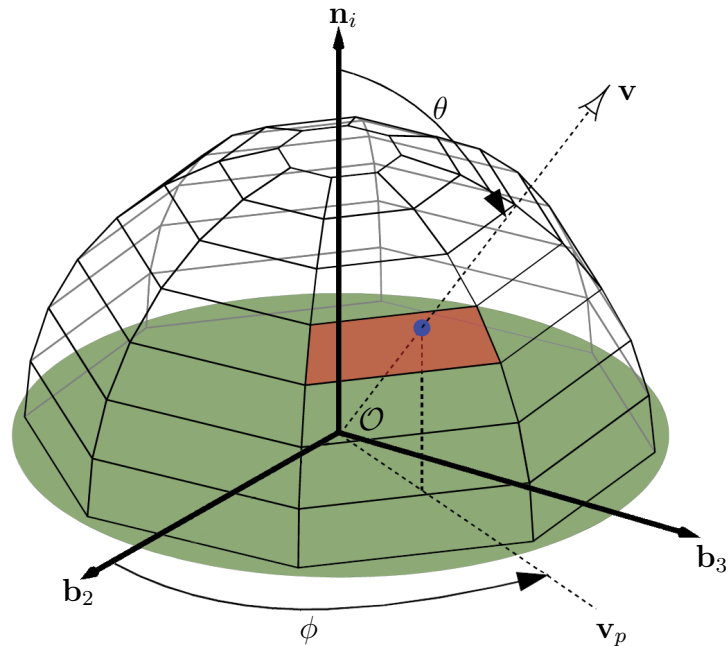


Abbildung 3.27: Verwendetes Kugelkoordinatensystem, um die Betrachtungsrichtung eines Oberflächenelements zu bestimmen, vgl. [27].

sind durch die zwei Winkel θ und ϕ des Koordinatensystems definiert und die Idee ist in Abbildung 3.27 skizziert. Zur Berechnung in welcher Klasse die aktuelle Beobachtungsrichtung zugehörig ist, müssen der Polarwinkel θ und der Azimutwinkel ϕ bestimmt werden. Der Polarwinkel kann durch die Projektion der Beobachtungsrichtung auf die Oberflächennormale bestimmt werden. Da die Vektoren \mathbf{v} und \mathbf{n}_i normiert sind, gilt

$$\theta = \arccos(\mathbf{v} \cdot \mathbf{n}_i). \quad (3.42)$$

Zur Berechnung des Azimutwinkels muss die Projektion der Beobachtungsrichtung in die Ebene des Oberflächenelements berechnet werden. Analog zu Gleichung 3.17 gilt

$$\hat{\mathbf{v}}_p = (\mathbf{v} \cdot \mathbf{b}_2)\mathbf{b}_2 + (\mathbf{v} \cdot \mathbf{b}_3)\mathbf{b}_3. \quad (3.43)$$

Der Azimutwinkel berechnet sich nach der Normierung

$$\mathbf{v}_p = \frac{\hat{\mathbf{v}}_p}{|\hat{\mathbf{v}}_p|} \quad (3.44)$$

durch

$$\phi = \arccos(\mathbf{v}_p \cdot \mathbf{b}_2). \quad (3.45)$$

Die berechneten Winkel θ und ϕ werden verwendet, um im Histogramm in der richtigen Klasse die Kennzeichnung zu setzen, dass das Oberflächenelement aus

der Richtung \mathbf{v} beobachtet wurde. Der Vertrauenswert v_i ist die Anzahl der gekennzeichneten Klassen im binären Histogramm.

Dieser Vertrauenswert wird verwendet, um nach der Integration von allen Aufnahmen, Ausreißer, die während der Erstellung nicht gefunden wurden, zu löschen. Dazu wird eine einstellbare Schranke n_{angle} verwendet. Wenn ein Oberflächenelement aus weniger als n_{angle} Richtungen beobachtet wird, wird es als Ausreißer markiert und aus dem Modell gelöscht. Dieser Schritt wird durchgeführt nachdem das Modell vollständig erzeugt und alle Richtungen in das Modell eingearbeitet wurden. Abbildung 3.28 zeigt ein generiertes Modell bevor die Elemente mit $v_i \leq n_{\text{angle}}$ gelöscht wurden. Es sind Oberflächenelemente im Boden erzeugt worden, die nicht zum Modell gehören. In Abbildung 3.29 wurden nur Oberflächenelemente beibehalten deren Vertrauenswert die Bedingung $v_i > n_{\text{angle}}$ erfüllen. Die Elemente des Bodens konnten identifiziert und gelöscht werden.



Abbildung 3.28: Durch Ungenauigkeiten beim Löschen des Bodens sind ein paar Oberflächenelemente ins Modell aufgenommen worden.



Abbildung 3.29: Durch den Vertrauenswert v_i können die Punkte vom Boden entfernt werden.

Erzeugen des finalen Modells

Als abschließender Schritt wird aus den vertrauenswürdigen Oberflächenelementen das finale Modell generiert. Dazu werden einerseits die Oberflächenelemente gefiltert, um die Anzahl der Elemente im erzeugten Modell zu reduzieren und andererseits werden die verbliebenen Oberflächenelemente geglättet.

Das finale Modell $\partial\mathcal{M}_{\text{final}}$ wird Element für Element aus einer Auswahl an Oberflächenelementen aus dem erzeugten Modell $\partial\mathcal{M}$ erzeugt. Die Oberflächenelemente $\mathcal{S}_i \in \partial\mathcal{M}$ des erzeugten Modells werden iterativ durchlaufen. Im Folgenden wird die Idee zur Erzeugung eines Oberflächenelements $\mathcal{S}_{\text{final};i}$ im finalen Modell beschreiben. Diese Schritte werden für alle Elemente des erzeugten Modells $\partial\mathcal{M}$ durchgeführt. Ähnlich wie beim Suchen von korrespondieren

Oberflächenelementen werden zwei Zylinder erzeugt, deren Hauptachsen die Oberflächennormale ist und deren Höhen einstellbar sind. Der erste Zylinder \mathcal{Z}_1 hat einen Radius der dreimal so groß wie der Radius des Oberflächenelements ist. Der Zylinder \mathcal{Z}_1 wird verwendet, um Oberflächenelemente zu finden mit deren Informationen das finale Oberflächenelement $\mathcal{S}_{\text{final};i}$ geglättet wird, siehe Abbildung 3.30. Der zweite Zylinder \mathcal{Z}_2 hat einen Radius der ein Drittel so groß ist wie der Radius des Oberflächenelements. Alle Oberflächenelemente, die im Suchzylinder \mathcal{Z}_2 zu finden sind, werden für die Erzeugung weiterer Oberflächenelemente des finalen Modells ignoriert, siehe Abbildung 3.31. Somit wird Zylinder \mathcal{Z}_2 verwendet um die finale Anzahl der Elemente im Model zu reduzieren. Zur Berechnung des finalen Oberflächenelements $\mathcal{S}_{\text{final};i}$ wird ein temporäres Element $\mathcal{S}_{\text{mean}}$ berechnet, das durch Mittelwertbildung aller Oberflächenelemente aus dem Suchzylinder \mathcal{Z}_1 berechnet wird. Dazu wird das arithmetische Mittel der Positionsvektoren und der Vektoren der Oberflächennormalen bestimmt. Das finale Element $\mathcal{S}_{\text{final};i}$ wird abschließend durch Mittelwertbildung von dem temporären Element $\mathcal{S}_{\text{mean}}$ und dem Oberflächenelement des bisherigen Modells \mathcal{S}_i bestimmt, siehe Abbildung 3.32. Durch die zweite Mittelwertbildung wird das Oberflächenelement abhängig von seiner Umgebung gefiltert. Abbildung 3.33 zeigt das finale Modell. Es ist optisch nicht von dem Modell aus Abbildung 3.29 zu unterscheiden, jedoch hat es mit 153315 Oberflächenelementen im Vergleich zu 211303 wesentlich weniger Elemente. Es wurden die redundanten Oberflächenelemente durch die Mittelwertbildung in den verbleibenden Elementen abgebildet. Somit kann die Anzahl der Elemente verringert werden, ohne wesentliche Information zu verlieren. Das gezeigte Modell ist nur ein Beispiel. In Kapitel 4 werden einige Ergebnisse dargestellt und unterschiedliche Einstellungen gegenübergestellt.

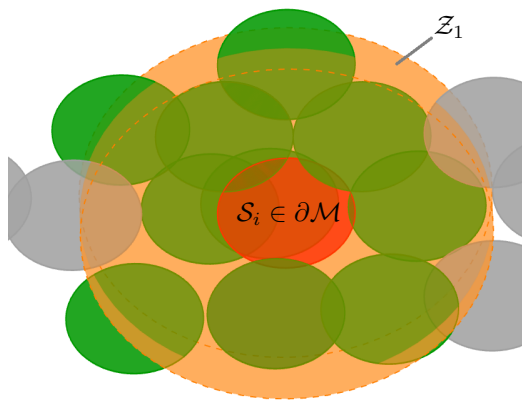


Abbildung 3.30: Darstellung des Suchzylinder Z_1 zum Glätten des finalen Modells.

