Entwicklung eines sequenzbasierten route-followings anhand biologisch inspirierter Navigationsalgorithmen

Bachelorthesis von

Philipp Schulz
Hochschule Heilbronn
Robotik und Automation
SS 2016

Prof. Dr. Dieter Maier
Dipl. Geographin Iris Grixa

DLR Deutsches Zentrum für Luft- und Raumfahrt

HHN Hochschule Heilbronn
TECHNIK WIRTSCHAFT INFORMATIK

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abstract

The autonomy of robots is an increasingly popular field of research, as this creates many new opportunities for the use of machines within the industry or our everyday life. Self-reliant pathfinding is one of the most important aspects for independency of mobile robots. For this purpose, optical systems are widely used, as they can determine the environmental information quickly and comprehensively.

In this field of research, long term navigation is one of the most challenging tasks for mobile robots, due to the fact that many of algorithms need to save location data. This is needed to determine where they have already been. Therefore the longer the robot navigates, the memory requirement increases (problem of scalability).

This work introduces a new route-following algorithm, which is especially suited for long distance routes, while maintaining low memory requirements and computing time. The method is based on the SequenceSLAM by Michael Milford and Gordon Wyeth, which uses sequences of frames for localisation on a previously recorded path. First of all, the method will be adjusted for real world application.

Subsequently, a homing method is applied, which will be used to compensate path deviations. For this the navigational abilities of ants were analysed and adapted for mobile robotic systems.

Finally the algorithm will be tested and evaluated based on the results and prospects of future work will be given.

Eidestaatliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

05.09.2016 P.Schulz

1 Einleitung

Die Autonomie von Robotern ist ein seit Jahren immer beliebter werdendes Forschungsgebiet, da hierdurch viele neue Möglichkeiten für den Einsatz von Maschinen im Alltag oder in der Industrie entstehen. Bei mobilen Robotersystemen ist dabei die eigenständige Navigation einer der entscheidensten Aspekte für deren Selbsständigkeit.

Hier haben sich in den letzten Jahren optische Verfahren immer mehr etabliert, da Kameras recht günstig sind und umfassende Informationen über die Umwelt des Roboters liefern. Um die Navigation zu ermöglichen, muss auf Basis der Sensordaten eine Karte der Umgebung erstellt werden, in der sich das System lokalisieren kann. Diese kann bereits im Vorfeld der Navigation oder während der Operationszeit erstellt werden. Verfahren, die parallel zur Laufzeit eine Karte erstellen und sich in ihr lokalisieren, werden SLAM-Verfahren (Simultaneous Localization And Mapping) genannt. Zu den bekanntesten gehören der D-SLAM [1] und der LSD-SLAM [2].

Bei den SLAM Verfahren akkumulieren sich mit der Zeit Fehler in der Geometrie der Karte. Daher müssen solche Algorithmen erkennen, ob der aktuelle Aufenthaltsort des Roboters bereits von ihm besucht wurde, um die Karte zu korrigieren. Dies wird Loop-Closure Detektion genannt. Für die Korrektur müssen sich solche Algorithmen an genügend vielen Stellen eine Repräsentation des aktuellen Ortes merken. Dadurch wächst der Speicherbedarf mit der Laufzeit des SLAM Verfahrens. Zudem kann sich das Erscheinungsbild eines Ortes aufgrund verschiedener Lichtverhältnisse oder beweglicher Objekte mit der Zeit ändern, was die Detektion von loop-closures und somit eine Langzeitnavigation zusätzlich erschweren.

Für viele Navigationsaufgaben muss die Karte der Umgebung nicht während der Operationszeit erstellt werden, da sie, zusammen mit der abzufahrenden Strecke, bereits im Vorfeld bekannt ist. Solche Verfahren besitzen den Vorteil, dass der Speicherbedarf nicht während der Laufzeit weiter steigt. Zudem benötigen sie meistens weniger Rechenleistung für die Lokalisierung, da die Karte nicht parallel berechnet werden muss.

Diese Arbeit beschäftigt sich mit der Entwicklung eines solchen Navigationsverfahrens. Hierfür wird zunächst die Lokalisierung aufbauend auf dem bereits bestehenden SequenceSLAM beschrieben, wobei vor allem auf dessen Robustheit gegenüber Bildänderungen und die Gründe dafür eingegangen wird. Anschließend erfolgt die Einbindung einer Navigationsstrategie. Diese ist biologisch inspiriert und soll eine Möglichkeit bieten, Abweichungen von der vorgegebenen Strecke zu korrigieren. Abschließend folgen Versuche und deren Auswertung, sowie ein Ausblick auf weitere Implementierungen und Verbesserung.

2 Lokalisierung auf einer bekannten Route

Damit eine Roboternavigation stattfinden kann, muss, wie bei Menschen auch, zunächst bekannt sein, an welchem Ort sich dieser befindet. Dieser Schritt wird allgemein Lokalisierung genannt. Es gibt bereits zahlreiche Verfahren, die diese Aufgabe auf verschiedenste Art und Weise erfüllen. Eines davon ist der sogenannte SequenceSLAM. Er wurde 2012 von Michael Milford und Gordon Wyeth entwickelt und auf der ICRA (International Conference on Robotics and Automation) in Minnesota vorgestellt [3].

Für dieses Verfahren existieren sowohl eine frei zugängliche Matlab-Implementierung von Nico Sünderhauf[1] und eine darauf basierende, ebenfalls frei zugängliche C++-Implementierung[2], die als Grundlage für die nachfolgende Arbeit verwendet wurden.

2.1 Der SequenceSLAM

Der SequenceSLAM ist in seiner ursprünglichen Form ein Verfahren, das die Bilder zweier Bildsequenzen miteinander vergleicht und die jeweils am gleichen Ort aufgenommenen Bilder einander zuordnet. Die Bildsequenzen müssen hierfür nicht zu gleichen Tages- oder Jahreszeiten oder unter gleichen Beleuchtungsbedingungen aufgenommen werden. Dies wird durch seine extrem hohe Robustheit gegenüber Veränderungen im Erscheindungsbild eines Ortes ermöglicht. Selbst wenn die eine Sequenz im Sommer, die Vergleichssequenz aber im Winter aufgenommen wurde, können noch in ca. 30 % der Fälle die Orte korrekt einander zugeordnet werden, wobei keine falschen Lokalisierungen stattfinden.

Aufgrund dessen und seiner Schnelligkeit ist der Algorithmus als Grundlage für eine optische Navigation geeignet. Allerdings sind hierfür noch einige Anpassungen nötig, die nachfolgend beschrieben werden sollen.

Bevor auf diese eingegangen werden kann, soll auf die Funktionsweise des Algorithmus, seine Stärken und Schwächen und die Gründe hierfür ausführlich eingegangen werden.

[1] https://svn.openslam.org/data/svn/openseqslam/
[2] https://github.com/subokita/OpenSeqSLAM/

2.1.1 Aufnahme der Bildsequenzen

Die Lokalisierung basiert auf dem Vergleich einer Bildsequenz mit einer Datenbank. Daher muss die Route vor dem ersten Lokalisierungsdurchlauf einmal komplett abgefahren werden. Währenddessen nimmt eine Kamera in einem Video-Stream die Sicht des Roboters auf. Das Video wird anschließend in einzelne Bilder aufgespalten, welche daraufhin in einem Vektor in chronologischer Reihenfolge gespeichert werden. Die Frequenz für die Bildvektoren ist dabei so zu wählen, dass zwischen den Bildern ein ausreichend großer Abstand liegt. Ansonsten werden die Bildsequenzen zu unspezifisch und das Matching, also der Prozess bei dem nach Übereinstimmungen zwischen den beiden Datensätzen gesucht wird, unpräziser. Gleichzeitig darf der Abstand zwischen den Bildern auch nicht zu groß sein, da sonst die Auslösung der Strecke zu gering ist. Der Mindest- und der Maximalabstand hängen wiederum von der Änderungsrate der Umgebung ab. Je nach gefahrener Geschwindigkeit kann anhand der anderen Faktoren die Aufnahmefrequenz bestimmt werden, wobei es hierfür keine normierte Formel gibt. Die Bildfrequenz wurde bei den getesteten Datensätzen meist empirisch bestimmt.

Die zweite Sequenz, über die die Route gematcht werden soll, wurde in der ursprünglichen Veröffentlichung ebenfalls vorher komplett aufgenommen, bevor der Matching-Algorithmus startet. Wie bereits zuvor, wird die Route als Video-Stream aufgenommen und anschließend in einzelne Bilder aufgespalten, die danach in einem Vektor in chronologischer Reihenfolge gespeichert werden. Hierbei muss beachtet werden, dass die Bildfrequenz sowohl für den Referenzvektor, als auch für den Vergleichsvektor gleich sind, um plausible Ergebnisse zu erzielen.

2.1.2 Präprozessierung des Bildes

Um die Lokalisierungsergebnisse und die Rechenergebnisse zu verbessern, werden die verwendeten Bilder aus der Datenbank, sowie die Vergleichsbilder vorverarbeitet. Hierbei werden zunächst Farbbilder in Intensitätsbilder umgewandelt. Da für die Umsetzung des Programms die kostenlos verfügbare Bildverarbeitungsbibliothek „openCV“verwendet wurde, wurde das standardmäßig implementierte Luminanzverfahren für die Umwandlung der Farbwerte verwendet. Hierbei werden die verschiedenen Farbkanäle nach folgender Formel gewichtet:

$$D(x, y) = 0{,}299 * R + 0{,}587 * G + 0{,}114 * B \quad (1)$$

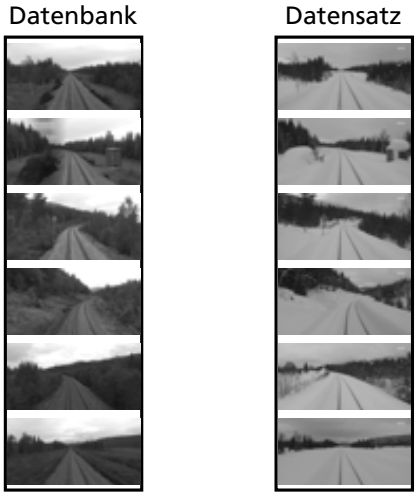


Abbildung 1: Beispielhafter Ausschnitt der beiden Vektoren für den SequenceSLAM. Links ist der Datenbank-Vektor, rechts der danach aufgenommene Vergleichsdatensatz. Die Bilder sind aus dem sogenannten Nordland-Datensatz (Näheres in Unterabschnitt 5.1.1)

Es können auch andere Umwandlungsverfahren verwendet werden, wobei sowohl bei den Referenzbildern der Datenbank als auch bei den Vergleichsbildern aus dem zweiten Datensatz die gleiche Methode gewählt werden muss. Ansonsten werden die resultierenden Ähnlichkeitswerte beeinflusst und können vom erwünschten Ergebnis abweichen.

Anschließend wird das Bild auf eine Größe von 64x32 Pixeln skaliert. Dieser Wert wurde in der ursprünglichen Veröffentlichung verwendet. In weiteren Studien wurde zudem gezeigt, dass höhere Auflösungen keine nennenswerten Verbesserungen der Ergebnisse zur Folge hatten [4]. Im Gegenteil wurden sogar mit einer Auflösung von nur 32 Bildpixeln noch beeindruckende Ergebnisse erzielt. Aufgrund der geringen Bildauflösung wird die Rechenzeit für die Berechnung der absoluten Summe der Grauwertdifferenzen stark verringert. Zudem kann so der Speicherbedarf für die Datenbanken verringert werden.

Der letzte Schritt ist eine sogenannte Patch-Normalisierung des Bildes. Dafür wird das Bild in kleinere Bereiche, sogenannte Patches, unterteilt. Jeder Patch wird anschließend anhand folgender Formel normalisiert:

$$g_{i,norm} = c_1 * (g_i - c_2) \tag{2}$$

Hierbei gilt:

$$c_1 = \frac{255}{g_{max} - g_{min}}$$

$$c_2 = g_{min}$$

g_i ist der i-te, g_{max} der maximale und g_{min} der minimale Grauwert des aktuellen Bereichs. Das normalisierte Ergebnis wird in $g_{i,norm}$ gespeichert. Dieser Rechenschritt erhöht den Kontrast des Bildes und ermöglicht dadurch eine Zuordnung von zwei Bildern, selbst wenn diese fast gar keine Ähnlichkeit mehr besitzen. Zudem wirkt die Normalisierung wie ein Kantenfilter, wodurch die gerade aufgenommene Szenerie besser von den anderen Bildern unterschieden werden kann.

Das so gewonnene Bild wird anschließend wieder für die weitere Nutzung in seinem zugehörigem Vektor gespeichert. Das Ergebnis der Präprozessierung ist in folgender Abbildung anhand eines Beispiels nochmals dargestellt.

Abbildung 2: Präprozessierung des Bildes für den SequenceSLAM. Das Bild stammt aus einem Drohnen-Rennen. Für die Visualisierung sind die Größenrelationen zueinander verschoben. Quelle: www.youtube.com/watch?v=EcLk_uZe33w

2.1.3 Lokalisierung der Datensätze

Nach der Präprozessierung der Bilder wird die Ähnlichkeit von jedem Bild aus dem Vergleichsvektor mit jedem Bild aus dem Referenzvektor bestimmt. Dies geschieht mithilfe der Summe der absoluten Grauwertdifferenzen.

$$SAD = \frac{1}{N} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} g_{Ref}(x,y) - g_{Match}(x,y) \tag{3}$$

W und H sind die Breite und die Höhe des Bildes in Pixeln. Bildet man das Produkt aus diesen beiden Größen, erhält man die Anzahl der Pixel N. g ist der Grauwert an der Stelle (x, y).

Außer dieser Formel wurden keine weiterführenden Filteroperationen für die Ähnlichkeitsbestimmung verwendet. Die für den Bildvergleich verwendete Methode ist allerdings nicht relevant. Wichtig ist nur, dass ein numerischer Wert bestimmt wird, der die

Ähnlichkeit zwischen den beiden Bildern beschreibt [3], da diese später für die Lokalisierung der besten Sequenz genutzt werden.

Anschließend wird jedes Ergebnis in eine Matrix eingetragen. Der Datenbankvektor wird in die Matrixreihen übertragen, während der Vergleichsvektor die Matrixspalten definiert. Das Ergebnis ist eine Differenzenmatrix, gefüllt mit numerischen Ähnlichkeitswerten. Diese wird in Abbildung 3 zur besseren Visualisierung als Grauwertbild dargestellt.

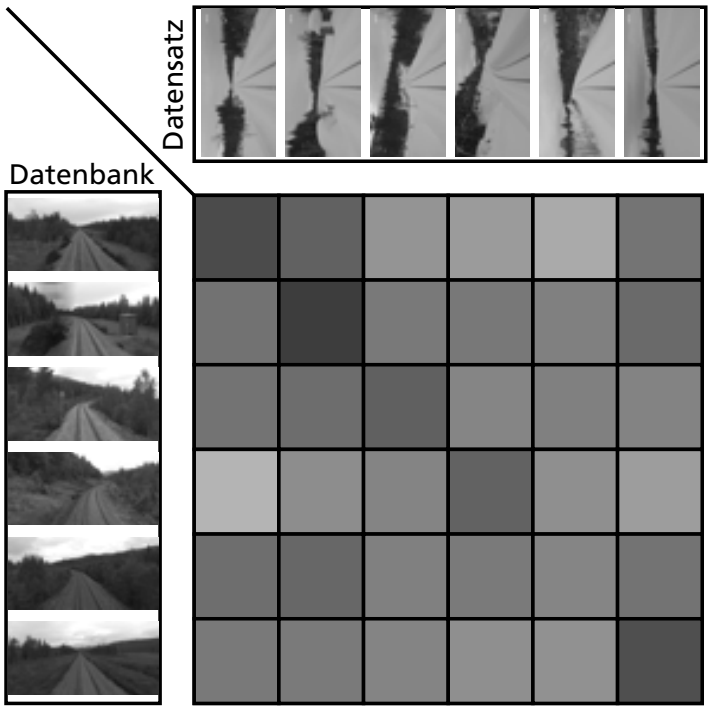


Abbildung 3: Schematische Darstellung der Differenzenmatrix aus den vorher gezeigten Datenvektoren. Je dunkler das Rasterfeld, desto ähnlicher sind sich die Bilder und desto wahrscheinlicher ist ein gefundener Match

Danach könnte bereits die Lokalisierung gestartet werden. Da sich die Bilder zwischen zwei Aufnahmen ändern können, wurde noch ein Zwischenschritt eingefügt, um die Orte trotzdem korrekt einander zuzuordnen. Dabei handelt es sich um ein Verfahren zur lokalen Kontrastverstärkung. Jedes Matrixelement wird mit seinen direkten Nachbarn aus der gleichen Spalte verglichen und mithilfe des Mittelwerts $\overline{D}_i$ und der Standardabweichung σ_i normalisiert (siehe Gleichung 4).

$$D_i = \frac{\overline{D}_i - D_i}{\sigma_i} \tag{4}$$

Die so ermittelten Werte werden anschließend positionsgetreu in eine neue Matrix geschrieben.

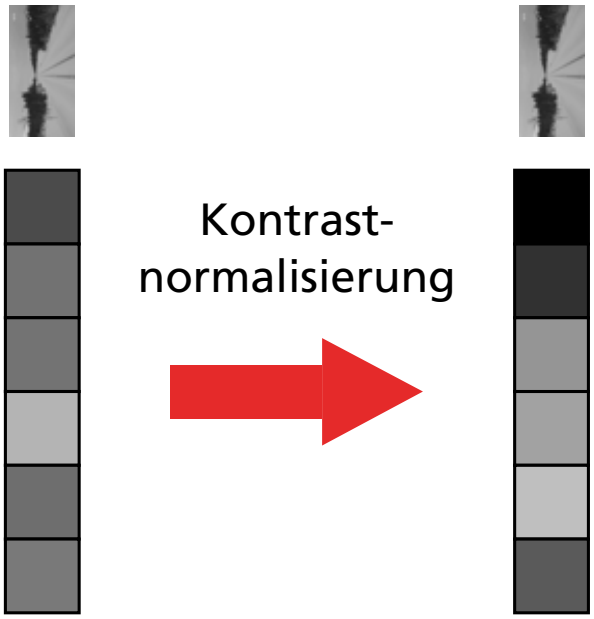


Abbildung 4: Kontrastnormalisierung eines Differenzenvektors aus der vorhergehenden Matrix. Dieser wird nach Gleichung 4 normalisiert, wobei hier zur besseren Übersichtlichkeit die restlichen Werte aus der Spalte, die für die Berechnung nötig sind, nicht dargestellt werden.

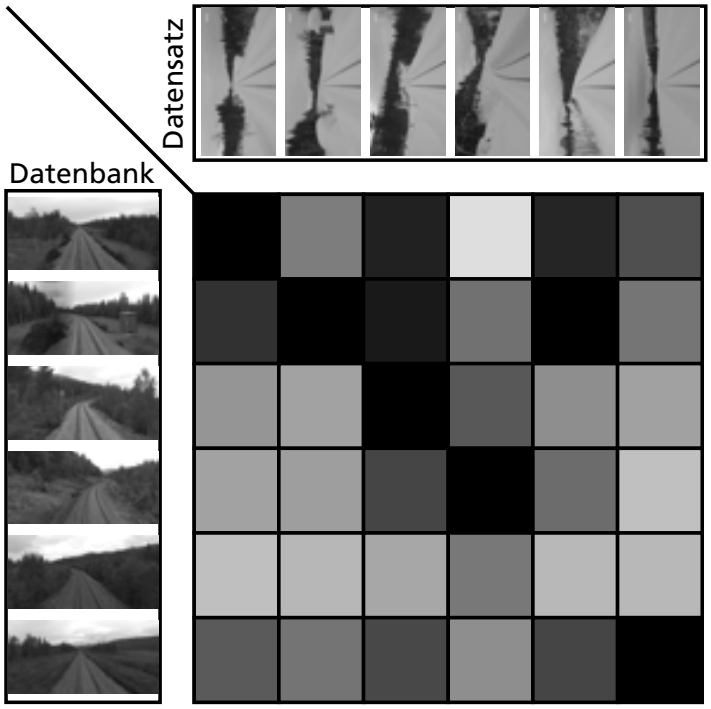


Abbildung 5: In dieser Abbildung ist die nach der Kontrastnormalisierung wieder zusammengesetzte Differenzenmatrix zu sehen. Die einzelnen Vektoren wurden wieder an ihre ursprünglichen Positionen zurückgeschrieben.

Abschließend findet die eigentliche Lokalisierung statt. Wie bereits erwähnt wird hierfür nach einer passenden Sequenz in der Datenbank gesucht. Da die Bilder chronologisch und in den gleichen Abständen in den Vektoren gespeichert wurden, bedeutet

das, dass die ähnlichsten Orte entlang einer Geraden mit dem Steigungswinkel -45° in der Matrix liegen müssen. Bei den Aufnahmen konnten jedoch Abweichungen in der Fahrgeschwindigkeit auftreten, weshalb noch weitere Geraden mit anderen Steigungen verwendet wurden.

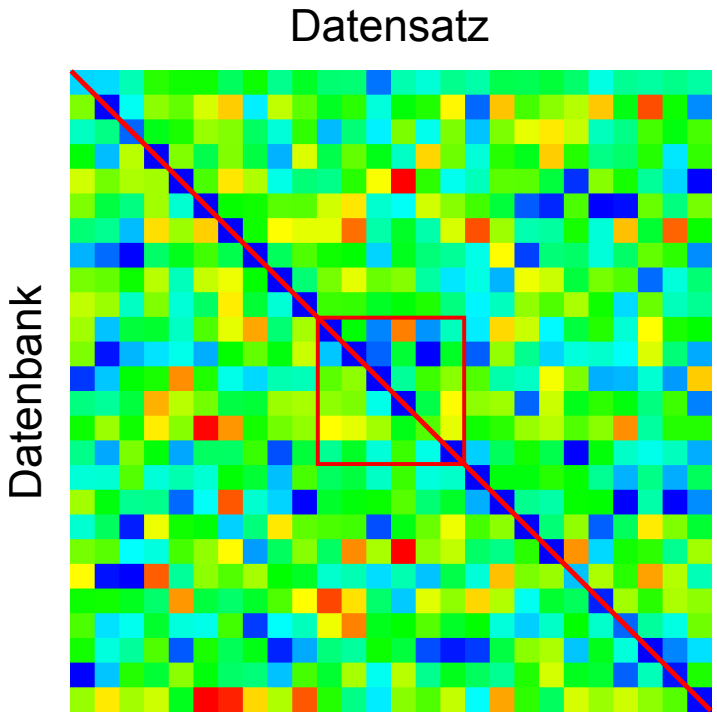


Abbildung 6: Schematische Darstellung der Sequenzfindung. Die Differenzenmatrix wird hierbei in Falschfarben dargestellt (blau entspricht einer sehr hohen Ähnlichkeit, rot einer sehr geringen). Die beste Sequenz ist entlang der roten Diagonale deutlich zu erkennen. Im roten Kasten ist die Differenzenmatrix aus dem vorherigen Beispiel.

Für das Matching werden die Geradengleichungen definiert und diskretisiert, sodass die x- und y-Werte der Geraden den Indizes der Matrix entsprechen. Anschließend wird der Startpunkt der Geraden nach und nach auf jeden Differenzenwert in der Matrix gelegt. Die Gerade, entlang derer der geringste Wert auftritt, wird als Resultat für den aktuellen Wert an dieser Position gespeichert. Für jede Spalte, also jeden Ort im Vergleichsvektor, werden so Vektoren mit Ergebniswerten erstellt.

Da vorher die höchste Ähnlichkeit zwischen zwei Werten als 0 definiert wurde, muss auch die Summe der korrekten Sequenz möglichst gering sein. Aus diesem Grund wird in jedem Ergebnisvektor der Minimalwert gesucht. Der gefundene Datenbankindex beschreibt den Ort, an dem sich die mobile Plattform momentan wahrscheinlich befindet und wird als Ergebnis der Lokalisierung ausgegeben.

Um jedoch zu verhindern, dass ein falsches Lokalisierungsergebnis ausgegeben wird,

wird dieses vorher noch überprüft. Dafür wird das zweitbeste Resultat aus dem Ergebnisvektor gesucht, wobei die Werte in der direkten Umgebung des besten Matches nicht berücksichtigt werden. Anschließend wird der beste Wert durch den zweitbesten dividiert. Das Resultat ist eine Faktor zwischen null und eins, der die Einzigartigkeit der aktuellen Lokalisierung angibt. Sollte es außer dem besten Match noch ein ähnlich gutes Ergebnis geben, geht der Wert gegen 1. Daher bedeutet ein kleiner Wert, dass die Lokalisierung mit einer höheren Wahrscheinlichkeit korrekt ist. Über einen Schwellwert können anschließend die falschen oder unsicheren Ergebnisse aussortiert werden.