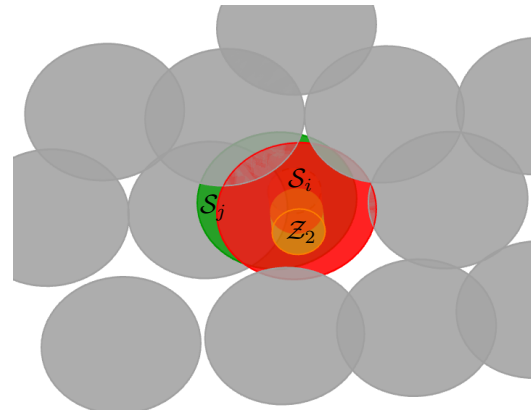


Abbildung 3.31: Darstellung des Suchzylinder Z_2 zum Filtern des finalen Modells. Das grüne Element S_j wird für die weitere Generierung des finalen Modells ignoriert.

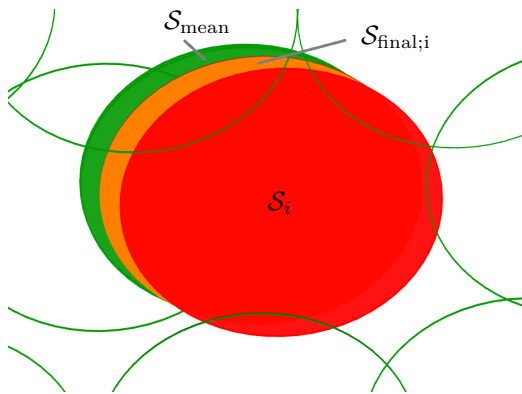


Abbildung 3.32: Berechnung des finalen Oberflächenelements $S_{\text{final};i} \in \partial\mathcal{M}_{\text{final}}$. S_i ist das Oberflächenelement des Modells $\partial\mathcal{M}$. S_{mean} ist durch Mittelwertbildung der Elemente im Zylinder Z_1 berechnet worden.



Abbildung 3.33: Finales Modell nach der Erzeugung der Oberflächenelemente $S_{\text{final};i}$.

3.3.3 Auswahl der verwendeten Aufnahmen zur Modellerstellung

In Abschnitt 3.3.2 wird angenommen, dass eine Menge $\mathcal{V}_s = \{\mathcal{A}_\tau \mid 0 \leq \tau \leq N_s, \tau \in \mathbb{N}\}$ an Aufnahmen aus der Menge aller Aufnahmen \mathcal{V} ausgewählt wurde. Die Aufnahmen aus \mathcal{V}_s werden verwendet, um das Modell iterativ zu erzeugen. Es ist möglich alle Aufnahmen, die von der Microsoft Kinect erzeugt werden, für die Modellerstellung zu verwenden indem $\mathcal{V}_s = \mathcal{V}$ gewählt wird. Jedoch gibt es Gründe warum eine Vorauswahl an Aufnahmen sinnvoll sein kann. Ein offensichtlicher Vorteil ist die Reduzierung der Laufzeit. Für jede Aufnahme $\mathcal{A}_\tau \in \mathcal{V}_s$ müssen die Schritte, die in den vorherigen Abschnitten beschrieben wurden, berechnet werden. Dabei ist nicht sichergestellt, dass das Modell sinnvoll erweitert wird, wenn jede Aufnahme verwendet wird. Falls zum Beispiel die Kamera für einige Zeit still gehalten werden würde, wird versucht das Modell mit sehr ähnlichen Aufnahmen zu erweitern. Im Extremfall würden sich die Aufnahmen nur aufgrund des Rauschens unterscheiden und keine sinnvolle Information zum Modell beitragen. Außerdem wird in Kapitel 4 gezeigt, dass es nicht unbedingt sinnvoll ist die Anzahl der verwendeten Aufnahmen so hoch wie möglich zu wählen. Ein zweiter Grund warum eine Vorauswahl der Aufnahmen sinnvoll ist, ist der verwendete Vertrauenswert u_i . Dieser Vertrauenswert gibt an wie oft ein Oberflächenelement mit der Information von anderen Aufnahmen aktualisiert wurde und ist nur aussagekräftig, wenn sich die Aufnahmen unterscheiden. Falls sich die Kamera nicht wesentlich zwischen den Aufnahmen bewegt, kann es vorkommen, dass falsche Punkte, wie sie zum Beispiel an Kanten auftreten können, oft beobachtet werden und somit wäre der erste Vertrauenswert nicht mehr aussagekräftig.

Ziel ist Aufnahmen zu finden, die neue Informationen für das Modell beinhalten. Dies ist im Allgemeinen der Fall wenn Aufnahmen aus unterschiedlichen Richtungen oder Positionen aufgenommen wurden. Falls die Bewegung der Kamera nicht eingeschränkt ist, wäre eine Möglichkeit alle Schlüsselbilder zum Erstellen des Modells zu verwenden. Somit wäre sichergestellt, dass sich die Kamera zwischen den Aufnahmen ausreichend bewegt hat und somit neue Informationen beinhalten. Durch die vorausgesetzte Kreisbewegung der Kamera um das Objekt wird eine andere Möglichkeit zur Auswahl der Aufnahmen verwendet. Die Idee ist, dass sich Aufnahmen nur unterscheiden wenn sie aus unterschiedlichen Drehwinkel um die y -Achse des Objekts aufgenommen werden. Es ist naheliegend den Kreis in Kreissektoren gleicher Größe einzuteilen und Aufnahmen zu verwenden, die an den Grenzen der Kreissektoren liegen, siehe Abbildung 3.34. Der Zentriwinkel $\psi < 360^\circ$ der Kreissektoren ist dabei frei wählbar und in Abschnitt 4.2.2 wird untersucht wie dieser Winkel gewählt werden kann.

Für die Erstellung der ausgewählten Menge \mathcal{V}_s müssen jene Aufnahmen gefunden werden, die an den Grenzen zwischen den Kreissektoren liegen. In der Praxis werden die Aufnahmen nicht exakt auf den Grenzen zu liegen kommen, aber durch die Bewegung der Kamera und der begrenzten Aufnahmezeit wird es Aufnahmen geben, die in der Nähe dieser Grenzen liegen. Da die Grenzen der Kreissektoren an ganzzahligen Vielfachen des Zentriwinkels $k\psi$ mit $k \in \mathbb{N}$ liegen, werden für die tatsächliche Auswahl jene Aufnahmen gesucht deren Rotation um die y -Achse des Objekts nahe an $k\psi$ liegen. Dazu wird für jede Aufnahme der Winkelabstand

$$\Delta\psi_{\tau;k} = |k\psi - \beta_{\tau}| \quad (3.46)$$

berechnet, wobei β_{τ} die Rotation um die y -Achse des Objekts der Aufnahme \mathcal{A}_{τ} beschreibt. Die Aufnahme \mathcal{A}_{τ} deren Winkelabstand $\Delta\psi_{\tau;k}$ am geringsten zum Winkel $k\psi$ ist wird mit \mathcal{A}_k bezeichnet. Die Auswahl der Aufnahmen ist somit

$$\mathcal{V}_s = \{\mathcal{A}_k \mid 0 \leq k < K, k \in \mathbb{N}, K = \frac{360}{\psi}\}. \quad (3.47)$$

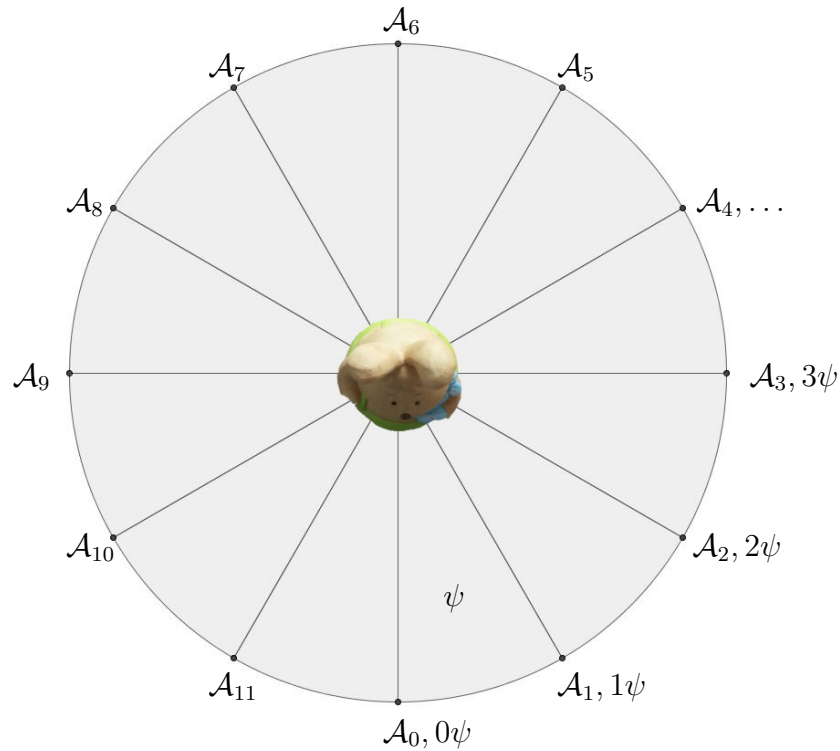


Abbildung 3.34: Darstellung der ausgewählten Aufnahmen zur Erstellung des Modells. Die Aufnahmen \mathcal{A}_k werden zu \mathcal{V}_s zusammengefasst.

3.3.4 Parameter des Frameworks

Im Zuge der bisherigen Erklärungen wurden bereits einige wählbare Schranken und Parameter erwähnt. Alle frei einstellbaren Parameter sind in der Tabelle 3.1 nochmals zusammengefasst und erklärt. Die angegebenen Werte wurden verwendet um die, in Kapitel 4 vorgestellten, Modelle zu generieren. Dabei sei hier darauf hingewiesen, dass jedes Modell mit den selben Parametereinstellungen modelliert wurde.

Parameter des DVO Algorithmus aus Abschnitt 3.2.1		
Name	Beschreibung	Wert
c_w	Anzahl der Pixel in x -Richtung.	640 Pixel
c_h	Anzahl der Pixel in y -Richtung.	480 Pixel
o_x	Position des Mittelpunktes in x -Richtung.	320 Pixel
o_y	Position des Mittelpunktes in y -Richtung.	240 Pixel
f_x	Fokale Länge der verwendeten Kamera in x -Richtung.	525 Pixel
f_y	Fokale Länge der verwendeten Kamera in y -Richtung.	525 Pixel
Parameter des Frame-To-Keyframe Tracking aus Abschnitt 3.2.2		
Name	Beschreibung	Wert
m_t	Maximal zulässige Translation zwischen Schlüsselbildern.	2.0 cm
m_r	Maximal zulässige Rotation um eine beliebige Achse zwischen Schlüsselbildern.	2°
y_s	Bei einer Rotation von $360^\circ - y_s$ um die y -Achse wird mit der Suche des Schleifenschlusses begonnen.	10°
Parameter für die Modellerstellung aus Abschnitt 3.3.2		
Name	Beschreibung	Wert
s_r	Radius aller Oberflächenelemente.	1 mm
r_r	Radius des ROI.	0.5 m
r_d	Schranke für die Abstandsberechnung des RANSAC Algorithmus zum Auffinden des Bodens.	1.0 cm
r_n	Suchradius zum Berechnen der Oberflächennormalen	2 cm.
c_l	Untere Schranke bei der Suche von Kanten mit dem Canny-Algorithmus.	10
c_h	Obere Schranke bei der Suche von Kanten mit dem Canny-Algorithmus.	30
c_s	Größe des Kernels für die Kantensuchen mit dem Canny-Algorithmus.	3 Pixel
a_i	Iterationen des anisotropen Filters zur Verbreitung der Kanten.	1
a_Δ	Parameter Δ der anisotropen Filterung.	0.5
a_κ	Parameter κ der anisotropen Filterung.	1.0
z_h	Höhe des Suchzylinders zum Suchen der korrespondierenden Oberflächenelemente.	2 cm
$u_{i,\min}$	Minimale Anzahl der Aktualisierungen bis t_{max} . Ansonsten wird das Oberflächenelement gelöscht.	7
t_{max}	Lebenszeit bis minimale Anzahl an Aktualisierungen erfolgt sein müssen.	12
n_{angle}	Anzahl der Richtungen aus denen ein Oberflächenelement beobachtet werden muss, damit es ins endgültigen Modell aufgenommen wird.	4

Tabelle 3.1: Zusammenfassung der Parameter des vorgestellten Frameworks. Die angegebenen Werte werden für die Modellierung der Modelle aus Kapitel 4 verwendet.

4 Auswertung

In diesem Kapitel wird die Genauigkeit des, in Kapitel 3 vorgestellten Frameworks, untersucht. Zu Beginn wird die Genauigkeit untersucht indem einfache Objekte modelliert und mit der Realität verglichen werden. Zusätzlich wird ein Modell eines Objekts einem Laserscan gegenübergestellt und die Unterschiede analysiert. Im zweiten Teil wird untersucht wie viele Aufnahmen für die Erstellung eines Modells sinnvoll sind. Zu diesem Zweck werden Objekte mit einer unterschiedlichen Anzahl an Aufnahmen modelliert und die Ergebnisse verwendet, um ein Optimierungsproblem zu formulieren.

4.1 Untersuchung der Genauigkeit

Die Untersuchung der Genauigkeit erfolgt in zwei Schritten. Zu Beginn wird die Genauigkeit anhand von einfachen Objekten untersucht und anschließend wird ein Modell mit einem Laserscan verglichen.

4.1.1 Untersuchung anhand einfacher Objekte

In diesem Abschnitt wird die Genauigkeit des vorgestellten Frameworks anhand eines Würfels, eines Zylinders und eines sechsseitigen Prismas untersucht. Die Oberflächen der Objekte sind aus Kunststoff, das unter einer 3mm dicken Schicht aus Plexiglas liegt und die Texturen weisen keine markanten Punkte auf. Bei der Modellierung wurde ein Zentriwinkel $\psi = 1^\circ$ verwendet und somit wurden 360 Aufnahmen zur Modellerstellung verwendet, vgl. Abschnitt 3.3.3. Die verwendeten Werte der restlichen Parameter sind in Tabelle 3.1 zusammengefasst. Eine Fotografie des Würfels ist in Abbildung 4.1 und das erstellte Modell in Abbildung 4.2 dargestellt. Der Zylinder und das sechseitige Prisma sind in Abbildung 4.3 bzw. Abbildung 4.5 abgebildet. Die zugehörigen Modelle sind in Abbildung 4.4 bzw. Abbildung 4.6 zu finden. Die eingezeichneten, markanten Längen wurden einerseits am realen Objekt mit einer Schublehre und andererseits digital, mithilfe der Software CloudCompare, am Modell gemessen.

CloudCompare ist ein Programm, um mit 3D Punktwolken zu arbeiten und wurde entworfen um Punktwolken miteinander zu vergleichen. Mittlerweile



Abbildung 4.1: Fotografie des Würfels, der zur Untersuchung der Genauigkeit verwendet wird.

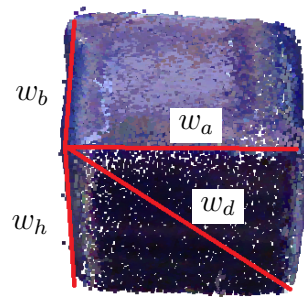


Abbildung 4.2: Modell des Würfels aus Abbildung 4.1. Die gemessenen, charakteristischen Längen sind eingezeichnet.



Abbildung 4.3: Fotografie des Zylinders der zur Untersuchung, der Genauigkeit verwendet wird.

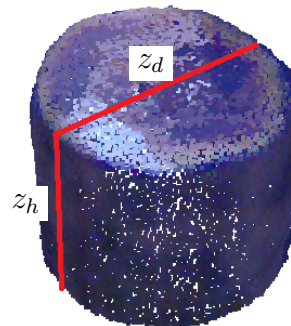


Abbildung 4.4: Modell des Zylinders aus Abbildung 4.3. Die gemessenen, charakteristischen Längen sind eingezeichnet.



Abbildung 4.5: Fotografie des sechsseitigen Prismas, das zur Untersuchung der Genauigkeit verwendet wird.

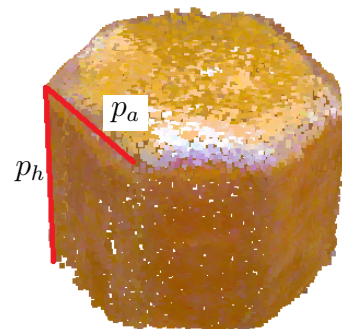


Abbildung 4.6: Modell des sechsseitigen Prismas aus Abbildung 4.5. Die gemessenen, charakteristischen Längen sind eingezeichnet.

wurde die Funktionalität wesentlich erweitert und für eine detaillierte Erklärung sei auf [47] verwiesen.

Die gemessenen Werte werden in Tabelle 4.1 verglichen. Zum Messen am digitalen Modell müssen zwei Punkte ausgewählt werden. Da diese Auswahl ungenau sein kann, wurden für jede Länge jeweils vier Messungen durchgeführt, um daraus einen Mittelwert \bar{x} zu berechnen. Der Unterschied zwischen der Messung am realen Objekt und dem Modell wird mit

$$\Delta_x = x_{real} - \bar{x} \quad (4.1)$$

berechnet, wobei x für die jeweilige charakteristische Länge steht. Beim Betrachten der Unterschiede Δ_x fällt auf, dass das reale Objekt meistens kleiner ist als das Modell. Ausgenommen davon sind die Höhen der Modelle w_h , z_h und p_h . Die niedrigere Höhe der Modelle im Vergleich zu den realen Objekten kann auf die Segmentierung des Objekts zurückgeführt werden. Wie in Abschnitt 3.3.2 gezeigt wird, wird der Boden der Aufnahme mit dem RANSAC Algorithmus erkannt und anschließend gelöscht. Beim Löschen des Bodens werden alle Punkte in einem Abstand r_d zur gefundenen Ebene gelöscht, siehe Tabelle 3.1. Dadurch werden somit auch einige Punkte des Objekts aus der Aufnahme gelöscht und sind für die weitere Modellierung nicht mehr verfügbar. Dieses Problem kann verringert werden indem der Parameter r_d kleiner gewählt wird. Jedoch erhöht sich dadurch die Wahrscheinlichkeit Punkte des Bodens nicht korrekt zu identifizieren und somit nicht korrekt aus der Aufnahme zu löschen. Der Mittelwert der Abweichung der Modelle zu den realen Objekten beträgt

$$\bar{\Delta}_1 = 0.046 \text{ mm.} \quad (4.2)$$

Unter der Annahme einer normalverteilten Abweichung lässt sich die Standardabweichung über

$$\sigma_{n1} = \sqrt{\frac{1}{n_1} \sum_{x \in \mathcal{X}} (\Delta_x - \bar{\Delta}_1)^2} \quad (4.3)$$

abschätzen, wobei

$$\mathcal{X} = \{w_a, w_b, w_h, w_d, z_d, z_h, p_a, p_h\} \quad (4.4)$$

die Menge aller charakteristischen Längen und $n_1 = |\mathcal{X}|$ die Mächtigkeit von \mathcal{X} ist. Die berechnete Standardabweichung beträgt

$$\sigma_{n1} = 5.418 \text{ mm.} \quad (4.5)$$

Aufgrund der geringen Anzahl an Messungen ist die Aussagekraft der berechneten Werte $\bar{\Delta}_1$ und σ_{n1} nicht gegeben. Sie dienen als grobe Abschätzung für die Genauigkeit. Im Folgenden wird ein komplexeres Modell mit einem