2.2 Stärken und Schwächen

Tabelle 1: Auflistung der Stärken und Schwächen des SequenceSLAM Algorithmus

Stärken	Schwächen
• Der SequenceSLAM ist extrem robust gegenüber Helligkeitsänderungen	• Es handelt sich um ein rein passives Lokalisierungsverfahren, es können also keine Bewegungsinformationen anhand der Bilder ermittelt werden.
• Das Matching-Verfahren ist aufgrund des Grauwertvergleichs und der geringen Bildauflösung vergleichsweise schnell	• Es ist nur teilautonom, da die Route vorher einmal aufgenommen werden muss.
• Der Speicherbedarf (und damit die Matching-Zeit) hängt linear von der Größe der Datenbank ab und ändert sich nicht während der Laufzeit	• Der Blickwinkel und andere Kameraeinstellungen beeinflussen stark die Matching-Ergebnisse

In Tabelle 1 sind die Stärken und die Schwächen des SequenceSLAM aufgelistet. Dabei ist vor allem die Robustheit gegenüber Helligkeitsschwankungen des Algorithmus hervorzuheben. Ein entscheidender Faktor hierfür ist das Verwenden der lokal besten Ergebnisse anstatt der global besten. Dabei wird angenommen, dass der richtige Ort auf der Route ein deutlich besseres Ergebnis als seine direkten Nachbarn liefert. Gleich-

zeitig werden global sehr gute Ergebnisse dadurch entfernt, wenn sie in ihrer Umgebung nicht einzigartig genug sind. Dieses Verfahren ist nochmal grafisch in Abbildung 7 dargestellt.

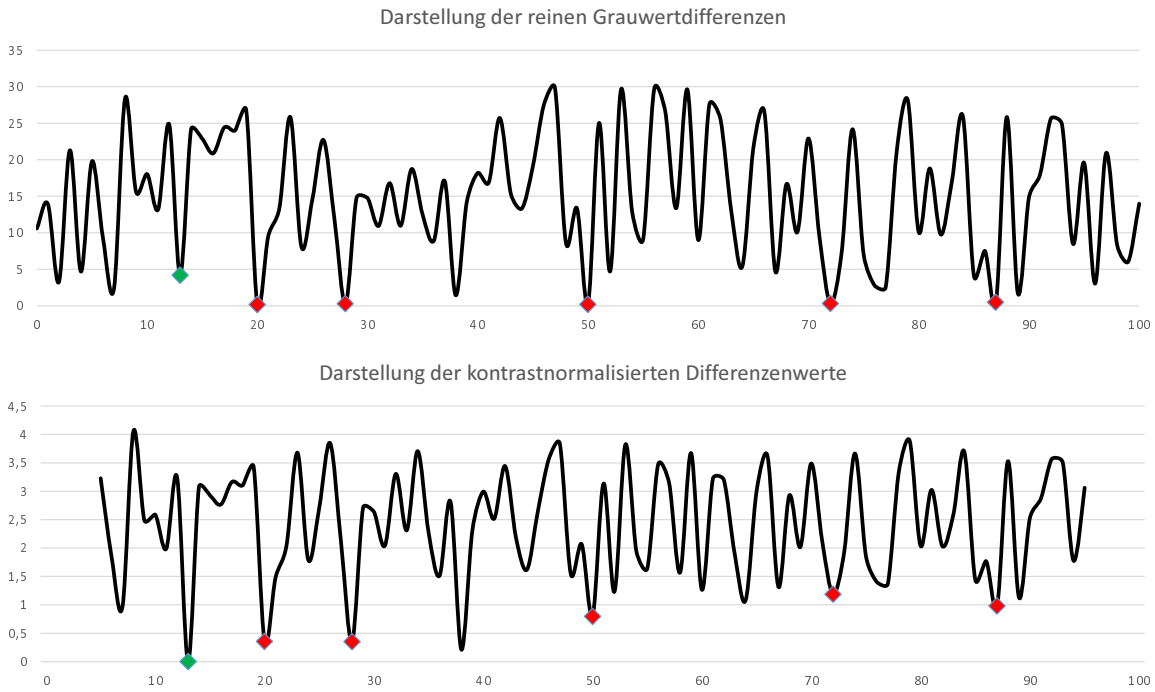


Abbildung 7: Schematische Darstellung des Differenzenvektors als Funktion der Differenzen über dem Bildindex. Oben ist der Ähnlichkeitsvektor vor der Kontrastverstärkung dargestellt, unten danach. Der grüne Punkt ist dabei der lokal beste Match, die roten Punkte die global besten Matches. Die Randwerte wurden in diesem Beispiel nicht miteinbezogen, da dort weniger als 10 Werte für die Kontrastnormalisierung vorlagen.

Allerdings werden durch diesen Schritt auch einige falsche Positionen als mögliche Lokalisierungen gewertet. Anhand der Sequenz können jedoch einzelne Ausreißer herausgefiltert werden (siehe Abbildung 6). Im optimalen Fall gibt es genau eine beste Sequenz von Bildern in der Differenzenmatrix. Damit dieser Fall sichergestellt werden kann, wird die Güte der Lokalisierung mithilfe der zweitbesten Sequenz bestimmt. Ist diese zu schlecht, wird der Ort als „false positive" deklariert und verworfen.

Die Bildunterschiede, die durch dieses Verfahren tatsächlich überwunden werden können, sind enorm. So können Orte, die einmal bei Tag und einmal bei Nacht aufgenommen wurden, tatsächlich noch korrekt einander zugeordnet werden. Ein weiterer Anwendungsfall war die Lokalisierung zu verschiedenen Jahreszeiten (siehe [5]). Auch hier konnte der SequenceSLAM mit sehr guten Ergebnissen überzeugen.

Sünderhauf erwähnt in seiner Arbeit zudem, dass die Ergebnisse stark vom Blickwinkel

und der perspektivischen Verzerrung der Kamera abhängen. Dadurch können Bilder, die nur um 1-2 Pixel zueinander verschoben sind, nicht mehr korrekt einander zugeordnet werden, auch wenn alle anderen Aufnahmebedingungen gleich bleiben.

Zusätzlich wurde das Verfahren ursprünglich nicht für ein autonomes Verhalten ausgelegt. Der Algorithmus lokalisiert sich rein passiv und wird nicht als live Anwendung verwendet. Schlussendlich ist auch noch keine Methode integriert, die eine Navigation nach erfolgreicher Lokalisierung ermöglicht.

3 Anpassungen des Algorithmus

3.1 Schwärzen des Himmels

Der Himmel hat einen starken Einfluss auf die Ermittlung der Ähnlichkeit zwischen den einzelnen Aufnahmen. Das Bild eines Ortes kann je nach Wetterlage stark in Helligkeit und Kontrast variieren. Das Sequenzmatching soll zwar gerade diese Störeffekte weitestgehend elminieren, trotzdem wäre es von Vorteil, wenn der Einfluss des Himmels auf die Lokalisierungsergebnisse eingedämmt werden kann.

Dafür haben Pepperel et al. das sogenannte „sky blackening“ eingeführt [6]. Hierbei handelt es sich um einen Prozess, der zuerst die Himmelsregion von der restlichen Umgebung separiert und anschließend komplett einschwärzt. Dadurch wird der Algorithmus noch robuster gegenüber Wetter- oder Tageszeitenschwankungen.

3.1.1 Bildtransformation zur Kontrasterhöhung des Himmels

Für die Detektion des Himmels wird zunächst ein RGB-Farbbild benötigt. Anschließend wird eine Transformation angewendet, die mithilfe der drei Farbkanäle den Kontrast der Himmelsregion erhöht [7].

$$C = -1{,}16 * R + 0{,}363 * G + 1{,}43 * B - 82{,}3 \tag{5}$$

Anhand dieser Transformation werden Bildpixel mit einem hohen Blau- und Grünanteil aufgehellt, während solche mit einem hohen Rotanteil verdunkelt werden. Da der Himmel von Natur aus einen sehr hohen Blau-Anteil besitzt und im Allgemeinen auch heller als die Umgebung ist, wird er aufgehellt. Der Boden hingegen ist dunkler und besitzt höhere Rotanteile als der Himmel. Durch die Transformation werden die Pixelwerte

vom normalen Grauwertbereich in einen neuen Wertebereich von −378,1 bis 374,915 verschoben.

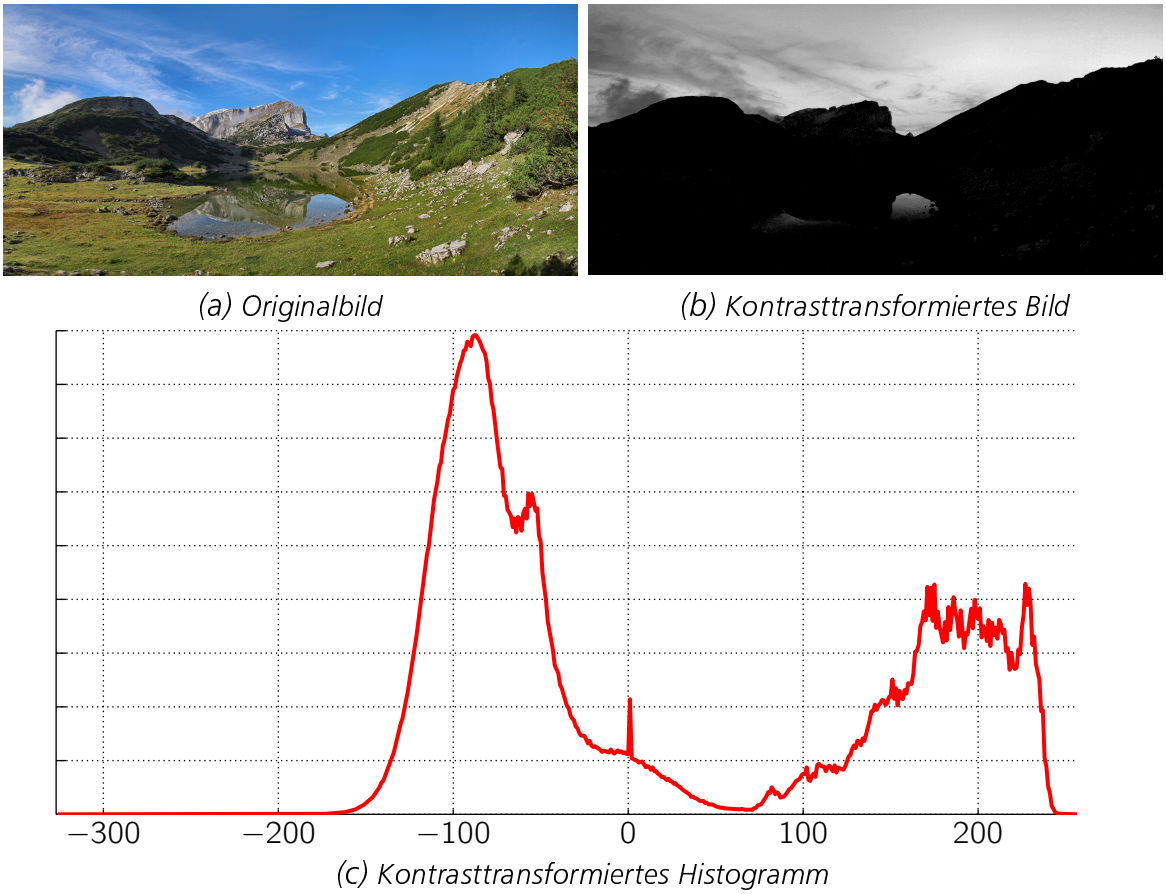


Abbildung 8: Kontrasttransformation einer Landschaftsaufnahme. (a) zeigt das Originalbild, (b) das transformierte Bild und (c) das Histogramm des Transformationsbildes. Das Bild stammt von https://commons.wikimedia.org/w/index.php?curid=1262156 (fotografiert und hochgeladen von Karsten Dörre)

Damit die standardisierten Bildverarbeitungsalgorithmen auf dieses Histogramm angewendet werden können, müssen die Werte noch in einen 8-Bit Grauwertbereich transformiert werden. Hierfür kann entweder das Autoscaling-Verfahren angewendet werden, oder die Histogrammwerte links und rechts der unteren und oberen Grauwertgrenze abgeschnitten werden.

In diesem Fall wurde das Histogramm auf einen bestimmten Bereich zugeschnitten. Grauwerte, die sich außerhalb der Bereiche befanden wurden auf den am nächsten liegenden Grenzwert transformiert. Somit wird beinahe der komplette Vordergrund im so entstandenen Bild schwarz, während der Himmelsbereich in helleren Grauwertstufen dargestellt wird (siehe Abbildung 8b).

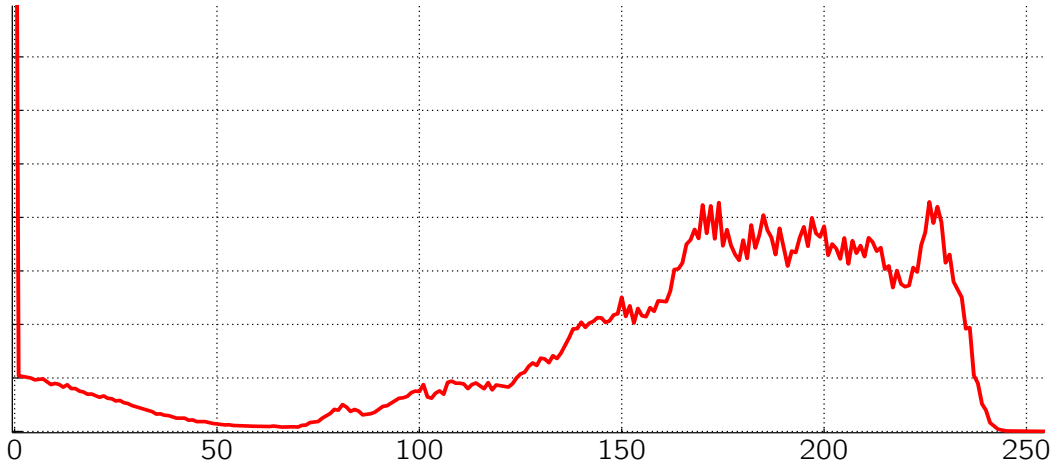


Abbildung 9: Das Histogramm nach der Transformation auf den 8-Bit Grauwertbereich. Alle Pixel, deren Werte vorher kleiner als 0 waren, sind nun genau auf der Nullachse. Daher verlässt das Histogramm an dieser Stelle den dargestellten Wertebereich.

3.1.2 Segmentation des Himmels

Für die Segmentierung der Himmelsregion wird eine binäre Maske berechnet. Diese sorgt dafür, dass der Himmel komplett geschwärzt wird, während die restlichen Bereiche ihre Grauwerte beibehalten. Der Schwellwert für die Binarisierung wird dabei für jedes Bild einzeln und automatisch berechnet. Das wohl bekannteste Verfahren zur Berechnung des Schwellwerts ist die sogenannte Otsu Methode [8], die im folgenden kurz beschrieben werden soll.

Das Verfahren wurde 1975 von Nobuyuki Otsu entwickelt und berechnet automatisiert die optimale Binarisierungsschwelle für ein Grauwertbild. Hierfür wird zunächst das Histogramm des Bildes berechnet. Dieses wird dann mithilfe eines Schwellwerts in zwei Klassen unterteilt. Somit enthält die erste Klasse die Grauwertverteilung unter, die zweite hingegen die Verteilung über dem Schwellwert.

$$P_0(t) = \sum_{i=0}^{t} p_i \text{ und } P_1(t) = \sum_{i=t+1}^{255} p_i = 1 - P_0(t) \tag{6}$$

$$\overline{g}_0 = \frac{1}{P_0(t)} \sum_{i=0}^{t} p_i * i \tag{7}$$

$$\overline{g}_1 = \frac{1}{P_1(t)} \sum_{i=t+1}^{255} p_i * i \tag{8}$$

Dabei ist t der aktuelle Schwellwert im Histogramm, P_0 und P_1 sind die aufaddierten Häufigkeiten der beiden Klassen und $p(i)$ ist die Häufigkeit eines einzelnen Grauwerts.

Der nächste Schritt ist die Berechnung der Varianzen, also der Streuung der Werte, innerhalb der beiden Klassen (siehe [8]).

$$\sigma_0^2 = \frac{1}{P_0} \sum_{i=0}^{t} (i - \overline{g}_0)^2 * p_i \tag{9}$$

$$\sigma_1^2 = \frac{1}{P_1} \sum_{i=t+1}^{255} (i - \overline{g}_1)^2 * p_i \tag{10}$$

Bei σ_0 und σ_1 handelt es sich um die Varianzen innerhalb der beiden Klassen. Das Ziel der Otsu-Methode ist herauszufinden, bei welchem Schwellwert t die beiden Varianzen innerhalb der beiden Klassen minimal werden. Dafür wird folgende Formel angewendet:

$$\sigma_i^2(t) = P_0(t) * \sigma_0^2(t) + P_1(t) * \sigma_1^2(t) \tag{11}$$

Diese Berechnung benötigt allerdings sehr viele Rechenschritte, bis alle Schwellwerte geprüft und der minimale gefunden werden kann, weswegen er programmatisch nicht allzu effizient arbeitet. Eine schnellere Methode ist das Berechnen der Varianz zwischen den beiden Klassen, da hierfür nur die Mittelwerte und die Wahrscheinlichkeiten der beiden Klassen benötigt wird.

$$\begin{aligned} \sigma_z^2(t) &= P_0(\overline{g}_0 - \overline{g}_{ges})^2 + P_1(\overline{g}_1 - \overline{g}_{ges})^2 \\ &= P_0 P_1(\overline{g}_0 - \overline{g}_1)^2 \end{aligned} \tag{12}$$

Hierbei hat Otsu gezeigt, dass der Grenzwert t, an dem die Varianz innerhalb der Klassen minimal wird, gleichzeitig auch die Stelle ist, an der die Varianz zwischen den Klassen maximal wird. Da die zweite Methode aufgrund der eingesparten Rechenschritte und der einfach umzusetzenden Formel deutlich effizienter arbeitet, wird diese in den meisten Fällen zur Schwellwertsberechnung verwendet.

Aufgrund seiner Einfachheit und den guten Ergebnissen, wird die Otsu Methode oft für automatisierte Schwellwertdetektion eingesetzt. Aufgrund der Verwendung von zwei Klassenvarianzen kann sie allerdings nur für bimodale Grauwertverteilungen, also Verteilungen mit zwei lokalen Maxima, verwendet werden. Gehören sehr kleine Bereiche des Bildes dem Himmel oder dem Boden an, wird die Grauwertverteilung annähernd unimodal.

Daher wurde anstatt der Otsu Methode die sogenannte valley emphasis Methode angewendet. Diese baut auf der Otsu-Methode auf und verwendet bis zu Gleichung 12 die gleichen Rechenschritte. Bei der Berechnung der maximalen Varianz zwischen den

Klassen wird jedoch noch eine Gewichtung W hinzugefügt [9].

$$W = (1 - p_t) \tag{13}$$

Diese Gewichtung ist antiproportional zum Häufigkeitswert der aktuellen Schwelle. Dadurch werden die Ergebnisse für Schwellwerte in globalen Minima des Histogramms deutlich verbessert, während alle anderen Werte verschlechtert werden.

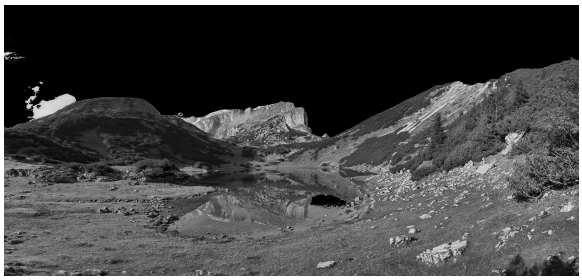
$$\sigma_z^2 = W * P_0 P_1 (\overline{g}_0 - \overline{g}_1)^2 \tag{14}$$

Wie zuvor wird nach dem Schwellwert gesucht, bei dem die gewichtete Varianz zwischen den Klassen maximal wird. Dieser wird anschließend zur Berechnung der Bildmaske verwendet. Bei Nachtaufnahmen ist das Verfahren nicht nötig, da hier der Himmel bereits schwarz erscheint. Sollten zudem Bilder in Gebäuden oder anderen Gegenden, wo kein Himmel zu sehen ist, aufgenommen werden, sollte es ebenso nicht angewendet werden, da ansonsten Bildinformationen unnötig verloren gehen.

(a) Originalbild

(b) Kontrasttransformiertes Bild

(c) Bildmaske



(d) Bild mit eingeschwärztem Himmel

Abbildung 10: Die vier Transformationsschritte vom Originalbild (a) bis zum Grauwertbild mit eingeschwärztem Himmel (d)

Durch das Verfahren wird zuverlässig der Himmel vom restlichen Bild separiert und eingeschwärzt. In Abbildung 10 ist allerdings zu sehen, dass Bereiche des Sees fälschlicherweise dem Himmel zugeordnet werden, während eine der Wolken nicht komplett geschwärzt wird. Diese falschen Segmentierungen machen hierbei nur einen kleinen Anteil des Bildes aus, weswegen sie nur einen geringen Einfluss auf das Matching-Er-

gebnis besitzen. Trotzdem sollten diese Bilder nicht an Seen oder Landschaften mit sehr hellen Flächen verwendet werden, da sonst zu viele Bildinformationen fälschlicherweise eingeschwärzt werden. Das so erhaltene Grauwertbild wird anschließend wie zuvor vorverarbeitet und gespeichert. Auf diese Weise wird mit allen Bildern der Datenbank und der Datensätze verfahren, vorausgesetzt sie erfüllen die bereits erwähnten Bedingungen.

3.2 Vorbereitungen für die Live-Anwendung

Damit der Algorithmus zum autonomen Folgen einer Route verwendet werden kann, müssen zuerst die Verfahren auf eine Anwendung mit Live-Bildern angepasst werden. Hierfür wurde anstatt einer statischen Matrix, die einmal erstellt und anschließend nicht mehr verändert wird, eine dynamische Matrixerzeugung gewählt. Die Matrix wird während der Laufzeit des Algorithmus ständig angepasst und durch den Differenzenvektor des aktuellen Frames mit allen Datenbankbildern ergänzt.

3.2.1 Dynamische Matrixerzeugung

Im Gegensatz zum Original wird der Vergleichsdatensatz während der Laufzeit erstellt. Dafür werden die Bilder einer Kamera ausgelesen und, nachdem sie vorverarbeitet wurden, in einem Vektor mit vordefinierter Länge gespeichert. Sobald der Vektor komplett gefüllt ist, wird das älteste Bild gelöscht und dafür die neueste Aufnahme gespeichert. Dadurch entsteht nach und nach ein eindimensionaler Vektor, aus dem zusammen mit der Datenbank, die wie im originalen Algorithmus vorher aufgenommen und vorverarbeitet wurde, wieder eine Ähnlichkeitsmatrix berechnet werden kann.

Diese wird allerdings nicht bei jeder Aufnahme eines neuen Bildes komplett neu berechnet. Stattdessen wird auch hier eine feste Größe für die Matrix vorgegeben. Die Anzahl der Zeilen entspricht der Menge der Bilder in der Datenbank und die Spalten werden auf die gleiche Größe wie der Vergleichsvektor begrenzt. Sobald das Bild ausgelesen und im Vergleichsvektor gespeichert wurde, wird die Ähnlichkeit des aktuellen Bildes zu jedem einzelnen Datenbankbild bestimmt. Diese werden wiederum temporär in einem eindimensionalen Vektor gespeichert.