Charakteristische Längen des Würfels in mm							
Länge	Messungen des Modells				\bar{x}	Real	Δ_x
w_a	86.9	87.2	86.5	87.5	87.03	82.2	-4.83
w_b	83.3	84.2	82.5	83.6	83.4	82.2	-1.2
w_h	74.1	75.2	73.9	74.2	74.35	82.2	2.95
w_d	108.3	109.7	107.5	108.9	108.6	116.2	7.6
Charakteristische Längen des Zylinders in mm							
Länge	Messungen des Modells				\bar{x}	Real	Δ_x
z_d	90.8	88.7	91.6	86.5	89.4	80.6	-8.8
z_h	71.9	73.4	71.1	71.4	76.6	76.6	4.65
Charakteristische Längen des sechseckigen Prismas in mm							
Länge	Messungen des Modells				\bar{x}	Real	Δ_x
p_a	40.8	42.4	41.4	47.3	42.96	40.2	-2.76
p_h	72.1	6.94	71.6	70.9	71.0	77.1	6.1

Tabelle 4.1: Vergleich der charakteristischen Längen zwischen den realen Objekten und den digitalen Modellen (Werte in mm).

Laserscan verglichen. Die daraus resultierenden Ergebnisse werden am Ende des Abschnitts zusammengefasst, um einen möglichst aussagekräftigen Wert zu erhalten, siehe Gleichung 4.8 und Gleichung 4.9.

4.1.2 Vergleich mit einem Laserscan

Der nächste Schritt, in der Untersuchung der Genauigkeit des Frameworks, ist der Vergleich eines Modells mit einem Laserscan. Für die Aufnahme des Scans wurde ein Konica Minolta VI-9i Laserscanner verwendet, vgl. [12]. Dieser Laserscanner funktioniert nach dem Prinzip der Triangulation. Mit Hilfe eines Linienlasers wird die Oberfläche des Objekts mit einer vertikalen Bewegung gescannt. Das vom Objekt reflektierte Licht gelangt über eine Linse zu einem CCD und anschließend wird über Triangulation der Abstand berechnet. Zusätzlich kann über einen RGB Filter ein Farbbild des Objektes aufgenommen werden, das verwendet werden kann um Textur bereitzustellen, vgl. [12].

Zum Erstellen des Laserscans wurden 15 Aufnahmen des Objekts gemacht, um alle Bereiche aufzunehmen. Begonnen wurde mit einem Rundumscan, bei dem das Objekt während der Aufnahme auf einem geregelten Drehteller platziert war. Mit diesem Drehteller kann die Rotation des Objekts sehr exakt eingestellt werden und es wurden zwölf Aufnahmen mit einem Winkelabstand von 30° erstellt. Zusätzlich wurde jeweils eine Aufnahme von oben, eine von unten gemacht und eine Detailaufnahme von einem Bereich mit vielen Details erstellt.



Abbildung 4.7: Lasergescanntes Objekt.



Abbildung 4.8: Modell des Objekts.

All diese Aufnahmen wurden anschließend per Hand zueinander konsistent platziert und etwaige Artefakte, die zum Beispiel durch Reflexion entstehen, wurden entfernt. In Abbildung 4.7 ist der Laserscan und in Abbildung 4.8 ist das Modell des gleichen Objekts, generiert mit dem hier vorgestellten Framework, dargestellt. Ein Vergleich dieser beiden Modelle wurde mit der Software CloudCompare durchgeführt und das Ergebnis ist in Abbildung 4.9 zu sehen. Ein Histogramm über die Verteilung der Abweichung zwischen Laserscan und erstelltem Modell ist in Abbildung 4.10 dargestellt. Zusätzlich wurde eine gaußsche Normalverteilung, die die Abweichung approximiert, eingezeichnet. Der Mittelwert des Fehlers ist

$$\bar{\Delta}_2 = 0.667 \text{ mm} \quad (4.6)$$

und die Standardabweichung der Abweichung beträgt

$$\sigma_{n2} = 2.164 \text{ mm}. \quad (4.7)$$

Abgesehen vom Fehler der Genauigkeit, der durch $\bar{\Delta}_2$ und σ_{n2} ausgedrückt wird, fällt auf, dass in einigen Bereichen des Modells die Oberfläche Löcher aufweist. Dieses Problem kann damit erklärt werden, dass diese Bereiche für die Kamera während der Aufnahme, aufgrund von Eigenverdeckung, nicht sichtbar waren. Dies ist zum Beispiel im Bereich des Bauchs des Hasen ersichtlich. Die Wölbung verdeckt die dahinter liegende Oberfläche und somit ist dieser Bereich nicht für die Modellierung verfügbar. Dieses Problem wurde beim Erstellen des Laserscans durch ein gezieltes Bewegen des Objekts behoben. Für den hier verfolgten Ansatz, dass das Objekt still am Boden vor der Kamera steht und

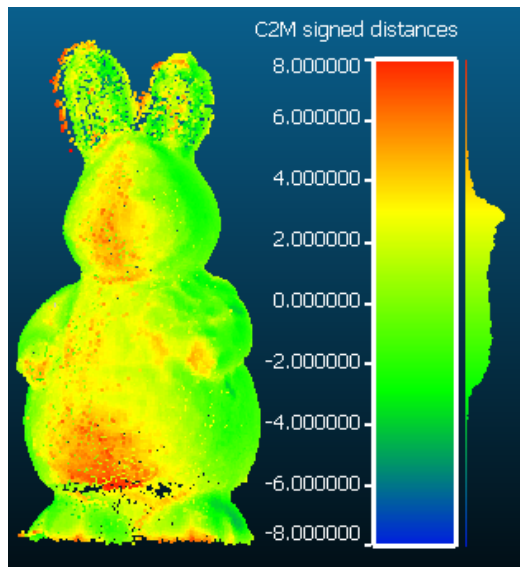


Abbildung 4.9: Vergleich zwischen dem Laserscan und dem erzeugten Modell (Werte in mm).

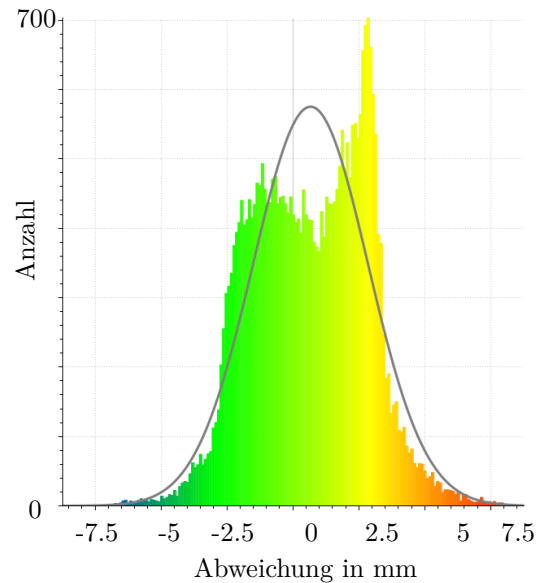


Abbildung 4.10: Histogramm und approximative Normalverteilung des Modellierungsfehlers.

die Kamera eine kreisförmige Trajektorie um das Objekt absolviert gibt es keine Lösung für diese Schwierigkeit.

Ein weiteres Problem tritt im Bereich der Ohren auf. Die große Anzahl an falschen Punkten ist auf ein fehlerbehaftetes Kameratracking zurückzuführen. Denn, obwohl mit den verwendeten Schlüsselbildern und der Fehlerreduktion mittels Schleifenschlusses die Genauigkeit erhöht wurde, besteht weiterhin ein wesentlicher Fehler im Schätzen der Position der Kamera. Ein Grund dafür ist, dass beim Korrigieren des Fehlers durch den Schleifenschluss keine vertrauenswürdige Information darüber existiert wo der Fehler entstanden ist. Somit wird die berechnete Transformation ΔT aus Gleichung 3.18 gleichmäßig auf alle Schlüsselbilder verteilt. Mit der Transformation ΔT wird sichergestellt, dass die geschätzte Position beim Schleifenschluss lokal konsistent ist. Jedoch ist dadurch nicht sichergestellt, dass die Position der restlichen Schlüsselbilder tatsächlich korrigiert werden. Zusätzlich können zwischendurch Fehler in der Transformation auftreten, die sich im weiteren Verlauf des Kameratracking wieder aufheben. Solche Fehler können beim Schleifenschluss nicht gefunden werden und bleiben in der geschätzten Trajektorie bestehen. In Bereichen mit vielen Punkten und gleichmäßigen Oberflächen kann ein gewisser Fehler in der Kameraposition durch das Suchen der korrespondierenden Oberflächenelementen ausgeglichen werden. In Bereichen mit wenig Punkten oder ungleichmäßigen Oberflächen, wie es bei den Ohren des Hasen der Fall ist, kann der Fehler

nicht behoben werden. Somit kann es passieren, dass in diesen Bereichen mehr fehlerhafte Punkte auftreten, als in Bereichen mit gleichmäßigen Oberflächen. Zusammenfassend lässt sich eine Genauigkeit des Frameworks durch Mittelwertbildung der berechneten Fehler aus den Gleichungen 4.2, 4.5, 4.6 und 4.7 berechnen. Der Mittelwert des kombinierten Fehlers berechnet sich zu

$$\bar{\Delta} = \frac{1}{2}(\bar{\Delta}_1 + \bar{\Delta}_2) = 0.3565 \text{ mm} \quad (4.8)$$

und die Standardabweichung beträgt

$$\sigma_n = \frac{1}{2}(\sigma_{n1} + \sigma_{n2}) = 3.791 \text{ mm}. \quad (4.9)$$

Die in diesem Abschnitt betrachteten Modelle wurde mit einem eingestellten Zentriwinkel von $\psi = 1^\circ$ (entspricht 360 Aufnahme) modelliert. Der Nachteil der hohen Anzahl der verwendeten Aufnahmen ist die benötigte Laufzeit. In Abschnitt 4.2 wird ein Optimierungsproblem erstellt, anhand dessen die Einstellung des Zentriwinkel, unter Berücksichtigung der Laufzeit, des Detailgrads und der Fehlerrate, gewählt werden kann.

4.2 Untersuchung zum Einfluss des Winkelabstands

Nachdem in Abschnitt 4.1 die Genauigkeit des Frameworks untersucht wurde, wird in diesem Abschnitt der Einfluss des verwendeten Winkelabstands auf das endgültige Modell untersucht. Dazu wird der Einfluss der Anzahl an verwendeten Aufnahmen auf den Detailgrad des Modells, die Laufzeit und die Anzahl an fehlerhaften Punkten untersucht. Abschließend wird ein Optimierungsproblem erstellt, um den optimalen Winkelabstand in Bezug auf Laufzeit, Detailgrad und Fehlerrate zu bestimmen. Zu diesem Zweck wurden 15 Modelle von Objekten mit unterschiedlichen Eigenschaften erstellt. Alle verwendeten Objekte sind, abgesehen von den in Abschnitt 4.1.1 bereits abgebildeten, in Tabelle 4.2 dargestellt und ihren Modellen gegenübergestellt. Bei der Auswahl der Objekte wurde darauf geachtet, dass sie unterschiedliche Eigenschaften, in Bezug auf Symmetrie, Größe, Oberflächenbeschaffenheit, Textur und Materialien, aufweisen um möglichst repräsentativ für eine Vielzahl anderer Objekte zu sein.

Abbildung 4.11 zeigt Modelle eines Objekts die mit unterschiedlichen Winkelabständen erzeugt wurden. Man erkennt, dass die Anzahl der Punkte mit Erhöhung des Winkelabstands abnimmt, die allgemeine Form des Objekts jedoch auch bei großen Winkelabständen weiterhin bestehen bleibt.

























Objekt und Modell		Objekt und Modell	
			
			
			
			
			
			

Tabelle 4.2: Abbildungen der verwendeten Objekte und ihrer Modelle. Alle dargestellten Modelle wurden mit einem Zentriwinkel von $\psi = 1^\circ$ erstellt.



Abbildung 4.11: Auswirkung des Winkelabstands. (links) Modellierung erfolgte mit $\psi = 1^\circ$, (Mitte) Modellierung erfolgte mit $\psi = 5^\circ$, (rechts) Modellierung erfolgte mit $\psi = 15^\circ$.

4.2.1 Erstellung der Untersuchungsdaten

Für die nachfolgende Untersuchung wurde jedes Objekte mit unterschiedlichen Anzahlen an Aufnahmen modelliert. Dabei wurden für jeden Zentriwinkel

$$\psi_j \in \Psi = \{1^\circ, 2^\circ, 3^\circ, 4^\circ, 5^\circ, 6^\circ, 7^\circ, 8^\circ, 9^\circ, 10^\circ, 12^\circ, 15^\circ, 20^\circ, 25^\circ, 30^\circ, 45^\circ\} \quad (4.10)$$

ein Modell erstellt und die Laufzeit, die verwendete Rechenzeit, die Anzahl der erstellten Datenpunkte und die Anzahl der Fehler gespeichert. Die absolute Laufzeit ist von der Rechenzeit verschieden, da durch Parallelisierung die mögliche Rechenzeit auf der CPU vervielfacht wird. Als Maß für den Detailgrad des Modells werden die Anzahl der Punkte im Modell verwendet. Dabei wird davon ausgegangen, dass eine größere Anzahl an Datenpunkten einen höheren Detailgrad bedeutet. Da diese Annahme nur gerechtfertigt ist, wenn alle Punkte korrekt sind, wird zusätzlich ein Maß für die Anzahl der falschen Punkte verwendet. Dazu werden mögliche Ausreißer mit einem „statistical outlier removal“ Filter (SOR Filter) bestimmt. Beim SOR Filter wird für jeden Punkt des Modells der durchschnittliche Abstand zu seinen k Nachbarn berechnet. Es wird angenommen, dass diese mittleren Abstände einer Normalverteilung genügen und von allen berechneten mittleren Abständen wird der globaler Mittelwert und die Standardabweichung bestimmt, vgl. [48]. Mit Hilfe der Standardabweichung kann für jeden Punkt entschieden werden, ob sein mittlerer Abstand zu weit vom globalen Mittelwert entfernt ist. Bei der Berechnung werden die $k = 30$ nächsten Punkte zur Bestimmung des mittleren Abstands verwendet und jeder Punkt, dessen mittlerer Abstand mehr als vier Standardabweichungen entfernt vom globalen Mittel ist, wird als Ausreißer identifiziert.

Damit die aufgenommen Daten zwischen den Objekten verglichen und gemittelt

werden können, wird jeder Messwert auf den Wert für die Modellerstellung mit $\psi = 1^\circ$ bezogen. Für das Objekt \mathcal{M}_i mit $i \in [1,15]$, $i \in \mathbb{N}$ wird für jedes Modell $\partial\mathcal{M}_{i;\psi_j}$ mit $\psi_j \in \Psi$ die Laufzeit $t_{i;\psi_j}$, die Rechenzeit $\tau_{i;\psi_j}$, die Punktzahl $p_{i;\psi_j}$ und die Ausreißerzahl $f_{i;\psi_j}$ bestimmt. Die Laufzeit, die Rechenzeit und die Punktzahl

$$t'_{i;\psi_j} = \frac{t_{i;\psi_j}}{t_{i;\psi_1}}, \quad \tau'_{i;\psi_j} = \frac{\tau_{i;\psi_j}}{\tau_{i;\psi_1}}, \quad p'_{i;\psi_j} = \frac{p_{i;\psi_j}}{p_{i;\psi_1}} \quad (4.11)$$

werden jeweils auf den Messwert für $\psi_1 = 1^\circ$ bezogen. Als Fehlerrate wird die Anzahl der Ausreißer bezogen auf die Gesamtanzahl an Punkten des Modells verwendet. Deswegen ist die Fehlerrate

$$f'_{i;\psi_j} = \frac{f_{i;\psi_j}}{p_{i;\psi_j}} \quad (4.12)$$

bereits eine bezogene Größe und kann direkt verwendet werden. Aus den bezogenen Werten der einzelnen Objekte wird ein Mittelwert

$$\bar{t}_{\psi_j} = \frac{1}{15} \sum_{i=1}^{15} t'_{i;\psi_j}, \quad (4.13)$$

$$\bar{\tau}_{\psi_j} = \frac{1}{15} \sum_{i=1}^{15} \tau'_{i;\psi_j}, \quad (4.14)$$

$$\bar{p}_{\psi_j} = \frac{1}{15} \sum_{i=1}^{15} p'_{i;\psi_j}, \quad (4.15)$$

$$\bar{f}_{\psi_j} = \frac{1}{15} \sum_{i=1}^{15} f'_{i;\psi_j} \quad (4.16)$$

über alle Objekte \mathcal{M}_i bestimmt. Die berechneten Mittelwerte können nun in Abhängigkeit des Zenitwinkel ψ dargestellt werden. In Abbildung 4.12 ist die gemittelte, bezogene Laufzeit \bar{t}_{ψ_j} und die gemittelte, bezogene Rechenzeit $\bar{\tau}_{\psi_j}$ in Abhängigkeit des Zenitwinkels ψ dargestellt. Dabei wird zwischen den Messpunkten linear interpoliert. Der Verlauf der Laufzeiten ist einer Hyperbel sehr ähnlich, siehe Abbildung 4.14, und die Anzahl der verwendeten Aufnahmen

$$n_{\mathcal{A};j} = \frac{360^\circ}{\psi_j} \quad (4.17)$$

ist reziprok zu dem verwendeten Winkelabstand. Da

$$\bar{t}_{\psi_j} \approx \frac{1}{\psi_j} \approx \frac{1}{1/n_{\mathcal{A};j}} = n_{\mathcal{A};j} \quad (4.18)$$

gilt, ist gezeigt, dass der Aufwand des vorgestellten Frameworks demnach annähernd linear in der Anzahl der verwendeten Aufnahmen ist.