Anschließend wird wieder die Kontrastnormalisierung, wie beim originalen Algorithmus, durchgeführt. Das Ergebnis ist ein eindimensionaler, kontrastnormalisierter Differenzenvektor, der die Ähnlichkeitsinformationen des aktuellen Bildes mit den Datenbankbildern enthält. Anhand der Anzahl der vorher aufgenommenen Bilder wird der

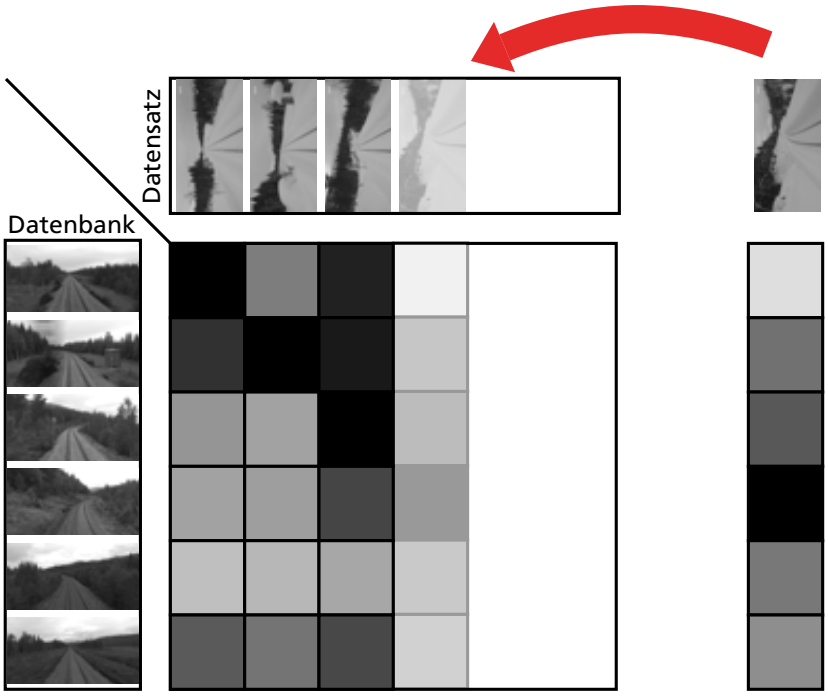


Abbildung 11: Darstellung der dynamischen Matrixerzeugung. Der Vektor wird direkt nach der Bildaufnahme berechnet und normalisiert. Die Differenzenmatrix wird während der Laufzeit nach und nach erstellt.

Vektor in die entsprechende Spalte der Matrix geschrieben. Dadurch wird die Rechenzeit verringert, da statt einer $N \times M$ Matrix nur noch ein Vektor pro Durchlauf berechnet werden muss.

Wie zuvor beim Datensatzvektor auch, wird auch hier die älteste Spalte gelöscht, sobald die Anzahl der Bilder die maximale Datensatzgröße überschreitet. Damit jedoch die Lokalisierung korrekt durchgeführt werden kann, muss sichergestellt werden, dass die Bilder aus der Datenbank und aus dem Vergleichsdatensatz ungefähr an den gleichen Positionen aufgenommen werden. Sollte dies nicht der Fall sein, können die Ähnlichkeitswerte nicht mehr korrekt ermittelt werden und die Ergebnisse verschlechtern sich dementsprechend. Im Folgenden sollen einige Verfahren vorgestellt werden, die die Aufnahmepunkte bei beiden Routen synchronisieren.

3.2.2 Synchronisation der Aufnahmepunkte

Die einfachste Methode ist hierbei das Verwenden einer zeitlichen Synchronisation zwischen Datensatz und Datenbank. Diese ist allerdings nicht immer optimal, da sich die Aufnahmepunkte immer weiter voneinander entfernen, sollten die Geschwindigkeiten nicht komplett gleich sein. Zudem besitzt sie den Nachteil, dass der Startpunkt immer gleich sein muss, da sonst die nachfolgenden Aufnahmepunkte, selbst bei gleichen Fahrgeschwindigkeiten, genau zwischen den Datenbankbildern liegen. Bei hohen

Aufnahmeraten und bekannten Geschwindigkeiten während der Aufnahme der Datenbank und des Datensatzes, ist die zeitliche Synchronisation trotzdem anwendbar.

Eine weitere Methode, die bereits in früheren Arbeiten zum SequenceSLAM verwendet wurde, ist das Verwenden von Odometrie-Daten. Da inzwischen jedes Fahrzeug, sowohl zu Land, als auch in der Luft, eine Sensorik zur Überwachung der Beschleunigungen und der Geschwindigkeiten besitzt, können die Positionsdaten genau berechnet werden. Dafür werden zunächst die Abstände zwischen den Aufnahmepunkten der Referenzfahrt bestimmt und gespeichert. Bei der zweiten Fahrt werden ebenfalls die Abstände des Fahrzeugs zum letzten Aufnahmepunkt gemessen und, sobald dieser einen bestimmten Schwellwert überschreitet, die Aufnahme ausgelöst.

Dieses Verfahren liefert zwar nicht die genaue Absolutposition (ground-truth), ist aber in den meisten Fällen ausreichend, um die Aufnahmepositionen miteinander zu synchronisieren. Hierfür müssen nicht korrigierbare Störfaktoren, wie ein Rutschen der Räder, ausgeschlossen werden können. Zudem können hierdurch im Gegensatz zur zeitlichen Synchronisation Geschwindigkeitsunterschiede während der Aufnahme ausgeglichen werden. Da jedoch keine ground-truth-Daten vorliegen, bleibt auch hier das Problem, dass der Startpunkt bei beiden Aufnahmen gleich sein muss, um ein korrektes Matching zu gewährleisten.

Die letzte Methode zur Synchronisation ist das Verwenden von übergeordneten Lokalisierungsmethoden wie zum Beispiel das Global Positioning System (GPS). Dadurch können für jeden Aufnahmepunkt ground-truth-Daten bereitgestellt werden und somit eine perfekte Synchronisation unabhängig von den Geschwindigkeiten und dem genauen Startpunkt gewährleistet werden. Der Nachteil hierbei ist, dass der Algorithmus dann nicht in Gegenden eingesetzt werden kann, wo es keine solch übergeordnete Systeme gibt. Außerdem wäre dann in jedem Fall aufwendige, externe Sensorik notwendig, wodurch die Komplexität des Aufbaus gesteigert werden würde. Die vorherigen Methoden würden mit der bordeigenen Sensorik funktionieren.

Die Vor- und Nachteile von allen drei vorgestellten Verfahren sind zur besseren Visualisierung nochmals in nachfolgender Tabelle aufgelistet.

Aufgrund der aufgezeigten Stärken und Schwächen wird empfohlen, eine odometrische Synchronisation für eine reale Anwendung zu verwenden. In Simulationsumgebungen, in denen alle Parameter bekannt sind und die Experimentierbedingungen dementsprechend reproduziert werden können, reicht auch eine zeitliche Synchronisation der Aufnahmepunkte.

Tabelle 2: Übersichtstabelle über die Vor- und Nachteile der verschiedenen Synchronisationsarten

Art der Synchronisation	Startplatz-unabhängig	Geschwindigkeits-unabhängig	mit bordeigener Sensorik
zeitlich	✗	✗	✓
odmetrisch	✗	✓	✓
extern	✓	✓	✗

Die externe Synchronisation ist nur in wenigen Einzelfällen sinnvoll, da die benötigte Sensorik und deren Einbindung extrem hohen Aufwand und zusätzliche Kosten bedeuten würde. Zudem werden optische Navigationsverfahren meistens in Gegenden benötigt, wo solch eine externe Lokalisierung nicht möglich ist.

3.3 Dynamische Ermittlung der Sequenzlänge

Milford und Wyeth zeigen, dass mithilfe des SequenceSLAM eine Matching-Quote von ca. 33% erreicht werden konnte, ohne dass dabei falsche Lokalisierungen stattfanden [3]. Obwohl dieses Ergebnis im Hinblick auf die großen Unterschiede zwischen den Aufnahmen erstaunlich war, wird für eine erfolgreiche Navigation eine deutlich höhere Lokalisierungsrate benötigt.

Sünderhauf et al. [5] haben den Einfluss einzelner Parameter auf die Matchingergebnisse ausführlich untersucht. Die im Hinblick auf die Lokalisierungsergebnisse entscheidenden Parameter sind demnach die Länge der verwendeten Sequenz und das Vergleichsverfahren. Eine längere Sequenz führt zu einer höheren Lokalisierungsrate, aber auch zu ungenaueren Ergebnissen und einer größeren Zahl an falschen Matches. Zudem wird gezeigt, dass es eine maximale Sequenzlänge gibt, ab der die Zahl an gefundenen Matches sogar wieder abnimmt, da die Sequenzen ab da zu unspezifisch werden. In ihren Experimenten lag diese Schwelle bei 100 Bildern. Zudem bedeutet eine längere Sequenz an Bildern auch mehr Rechenoperationen und damit eine erhöhte Rechenzeit.

Um die Vorteile kurzer (geringere Rechenzeit und genauere Lokalisierung) und langer (robustere Lokalisierung) Sequenzen zu kombinieren, wurde eine dynamische Anpassung dieser Größe während der Laufzeit implementiert. Jedes Bild wird dabei zuerst

mithilfe der Standardsequenzlänge überprüft. Sollte das Ergebnis unter dem gewählten Schwellwert sein, wird es als wahr gewertet und die Position in der Datenbank ausgegeben. Liegt es dahingegen über dem Schwellwert, werden im nächsten Schritt doppelt so viele Bilder in die Sequenz geladen. Anschließend wird wieder gematcht und das Ergebnis ausgewertet. Sollte die aktuelle Lokalisierung wieder zu schlecht sein, wird der Vorgang solange wiederholt, bis die maximale Sequenzlänge von 100 Bildern erreicht wurde, oder keine weiteren Datensatzbilder im Vektor mehr gespeichert sind.

Wenn ab diesem Punkt immer noch kein zufriedenstellendes Ergebnis ermittelt werden konnte, wird das aktuelle Bild als „nicht lokalisierbar" deklariert und das nächste Bild in den Vergleichsvektor geladen. Auf diese Weise wird jeder Ort so präzise wie möglich lokalisiert, während die Laufzeit so gering wie möglich gehalten wird.

3.4 Sequenzbildung

Ein weiterer Aspekt, der die Live-Anwendung des Algorithmus behinderte, war die mathematische Berechnung der Matching-Geraden. Diese besaßen, egal mit welcher Steigung, immer den gleichen Startpunkt. Der Start wurde dabei vom aktuell zu matchenden Bild um 10 Bilder, oder der aktuell gewählten Sequenzlänge, in der Datenbank nach hinten gelegt.

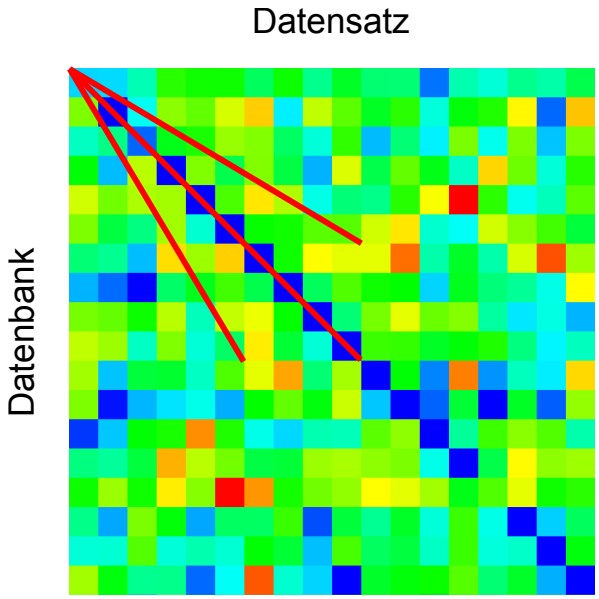


Abbildung 12: Beispiel für die originale Berechnung der Geraden. Der Startpunkt der Geraden wird vom eigentlichen Bild (Endpunkt der Geraden mit 45° Steigung) die Anzahl der Bilder in einer Sequenz in die Vergangenheit gelegt.

$$\vec{g}_i = (n + v * i) + (m - 10 + i) * D_{max} \tag{15}$$

Bei einer Geraden mit 45° Steigung würde sie also genau den aktuell zu lokalisierenden Punkt in der Differenzenmatrix schneiden. Für die weiteren Geraden gilt diese Tatsache allerdings nicht. Genau genommen wird somit das Bild in der Vergangenheit und nicht das aktuelle Bild lokalisiert. Um Fehlern durch diese Art der Berechnung vorzubeugen, wurde der Startpunkt, ähnlich wie in [10] auf das aktuelle Bild gelegt und die Gerade von dort in die Vergangenheit gebildet. Dadurch ergibt sich folgende Formel für die jeweiligen Geraden.

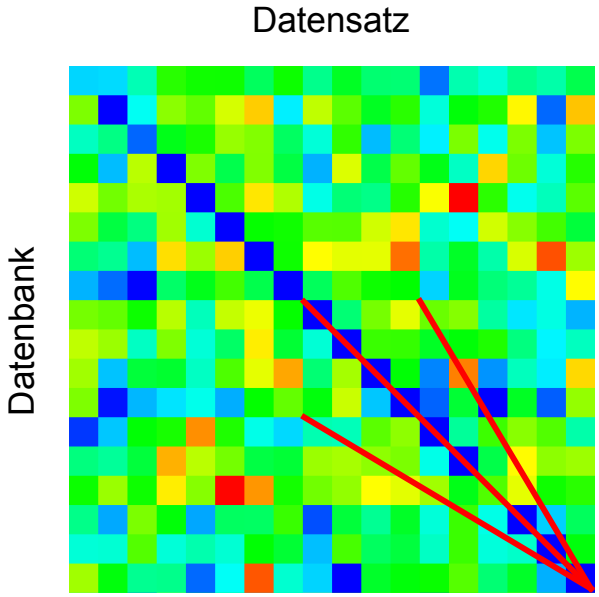


Abbildung 13: Beispiel für die an die live-Anwendung angepasste Berechnung der Geraden. Der Startpunkt liegt nun immer beim aktuellen Bild und die Geraden werden in die Vergangenheit gelegt.

$$\vec{g}_i = (n - v * i) + (m - i) * D_{max} \tag{16}$$

Hierbei ist $\vec{g}$ der Vektor mit den Indizes der jeweiligen Bilder entlang der Geraden, m das aktuelle Sequenzbild und n das aktuell gematchte Datenbankbild. Um verschiedene Geraden mit verschiedenen Geschwindigkeiten zu ermitteln, wird diese noch als Steigungsfaktor in die Berechnung mit einbezogen. D_{max} beschreibt die maximale Größe des Vergleichsvektors. Auf diese Weise wird ein Vektor mit den Indizes der Differenzenwerte in der Matrix berechnet. Da diese immer natürliche Zahlen sein müssen, werden die ermittelten Ergebnisse noch abgerundet. Zudem findet eine Bereichsüberprüfung statt, damit nicht auf Speicher außerhalb der Matrix zugegriffen wird.

Der nächste Schritt ist dann, wie bereits zuvor, die Addition der einzelnen Werte und das Ermitteln des besten Ergebnisses. Werden die Aufnahmepunkte der beiden Fahrten

zudem perfekt synchronisiert, liegt der beste Match genau auf einer Geraden mit der Steigung -1. Dadurch kann dann die Geschwindigkeit als Steigung für die Berechnung weggelassen werden. Da dies allerdings nicht sichergestellt werden kann und die zusätzliche Rechenzeit keinen großen Einfluss auf die Gesamtrechenzeit besitzt, wurde dieser Faktor trotzdem beibehalten.

3.5 Umgang mit sich selbst wiederholenden Routen

Je nachdem wie die Route aufgebaut ist, kann es geschehen, dass einzelne Streckenabschnitte mehrmals passiert werden müssen. Dementsprechend wird dieser Abschnitt auch mehrfach aufgenommen und für eine Karte oder eine Datenbank gespeichert.

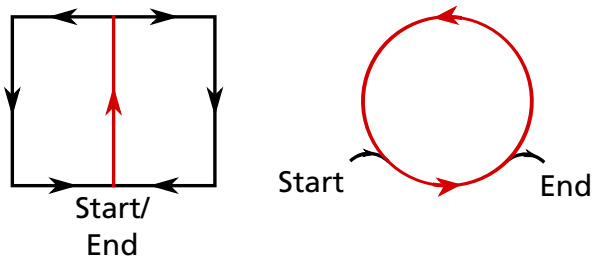


Abbildung 14: Beispielhafte Darstellung zweier Routen, bei denen Streckenabschnitte mehrfach nachgefahren werden. Beim linken Beispiel wird der mittlere Streckenabschnitt mehrfach passiert, beim rechten wird der gesamte Kreis nehrfahch umfahren, bevor das Fahrzeug beim Endpunkt ankommt.

Der ursprüngliche SequenceSLAM kann solche mehrfach vorkommenden Routenabschnitte nicht korrekt zuordnen. Sind zwei gleiche Orte in der Datenbank vorhanden können auch zwei gleich gute Ergebnisse beim Matching gefunden werden. Während eines der beiden Ergebnisse als bester Match gewertet wird, wird das andere als zweitbester Match deklariert. Die anschließende Überprüfung würde daher einen Wert von 1 ergeben, wodurch die Lokalisierung nicht gewertet wird.

Zur besseren Visualisierung wurde die Fahrt der linken Route in Abbildung 15 als Zeitstrahl dargestellt und die einzelnen Bereiche beschriftet. „M“ ist der mittlere, „L“ der linke und „R“ der rechte Streckenabschnitt. Die einzelnen Punkte markieren hierbei die Startpunkte der Geraden.

Aus diesem Grund wurden zwei Möglichkeiten erdacht, die dem Algorithmus einen Anhaltspunkt geben, wo er sich gerade in der Datenbank befindet. Bei der ersten Methode wird die Datenbank vorher bearbeitet und die mehrfach vorkommenden Bereiche speziell markiert. Beim anschließenden Matching wird dem Algorithmus mitgeteilt,

Abbildung 15: Aufteilung der linken Route in einen Zeitstrahl. Die sich wiederholenden Bereiche sind grün markiert.

wie oft er bereits den jeweiligen Abschnitt passiert hat. Allein dieser wird aus der Liste der Abschnitte für das Matching berücksichtigt. Die anderen mehrfachen Abschnitte werden nicht berücksichtigt, bis der Abschnitt komplett passiert wurde.

Abbildung 16: Zeitstrahl der Strecke mit nummerierter Beschriftung. Der Abschnitt, der nicht beachtet wird, wurde ausgegraut.

Eine weitere Methode ist die Begrenzung des Suchbereichs. Hierbei werden die Matching-Geraden nur innerhalb dieses Bereichs ermittelt und ausgewertet. Der Bereich wird dabei nach und nach innerhalb der Datenbank verschoben. Alle Datenbankbilder außerhalb der Grenzen werden für das Matching nicht beachtet.

Abbildung 17: Zeitstrahl der Strecke mit rot markiertem Suchbereich.

Für diese Arbeit wurde die zweite Methode verwendet. Dabei war ausschlaggebend, dass keine aufwendige Bearbeitung der Datenbank vor dem ersten Durchlauf nötig ist. Daher funktioniert der Algorithmus auch ohne Schwierigkeiten mit jeder beliebigen Route verwendet werden. Zudem werden Störungen beim Zählen der bereits passierten Abschnitte vermieden.

Die aktuellen Grenzen des Suchbereichs werden anhand der letzten bekannten Position des Roboters in der Datenbank festgelegt. Diese wird jedesmal, wenn ein Sequenzbild erfolgreich gematcht wird, aktualisiert. Da das Fahrzeug einer fest vorgegebenen Route folgen soll, kann die nächste Position in der Datenbank genau eingegrenzt werden. Um Abweichungen von der erwarteten Position abzufangen, wird noch ein Sicherheitsbereich hinzugefügt. Zudem werden noch Vergleichssequenzen benötigt, um die Güte des Matchings zu berechnen.

Der Suchbereich darf dabei nicht zu groß sein, da sonst die Vorteile der Eingrenzung wieder verloren gehen. Hierbei darf die Anzahl der Bilder innerhalb der Grenzen nicht

den minimalen Abstand zwischen zwei sich wiederholenden Orten in der Datenbank übersteigen.

Das Matching findet daraufhin allein in diesem Bereich statt. Ansonsten werden keine Änderungen vorgenommen. Zuerst wird das beste Ergebnis im Bereich gesucht und danach der Quotient mit dem zweitbesten Ergebnis berechnet. Liegt dieser unter einem bestimmten Schwellwert, wird die Lokalisierung als korrekt gewertet.

Neben der höheren Robustheit gegenüber langen Schleifenfahrten, werden aber auch andere Störeffekte eliminiert. So werden zum Beispiel sehr ähnliche Routenabschnitte nicht mehr miteinander verwechselt. Ein weiterer Vorteil ist die verringerte Rechenzeit. Anstatt über die gesamte Datenbank den aktuellen Ort zu suchen, ist der Abschnitt der Datenbank immer gleich klein. Somit kann gerade bei sehr großen Datenbanken von 1000 Bildern oder mehr die Laufzeit verringert werden.

4 Biologisch inspirierte Navigation

Für die autonome Navigation eines mobilen Roboters muss nach der erfolgreichen Lokalisierung eine Kursvorgabe erfolgen. Im Fall des route-followings bedeutet das eine Richtungsvorgabe um die vorgegebene Route nicht zu verlassen. Weicht das Fahrzeug trotzdem einmal von der Route ab muss der Algorithmus vorgeben in welche Richtung diese liegt, damit das Fahrzeug dorthin zurückkehren kann.

Klassische optische Navigationsalgorithmen arbeiten mit 3D-Informationen oder hoch auflösenden Kamerabildern, um die Eigenbewegung zu schätzen (visuelle Odometrie), eine Karte der Umgebung aufzubauen und sich in dieser zu lokalisieren. Aufgrund der hohen Komplexität der derart gewonnenen Daten und den sehr aufwendigen Umrechnungen, steigt die Rechenleistung auf ein hohes Pensum an. Dadurch müssen entweder leistungsfähige Computer auf dem Fahrzeug mitgeführt werden, was das Gewicht und den Energieverbrauch extrem steigert. Oder das Fahrzeug muss ständig mit einem externen Rechner, der die Navigation übernimmt, kommunizieren, wodurch die Reaktionszeit stark erhöht wird. Steht beides nicht zur Verfügung, muss bei Verwendung weniger leistungsfähiger Computer, die auf dem Fahrzeug mitgeführt werden können, die Auflösung und damit die Genauigkeit der Navigation verringert werden.

Im Lauf der Jahre haben Lebewesen zuverlässige und gleichzeitig einfache Verfahren zur Lösung dieses Problems entwickelt. Insekten wie die Ameise können beispielsweise Navigationsaufgaben sehr effizient lösen, für die moderne Algorithmen eine immens hohe Rechenleistung benötigen. Dafür benutzen sie vor allem optisch gewonnene Informationen. Diese Tatsache ist besonders beeindruckend, da die Zahl der Nervenzellen im Vergleich zu Säugetieren sehr gering ist und die Augen einer Ameise nur schlecht auflösende Bilder bereitstellen. Hierbei schwankt die Auflösung von 0 bis 10000 Bildpunkten pro Auge, wobei die meisten Arten einige Hundert Bildpunkte besitzen. Die Bilder decken dabei einen Bereich von beinahe 360° Blickwinkel ab. Zudem können die meisten Gattungen nur in zwei Farben (bichromatisch) sehen. Viele sind sogar farbenblind. Werden diese Tatsachen auf eine technische Ebene transformiert, bedeutet das eine sichere und zuverlässige Navigation mit niedrig auflösenden Grauwertbildern, bei gleichzeitig sehr geringem Speicher- und Rechenbedarf.

Diese Vorteile haben bereits viele Untersuchungen inspiriert, deren Ziel das Verstehen der Navigationsstrategien von Ameisen ist. Daraus wurden dann einige Modelle erstellt, die eine technische Umsetzung dieser Strategie ermöglichen sollen. Eines dieser Modelle von Baddeley et al. soll im nachfolgenden Kapitel genauer erläutert werden, bevor die Implementierung für den SequenceSLAM beschrieben wird.

4.1 Navigation der Ameise

Die Augen der Ameise besitzen, wie bei allen Insekten, keine Muskeln, die eine Augenbewegung ermöglichen. Dadurch muss die Ameise jedesmal, wenn sie einen anderen Bereich ihrer Umgebung sehen will, ihren Kopf oder ihren Körper drehen. Zudem besitzen sie Facetten- oder sogenannte Komplexaugen, die einen Blickwinkel von beinahe 360° ermöglichen. Die Auflösung dieser Bilder ist vergleichsweise gering.

Modelle der Ameisennavigation nehmen an, dass sich die Ameisen bereits beim ersten Lauf die Umgebungsbilder merken. Zusätzlich merken sie sich andere Merkmale, wie zum Beispiel die Himmelsrichtung über eine Art einen inneren Kompass. Jedes Mal, wenn sie anschließend die Strecke wiederholen, folgen sie diesen Merkmalen, wobei sie die bereits gespeicherten, optischen Informationen aktualisieren. Dadurch werden sie nach und nach immer sicherer, sodass sie sich bereits nach ein paar Wiederholungen beinahe nur noch auf ihre Augen zur Navigation verlassen [11].

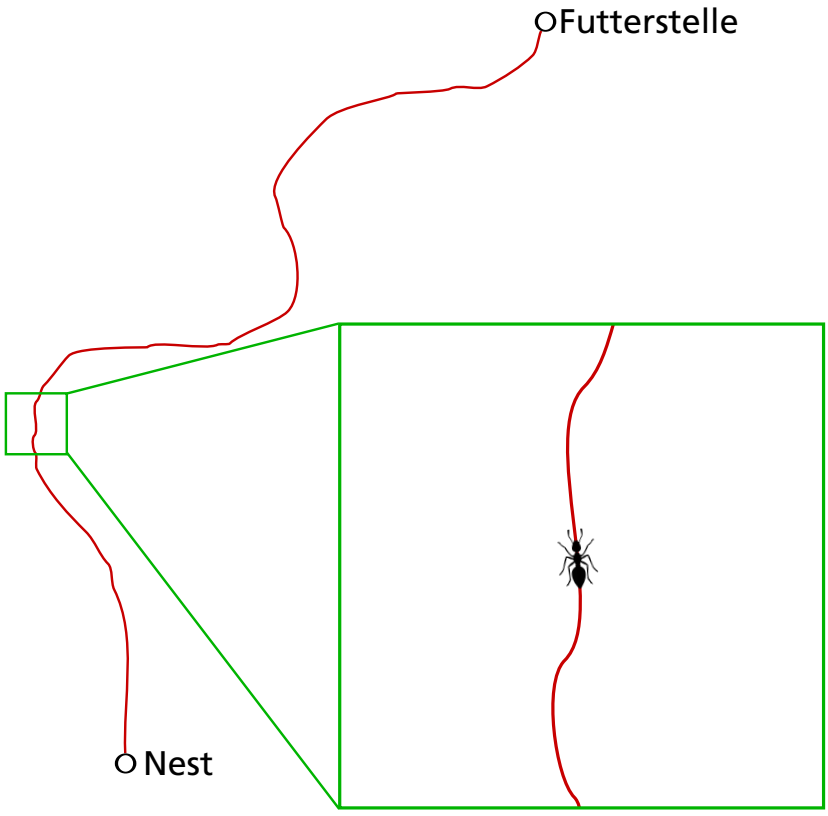


Abbildung 18: Schematische Darstellung der Wegfindung von Ameisen. Die Ameise folgt der rot markierten Route. Mit jedem Lauf wird die Navigation anhand optischer Informationen sicherer, bis sie keine anderen Informationen zur Wegfindung benötigt.

Dabei nutzen sie, dass das gespeicherte Bild der Umgebung immer in die Laufrichtung zeigt. Dies ermöglicht nach der Lokalisierung ein schnelles Weiterlaufen, indem sie sich

in die Blickrichtung weiterbewegen. Um herauszufinden, welche Richtung dabei die richtige ist, drehen sie sich ein wenig nach links und nach rechts. Anschließend entscheiden sie, in welche Richtung das Bild am ähnlichsten mit einem der gemerkten Bilder aus der „Datenbank“ war und setzen ihren Weg dorthin fort.

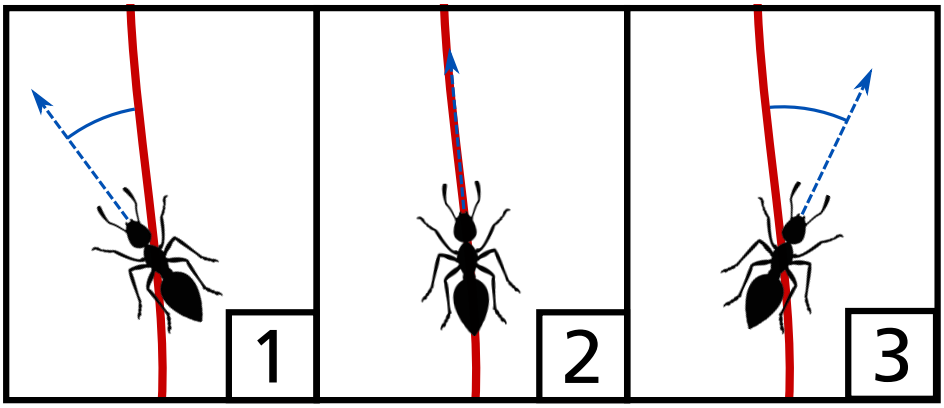


Abbildung 19: Sollte die Ameise einmal nicht wissen, in welche Richtung die Route weitergeht, dreht sie sich auf der Stelle und vergleicht die so gewonnenen Bilder mit ihrer „Datenbank“. Anschließend bewegt sie sich in die Richtung, deren Bild die höchste Ähnlichkeit mit einem der Datenbankbilder hat.

Die aufgenommene Route ist nicht bidirektional. Kennt also die Ameise den Weg vom Nest zum Futterplatz, kann sie deswegen nicht automatisch entlang der gleichen Strecke zurück laufen. Daher können die Hin- und die Rückroute durchaus unterschiedlich sein.

Der gesamte Prozess lässt sich aufgrund seiner Einfachheit und Schnelligkeit gut mit dem SequenceSLAM kombinieren. Hierbei findet die Lokalisierung mithilfe des SequenceSLAM statt und anhand der Rotation des Bildes soll anschließend ein Zurückfinden auf die Route ermöglicht werden, sollte das Fahrzeug von ihr abweichen.

4.2 Vorbereitungen für die technische Umsetzung

Für die Anwendung der Navigation der Ameise im SequenceSLAM mussten sowohl Hardware, als auch Software angepasst werden. Bei der Hardware handelte es sich vor allem um die Kamera, die zur Aufnahme verwendet werden sollte. Vorher wurde eine Kamera mit einem Öffnungswinkel von 90° in der Höhe und Breite verwendet. Diese besaß den Nachteil, dass beim Folgen der Route entweder der Roboter an sich, oder die Kamera selbst gedreht werden mussten, um die komplette Umgebung aufnehmen und vergleichen zu können. Das kostet Zeit und macht den Algorithmus undynamisch, da

die mobile Plattform an jedem Aufnahmepunkt stehen bleiben müsste, um die weitere Bewegungsrichtung zu ermitteln.

Zudem hat der Algorithmus aufgrund des Grauwertvergleichs Schwierigkeiten einen guten Match zu finden, sollte sich die Position und der Blickwinkel der Kamera zwischen den Aufnahmen der Route ändern. Eine gute Lokalisierung ist dementsprechend nur dann möglich, wenn die einzelnen Routenfahrten immer exakt gleich sind. Dies ist bei live-Applikationen kaum möglich. Daher muss eine Möglichkeit implementiert werden, um die Empfindlichkeit des Matchings zu verringern.

In der Simulation, die für das Testen des Algorithmus verwendet wurde, wurden für die Kameras eigene Koordinatensysteme definiert, bei denen die x-Achse immer in Blickrichtung liegt. Die y-Achse zeigt dabei nach links und die z-Achse nach oben. Für alle weiteren Betrachtungen der Verschiebungen und Rotationen der Kamera wurde diese Notation übernommen.

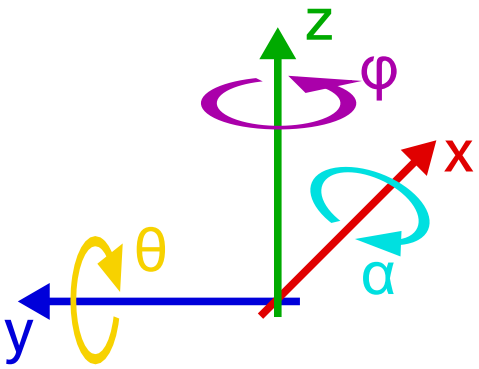


Abbildung 20: Koordinatensystem der Kamera mit den drei Achsrichtungen und den Rotationen. α ist der Rollwinkel, θ ist der Nickwinkel und ϕ ist der Gierwinkel.

Der Algorithmus ist dabei gegenüber allen drei Rotationen, also Roll-, Nick- und Gierwinkel, empfindlich. Zudem haben Verschiebungen entlang der y- und z-Achse der Kamera großen Einfluss auf das Matching. Verschiebungen entlang der x-Achse der Kamera sind differenzierter zu betrachten. Im Normalfall handelt es sich hierbei um Positionsänderungen entlang der Route, deren Einfluss maßgeblich vom metrischen Abstand zwischen den einzelnen Bildern in der Datenbank und der Größe der aufgenommenen Szenerie abhängt.

Sind die Abstände zwischen den Bildern der Datenbank im Vergleich zur Größe der aufgenommenen Landschaft sehr groß, können Verschiebungen entlang der x-Achse bei der Aufnahme des Vergleichsdatensatzes zu Problemen beim Matching führen.

Dabei werden unter Umständen komplett andere Szenerien aufgenommen, die keine Ähnlichkeit zu einem der Datenbankbilder besitzen.

Abbildung 21: 3 Aufnahmen aus dem Nordland-Datensatz mit großen Abständen. Das erste und letzte Bild sind die Datenbankbilder, das mittlere wurde dazwischen aufgenommen. Das mittlere besitzt kaum Ähnlichkeit mit dem vorherigen oder nachfolgenden Bild.

Im Gegensatz dazu sind die Änderungen der Bilder bei kleinen Abständen im Vergleich zur Größe der aufgenommenen Landschaft relativ gering. Somit haben die Verschiebungen hierbei keinen großen Einfluss mehr. Sollte sie zu groß werden, kann das aktuelle Vergleichsbild an einer neuen Stelle der Datenbank gematcht werden.

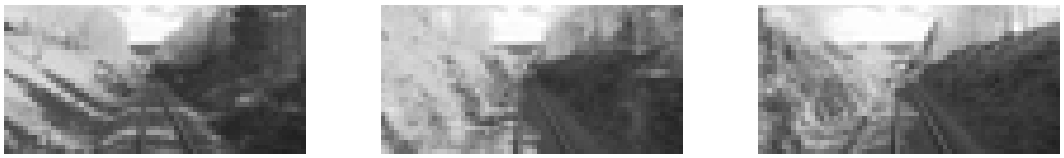


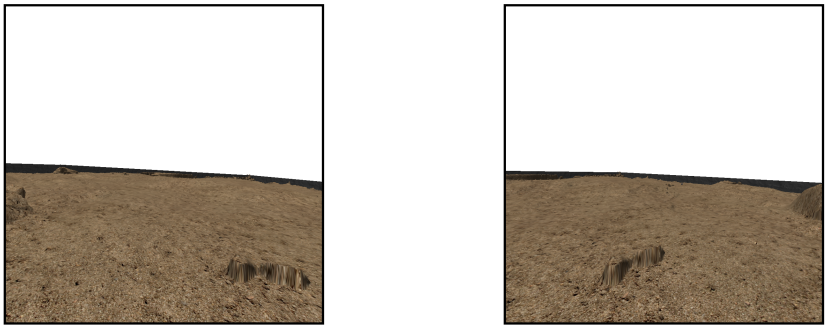
Abbildung 22: 3 Aufnahmen aus dem Nordland-Datensatz mit kleinen Abständen. Das erste und letzte Bild sind wieder die Datenbankbilder, das mittlere wurde dazwischen aufgenommen. Im Gegensatz zu vorher besitzt das mittlere Bild sowohl mit dem vorherigen als auch dem nachfolgenden Bild eine hohe Ähnlichkeit.

Für diese Arbeit wird davon ausgegangen, dass die Abstände im Vergleich zur Größe der aufgenommenen Szenerie gering sind. Daher wird keine zusätzliche Implementierung für eine Unabhängigkeit gegenüber Verschiebungen in der x-Achse benötigt.

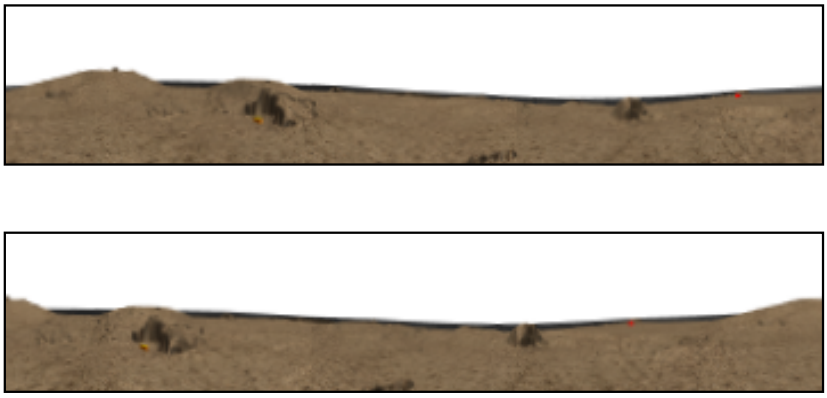
4.2.1 Installation einer Omni-Kamera

Die erste Maßnahme, um die Aufnahme robuster gegenüber Positions- und Orientierungsnänderungen zu machen, war die Verwendung einer Omnidirektionalen Kamera (Omni-Kamera). Im Gegensatz zu herkömmlichen Kameras können hiermit 360° Aufnahmen für den Algorithmus bereitgestellt werden. Der vertikale Öffnungswinkel liegt dabei bei 70°. Da die verwendete Omni-Kamera bisher nur simuliert wurde, wurde das Panoramabild aus vier einzelnen Kamerabildern, die in einem Winkel von 90° zueinander angeordnet waren, zusammengesetzt. Diese werden anschließend in ein Panora-

mabild (Rektangulärprojektion) umgerechnet. Die so gewonnenen Bilder werden dann wieder für den SequenceSLAM in einem Bildvektor gespeichert.



(a) Bild der Frontkamera bei Verdrehungen



(b) Omniansicht bei Verdrehungen des Fahrzeugs

Abbildung 23: In (a) ist das Bild einer einzelnen Kamera mit einem vertikalen Öfnnungswinkel von 95° bei einer Drehung des Fahrzeugs um ca. 20° nach rechts dargestellt. Darunter ist zum Vergleich das Panoramabild aus der gleichen Position und bei der gleichen Verdrehung abgebildet. Trotz der Eintönigkeit der Landschaft kann anhand des Felsens in (a) gezeigt werden, wie stark sich die abgebildete Landschaft bei Verdrehungen ändert. Die Bilder stammen aus einer am DLR entwickelten Simulationsumgebung.

Da der gesamte Bereich um die Kamera herum aufgenommen wird, wird das Bild komplett unabhängig gegenüber Drehungen um die z-Achse der Kamera. Wenn die z-Achse des Fahrzeugs und der Kamera gleich sind, sind die Bilder zudem gegen Fahrzeugrotationen auf der Stelle unabhängig. Das bedeutet, dass das Verwenden einer Omni-Kamera die Empfindlichkeit gegenüber Gier-Rotationen komplett entfernt.

Desweiteren wird durch die Omni-Kamera die Abhängigkeit gegenüber Verschiebungen in der y-Achse verringert. Schon bei sehr kleinen Verschiebungen passen die Positionen der Pixel nicht mehr zueinander, wodurch komplett gleiche Szenerien nicht mehr gematcht werden können.

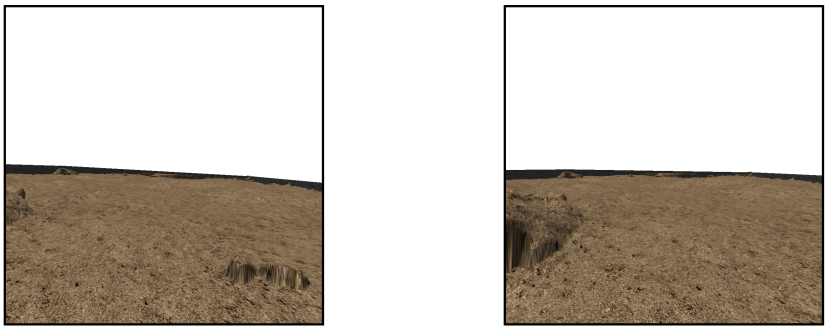


Abbildung 24: Ansicht einer normalen Kamera mit einem Öffnungswinkel von 95°, bei einer Verschiebung nach links. Der kleine Felsen, der vorher rechts unten lag, ist komplett aus dem Bild verschwunden, während links ein Felsen hinzugekommen ist.

Dieser Effekt tritt bei einer Omni-Kamera zwar ebenfalls auf, aber nur in der Richtung orthogonal zur Bewegungsrichtung. Hier findet die größte Änderung statt, auch schon bei kleinen Verschiebungen. In der Bewegungsrichtung ändert sich das Bild jedoch kaum. Hier greift der gleiche Effekt, der vorher bei der x-Achsenverschiebung auftrat, der besagt, dass die Bildänderungen bei Verschiebungen in Blickrichtungen nur gering sind.

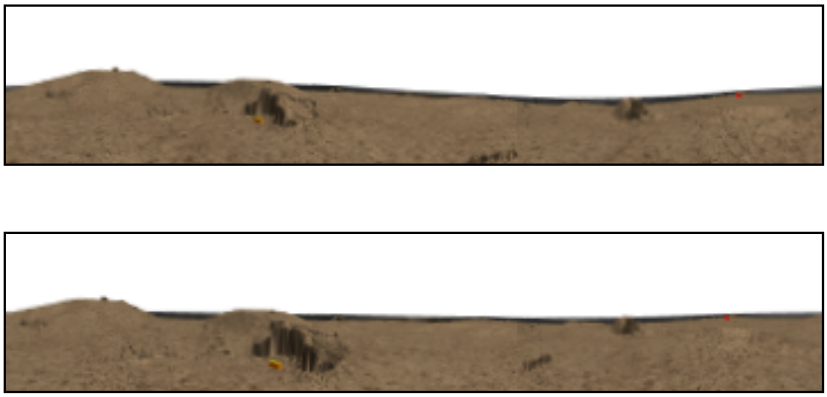


Abbildung 25: Ansicht einer Omni-Kamera an den gleichen Positionen, wie vorher die Einzelkameras. Die Bilder sind gerade am linken und rechten Bildrand noch sehr ähnlich, obwohl die gleiche Verschiebung wie beim vorigen Bild vorliegt.

Allerdings kann durch eine Omni-Kamera nicht die Verschiebung entlang der z-Achse der Kamera ausgeglichen werden. Auch der Roll- und Nickwinkel haben noch einen großen Einfluss auf das Matching-Ergebnis. Bei Bodenfahrzeugen, die in Gebäuden unterwegs sind, sind diese Parameter irrelevant, da Wertänderungen in diesen Achsen in einem sehr geringen Maß auftreten. Für andere Anwendungen kann noch ein sogenannter gimbal verwendet werden, der die Kamera immer entlang der xy-Ebene ausrichtet. Dadurch hätten der Roll- und Nickwinkel des Fahrzeugs ebenfalls keinen Ein-

fluss mehr auf die Vergleichsergebnisse. Allein die Verschiebung entlang der z-Achse kann nicht hardwaretechnisch kompensiert werden. Da für den SequenceSLAM bisher nur Boden-Fahrzeuge verwendet wurden, kann dieser Aspekt zunächst vernachlässigt werden.

4.3 Einbindung der Navigation in den SequenceSLAM

Die Navigation beim verbesserten SequenceSLAM besteht vor allem aus zwei Punkten:

- Aktives Folgen einer Route
- Korrektur der Bewegungsrichtung bei Abweichungen von der Route

Für den ersten der beiden Punkte wird für jedes einzelne Bild aus der Datenbank die Odometrie aufgenommen und gespeichert. Daraus lässt sich dann für jeden Ort entlang der Route die relative Entfernung zum nächsten, anzufahrenden Punkt bestimmen. Nach der erfolgreichen Lokalisierung kann die entsprechende Bewegungsrichtung und Entfernung ausgegeben werden und das Fahrzeug bewegt sich an die nächste Position in der Route.

Hierbei geht dieser Prozess davon aus, dass während der Fahrt keinerlei Fehler auftreten. In der Realität können sich hingegen selbst geringste Abweichungen nach und nach aufsummieren , sodass schließlich keine korrekte Lokalisierung mehr möglich ist, oder das Fahrzeug von der Route abweicht.

Aus diesem Grund musste ein Verfahren zur Bestimmung und Korrektur der aktuellen Position des Fahrzeugs implementiert werden. An dieser Stelle setzt die Navigation nach dem Vorbild der Ameisen ein. Diese wurde kombiniert mit den Odometrieinformationen oder alleinstehend eingesetzt. Bei der Kombination wurde jedoch nur die Richtung zum nächsten Punkt in der Datenbank ausgegeben, da die Ameisen-inspirierte Navigation später nur diese Information benötigt.

4.3.1 Vorgehensweise

Um die Navigation des Fahrzeugs so einfach wie möglich zu halten, wurde die Bewegung aus zwei Komponenten zusammengesetzt: Eine Geschwindigkeit v entlang der x-Achse des Fahrzeugs und eine Winkelgeschwindigkeit ω um die z-Achse. Bei dem

Fahrzeug wurde ein omnidirektionaler Rover gewählt. Dieser ist für die zu bewältigende Aufgabe gut geeignet, da er die Bewegungsmuster der Ameise nachvollziehen kann.

Als nächstes wurde festgelegt, dass die Blickrichtung der Kamera immer der Bewegungsrichtung des Fahrzeugs entspricht. Wie bei dem Vorbild aus der Natur auch, kann somit anhand des Bildmatchings direkt ermittelt werden, in welche Richtung die Route fortgesetzt werden muss.

Anhand dieser Bedingungen wurde die Vorgehensweise für die an die Panoramabilder angepasste Lokalisierung festgelegt. Die einzelnen Schritte sind in Abbildung 26 zur besseren Visualisierung in einem Flussdiagramm dargestellt. Dieser Programmabschnitt wird für jedes neu aufgenommene Datensatzbild anstatt der Berechnung der absoluten Summe der Grauwertdifferenzen durchgeführt. Anschließend folgen unverändert die weiteren Lokalisierungsschritte, also die Kontrastnormalisierung des Differenzenvektors und die anschließende Suche der besten Sequenz.

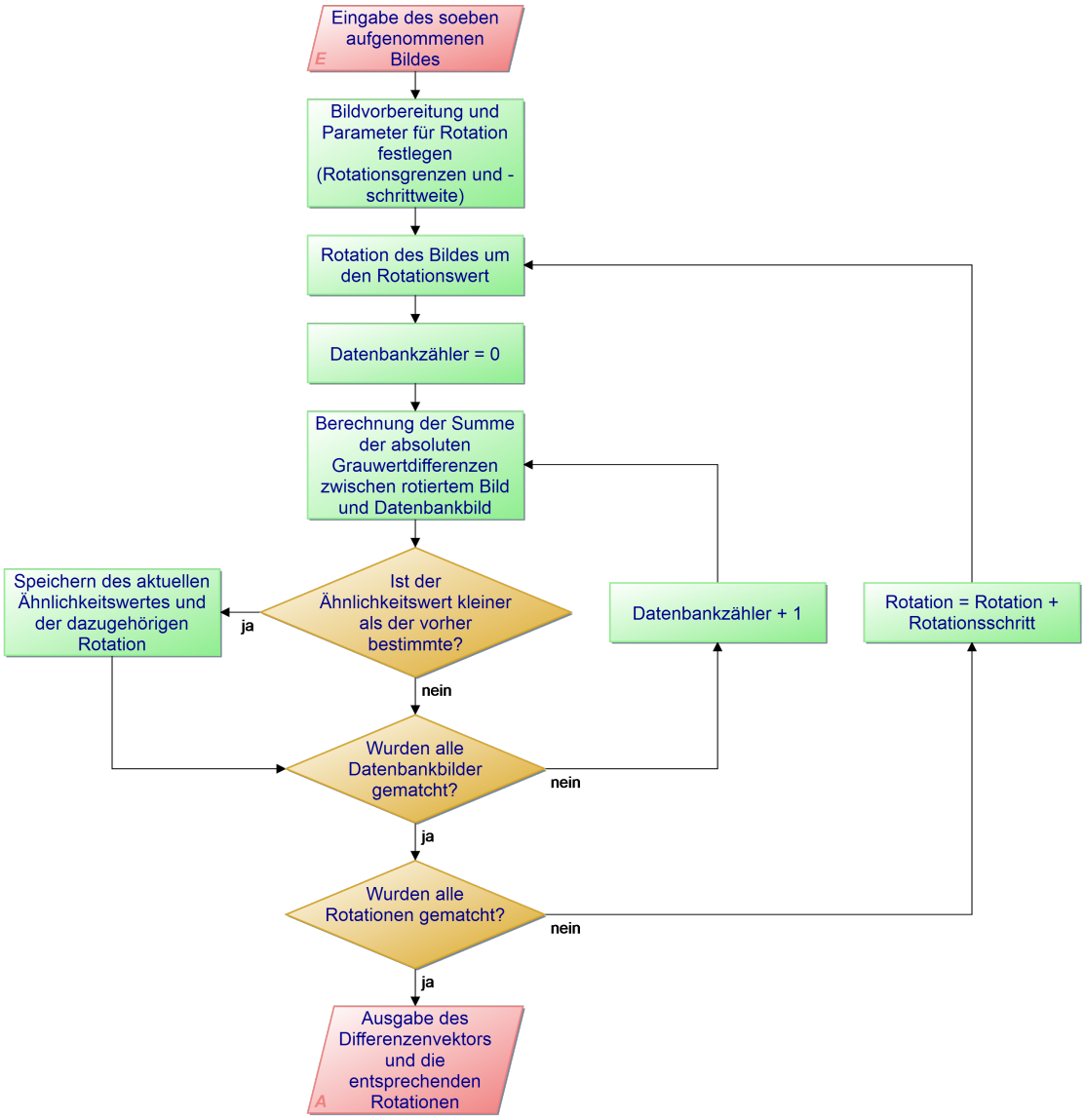


Abbildung 26: Flussdiagramm für die Navigation des Fahrzeugs.