Die Anzahl der erzeugten Punkte im finalen Modell ist der Fehlerrate in Abbildung 4.13 gegenübergestellt. Die Anzahl der erzeugten Punkte weist zwei

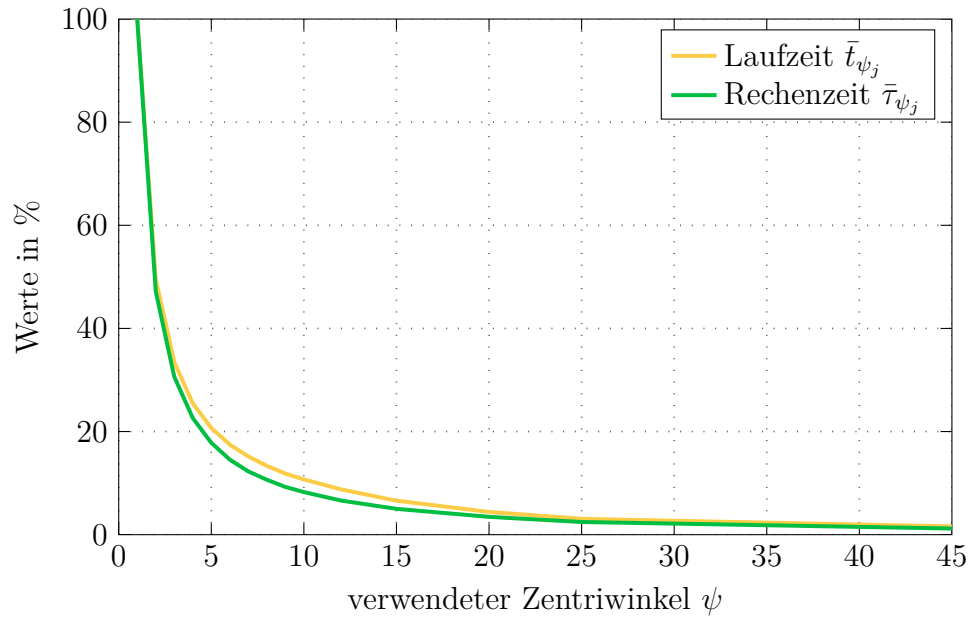


Abbildung 4.12: Darstellung der Laufzeit und der Rechenzeit in Abhängigkeit vom Winkelabstand ψ .

Bereiche mit annähernd linearem Abfall vor. Der Bereich zwischen $\psi = 1^\circ$ und $\psi = 25^\circ$ ist für die nachfolgenden Überlegungen interessant. In dem Bereich nach dem Knick wurden lediglich drei Messpunkte erzeugt und das erzeugte Modell verfügt nur mehr über sehr wenige Punkte. Deswegen ist der Bereich nach dem Knick wenig aussagekräftig. Die Fehlerrate weist einen leicht ansteigenden Verlauf auf und steigt von 0.78% bei $\psi_1 = 1^\circ$ auf 1.14% bei $\psi_1 = 45^\circ$ an.

4.2.2 Formulierung des Optimierungsproblems

Der Verlauf der Laufzeitkurve im Vergleich zum Verlauf der erzeugten Punktzahl motiviert die Suche nach einem Zentriwinkel ψ der einen möglichst hohen Detailgrad bei geringen Laufzeiten und Fehlerraten erzeugt. Damit dieser Zentriwinkel gefunden werden kann, wird ein Optimierungsproblem formuliert, das die Anzahl der erzeugten Punkte, die Laufzeit und die Fehlerrate in einer Kostenfunktion zusammenfasst. Zur Formulierung des Optimierungsproblems werden die Verläufe der Laufzeit, der Punktzahl und der Fehlerrate mittels der Methode der kleinsten Fehlerquadrate approximiert. Da die Laufzeit stark

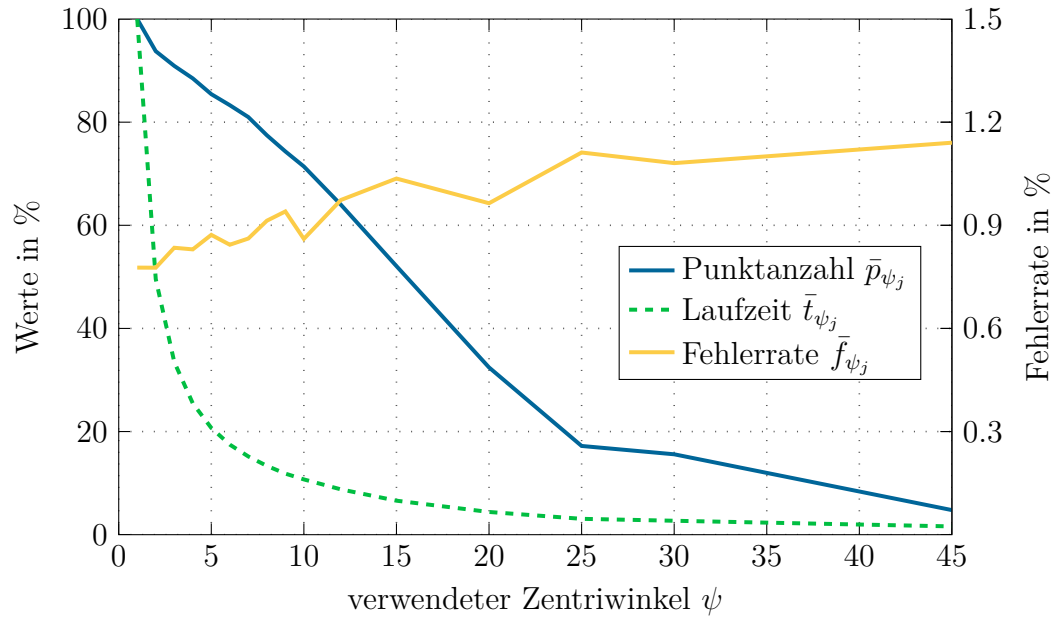


Abbildung 4.13: Darstellung der erzeugten Punktzahl und der Fehlerrate in Abhängigkeit vom Winkelabstand ψ .

einer Hyperbel ähnelt wird versucht den Verlauf der Laufzeit mit der Funktion

$$\bar{t}_{\psi_j} \approx a + \frac{b}{\psi_j} \quad (4.19)$$

zu approximieren. Die Parameter $a \in \mathbb{R}$ und $b \in \mathbb{R}$ werden so gesucht, dass die Fehlerquadrate zwischen den Messwerten und der Modellfunktion minimiert werden. Fasst man alle Messungen im Vektor \mathbf{y} und die Parameter im Vektor \mathbf{p} zusammen, dann lässt sich ein überbestimmtes Gleichungssystem

$$\mathbf{y} = \mathbf{S}\mathbf{p} \quad (4.20)$$

erzeugen, wobei \mathbf{S} die entsprechende Einträge der Modellfunktion beinhaltet. Mit dieser Schreibweise können die Parameter über

$$\mathbf{p}_0 = (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{y} \quad (4.21)$$

bestimmt werden, vgl. [49]. Die berechneten Parameter führen zu der Funktion

$$\bar{t}_{\psi_j} \approx 0.51\% + \frac{99.58\%}{\psi_j}, \quad (4.22)$$

die in Abbildung 4.14 mit den Messdaten verglichen wird.

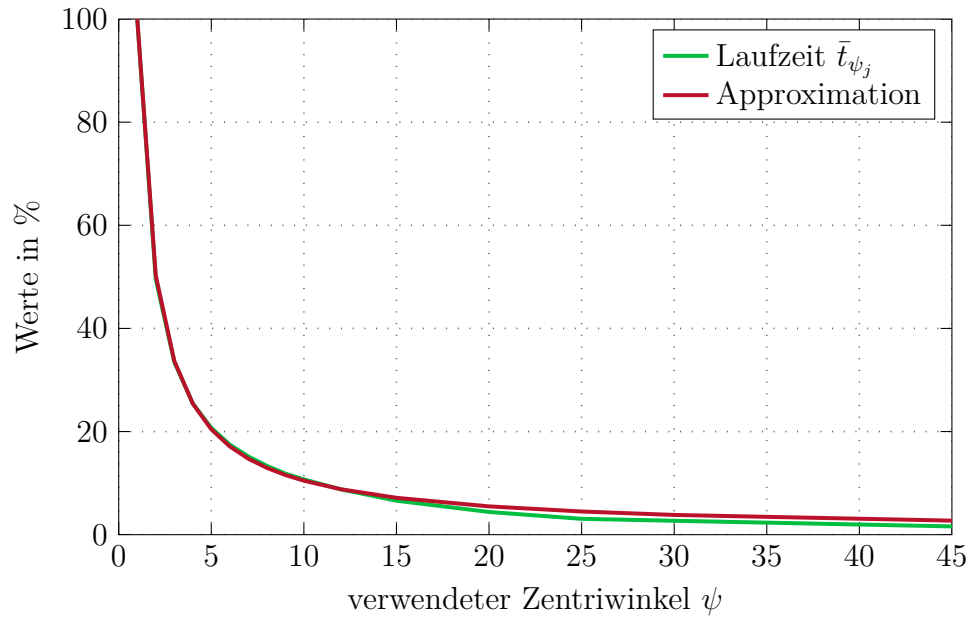


Abbildung 4.14: Darstellung der Laufzeit und der approximierenden Funktion in Abhängigkeit vom Winkelabstand ψ .

Für die Approximation der Punktzahl und der Fehlerrate wird jeweils eine lineare Funktion der Form

$$y(\psi_j) = a + b\psi_j \quad (4.23)$$

angesetzt. Nach der Bestimmung der freien Parameter, lässt sich der Verlauf der Punktzahl im Bereich zwischen $\psi = 1^\circ$ und $\psi = 25^\circ$ mit

$$\bar{p}_{\psi;j} \approx 103.23\% - 3.41\%\psi_j \quad (4.24)$$

approximieren. Die Funktion zur Approximation der Fehlerrate wurde mit

$$\bar{f}_{\psi;j} \approx 0.78\% + 0.01\%\psi_j \quad (4.25)$$

bestimmt. Der Vergleich zwischen den Näherungen und den Messdaten ist in Abbildung 4.15 dargestellt.