4.3.2 Bildvorbereitung

Der Punkt „Bildvorbereitung" beinhaltet dabei die Präprozessierung des Bildes. Dazu gehören alle in Unterabschnitt 2.1.2 beschriebenen Schritte, sowie das Einschwärzen des Himmels, sofern es sich um ein bei Tag aufgenommenes Bild handelt. Anschließend wird das Bild im Datensatzvektor an der aktuellen Position gespeichert.

Zudem werden hier die Rotationsparameter für die Navigation festgelegt. Diese bestehen aus einer oberen und einer unteren Winkelgrenze. Diese markieren den maximalen Rotationsbereich, in dem die Richtung zum nächsten Routenpunkt gesucht wird. Wie bei dem Ameisen-inspirierten Modell wird auch hier ein rotiertes Bild von

der Umgebung gematcht und die Richtung, in der die höchste Ähnlichkeit auftritt, als weitere Bewegungsrichtung gewertet (s. Unterabschnitt 4.3.3). Zudem muss eine Winkelschrittweite bestimmt werden, die die Auflösung der Rotation festlegt.

4.3.3 Matchen des Panoramabildes

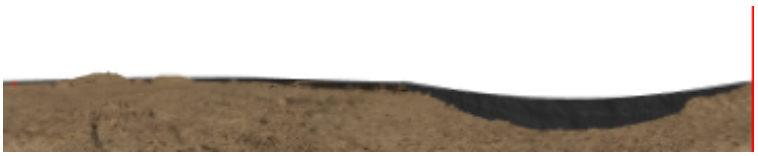
Für die Bestimmung der Ähnlichkeit der aktuellen Szenerie mit den Orten aus der Datenbank wurde wieder die Summe der absoluten Grauwertdifferenzen verwendet. Allerdings wurde auch mitberücksichtigt, dass das Fahrzeug im Vergleich zur echten Aufnahmeposition verdreht oder verschoben sein kann.

Um die aktuelle Position des Fahrzeugs zu ermitteln, gibt es mehrere Möglichkeiten. Zum Einen könnte das Fahrzeug rotiert und in kleinen Abständen verfahren werden. Dabei werden ständig Bilder von der Umgebung aufgenommen und mit den Datenbankbildern verglichen. Sobald das Lokalisierungsergebnis wieder einen ausreichend guten Wert aufweist, kann davon ausgegangen werden, dass sich das Fahrzeug wieder auf der Route befindet. Zudem sorgt die achsfeste Positionierung der Kamera dafür, dass die weitere Bewegungsrichtung und damit die nächste Position in der Route bekannt sind. Dieser Prozess ist jedoch sehr zeitaufwendig und unterbricht den Folgeprozess.

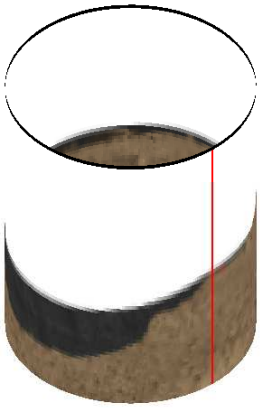
An dieser Stelle kommen die Vorteile der Omni-Kamera zum Tragen. Da die gesamte Szenerie um das Fahrzeug herum aufgenommen wurde, kann die Drehung des Fahrzeugs simuliert werden. Dies setzt voraus, dass die z-Achse der Kamera und die Drehachse des Fahrzeugs gleich sind. Für die weitere Betrachtung wird davon ausgegangen, dass die Kamera gegenüber dem Roll- und Nickwinkel fest angebracht ist. Das bedeutet, dass nur die Drehungen um die z-Achse des Fahrzeugs Auswirkungen auf das Kamerabild haben, wie das zum Beispiel bei der Verwendung eines gimbals der Fall ist.

Um die Rotation des Fahrzeugs um die eigene z-Achse zu simulieren, wird das Panoramabild nicht mehr als 2D-Bild betrachtet. Stattdessen wird es gedanklich auf eine Zylinderoberfläche projiziert, wobei der linke und der rechte Rand aneinander angeschlossen werden. Die Linie, die von den beiden Rändern gebildet wird, wird im Folgenden als Schnittlinie des Bildes betrachtet.

In Abbildung 27 ist deutlich zu erkennen, was mit der Rotation des Bildes gemeint ist. Hierfür wird der Zylinder um seine z-Achse gedreht, während die Schnittlinie ihre Position beibehält. Diese markiert hierbei eine Drehung um genau 180° zur frontalen



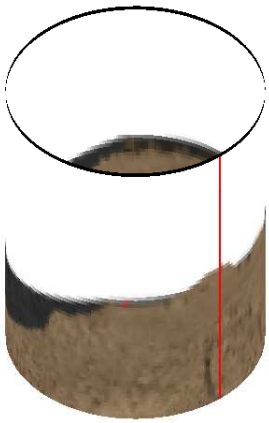
(a) 2D-Bild aus der Simulation



(b) Bild auf den Zylinder projiziert

Abbildung 27: Das Bild links wird auf den rechten Zylinder projiziert. Am rechten Rand wurde eine rote Schnittlinie eingefügt, um den Rand des Bildes auch beim Zylinder darstellen zu können. Das Bild stammt aus einer am DLR entwickelten Simulationsumgebung (s. Unterabschnitt 5.2.1)

Blickrichtung der Kamera. Das anschließend wieder auseinandergeschnittene Bild zeigt, dass die Rotation des Bildes einer Pixelumverteilung entspricht. Hierfür wird ein Pixelbereich vom einen Rand entfernt und an den anderen Rand in der gleichen Reihenfolge wieder angefügt.



(a) gedrehter Bild-Zylinder

(b) Aufgeschnittenes 2D-Bild

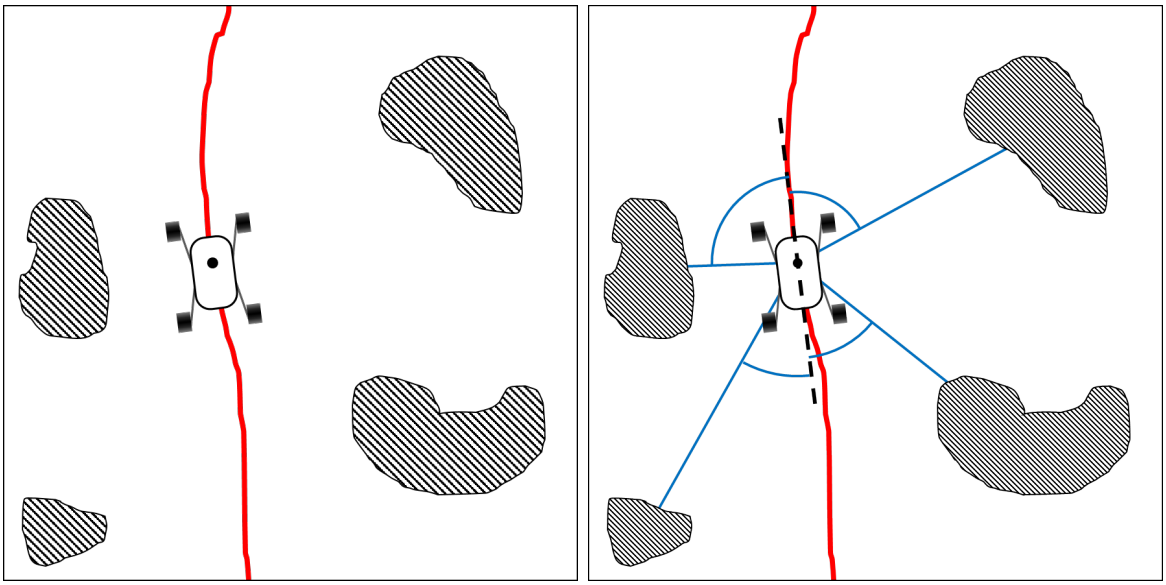
Abbildung 28: Der Zylinder von vorher wurde gedreht, während die Schnittlinie sich nicht bewegt hat. Das daraus entstehende 2D-Bild ist in Abbildung 28b dargestellt.

Das Verschieben der Pixel lässt sich wiederum in mathematischen Formeln beschreiben. Dafür wird mithilfe der Bildweite W nach der Vorverarbeitung die Anzahl der Pixel pro

Grad bestimmt. Damit kann die Bildrotation i_{rot} in Pixel berechnet werden.

$$i_{rot} = \frac{W}{360°} * s_{rot} \tag{17}$$

s_{rot} ist dabei die vorher festgelegte Schrittweite in Grad. Anhand dieser Transformation kann ein zeitaufwendiges Drehen des Fahrzeugs effizient ersetzt werden. Allerdings tritt eine reine Verdrehung ohne eine Verschiebung nur selten auf. Damit auch das Abweichen von der Route gleichzeitig schnell und sicher erkannt werden kann, wurden weitere Überlegungen angestellt, wie mit der simulierten Bildrotation die Abweichung erkannt und korrigiert werden konnte.



(a) Rover in Landschaft (Schema) *(b) Verbindungslinien zu den Landmarken*

Abbildung 29: Schematische Darstellung der Positionen der Landmarken zum Rover. Die Verbindungslinien zeigen die Winkel relativ zur Blickachse des Rovers an.

Hierfür wurde die Umgebung zunächst als eine Ansammlung von Landmarken aus der Vogelperspektive betrachtet. Das Fahrzeug fährt durch diese Ansammlung und nimmt alle Marken mit der Omni-Kamera auf. Jedes Objekt hat dabei in jedem Bild eine fest zugewiesene Position. Betrachtet man die Blickachse des Fahrzeugs als Nullposition, können die einzelnen Landmarken in der Landschaft mithilfe von Winkeln markiert werden.

Wird das Fahrzeug von seiner ursprünglichen Position verschoben, ändern sich die Winkel der Landmarken und damit deren Position in den Bildern. Wie sehr sich diese Winkel ändern, hängt wiederum von der Position und der Entfernung der Landmarke ab. Liegt das markante Objekt genau auf der Verschiebungsachse, so ändert sich der Winkel

nicht. Zudem werden die Winkeländerungen und damit die Positionsänderungen im Bild immer kleiner, je weiter das Objekt entfernt ist.

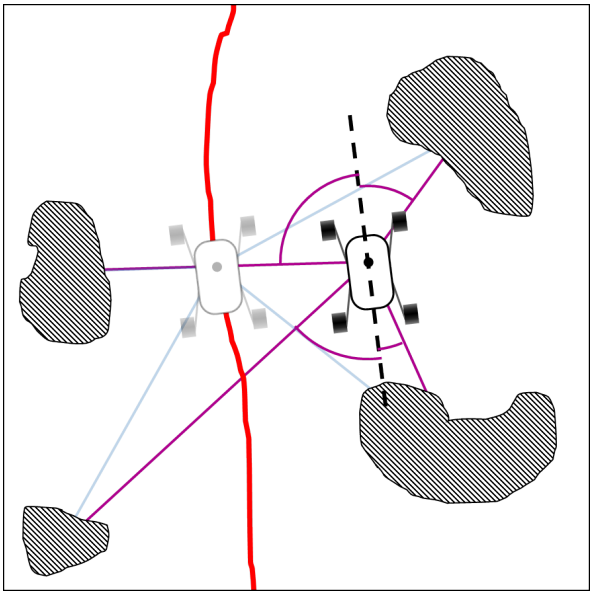


Abbildung 30: Der Rover wurde verschoben, sodass er sich neben der Route befindet. Dadurch ändern sich die Positionen der Landmarken im Bild (dargestellt durch die neuen, violetten Linien) im Vergleich zu den alten Positionen.

Anhand Abbildung 30 kann eine Gesetzmäßigkeit für die Veränderung der Position in den Bildern erkannt werden. Bei einer y-Verschiebung verschieben sich auch die Landmarken im Bild in die gleiche Richtung (anhand der Winkeländerungen im Bild zu erkennen). Das selbe Prinzip gilt spiegelverkehrt auch für die Landmarken hinter dem Fahrzeug. Dies wird im Folgenden genutzt, um die Verschiebung des Rovers ausgleichen zu können.

Dabei wurde angenommen, dass durch die simulierte Fahrzeugdrehung das aktuelle Bild wieder lokalisiert werden kann. Hierfür wurden zwei Herangehensweisen als möglich erachtet. Bei der ersten wurde das komplette Panoramabild um einen bestimmten Winkel gedreht. Anschließend wird die Ähnlichkeit des gedrehten Bildes mit dem Datensatzvektor ermittelt. Die Rotation, bei der die größte Ähnlichkeit auftritt, wird anschließend, wie bei einer Verdrehung des Fahrzeugs, gespeichert. Diese Methode wäre zwar schnell und einfach zu implementieren. Sie beachtet allerdings nicht das gegensätzliche Verhalten von Bildern vor und hinter der Kamera.

Um dies mit zu berücksichtigen, wird das Bild in einen vorderen und einen hinteren Bereich unterteilt. Die Bereiche umfassen jeweils 160° und sind symmetrisch um die Blickachse der Kamera gelegt. Anschließend werden die Datenbankbilder ebenfalls wieder in vordere und hintere Bereiche unterteilt.

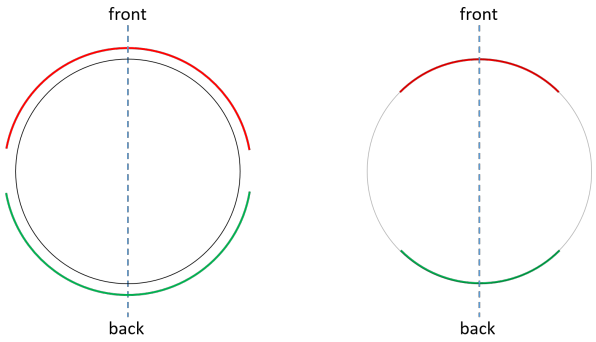


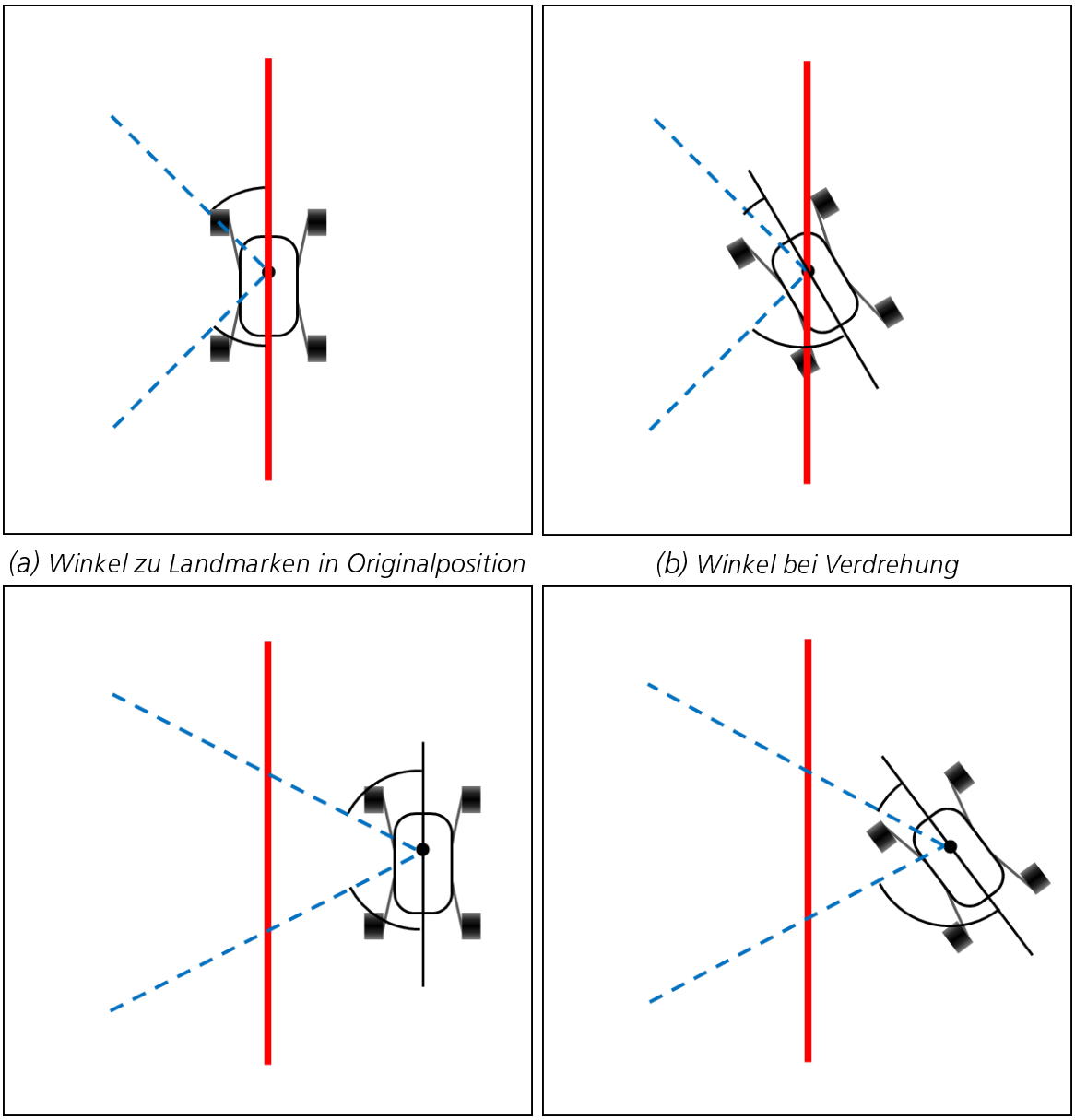
Abbildung 31: Die nach vorne und hinten getrennten Bildbereiche der Omni-Kam. Links das Datensatzbild, rechts das ausgeschnittene Datenbankbild. Die blau gestrichelte Linie markiert die Blickachse der Kamera.

Die aus der Datenbank extrahierten Bildbereiche werden anschließend nacheinander über das aktuelle Szeneriebild geschoben. An jeder Stelle wird die Ähnlichkeit ermittelt und der größte Wert zusammen mit seinem Rotationsbetrag gespeichert. Hierbei können vorne und hinten verschiedene Winkel auftreten. Die drei Varianten, in denen sich Winkel ändern können, sind in Abbildung 32 b-d aufgelistet.

In Abbildung 32a sind zum Vergleich die Originalwinkel zwischen Rover und zwei gedachten Landmarken angezeigt. Diese dienen als Referenzwinkel für die weiteren Betrachtungen der Einzelfälle. Bei einer reinen Verdrehung, ohne eine Verschiebung von der Route, ändern sich beide Winkel zu den Landmarken um den gleichen Betrag und in die gleiche Richtung. Das bedeutet, dass der Winkel, um den das Fahrzeug gedreht wurde, von den vorherigen Landmarkenwinkeln (in Bezug auf die Blickachse) abgezogen wird. Um wieder das Originalbild zu erhalten, wird nur eine Drehung des gesamten Bildes um einen bestimmten Winkel benötigt. Bei diesem handelt es sich um genau den Betrag, um den das Fahrzeug zur Route verdreht ist.

Der zweite Fall ist eine reine Verschiebung weg von der Route, wobei die Verschiebung entlang der Route aufgrund der Bilddichte nicht betrachtet werden. Hierbei ändern sich die Winkel für Landmarken vor und hinter der Kamera in die entgegengesetzte Richtung. Bei Verschiebungen nach rechts von der Route ändern sich die Winkel im vorderen Teil des Kamerabildes gegen den Uhrzeigersinn, während sie sich im hinteren Teil mit dem Uhrzeigersinn ändern. Bei Verschiebungen nach links von der Route gilt dasselbe umgekehrt. Somit kann bei den separat gedrehten vorderen und hinteren Bildern eine Verschiebung anhand der Vorzeichen der vorne und hinten ermittelten Positionsänderungen bestimmt werden.

Das letzte Beispiel ist die Kombination aus Verschiebung und Verdrehung, wobei die vorherigen Auswirkungen auf das Bild ebenfalls kombiniert auftreten können. Es finden sowohl die Winkeländerungen für die Verschiebung, als auch eine Verdrehung der



(a) Winkel zu Landmarken in Originalposition

(b) Winkel bei Verdrehung

(c) Winkel bei Verschiebung

(d) Winkel bei Verschiebung & Verdrehung

Abbildung 32: Schematische Darstellung der Positionen der Landmarken zum Rover bei verschiedenen Roverpositionen zur Route. Die Verbindungslinien zeigen die gedachten Positionen von Landmarken an.

Kamera statt, die alle Landmarken im Bild um den gleichen Betrag und in die gleiche Richtung verschiebt. Der Winkel, der durch die simulierte Rotation des Bildes ermittelt wird, kann anschließend zur Korrektur des Kurses verwendet werden.

4.3.4 Kurskorrektur bei Abweichungen

Die vorher durchgeführte Navigation ermittelt die Abweichung von der eigentlichen Route in Winkeln. Daher wurde die Korrektur ebenfalls über die Winkelgeschwindigkeit

vorgegeben. Hierfür wird zunächst eine Zielrotation berechnet, die bis zum nächsten Aufnahmepunkt erreicht werden soll. Dafür wird zunächst überprüft, ob die Vorzeichen der beiden ermittelten Winkel gleich sind. Ist dies der Fall, so wird der weitere Fahrtwinkel nach folgender Formel berechnet:

$$\alpha_{ziel} = \frac{\alpha_1 + \alpha_2}{2} \tag{18}$$

Hierbei wird angenommen, dass bei gleichen Vorzeichen der Winkel eine Verdrehung vorliegt. Bei ungleichen Vorzeichen liegt eine Verschiebung vor, weshalb dort dann folgende Formel angewendet werden muss:

$$\alpha_{ziel} = \frac{\alpha_1 - \alpha_2}{2} \tag{19}$$

Hierbei bezeichnet α_1 den vorderen Winkel und α_2 den hinteren. Der so erhaltene Winkel wird anschließend als Richtungsangabe zum nächsten Punkt in der Datenbank verwendet. Damit die Fahrt nicht unterbrochen werden muss, um das Fahrzeug in die richtige Richtung zu drehen, wird die neue Richtung während der Fahrt mithilfe einer Winkelgeschwindigkeit ω um die z-Achse eingebunden. Zudem wird sie mit der x-Geschwindigkeit v des Fahrzeugs verrechnet, um eine ruhige Kurvenfahrt zu ermöglichen.

Für die Verrechnung werden die beiden Größen zunächst auf einen Wertbereich von -1 bis 1 normiert, wobei die maximal und minimal mögliche Geschwindigkeit als Normierungsgrenzen dienen. Anschließend werden sie in einem Einheitskreis aufgetragen, in dem die normierte Geschwindigkeit die reele Achse und die normierte Winkelgeschwindigkeit die imaginäre Achse bilden. Dieses Modell wurde bereits bei der Aufnahme der Route in der Simulation verwendet und basiert auf der Bedienung des Fahrzeugs mit einem Joystick.

Die resultierende Geschwindigkeit liegt dadurch immer auf einem Einheitskreis und besteht aus den beiden Geschwindigkeitskomponenten. Anhand der Geometrie und dem Satz des Pythagoras ergibt sich folgender Zusammenhang zwischen den beiden Geschwindigkeiten:

$$v_{\text{res}} = \sqrt{v^2 + \omega^2} = 1 \tag{20}$$

Mithilfe der maximalen Winkelgeschwindigkeit lässt sich zudem die größtmögliche Rotation zwischen zwei Zeitschritten berechnen.

$$\alpha_{\text{max}} = \omega_{\text{max}} * t_{\text{Step}} \tag{21}$$

Dieser Winkel wird im Folgenden „max angle“ genannt. Er bezeichnet die innerhalb eines Zeitschritts maximal mögliche Rotation. Die Winkelgeschwindigkeit wird mithil-

fe des max angle's und der vorher erhaltenen Rotation zum nächsten Punkt anhand folgender Formel berechnet.

$$\omega = \frac{\alpha}{\alpha_{max}} * \omega_{max} \tag{22}$$

v wird anschließend abhängig von der so erhaltenen Winkelgeschwindigkeit errechnet, damit der Radius der Kurve dementsprechend angepasst werden.

Ist die vorher erhaltene Rotation größer als der max angle, wird immer die maximale Winkelgeschwindigkeit ausgegeben, während v den Wert 0 zugewiesen bekommt. Dadurch dreht sich das Fahrzeug auf der Stelle. Aufgrund des Panoramabildes und den annähernd gleich positionierten Drehachsen des Fahrzeugs und der Kamera, wird dadurch auch das Bild gedreht. Das bedeutet, dass die gleiche Ähnlichkeit wie zuvor ermittelt wird. Allerdings sollte, zumindest bei ebenem Untergrund, der ermittelte Korrekturwinkel nun deutlich kleiner sein als vorher, wodurch eine weitere Vorwärtsbewegung des Fahrzeugs ermöglicht werden kann.

Durch das Drehen auf der Stelle wird zudem auch ein Bild in die Sequenz eingefügt, dass vorher nicht dort war. Dadurch können später Fehler beim Matching entstehen, da nun die diagonale Sequenz in der Differenzenmatrix unterbrochen wird. Um dies zu verhindern, wird das alte Bild von der neuen Aufnahme überschrieben. Die nicht benötigten Mehraufnahmen werden somit aus dem Vergleichsvektor entfernt.

5 Versuchsaufbau und -durchführung

Für den SequenceSLAM wurden zwei Versuchsszenarien aufgebaut, um die verschiedenen Bereiche genau zu testen. Beim ersten wurde nur getestet, inwiefern die reine Lokalisierung im Vergleich zum originalen SequenceSLAM verbessert wurde. Hierfür wurden verschiedene Datensätze verwendet, bei denen die korrekte Lokalisierung bereits bekannt war. Das zweite Szenario fand in einer Simulationsumgebung statt und sollte die Navigierfähigkeit des Algorithmus testen. Der Aufbau dieser beiden Versuchsumgebungen soll im Folgenden kurz beschrieben werden, bevor die erhaltenen Ergebnisse dokumentiert und evaluiert werden.

5.1 Lokalisierungsszenario

In diesem Szenario wurden noch keine Live-Bilder mit vorher erstellten Datenbänken gematcht. Stattdessen wurden zwei bereits aufgenommene Datensätze verwendet.

5.1.1 Nordland-Datensatz

Die erste Versuchsstrecke war der sogenannte Nordland-Datensatz. Dabei handelt es sich um die Aufnahme einer 728 km langen Bahnstrecke aus der Perspektive des Zugfahrers. Sie wurde insgesamt viermal aufgenommen, einmal zu jeder Jahreszeit. Die Aufnahme wurde im Rahmen der TV-Dokumentation „Norlandsbanen - Minutt for Minutt“ vom norwegischen Fernsehsender NRK aufgenommen und ist online[3] als „Common Creative license“ erhältlich.

Sünderhauf et al. verwendeten bereits diesen Datensatz, um die Robustheit des Algorithmus gegenüber jahreszeitlichen Schwankungen zu bewerten. Dafür hat er einen halbstündigen Ausschnitt aus jeder der vier Aufnahmen ausgeschnitten und mithilfe von ebenso verfügbaren GPS-Daten synchronisiert. Auf diese Weise erhielt er vier Datenbanken, die an jedem Index den genau gleichen Ort zu unterschiedlichen Jahreszeiten zeigte[4].

Da die Datensätze zu verschiedenen Jahreszeiten aufgenommen wurden, sind diese dafür geeignet, die Robustheit gegenüber Änderungen im Erscheinungsbild eines Ortes aufzuzeigen. So sind die Bilder im Winter aufgrund des Schnees deutlich heller, als zu den anderen Jahreszeiten. Allgemein sind die gleichen Orte in den vier Datenbanken teilweise so unterschiedlich, dass ein reiner Grauwertvergleich beinahe keine Ähnlichkeit mehr ermitteln kann.

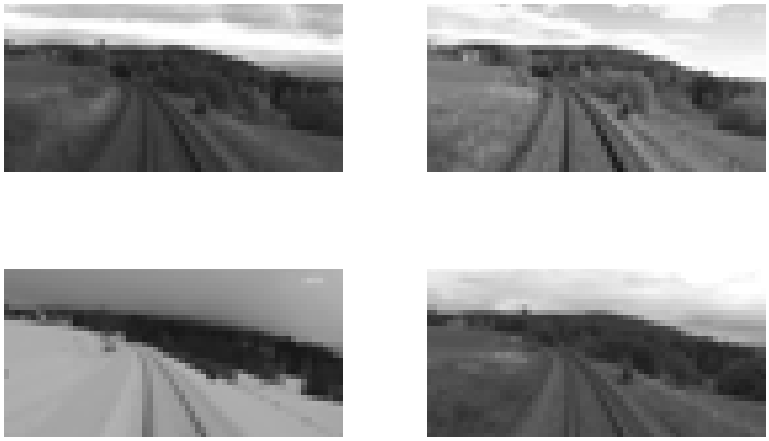


Abbildung 33: Einige Beispielbilder aus dem Nordland-Datensatz. Jedes Bild zeigt den gleichen Ort zu unterschiedlichen Jahreszeiten: Oben links im Herbst, Oben rechts im Frühling, unten links im Winter und unten rechts im Sommer.

[3] http://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/
[4] https://svn.openslam.org/data/svn/openseqslam/trunk/datasets/

Für die Versuche wurden jeweils 350 Bilder aus den vier Routen „Sommer“, „Winter“, „Herbst“, und „Frühling“ verwendet. Der Abstand zwischen den verwendeten Bildern der einzelnen Datensätze lag bei 100 Datenbankbildern. Da eine Bahnfahrt aufgenommen wurde, konnten verschiedene Phänomene, wie Tunnelfahrten oder Stopps an Haltestationen während der Fahrt beobachtet werden. Haltestellen führen dabei zu Mehrfachaufnahmen, die wiederum die Lokalisierung anhand einer Sequenz erschweren können. Tunnelfahrten sind deswegen problematisch, da die dort aufgenommenen Bilder komplett schwarz sind. Somit wird es schwer einen Tunnelabschnitt richtig zu lokalisieren, sobald mehrere davon an verschiedenen Stellen der Route aufgenommen wurden.

5.1.2 Milford-Datensatz

Bei der zweiten Versuchsstrecke handelte es sich um den Datensatz, der bereits von Milford und Wyeth in ihrer originalen Veröffentlichung verwendet wurde. Hierfür wurde eine Route in Brisbane, Australien zweimal aufgenommen. Die erste Fahrt fand in der Nacht und bei starkem Regen statt, die zweite hingegen am Morgen. Die Aufnahmen wurden in einem Video kombiniert, sodass beide Fahrten nacheinander abliefen und für weitere Tests online zur Verfügung gestellt[5].

Aufgrund der verschiedenen Tageszeiten und Wetterbedingungen, sind selbst mit menschlichem Auge beinahe keine Übereinstimmungen zwischen den einzelnen Orten zu erkennen. Trotzdem war der SequenceSLAM laut Milford et al. in der Lage, ungefähr 30% der Orte korrekt zu lokalisieren, wobei keine falschen Matches gefunden wurden.

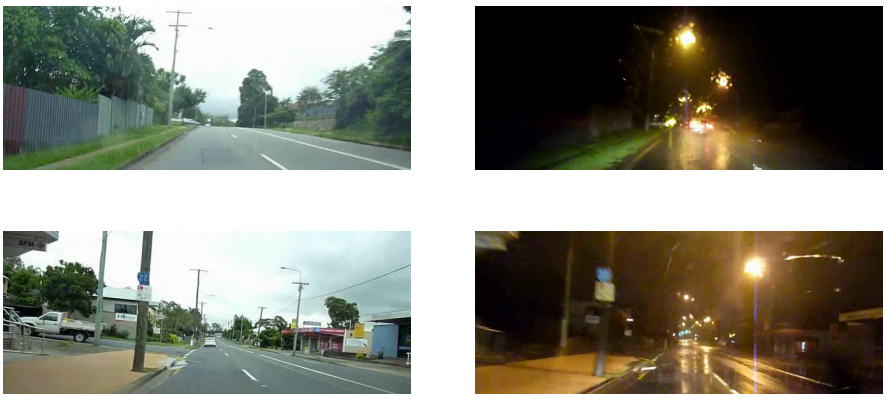


Abbildung 34: Zwei korrespondierende Orte, jeweils am Tag und in der Nacht aufgenommen.

[5] https://wiki.qut.edu.au/display/cyphy/Michael+Milford+Datasets+and+Downloads

Um aus dem Video eine Datenbank für diese Arbeit zu generieren, wurde zuerst die Nachtfahrt in einzelne Frames aufgeteilt. Hierfür wurden die ersten 10 Minuten des online erhältlichen Videos mit einer Frequenz von 0,5 Hz (alle zwei Sekunden ein Bild) in einzelne Bilder unterteilt. Anschließend wurden Stellen, an denen das Fahrzeug lange Zeit stillstand, wie Kreuzungen, oder enge Kurven, herausgesucht und die Mehrfachaufnahmen an diesen Stellen aus der Datenbank herausgeschnitten. Daraufhin wurde das am Tag aufgenommene Video in seine einzelnen Frames aufgeteilt, wobei hier jedes Frame verwendet wurde. Diese wurden dann nach den entsprechenden Aufnahmen der Orte aus dem ersten Datensatz durchsucht. Wurde ein Bild des gleichen Ortes gefunden, wurde es separat gespeichert. Nachdem allen Nachtaufnahmen genau eine Tagaufnahme zugeordnet werden konnte, wurden die übrig gebliebenen Frames wieder gelöscht. Auf diese Weise konnten die genauen Positionsdaten zwischen den einzelnen Aufnahmen synchronisiert und später überprüft werden.

Daraus ergab sich eine Datenbank mit 230 Bildern und ein ebenso großer Vergleichsdatensatz. Aufgrund der extrem großen Unterschiede zwischen den Aufnahmen, konnte hiermit gezeigt werden, wie robust der Algorithmus gegenüber Helligkeitsschwankungen ist.

5.2 Navigationsszenario

Die Navigationsfähigkeit konnte, im Gegensatz zur verbesserten Lokalisierung, nur mit live aufgenommenen Routen getestet werden. Für die ersten Tests wurde zunächst eine Simulationsumgebung, am Deutschen Zentrum für Luft- und Raumfahrt entwickelt, verwendet. Der Algorithmus wurde so angepasst, dass entsprechende Bewegungsdaten aus dem Programm in die Simulation übermittelt werden konnten. Auf diese Weise wurde getestet, inwiefern die Navigation des Roboters anhand des verbesserten SequenceSLAM und der biologischen Navigation möglich ist.

5.2.1 Simulation

Die Simulation basiert dabei auf der Programmiersprache „Modelica“ und ist modular aufgebaut, sodass einzelne Teile effizient ausgetauscht werden können. Neben einer 3D-Umgebung stellte sie auch ein simuliertes Fahrzeug in Form der ebenfalls am DLR entwickelten Lightweight Rover Unit (LRU) bereit. Für die Landschaft in der Simulation wurde ein 3D-Scan der SpacebotCup-Umgebung von 2014 verwendet. Der Rover

besteht aus einem Körper, an dem vier Radachsen befestigt sind. Jedes Rad lässt sich einzeln lenken und ansteuern, was einen omnidirektionalen Betrieb des Fahrzeugs ermöglicht. Zudem ist der Rover, wie auch in der Realität, mit verschiedensten Sensoren ausgestattet, angefangen bei Inertial Measurement Units (Inertiale Messeinheiten, IMUs), bis hin zu verschiedenen Kameras. Zusätzlich zur Landschaft und dem Rover, wird dort auch die Physik simuliert. Die komplette Simulation wird über mehrere Middlewares, also Software, die die Kommunikation zwischen all den komplexen Prozessen ermöglicht, gesteuert [12].

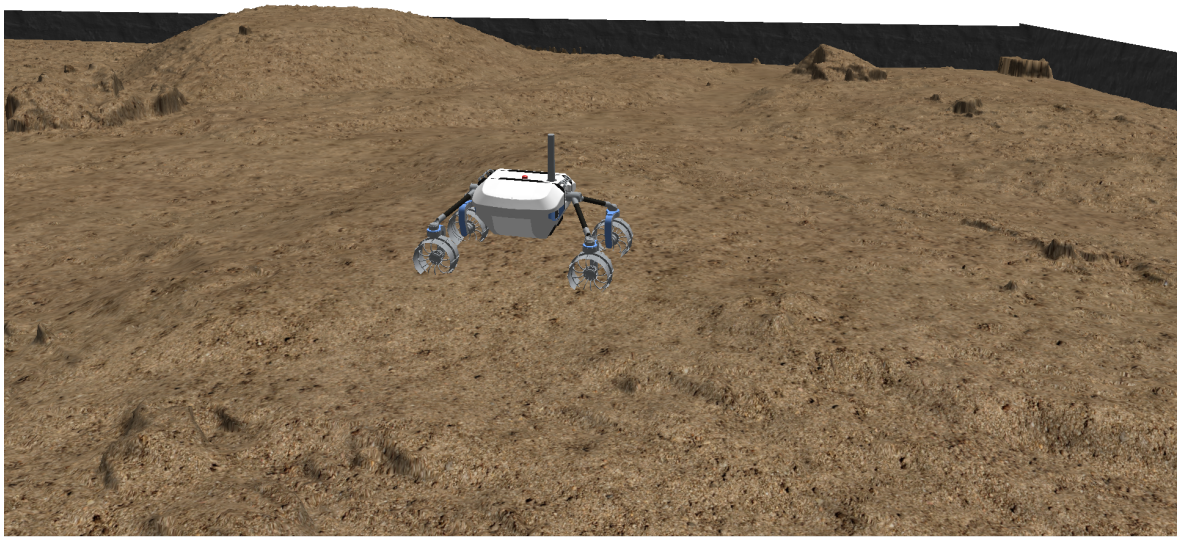


Abbildung 35: Simulationsumgebung für die Tests der Navigation. Die Spacebotcup-Umgebung wird durch die schwarzen Wände begrenzt. In der Mitte ist die LRU zu sehen. Auf dem Stab sitzt die Omni-Kamera, für die zum Zeitpunkt der Aufnahme noch kein 3D-Modell vorlag.

Zum einen wird der sogenannte „links and nodes"-Manager (ln-Manager) für den Start der einzelnen Prozesse und die Verwaltung der aktuell geladenen Module verwendet. Damit gerade optische Algorithmen getestet werden können, müssen zudem Sensordaten simuliert und in Form von lesbaren Daten an die Prozesse weitergeleitet werden. Dies geschieht mit der Software „SensorNet", die, wie der ln-Manager, ebenfalls am DLR entwickelt wurde. Um schließlich den Roboter anhand der gewonnenen Daten zu steuern, wird schlussendlich das Robot Operating System (ROS) verwendet, das als kostenlose Open Source Middleware eine effiziente Ansteuerung beinahe jeden Roboters ermöglicht.

Für die Versuche wurden ausschließlich die simulierte Omni-Kamera verwendet, die aus den vier Einzelbildern in jede Himmelsrichtung das Panoramabild zusammensetzt. Mithilfe der simulierten IMU konnten zudem die genauen Positionsdaten des Rovers zu

jedem Zeitpunkt ermittelt werden. Diese wurden später zur Verifizierung der Genauigkeit beim Folgen der Route verwendet.

5.2.2 Versuchsdurchführung

Der erste Schritt war die Aufnahme einer Route in der Simulation. Dabei wurden die Omni-Bilder (omnidirektional) der simulierten Kamera, sowie die genauen Positionsdaten des Roboter zu jedem Zeitpunkt aufgenommen. Da die Simulation mit ROS zur Steuerung des Roboters arbeitet, mussten die aufgezeichneten Daten anschließend aus dem ROS-spezifischen Datentyp extrahiert und in standardisierte Datentypen umgewandelt werden. Als Format für die Bilder wurde dabei „pgm“ gewählt. Der Grund hierfür lag in der Kommentarfunktion im Datenkopf (Header) des Bildes. Somit konnten die Positionsdaten des Roboters an jeder Stelle in die Bildheader geschrieben werden, um später im Algorithmus die Navigation zu unterstützen. Die Positionsdaten wurden zusätzlich in einer Textdatei gespeichert, wobei sie, wie die Bilder auch, mit ihren jeweiligen Zeitstempeln versehen wurden, um sie später wieder einander zuordnen zu können.

In der Simulation wurde eine Ausgabefrequenz von 15 Hz für die Bilder eingestellt. Dies war die maximal mögliche Frequenz, bei der die zeitaufwendige Umrechnung der vier Einzelaufnahmen in ein Panoramabild möglich war. Die Testroute wurde rechteckförmig und gegen den Uhrzeigersinn aufgenommen. Anhand der ersten Versuche konnte allerdings recht schnell erkannt werden, dass der Aufnahmeort zu uneben für die Lokalisierung des SequenceSLAM mit Omni-Bildern war. Dadurch konnten selbst bei kleinen Verschiebungen die Orte, aufgrund der Winkeländerungen des Fahrzeugs und der Kamera, nicht mehr korrekt lokalisiert werden.

Daher wurde noch eine zweite Route an einer flacheren Stelle aufgenommen. Diese wurde, im Gegensatz zur ersten Route, im Uhrzeigersinn gefahen und war nur am Anfang ungefähr rechteckig. Dies lag zum einen daran, dass der ebene Bereich in der Simulation keine große Rechteckroute erlaubte. Zum anderen konnte so getestet werden, ob die Lokalisierung auch bei anderen Routenformen funktioniert.

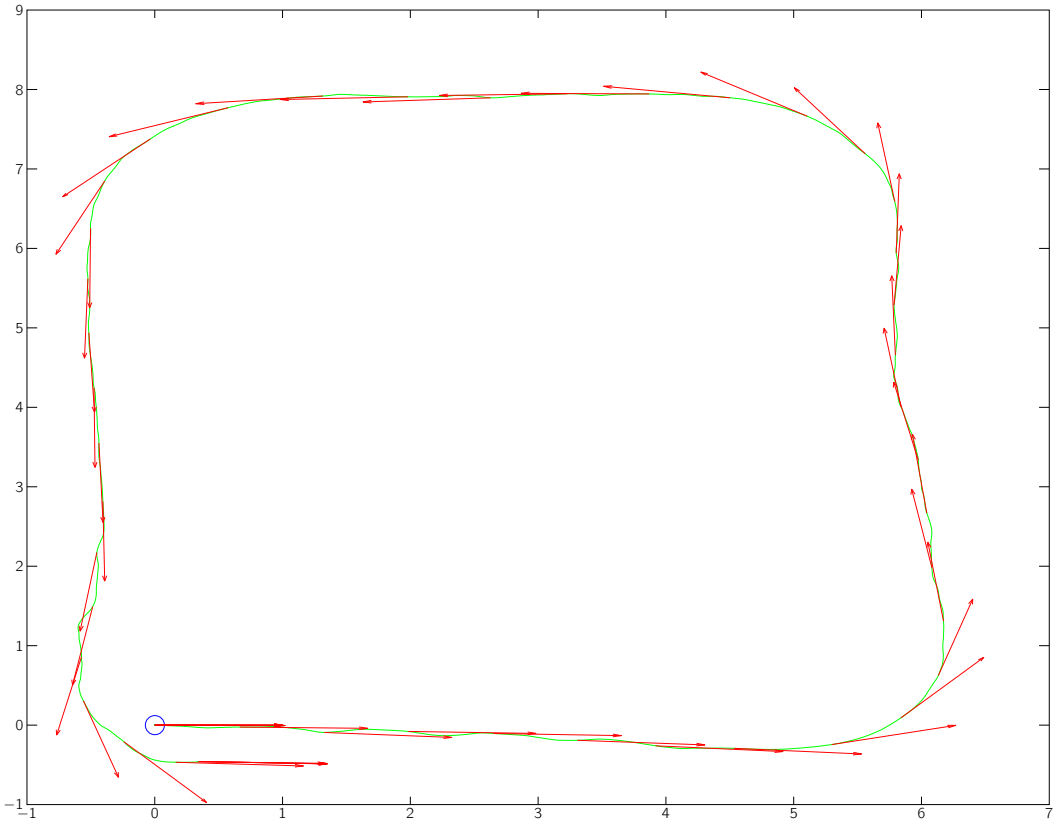


Abbildung 36: Ground-Truth Daten der aufgenommenen Route. Die Positionen des Fahrzeugs sind grün dargestellt, die roten Pfeile zeigen an, in welche Richtung der Rover aktuell fährt. Der blaue Kreis markiert den Startpunkt der Route.

Anschließend fanden die Versuche statt, deren Ergebnisse in nachfolgenden Kapitel noch vorgestellt werden. Hierfür wurde der Roboter auf den jeweiligen Routenstartpunkt gesetzt und der Algorithmus gestartet. Da dieser eine Mindestsequenz von Bilder benötigt, bevor er mit der Lokalisierung starten kann, ist es wichtig, dass er am Anfang nur geradeausfährt, damit er keine Kurve verpasst. Die Ergebnisse der Navigation wurden ebenfalls in Bildern und den Positionsdaten des Roboters festgehalten.

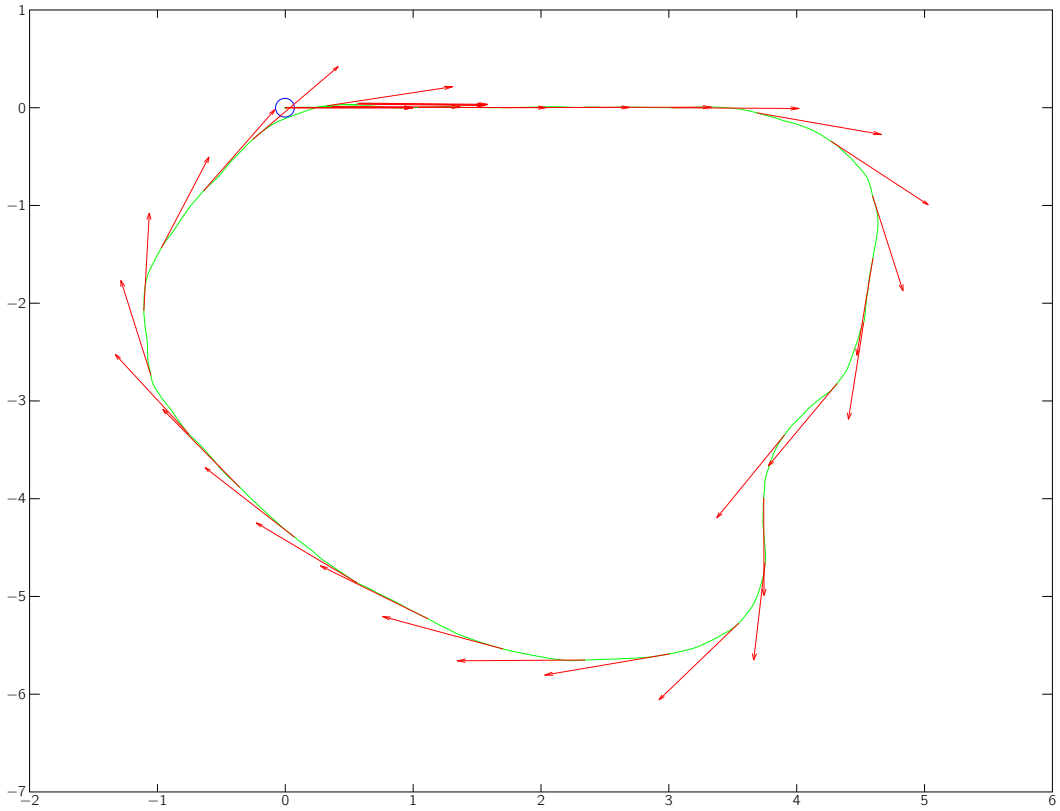


Abbildung 37: Ground-Truth Daten der zweiten, aufgenommenen Route. Die Positionen des Fahrzeugs sind grün dargestellt, die roten Pfeile zeigen an, in welche Richtung der Rover aktuell fährt. Der blaue Kreis markiert den Startpunkt der Route.