Mit den angenäherten Funktionen kann das Optimierungsproblem

$$\min_{\psi \in [1^\circ, 25^\circ]} f(\psi) = c_t \underbrace{\left(0.51 + \frac{99.58}{\psi}\right)}_{\text{Laufzeit}} - c_p \underbrace{\left(103.23 - 3.41\psi\right)}_{\text{Punktzahl}} + c_f \underbrace{\left(0.78 + 0.01\psi\right)}_{\text{Fehlerrate}} \quad (4.26)$$

mit der Kostenfunktion $f(\psi)$, der Gewichtung der Laufzeit c_t , der Gewichtung der Punktzahl c_p und der Gewichtung der Fehlerrate c_f formuliert werden.

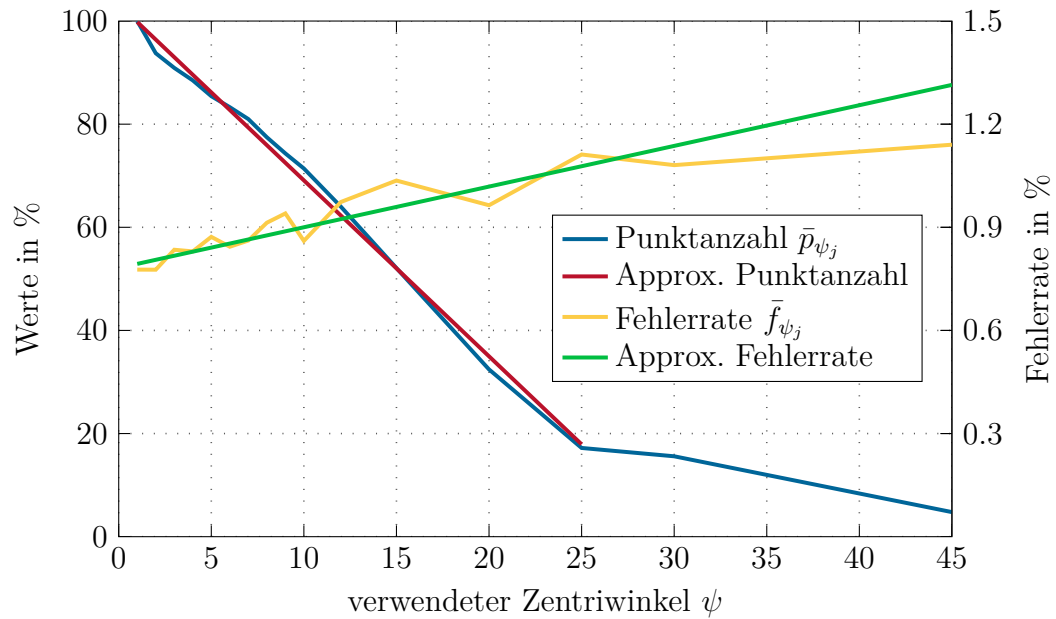


Abbildung 4.15: Darstellung der erzeugten Punktzahl und der Fehlerrate im Vergleich mit ihren Näherungen in Abhängigkeit vom Winkelabstand ψ .

Es wird jener Zentriwinkel gesucht der die Funktion $f(\psi)$ minimiert und dabei kann über die Gewichtungsfaktoren eingestellt werden welchen Aspekt der Kostenfunktion bei der Optimierung stärker berücksichtigt wird. Für den Fall, dass die Lösung innerhalb der definierten Grenzen $[1^\circ, 25^\circ]$ liegt ist es ausreichend

$$\frac{d}{d\psi}f(\psi) = 0 \quad (4.27)$$

zu bilden und auf ψ aufzulösen, vgl. [50]. Für den angegebenen Bereich berechnet sich die Lösung zu

$$\psi = \sqrt{\frac{99.58c_t}{100.05c_p + 0.01c_f}} \quad (4.28)$$

und die vollständige Lösung des Optimierungsproblems lautet

$$\psi_0 = \begin{cases} 1^\circ, & \psi \leq 1^\circ \\ \psi, & \psi \in (1^\circ, 25^\circ) . \\ 25^\circ, & \psi \geq 25^\circ \end{cases} \quad (4.29)$$

Mit Hilfe der, in Gleichung 4.26 eingeführten, Gewichtung kann der optimal zu wählende Zentriwinkel beeinflusst werden. Bei gleichen Gewichtungen für

Laufzeit, Punktzahl und Fehlerrate, d.h. für eine Wahl von $c_t = c_p = c_f = 1$, ergibt sich ein optimaler Zentriwinkel von

$$\psi_0 = 5.39^\circ. \quad (4.30)$$

In Tabelle 4.3 sind einige Möglichkeiten zur Wahl der Gewichtung und die entsprechenden Lösungen angeführt. Wenn die Gewichtung für die Laufzeit erhöht wird, wird der verwendete Zentriwinkel erhöht und somit die verwendete Anzahl an Aufnahmen reduziert. Bei der Erhöhung der Gewichtung für die Punktzahl und die Fehlerrate wird jeweils der berechnete Winkel reduziert. Dieses Verhalten deckt sich mit den Erwartungen.

c_t	c_p	c_f	ψ_0
1	1	1	5.39°
2	1	1	7.63°
10	1	1	17.05°
1	2	1	3.82°
1	10	1	1.71°
1	1	10	5.31°
1	1	100	4.66°

Tabelle 4.3: Berechnung des optimalen Zentriwinkels ψ_0 bei unterschiedlichen Gewichtungen.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Framework entwickelt mit dem es möglich ist vollautomatisch 3D Modelle zu erstellen. Dazu wurde angenommen, dass das zu modellierende Objekt auf dem Boden vor der RGB-D Kamera steht und sich die Kamera in einer kreisförmigen Bewegung um dieses Objekt bewegt. Zur Bestimmung der Kamerabewegung wurde ein Frame-To-Keyframe Kameratracking mit Hilfe des DVO Algorithmus entwickelt. Der DVO Algorithmus schätzt dabei die Bewegung der Kamera zwischen zwei Aufnahmen indem der photometrische Fehler minimiert wird und die Auswahl der Keyframes erfolgt nach der Berechnung der Translation und der Rotation um alle Achsen. Am Ende der kreisförmigen Bewegung wird das Problem des Schleifenschlusses durch die Anwendung des ELCH Algorithmus gelöst.

Nachdem die Kamerabewegung geschätzt wurde, wird das Modell Aufnahme für Aufnahme erzeugt. Dabei beschreibt das Modell die Oberfläche des Objekts, die durch viele scheibenförmige Oberflächenelemente approximiert wird. Mit jeder Aufnahme werden korrespondierende Oberflächenelemente verbessert und neue Elemente hinzugefügt, um das Modell zu erweitern. Die Behandlung von Ausreißern erfolgt einerseits während der Modellierung und andererseits am Ende, wobei alle Oberflächenelemente gelöscht werden, die nicht mindestens aus vier unterschiedlichen Richtungen beobachtet wurden.

Die Genauigkeit des Frameworks wurde untersucht, indem einfache Objekte modelliert und mit der Realität verglichen wurden. Zusätzlich wurde ein Modell einem Laserscan gegenübergestellt und die Unterschiede analysiert. Abschließend wurde ein Optimierungsproblem formuliert und gelöst, um den optimalen Winkelabstand zwischen verwendeten Aufnahmen, unter Berücksichtigung der Laufzeit, des Detailgrads und der Fehlerrate, zu finden.

Der Vorteil dieses Frameworks, im Vergleich zu anderen Lösungen zur Objektmodellierung, liegt darin, dass es vollautomatisch funktioniert. Durch die getroffenen Annahmen der Kamerabewegung und der Position vor dem Sensor ist es möglich die Segmentierung des Objekts vollautomatisch umzusetzen. Es ist keinerlei menschlicher Eingriff notwendig. Wenn die RGB-D Kamera auf einem fahrbaren Roboter platziert wird, kann mit dem vorgestellten Framework, der Roboter ein Modell des Objekts vollkommen selbstständig erstellen. Der limitierende Faktor, ohne einen Beweis zu geben, ist im umgesetzten Kameratracking zu finden. Die akkumulierten Fehler, die während der Verfolgung der

Kamerabewegung entstehen, können nicht ausreichend beim Schleifenschluss behoben werden. Deswegen müssen die Parameter beim Suchen von korrespondierenden Oberflächenelementen robust gewählt werden. Wenn die Position der Kamera exakter bestimmt werden kann, wäre es möglich die Suche nach korrespondierenden Elementen „schärfer“ einzustellen und die erstellten Modelle wären genauer. Durch die vorgestellte Suche nach korrespondierenden Elementen und die anschließende Aktualisierung ist es möglich die Charakteristik der Erstellung des Modells einzustellen. Zum Beispiel kann der Suchradius während der Korrespondenzsuche vergrößert werden, um Oberflächen stärker zu glätten, oder verkleinert werden, um mehr Details zuzulassen.