6 Ergebnisse und Evaluierung des Verfahrens

6.1 Lokalisierung

Zur Evaluierung des verbesserten SequenceSLAM wurden zwei Kriterien herangezogen. Ein Kriterium ist die Laufzeit, da der Algorithmus für die Echtzeit-Navigation schnell genug sein muss. Das zweite Kriterium für die Bewertung des Algorithmus sind die Matching-Ergebnisse. Diese werden im Folgenden in sogenannten Precision-Recall Plots visualisiert. Bei diesen wird auf der x-Achse der Recall-Wert, also die Anzahl der berücksichtigten Daten in Prozent, aufgetragen.

$$\text{Recall} = \frac{\text{True Positives} + \text{False Positives}}{\text{Gesamtanzahl der Daten}} \tag{23}$$

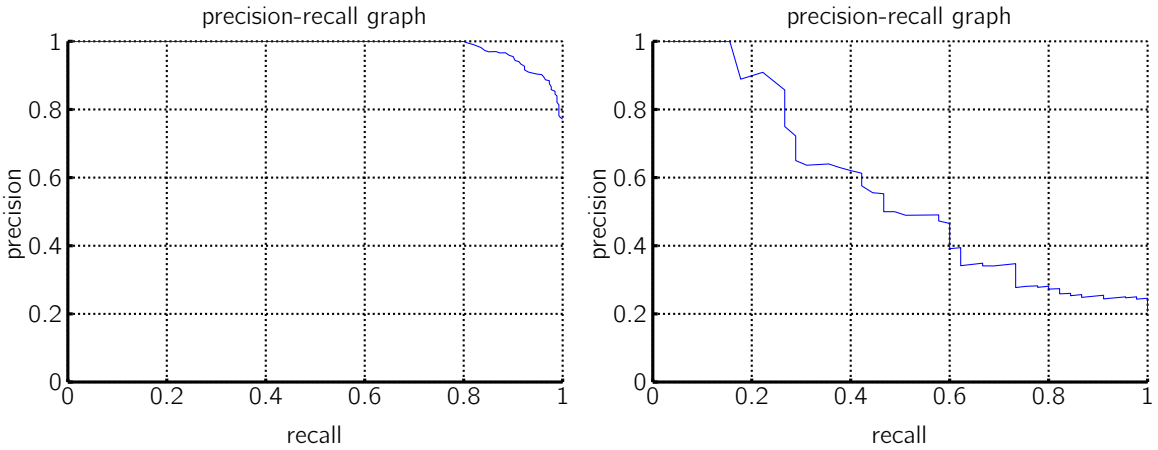
Der Precision-Wert beschreibt, wie viele der gefundenen Werte korrekt sind und gibt somit die Güte der aktuellen Ergebnisse an.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{24}$$

Dies soll durch folgendes Zahlenbeispiel verdeutlicht werden. Wenn von einer Datenbank mit 100 Bildern 40 Bilder als korrekte Lokalisierungen deklariert werden, so liegt der Recall-Wert bei 40%. Wenn dann bei den lokalisierten Bildern 5 falsche Matches dabei sind, liegt die Präzision bei $35/40 = 87{,}5\%$. Die Precision-Recall-Plots zeigen, ab wann und wie viele falsche Matches während des Vorgangs auftreten.

6.1.1 Ursprüngliche SequenceSLAM

Als Referenzwert für die weiteren Versuchsreihen wurden zunächst die Lokalisierungsergebnisse des originalen SequenceSLAM für die beiden Datensätze ausgewertet. Für die Nordland-Datenreihe werden hierbei allein die Ergebnisse mit der Winter-Route als Datenbank und der Sommerroute als Vergleichssdatensatz gezeigt.



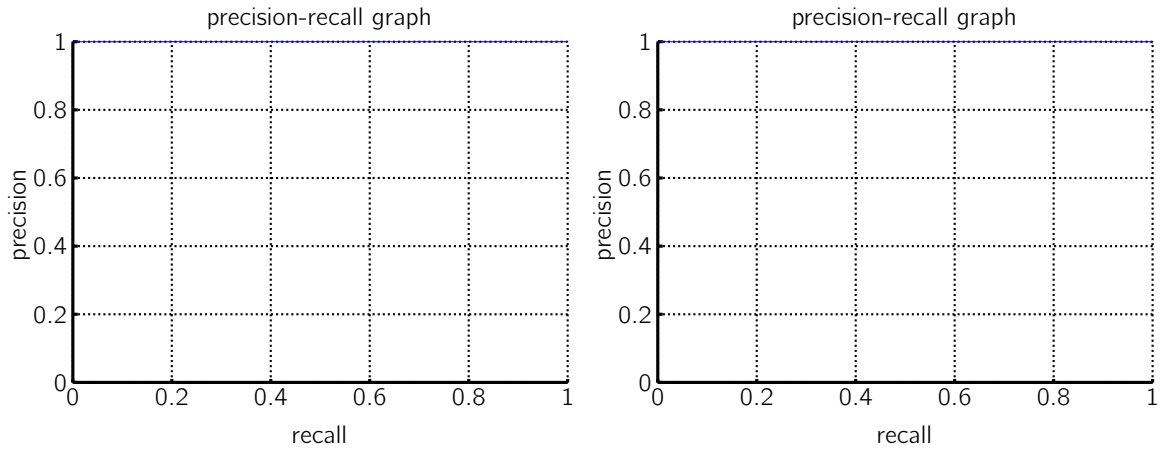
(a) Precision-Recall-Plot Nordland-Route *(b) Precision-Recall-Plot Milford-Route*

Abbildung 38: Precision-Recall-Plots des originalen SequenceSLAM-Algorithmus

Hierbei ist deutlich zu sehen, dass die Ergebnisse für die Nordland-Route schon beim originalen Algorithmus erstaunlich präzise sind. Allgemein können ca.70% der Bilder korrekt lokalisiert werden. Zudem erreicht der Algorithmus bis zu einer recall-Rate von 70% der Bilder noch 100% Präzision. Beim Milford-Datensatz können ungefähr 25% der Bilder richtig lokalisiert werden. Zudem sind nur bis zu einer Rate von ca. 20% der Bilder alle Matches korrekt. Die Laufzeit des Algorithmus, bis alle Bilder des Vergleichssdatensatzes gematcht worden waren, lag bei der Nordland-Reihe bei 5,1 s und bei der Milford-Route bei 4,5 s. Hierbei beeinflussen vor allem die Länge des Datensatzes und damit die Anzahl der Matching-Abläufe die Dauer.

6.1.2 Dynamische Anpassung der Sequenzlänge

Im nächsten Schritt wurde zunächst die dynamische Anpassung der Sequenzlänge hinzugefügt und die Ergebnisse wieder für beide Routen gemessen. Die Plots sahen dabei wie folgt aus:



(a) Precision-Recall-Plot Nordland-Route *(b) Precision-Recall-Plot Milford-Route*

Abbildung 39: Precision-Recall-Plots des SequenceSLAM-Algorithmus mit dynamisch anpassender Sequenzlänge

Bei beiden Plots konnte durch die Anpassung der Sequenzlänge eine Präzision von 100% bei nahezu jeder recall-Rate erreicht werden. Sogar bei der extrem anspruchsvollen Tag-Nacht-Route konnten alle bis auf drei Orte korrekt lokalisiert werden. Allerdings wurde dadurch auch die Laufzeit erhöht. Beim Nordland-Versuch lag sie nun bei 7,5 s, bei der Milford-Route bei 7,0 s. Auch wenn die Zeit immer noch sehr gering war, entsprach dies einer im Schnitt um 40% erhöhten Dauer für das Matching.

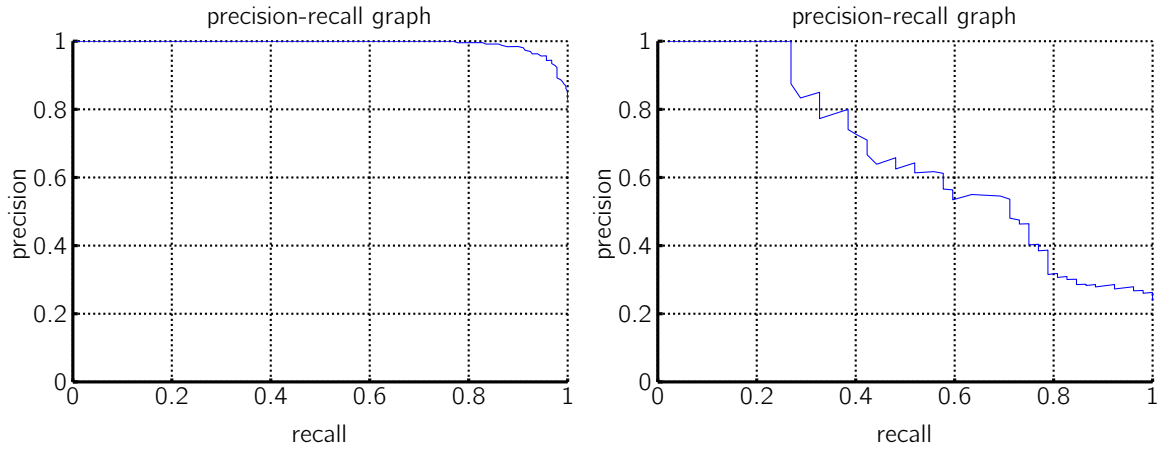
Zusätzlich werden die direkten Ergebniswerte an sich nicht verbessert. Das bedeutet, dass die abschließende Berechnung der Einzigartigkeit (s. Unterabschnitt 2.1.3) bei ca. 70% der Werte eine Güte von 0,9 oder schlechter ergab. Dies ist laut Milford der maximale Grenzwert, ab dem sichergestellt werden kann, dass alle Orte korrekt lokalisiert wurden, selbst wenn unbekannte Routenabschnitte gematcht werden [3].

Im Gegensatz dazu wurden beim Nordland-Datensatz die Ergebnisse durch das Verwenden der dynamischen Sequenzlänge deutlich verbessert. Hier waren nur noch 4 Bilder über der Marke von 0,9, von denen drei nicht korrekt gematcht wurden. Zudem waren alle Lokalisierungen mit einem kleineren Wert, im Gegensatz zur vorherigen Messung, korrekt. Somit konnten gerade für die jahreszeitlich schwankenden Datensätze deutliche Verbesserungen erzielt werden. So können auch schlechte Ergebnisse,

wenn fremde und nicht zugehörige Routenabschnitte aufgenommen werden, vermieden werden.

6.1.3 Begrenzung des Suchbereichs

Für den nächsten Versuch wurde die dynamische Sequenzlänge wieder abgeschaltet und stattdessen die Begrenzung des Suchbereichs eingeschaltet, um den Einfluss dieser Modifikation auf den Algorithmus zu testen. Dieser wurde nur dann eingeschränkt, wenn im vorhergehenden Durchlauf ein korrekter Match gefunden wurde.



(a) Precision-Recall-Plot Nordland-Route *(b) Precision-Recall-Plot Milford-Route*

Abbildung 40: Precision-Recall-Plots des SequenceSLAM-Algorithmus mit begrenztem Suchbereich nach gefundenem Match.

Anhand der Plots ist zu erkennen, dass die Ergebnisse im Vergleich zum originalen SequenceSLAM verbessert wurden. Allerdings ist diese nur minimal. Die Begrenzung wurde allerdings nicht eingefügt, um die allgemeine Präzision zu erhöhen. Stattdessen sollen bei Routen mit sich wiederholenden Streckenabschnitte die Lokalisierungsergebnisse verbessert werden.

Hierfür wurde eine weitere Versuchsreihe gestartet, in der der Datenbankvektor des Nordland-Satzes zweimal hintereinander geladen wurde. Der resultierende Vektor war dementsprechend doppelt so lang wie vorher und simulierte eine Schleifenfahrt über den gesamten Datensatz. Zu dieser Datenbank wurde anschließend der gleiche Vergleichsdatensatz wie davor gematcht. Durch die Schleifenfahrt konnten für jedes Bild zwei gleich gute Ergebnisse gefunden werden. Der Ähnlichkeitswert liegt somit für beinahe jeden Match bei 1,0. Da der Algorithmus chronologisch nach den minimalen Differenzwerten sucht, werden zwar trotzdem die gleichen Ergebnisse wie vorher gefunden. Bei einem Schwellwert von 0,9 werden diese jedoch nicht mehr berücksichtigt.

Durch die Begrenzung des Suchbereichs wird der zweite Durchlauf für das Matching nicht mehr berücksichtigt. Dadurch werden die Lokalisierungswerte wieder so verbessert, dass sie unter dem vorher festgelegten Schwellwert von 0,9 liegen. Somit können wieder die gleichen Ergebnisse wie bei der einfachen Schleifenfahrt gefunden werden, unabhängig davon, wie oft bestimmte Abschnitte in der Datenbank vorkommen.

6.1.4 Einschwärzen des Himmels

Da der Nordland-Datensatz schon in Form von Grauwertbildern vorlag, konnte dort nicht das Einschwärzen des Himmls angewandt und dessen Einfluss überprüft werden. Daher wurde der Test nur für den Milford-Datensatz durchgeführt, wobei nur die Tagroute eingeschwärzt wurde.

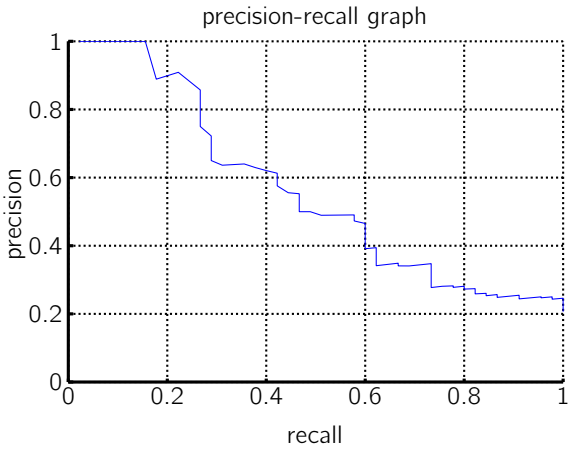


Abbildung 41: Precision-Recall-Plots des SequenceSLAM-Algorithmus mit geschwärztem Himmel. (Hier nur für Milford-Datensatz).

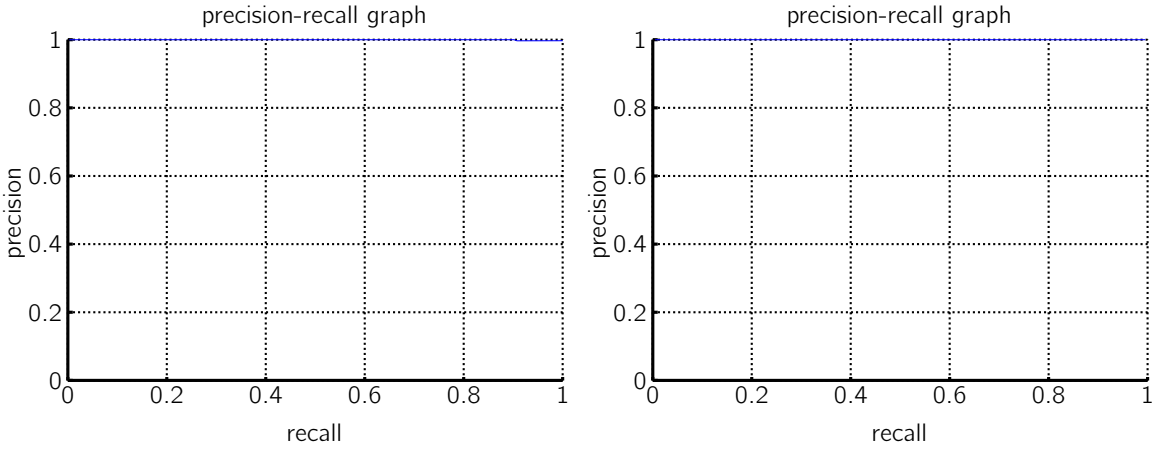
Hier hatte das Einschwärzen des Himmels leider nicht die erhoffte Wirkung auf das Matching, auch wenn der Himmel, wie in Abbildung 42 deutlich zu sehen, korrekt eingeschwärzt wird.

Die Nachtaufnahmen sind aufgrund der schlechten Sichtverhältnisse so dunkel, dass die Übereinstimmung trotz des eingeschwärzten Himmels nicht erhöht werden kann. Für die Tests lagen keine weiteren Datensätze mit bei Tag und bei Nacht aufgenommenen Routen vor, mit denen diese Modifikation besser getestet werden konnte. Trotz allem konnte gezeigt werden, dass das Verfahren zuverlässig den Himmel einschwärzt und somit Potenzial besitzt, Aufnahmen bei verschiedenen Tages- oder Jahreszeiten, sowie bei wetterbedingten Helligkeitsunterschieden anzugleichen.

Abbildung 42: Beispiele von Milford-Bildern mit eingeschwärztem Himmel.

6.1.5 Angepasster SequenceSLAM

Schlussendlich wurden alle Verbesserungen kombiniert und die Ergebnisse ebenfalls in Precision-Recall-Plots aufgetragen. Diese ähneln sehr stark den Ergebnissen mit der dynamischen Sequenzlänge, da diese die größte Auswirkung auf die Lokalisierungsergebnisse hatte. Allerdings werden durch die Kombination der verschiedenen Verfahren keine Matching-Ergebnisse verschlechtert.



(a) Precision-Recall-Plot Nordland-Route *(b) Precision-Recall-Plot Milford-Route*

Abbildung 43: Precision-Recall-Plots des SequenceSLAM-Algorithmus mit allen Verbesserungen.

6.2 Ergebnisse für die Navigation

Für die Validierung der Ergebnisse wurden mehrere Kriterien berücksichtigt:

- Erreicht der Roboter die Endposition?
- Wie weit weicht er während der Fahrt von der Route ab?
- Wie schnell erreicht der Roboter die Endposition?

Die ersten Versuche wurden zunächst mit halber Maximalgeschwindigkeit ausgeführt. Daher wurde die Zeitmessung anfangs nicht berücksichtigt. Die Abweichung von der Route wird mithilfe der xy-Graphen und der Orientierung an den jeweiligen Punkten dargestellt. Zunächst werden die Ergebnisse für die erste Route dargestellt.

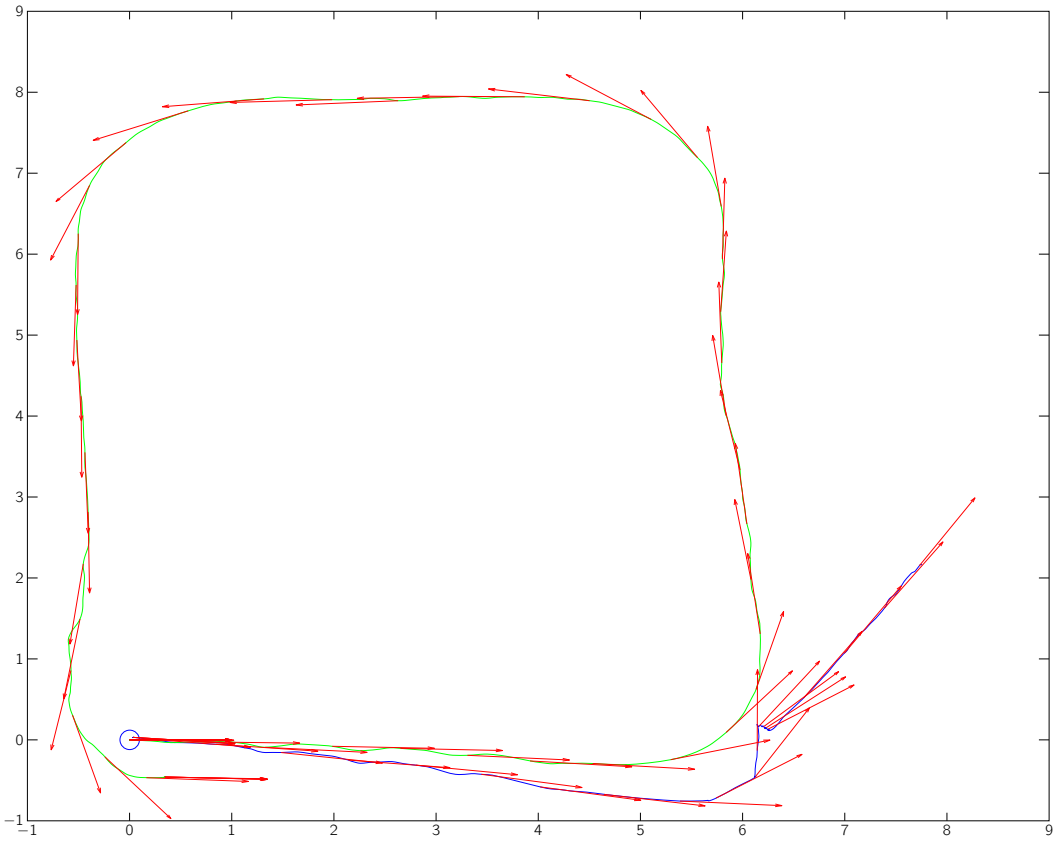


Abbildung 44: Positionsdaten des Rovers bei der Datenbankfahrt (grün) und der Navigationsfahrt (blau). Die roten Pfeile stellen die Blickrichtung des Rovers an den verschiedenen Positionen dar.

In Abbildung 44 ist der erste Navigationsversuch dargestellt. Der Roboter entfernt sich anfangs, wie vom Algorithmus vorgegeben, geradeaus von der Startposition in die vorgegebene Richtung. Aufgrund von kleineren Rechenfehlern konnte die genaue Anfangsorientierung nicht komplett reproduziert werden. Zudem war der Startpunkt auf einer leicht schiefen Ebene in der Simulation, weshalb der Roboter von der originalen Position etwas wegrutschte. Die Abweichung am Anfang war aber nicht groß genug, um das Matching des Algorithmus zu behindern.

Ab ca. der Hälfte der Geraden begann das Matching des Algorithmus und damit die Richtungsvorgabe für die weitere Bewegung. Zunächst wurde noch keine Korrektur vorgenommen, da der Roboter annähernd parallel zur Route und nicht zu weit von ihr entfernt entlang fuhr. Somit ergab der Vergleich der Panoramabilder keinen Korrekturwinkel. Dieser wurde zum ersten Mal auf Höhe der ersten Kurve ausgegeben.

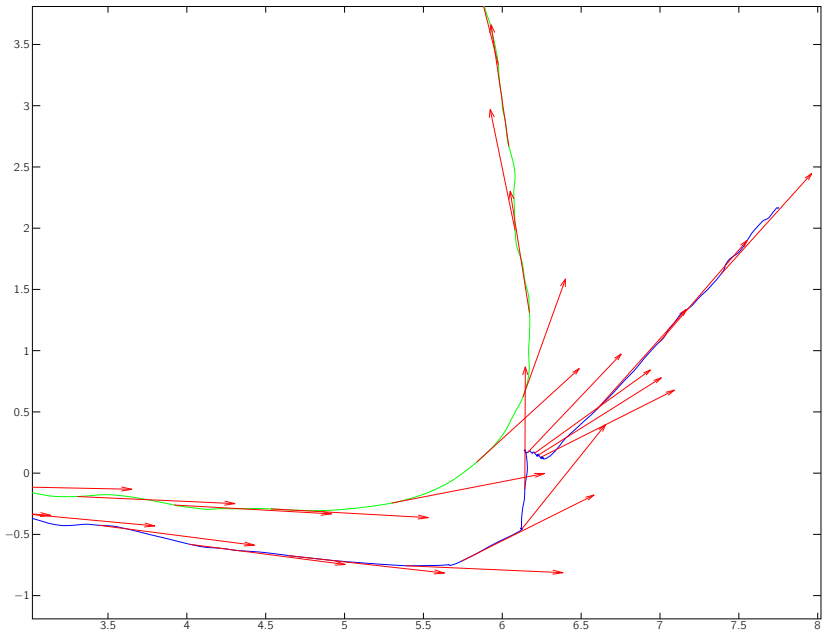


Abbildung 45: Zoom auf die erste Kurve des vorher gezeigten Versuchs.

Es ist deutlich zu erkennen, dass der Algorithmus versucht, die Fahrt in die richtige Richtung zu korrigieren, da die Abweichung von der Route zu groß ist. Der Roboter korrigiert seine Bewegungsrichtung nach links und fährt bis zum nächsten Aufnahmepunkt weiter. Kurz bevor er auf die Route zurückkehrt dreht er sich auf der Stelle, bis er sich schließlich von der Route entfernt.

Die anschließende Analyse zeigte, dass die Abweichung an dieser Stelle hauptsächlich dadurch entstand, dass der Roboter mehr Bilder in die Sequenz geladen hatte, als bei der ursprünglichen Aufnahme. Durch den größeren Radius der Kurve und weil hier Mehrfachaufnahmen bei Rotationen auf der Stelle noch nicht aus der Sequenz entfernt wurden, lokalisierte er sich weiter entfernt, als er wirklich war. Dies kann deutlich anhand eines weiteren Beispiels gezeigt werden.