Aktuelle Probleme des Frameworks sind die beschriebene Ungenauigkeiten im Kameratracking und die benötigte Laufzeit. Trotz Parallelisierung auf der CPU und Einteilung des ROI mittels eines Suchgitters, ist die Suche nach korrespondierenden Oberflächenelementen aufwändig und benötigt den größten Teil der Laufzeit. In einer nachfolgenden Arbeit sollte versucht werden die, gut parallelisierbare, Suche zu beschleunigen indem manche Berechnungen auf die Grafikkarte ausgelagert werden. Weiters kann das Framework auf einer steuer- und fahrbaren Plattform implementiert werden, um vollautomatisch Objekte zu erstellen. Dazu sollte eine flexible Suche nach dem Objekt dem Framework vorgeschaltet werden, damit sichergestellt wird, dass sich das Objekt vor der Kamera befindet.

Literatur

- [1] Z. Zhang, “Microsoft Kinect sensor and its effect.”, in *IEEE MultiMedia*. 2012, Bd. 19, S. 4–10.
- [2] T. Mörwald, “Object modelling for cognitive robotics”, Diss., Technische Universität Wien, Institut für Automatisierungs- und Regelungstechnik, 2013.
- [3] C. Kerl, J. Sturm und D. Cremers, “Robust odometry estimation for rgb-d cameras”, in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, IEEE, 2013, S. 3748–3754.
- [4] —, “Dense visual slam for rgb-d cameras”, in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, 2013, S. 2100–2106.
- [5] R. Jarvis, “A laser time-of-flight range scanner for robotic vision”, in. *IEEE Trans. Pattern Analysis Mach. Intell.*, 1983, Bd. PAMI-5, S. 505–512.
- [6] M. D. Adams, *Coaxial Range Measurement—Current Trends for Mobile Robotic Applications*. IEEE SENSORS JOURNAL, 2002, Bd. 2.
- [7] L. Marques, U. Nunes und A. T. de Almeida, “A new 3d optical triangulation sensor for robotics”, in *AMC’98 - Coimbra. 1998 5th International Workshop on Advanced Motion Control. Proceedings*. 3030 Coimbra, Portugal: IEEE, 1998, S. 512–517, ISBN: 0-7803-4484-7.
- [8] T. Kanade und T. Sommer, “An optical proximity sensor for measuring surface position and orientation for robot manipulation”, in *Proc. 3rd International Conference on Robot Vision and Sensory Control*. 1983, S. 667–674.
- [9] S. Lee, “Distributed optical proximity sensor system: hexeye”, in *Proc. IEEE Conf. on Robotics and Automation*. 1992, S. 1567–1572.
- [10] L. Marques, F. Moita, U. Nunes und A. T. de Almeida, “3d laser-based sensor for robotics”, in *Proc. IEEE 7th Mediterranean Electrotechnical Conf.* IEEE, 1998, S. 1328–1331.
- [11] S. Lee und J. Desai, “Implementation and evaluation of hexeye: a distributed optical proximity sensor system”, in *IEEE Conf. on Robotics and Automation*. 1995, S. 2353–2360.

- [12] (2006). Non-contact 3d digitizer vivid 9i/vi-9i, Adresse: http://www.konicaminolta.com/instruments/download/instruction_manual/3d/pdf/vivid-9i_vi-9i_instruction_eng.pdf (besucht am 21.09.2016).
- [13] G. Chesi, D. Prattichizzo und A. Vicino, “A visual servoing algorithm based on epipolar geometry”, in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, IEEE, Bd. 1, 2001, S. 737–742.
- [14] T. Brodsky, M. Lee und E. Cohen-Solal, *Self adjusting stereo camera system*, US Patent 8,085,293, 2011. Adresse: <https://www.google.com/patents/US8085293>.
- [15] A. Silverstein, *Single sensor chip digital stereo camera*, US Patent 7,061,532, 2006. Adresse: <https://www.google.com/patents/US7061532>.
- [16] D. Lee und I. Kweon, “A novel stereo camera system by a biprism”, *IEEE Transactions on Robotics and Automation*, Bd. 16, Nr. 5, S. 528–541, 2000.
- [17] K.-D. Kuhnert und M. Stommel, “Fusion of stereo-camera and pmd-camera data for real-time suited precise 3d environment reconstruction”, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, S. 4780–4785.
- [18] R. Valkenburg und A. McIvor, “Accurate 3d measurement using a structured light system”, in *Image and Vision Computing*. 1998, Bd. 16, S. 99–110.
- [19] E. Horn und N. Kiryati, “Toward optimal structured light patterns”, in *Image and Vision Computing*. 1999, Bd. 17, S. 87–97.
- [20] D. Nister, O. Naroditsky und J. Bergen, “Visual odometry”, in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2004.
- [21] P. Besl und N. McKay, “A method for registration of 3-d shapes”, in *IEEE Trans. Pattern Anal. March. Intell. (PAMI)*. 1992, Bd. 14.
- [22] S. Rusinkiewicz und M. Levoy, “Efficient variants of the ICP algorithm”, in *Intl. Conf. on 3-D Digital Imaging and Modeling (3DIM)*. 2001.
- [23] A. J. Davison, I. D. Reid, N. D. Molton und O. Stasse, “Monoslam: real-time single camera slam”, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Bd. 29, Nr. 6, S. 1052–1067, 2007.
- [24] A. Chiuso, P. Favaro, H. Jin und S. Soatto, “3-d motion and structure from 2-d motion causally integrated over time: implementation”, in *Computer Vision—ECCV 2000*, Springer, 2000, S. 734–750.
- [25] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox und N. Roy, “Visual odometry and mapping for autonomous flight using an rgb-d camera”, in *International Symposium on Robotics Research (ISRR)*, Bd. 2, 2011.

- [26] J. Prankl, A. Aldoma, A. Svejda und M. Vincze, “Rgb-d object modelling for object recognition and tracking”, in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, S. 96–103.
- [27] T. Weise, T. Wismer, B. Leibe und L. Van Gool, “Online loop closure for real-time interactive 3d scanning”, *Computer Vision and Image Understanding*, Bd. 115, Nr. 5, S. 635–648, 2011.
- [28] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison u. a., “Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera”, in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011, S. 559–568.
- [29] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges und A. Fitzgibbon, “Kinectfusion: real-time dense surface mapping and tracking”, in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, IEEE, 2011, S. 127–136.
- [30] J. Chandler und J. Fryer, “Autodesk 123d catch: how accurate is it”, *Geomatics World*, Bd. 2, Nr. 21, S. 28–30, 2013.
- [31] A. Collet, D. Berenson, S. S. Srinivasa und D. Ferguson, “Object recognition and full pose registration from a single image for robotic manipulation”, in *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*, IEEE, 2009, S. 48–55.
- [32] I. Tashev, “Recent advances in human-machine interfaces for gaming and entertainment”, *Int. J. Inform. Technol. Security*, Bd. 3, Nr. 3, S. 69–76, 2011.
- [33] S. Falahati, *OpenNI cookbook*. Packt Publishing Ltd, 2013.
- [34] R. B. Rusu und S. Cousins, “3d is here: point cloud library (pcl)”, in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, S. 1–4.
- [35] J. Sturm, N. Engelhard, F. Endres, W. Burgard und D. Cremers, “A benchmark for the evaluation of rgb-d slam systems”, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, S. 573–580.
- [36] S. Choppin. (2012). Coordinate system of a microsoft kinect, Adresse: <http://www.depthbiomechanics.co.uk/wp-content/uploads/2012/10/Kinect.jpg> (besucht am 10.08.2016).
- [37] P. Henry, M. Krainin, E. Herbst, X. Ren und D. Fox, “Rgb-d mapping: using depth cameras for dense 3d modeling of indoor environments”, in *Experimental robotics*, Springer, 2014, S. 477–491.

- [38] P. Newman und K. Ho, "Slam-loop closing with visually salient features", in *proceedings of the 2005 IEEE International Conference on Robotics and Automation*, IEEE, 2005, S. 635–642.
- [39] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige und W. Burgard, "G 2 o: a general framework for graph optimization", in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, S. 3607–3613.
- [40] J. Sprickerhof, A. Nüchter, K. Lingemann und J. Hertzberg, "An explicit loop closing technique for 6d slam.", in *ECMR*, 2009, S. 229–234.
- [41] J. Sprickerhof, "Effizientes schleifenschließen mit sechs freiheitsgraden in laserscans von mobilen robotern.", Diplomarbeit, Universität Osnabrück, 2009.
- [42] J. Canny, "A computational approach to edge detection", *IEEE Transactions on pattern analysis and machine intelligence*, Nr. 6, S. 679–698, 1986.
- [43] G. Bradski und A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. Ö'Reilly Media, Inc.", 2008.
- [44] J. Weickert, *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998, Bd. 1.
- [45] M. A. Fischler und R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", *Communications of the ACM*, Bd. 24, Nr. 6, S. 381–395, 1981.
- [46] L. Plunkett. (2010). Report: Here Are Kinect's Technical Specs, Adresse: <http://kotaku.com/5576002/here-are-kinects-technical-specs> (besucht am 21.09.2016).
- [47] D. Girardeau-Montaut. (2016). Introduction to cloudcompare, Adresse: <http://www.danielgm.net/cc/> (besucht am 11.09.2016).
- [48] B. Skinner, T. Vidal-Calleja, J. V. Miro, F. De Bruijn und R. Falque, "3d point cloud upsampling for accurate reconstruction of dense 2.5 d thickness maps", in *Australas. Conf. Robot. Autom.(ACRA)*, 2014.
- [49] W. Kemmetmüller und A. Kugi, *Regelungssysteme 1, Vorlesung*. TU Wien, ACIN, 2014.
- [50] A. Kugi, *Optimierung, Vorlesung und Übung*. TU Wien, ACIN, 2015.