Abbildung 46 zeigt den interessanten Ausschnitt eines weiteren Versuchs. Die Cyan-gefärbten Linien markieren die Punkte auf der Route, zu denen sich der Algorithmus jeweils lokalisiert. Die Richtungspfeile, die die aktuelle Orientierung des Roboters zeigen, sind wieder in Rot dargestellt. Sobald der Rover an den Anfang der Kurve kommt, bleibt er kurz stehen, um seine Orientierung zu korrigieren. Danach nimmt er ein wei-

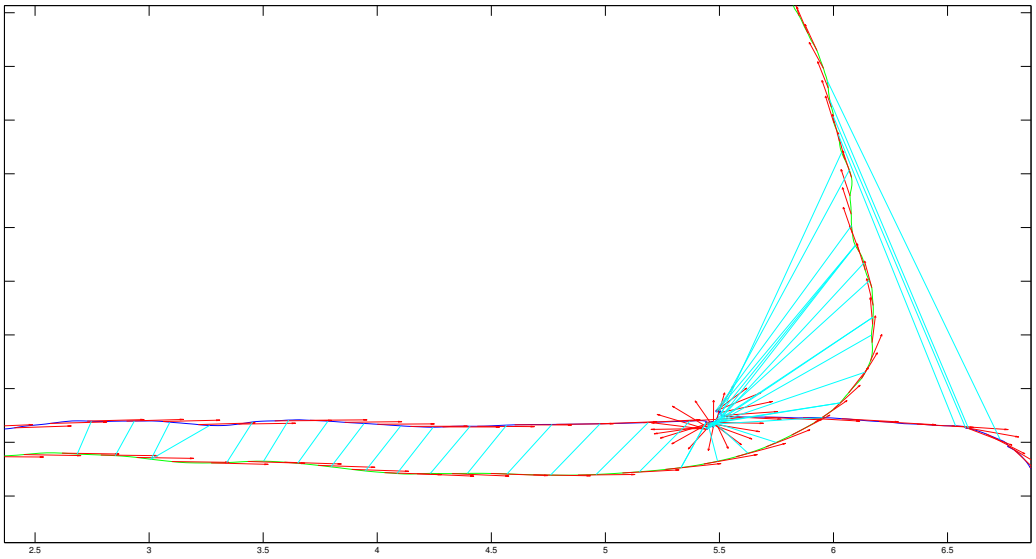


Abbildung 46: Weitere Route mit Verbindungslinien der Punkte in der Sequenz zu den gematchten Punkten aus der Datenbank.

teres Bild auf, ohne das vorherige zu löschen. Da sich die Position in der Kurve nicht so stark ändert, ist das neue Bild dem nächsten Datenbankbild ebenfalls sehr ähnlich, wodurch die beste Sequenz eine Position weiter ermittelt wird. Dieser Prozess wiederholt fortlaufend, während sich der Rover auf der Stelle dreht. So lokalisiert er sich schließlich am Ende der Kurve, obwohl er in Wirklichkeit seine Position nie verändert hat. Schlussendlich kann er sich nicht mehr lokalisieren, da die Sequenz nicht mehr zur Datenbanksequenz passt. Deshalb fährt er von seiner aktuellen Position aus geradeaus weg (rechts von der Kurve). Bei den nachfolgenden Matches handelt es sich um falsche Lokalisierungen.

Ansonsten konnten mit dieser Route keine aussagekräftigen Tests durchgeführt werden, da die Unebenheit des Bodens, gerade bei der ersten Kurve, ein korrektes Matching verhinderten und die noch in der Entwicklung befindliche Simulation nicht zeitnah genug angepasst werden konnte. Daher wurden weitere Tests mit der zweiten, ebeneren Route durchgeführt. Der erste Versuch soll in Abbildung 47 zunächst komplett dargestellt werden, ehe er wieder, wie zuvor, analysiert wird.

In diesem Versuch ist deutlich zu sehen, dass der Roboter sich um die erste Kurve navigieren kann. Er fängt an der ungefähr richtigen Stelle an, seine Orientierung zu ändern und entfernt sich während der Fahrt nicht weit von der Originalroute. Allerdings hat er Schwierigkeiten der kleinen Einbuchtung nach der ersten Kurve genau zu folgen. Bei der zweiten großen Kurve weicht er dann schließlich von der Route ab. Der Grund hierfür liegt ebenfalls in leichten Unebenheiten des Bodens. Dadurch lokalisiert sich der Roboter an einer falschen Stelle, was dazu führt, dass er sich in eine nicht vorhersagbare Richtung korrigiert. Dies kann schon durch leichte kleine Abweichungen

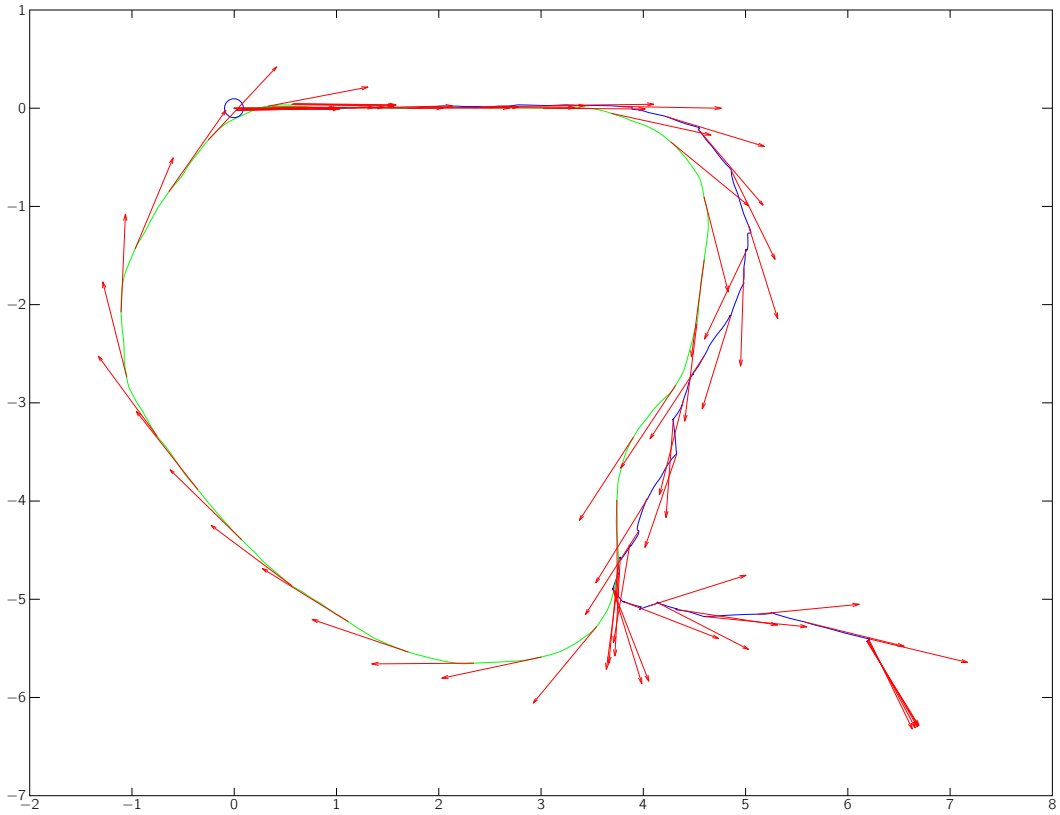


Abbildung 47: Positionsdaten des Roboters bei der zweiten Routenfahrt und dem anschließendem Versuch, dieser zu folgen. Die Originalroute ist wieder grün, der Folgeversuch blau dargestellt.

zur originalen Aufnahmeposition geschehen. So kann es sein, dass ein Rad bei einer kleinen Verdrehung zur Originalposition in eine Senke fährt. Dadurch würde sich der komplette Roboter verkippen und damit das aufgenommene Bild verzerren.

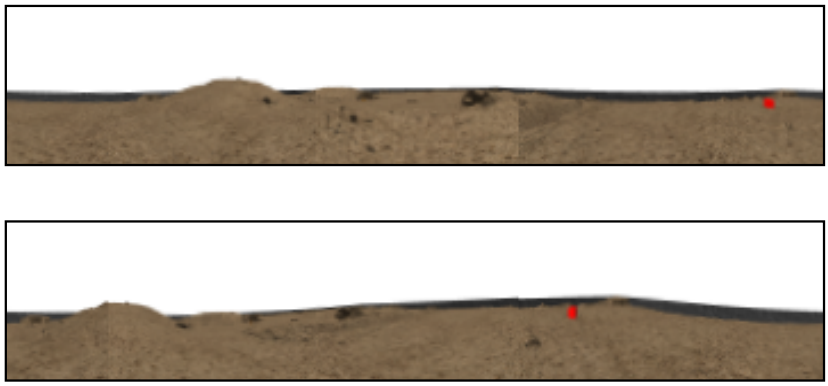


Abbildung 48: Darstellung eines der Lokalisierungsergebnisse anhand des aktuell aufgenommenen Bildes (oben) und des dazu gematchten Datenbankbildes (unten).

Abbildung 48 zeigt die Aufnahme an dem Moment, an dem der Roboter beginnt von der originalen Route abzuweichen. Die Szenerien besitzen eine hohe Ähnlichkeit, ge-

hören allerdings nicht zueinander. Das Bild, zu dem er sich normalerweise lokalisieren sollte, liegt weiter in der Zukunft, als das aktuell ausgewählte Bild. Das ist ein Effekt der linearen Sequenzbildung. Da die Route bei der Einbuchtung vor der zweiten Kurve etwas abgekürzt wird, hat er nicht so viele Bilder in der Sequenz, wie bei der Originalaufnahme. Dementsprechend lokalisiert er sich auch weiter in der Vergangenheit, weil dort die lokal beste, lineare Sequenz gefunden wird. Anhand des Bildes ermittelt der Algorithmus anschließend eine Abweichung vom Original in Form einer Verdrehung im Uhrzeigersinn. Um diese auszugleichen, gibt er eine Rotation gegen den Uhrzeigersinn vor. Dadurch weicht er, wie in Abbildung 49 zu sehen ist, nach links von der Route ab, bis er sich nicht mehr lokalisieren kann und in die letzte angegebene Richtung fährt.

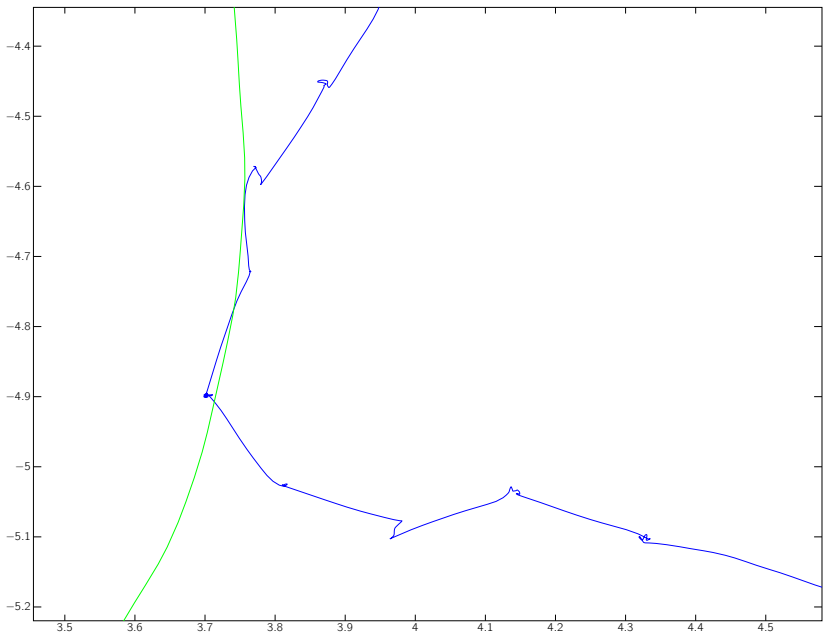


Abbildung 49: Position, an der der Rover von der vorgegebenen Route abweicht.

Dieser Effekt trat bei allen weiteren Versuchen so, oder so ähnlich auf, allerdings zu verschiedenen Zeitpunkten. So schaffte der Algorithmus bei einem weiteren Versuch fast die komplette Route, bis er schließlich davon abwich und sich nicht mehr lokalisieren konnte.

6.3 Evaluation

Bevor die dem SequenceSLAM hinzugefügte Navigation und deren Ergebnisse bewertet werden, wird zunächst auf die Modifikationen bezüglich der Lokalisierung des Roboters eingegangen. Diese zeigten einen überaus positiven Effekt auf die Ergebnisse. Gerade die dynamische Anpassung der Sequenzlänge ermöglichte es beinahe alle Bilder in einer Route korrekt zu lokalisieren. Auch die Einzigartigkeitswerte wurden teilweise stark

erhöht. Trotzdem muss für spätere Anwendungen berücksichtigt werden, dass ein gewisser Grenzwert für dieses Verfahren immer eingehalten werden muss, auch wenn die gewählten Beispiele immer korrekt gematcht wurden. Ansonsten kann nicht sichergestellt werden, dass keinerlei falsche Lokalisierungen auftreten.

Aber auch die anderen Modifikationen zeigten zufriedenstellende Ergebnisse. So konnten anhand der Begrenzung des Suchbereichs trotz einer simulierten Schleifenfahrt alle Orte korrekt und mit einem guten Wert lokalisiert werden. Anhand der Bilder kann zudem gezeigt werden, dass das Einschwärzen des Himmels die Ähnlichkeit einer Tagaufnahme mit einer Nachtaufnahme stark erhöht, solange die Umgebung ausreichend beleuchtet ist. Beispiele hierfür sind urbane Gegenden, sofern das Sichtverhältnis nicht zu sehr eingeschränkt wird (beispielsweise durch Nebel oder starke Regenfälle).

Die weiterentwickelte Lokalisierung eignete sich somit aufgrund ihrer hohen Stabilität und Lokalisierungsrate bestens als Grundlage einer Navigation. Durch die Installation einer Omni-Kamera wurde zudem die Empfindlichkeit des Algorithmus gegenüber Abweichungen von der Route verringert. Zudem passte sich die biologisch inspirierte Navigation gut in den Algorithmus mit ein und zeigte ansatzweise, dass das Prinzip funktionierte.

So konnten mehrere Tests belegen, dass der Algorithmus Abweichungen erkennen und allein anhand von Winkelvorgaben wieder korrigieren kann. Allerdings wurde die Navigation auch durch die Empfindlichkeit der Kamera gegenüber Neig- und Schwenkwinkel, sowie durch die lineare Sequenzbildung begrenzt. Diese führten teilweise zu Fehllokalisierungen oder zu Deadlocks, also Stellen, an denen der Algorithmus seine weitere Bewegungsrichtung nicht eindeutig bestimmen kann. Zwar kann er sich noch lokalisieren. Durch die Rotation auf der Stelle und der Bodenunebenheit entstehen aber Verkippungen, die das Bestimmen der weiteren Bewegungsrichtung verhindern. Um den Algorithmus in Zukunft zur Navigation von mobilen Plattformen einsetzen zu können, müssen genau diese Punkte bearbeitet werden.

7 Ausblick

Die in dieser Arbeit präsentierten Ergebnisse, zeigten dass der Algorithmus noch an vielen Stellen verbessert werden muss. Zusätzlich können auch noch weitere Modifikationen vorgenommen werden, die die Lokalisierung und Navigation weiter stabilisieren. Hierbei haben die folgenden drei Punkte den größten Einfluss:

- Bestimmung der Bildähnlichkeit
- Empfindlichkeit der Omni-Kamera gegenüber Verkippungen
- lineares Sequenzmatching

7.1 Methoden zur Bestimmung der Bildähnlichkeit

Bisher wurde über die Summe der absoluten Grauwertdifferenzen die Ähnlichkeit von zwei Orten bestimmt. Dieses ist zwar sehr schnell und bietet die Möglichkeit, Helligkeitsunterschiede auszugleichen. Der Nachteil ist jedoch eine hohe Empfindlichkeit gegenüber Änderungen der Aufnahmeposition. Zudem können die Verschiebungen und Rotationen nicht aus dem Bild zurückgerechnet werden.

Eine alternative Methode für das Matching wären Feature-Verfahren. Diese besitzen die entgegengesetzten Eigenschaften. Während mit ihnen Posenänderungen nicht nur ausgeglichen, sondern auch genau aus dem Bild errechnet werden können, sind die Verfahren sehr anfällig gegenüber Helligkeitsänderungen. Beispiele für verschiedene Feature-Deskriptoren sind SURF oder SIFT.

Dadurch würde der SequenceSLAM seine Stärke, nämlich die Robustheit gegenüber Helligkeitsschwankungen, verlieren. Andere, sequenzbasierte Verfahren haben für das Matching sogenannte HOG-Deskriptoren verwendet. Hierbei werden die Gradienten jedes Pixels mit Betrag und Richtung bestimmt. Anschließend wird das Bild in Zellen unterteilt und für jede Zelle das Histogram of Oriented Gradients (HOG) bestimmt. Diese Zellen besitzen den Vorteil, dass sie in gewisser Weise rotations- und translationsinvariant sind, wenn die ermittelten Gradienten auf die Hauptrichtung des Bildes bezogen werden.

Dadurch kann der Algorithmus gegenüber Positionsänderungen und Verkippungen bei der Aufnahme stabilisiert werden. Da dies immer zu Lasten der Rechenzeit geht, sollen im Folgenden Methoden zur Bildstabilisierung beschrieben werden.

7.2 Kamerabilder stabilisieren

Die Omni-Bilder sind bereits gegenüber Rotationen um die Bildachse komplett invariant. Auch kleine Verschiebungen entlang der Bodenebene können ausgeglichen werden. Der wichtigste Faktor, gegenüber den die Kamera stabilisiert werden muss, ist in diesem Fall die Rotation des Bildes um die x- und y-Achse.

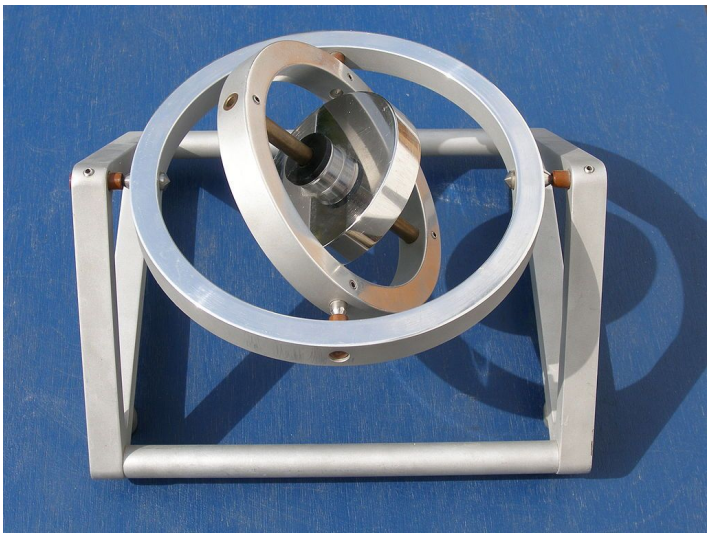


Abbildung 50: Grundlegender Aufbau eines mechanischen gimbals. Dieser kann dazu genutzt werden, die Scheibe in der Mitte gegenüber Verkippungen in jede Richtung zu stabilisieren. Quelle: http://www.drohnen.de/wp-content/uploads/2015/02/Gimbal.jpg

Hierfür gibt es ebenfalls zwei Möglichkeiten. Die erste ist eine softwareseitige Umrechnung des Bildes. Dafür muss zuerst die Verkippung des Bildes extern gemessen werden. Ist diese bekannt, kann ein neues Bild berechnet werden. Dabei werden die einzelnen Bildpunkte anhand der vorher gemessenen Winkel an eine neue Position innerhalb des Bildes geschoben. Allerdings gehen durch diesen Prozess auch Bildinformationen verloren, da an den Rändern teilweise Bildpixel außerhalb des Aufnahmebereichs benötigt werden. Um an diesen Stellen keine leeren Bildbereiche zu haben, werden die Bilder am oberen und unteren Rand abgeschnitten. Aus diesem Grund darf die Verkippung, die ausgeglichen werden soll, nicht zu groß sein, damit noch genügend Bildinformationen für das nachfolgende Matching übrig bleiben.

Eine andere Methode, die keine Informationsverluste zu Folge hätte, wäre die Verwendung eines sogenannten gimbals. Dabei handelt es sich um eine spezielle Art der Lagerung, die versucht die Kamera immer in Waage zu halten.

Hierfür werden zwei Lager in zwei Ebenen und rechtwinklig zueinander angeordnet. Wichtig ist zudem, dass der Schwerpunkt des gesamten Konstrukts, also Kamera und gimbal, genau mittig ist. Um auch schnelle Bewegungen, wie zum Beispiel bei Drohnen, ausgleichen zu können, reicht eine mechanische Lagerung aufgrund der Trägheit oft nicht mehr aus. In diesem Fall werden stattdessen kleine Elektromotoren verwendet, die den Ausgleich der Verkippungen aufgrund der Daten einer Inertial Measurement Unit (IMU) übernehmen.

Auf diese Weise könnten die Kamerabilder stabilisiert werden und es würden keine Verkippungseffekte, wie zum Beispiel das Festfahren des Roboters an bestimmten Stellen

der Route, mehr auftreten.

7.3 Anpassen der Sequenzsuche

Bisher wird die Sequenz als eine lineare Folge von Bildern in der Differenzenmatrix betrachtet. Dies setzt voraus, dass die Geschwindigkeit während der gesamten Aufnahme konstant bleibt. Außerdem ist diese Art der Aufnahme sehr anfällig gegenüber Abweichungen von der Route, durch die Bilder aus der Sequenz entfernt oder zu ihr hinzugefügt werden. Gerade bei dynamischen Prozessen in einer realen Umgebung, in der der Roboter nicht immer mit konstanter Geschwindigkeit fahren kann, würde das zu Problemen bei der Lokalisierung führen.

Eine Möglichkeit den SequenceSLAM an reale Bedingungen anzupassen, wäre die Verwendung einer nichtlinearen Sequenz. Hierbei würde keine Gerade sondern eine beliebige Kurve in der Matrix gesucht werden. Dadurch würde der Einfluss von schwankenden Geschwindigkeiten während der Fahrt verringert werden.

Ein weiterer Ansatz würde sich komplett vom Gedanken einer zusammenhängenden Sequenz lösen. Hierbei werden Wahrscheinlichkeiten für das Bestimmen der Sequenz verwendet. Ausgehend vom letzten bekannten Punkt auf der Route könnten die einzelnen Punkte in der Route zum Beispiel in Form einer Gaußschen Verteilung gewichtet werden. Somit würden Ergebnisse in der Nähe der letzten bekannten Roboterposition automatisch höher gewichtet werden, als weit entfernte Ergebnisse. Ein Beispiel hierfür ist das sogenannte Hidden-Markov-Modell [13].

Dadurch wäre das Matching komplett unabhängig gegenüber Geschwindigkeitsänderungen. Zudem hätten zusätzliche oder fehlende Bilder in der Sequenz keinen zu großen Einfluss mehr auf die Lokalisierung. Mit dieser Methode könnten alle Schwächen der Sequenzbildung beim SequenceSLAM effektiv entfernt werden.

Quellenverzeichnis

[1] Z. Wang, S. Huang, and G. Dissanayake, "D-slam: A decoupled solution to simultaneous localization and mapping," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 187–204, 2007.

[2] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*, pp. 834–849, Springer, 2014.

[3] M. J. Milford and G. F. Wyeth, "Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1643–1649, IEEE, 2012.

[4] M. Milford, "Vision-based place recognition: how low can you go?," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 766–789, 2013.

[5] N. Sünderhauf, P. Neubert, and P. Protzel, "Are we there yet? challenging seqslam on a 3000 km journey across all four seasons," in *Proc. of Workshop on Long-Term Autonomy, IEEE International Conference on Robotics and Automation (ICRA)*, p. 2013, 2013.

[6] E. Pepperell, P. Corke, and M. Milford, "Towards persistent visual navigation using smart," in *Proceedings of Australasian Conference on Robotics and Automation*, ARAA, 2013.

[7] S. Thurrowgood, D. Soccol, R. J. Moore, D. Bland, and M. V. Srinivasan, "A vision based system for attitude estimation of uavs," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5725–5730, IEEE, 2009.

[8] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[9] H.-F. Ng, "Automatic thresholding for defect detection," *Pattern recognition letters*, vol. 27, no. 14, pp. 1644–1649, 2006.

[10] Y. Wang, X. Hu, J. Lian, L. Zhang, and X. Kong, "Improved seq slam for real-time place recognition and navigation error correction," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference on*, vol. 1, pp. 260–264, IEEE, 2015.

[11] B. Baddeley, P. Graham, P. Husbands, and A. Philippides, "A model of ant route navigation driven by scene familiarity," *PLoS Comput Biol*, vol. 8, no. 1, p. e1002336, 2012.

[12] M. Hellerer, M. J. Schuster, and R. Lichtenheldt, "Software-in-the-loop simulation of a planetary rover," in *The International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2016)*, June 2016.

[13] P. Blunsom, "Hidden markov models," *Lecture notes, August*, vol. 15, pp. 18–19, 2004.