

Entwickeln eines sequenzbasierten Verfahrens zum Folgen einer Route anhand biologisch inspirierter Lokalisierungsalgorithmen

Philippe Schulz

Robotik und Automation

Hochschule Heilbronn

18. August 2016

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	III
1 Einleitung	1
2 Lokalisierung auf einer bekannten Route	2
2.1 Der SequenceSLAM	2
2.1.1 Aufnahme der Bildsequenzen	3
2.1.2 Präprozessierung des Bildes	3
2.1.3 Lokalisierung der Datensätze	5
2.2 Stärken und Schwächen	9
3 Anpassungen des Algorithmus	11
3.1 Vorbereitungen für die Live-Anwendung	11
3.1.1 Dynamische Matrixerzeugung	11
3.1.2 Synchronisation der Aufnahmepunkte	12
3.2 Dynamische Ermittlung der Sequenzlänge	14
3.2.1 Versuchsbedingungen	15
3.2.2 Versuchsergebnisse	16
3.3 Sequenzbildung	17
3.4 Umgang mit sich selbst wiederholenden Routen	18
3.5 Schwärzen des Himmels	20
3.5.1 Bildtransformation zur Kontrasterhöhung des Himmels	21
3.5.2 Segmentation des Himmels	21
4 Biologisch inspirierte Navigation	26
4.1 Navigation der Ameise	26
4.2 Technische Umsetzung	28
4.2.1 Installation einer Omni-Kamera	29
4.2.2 Einbindung der biologischen Navigation in die Simulation	30
5 Versuchsaufbau und -durchführung	32
6 Ergebnisse und Evaluierung des Verfahrens	33
7 Ausblick	34

Abbildungsverzeichnis

1	Bildvektoren für den SequenceSLAM	4
2	Präprozessierung des Bildes	5
3	Schematische Darstellung der Differenzenmatrix	6
4	Darstellung der Kontrastnormalisierung	7
5	Schematische Darstellung der Sequenzfindung	8
6	Lokale Kontrastnormalisierung der Grauwertvektoren	10
7	Dynamische Matrixerzeugung	12
8	Beispiel Bilder aus dem Nordland-Datensatz	16
9	Beispiele für Routen mit Loop Closure	18
10	Schematische Darstellung der Datenbank	19
11	Methode 1 zur Schleifenbehandlung	19
12	Methode 2 zur Schleifenbehandlung	19
13	Kontrasttransformation für sky blackening	22
14	Histogrammtransformation	22
15	Kontrasttransformation für sky blackening	24
16	Wegfindung der Ameise	27

Tabellenverzeichnis

1 Auflistung der Stärken und Schwächen des SequenceSLAM Algorithmus	9
2 Übersichtstabelle über die Vor- und Nachteile der verschiedenen Synchro- nisationsarten	14
3 Matchingergebnisse mit und ohne dynamischer Sequenzlänge	17

1 Einleitung

2 Lokalisierung auf einer bekannten Route

Damit eine Roboternavigation stattfinden kann, muss, wie bei uns 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 mehr oder weniger gut ausführen. 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 [1].

Für dieses Verfahren existieren sowohl eine frei zugängliche Matlab-Implementierung von Nico Sünderhauf (<https://svn.openslam.org/data/svn/openseqlslam/>) und eine darauf basierende, ebenfalls frei zugängliche C++-Implementierung (<https://github.com/subokita/OpenSeqSLAM/>), die als Grundlage für die nachfolgende Arbeit verwendet wurden.

2.1 Der SequenceSLAM

Der SequenceSLAM ist in seiner ursprünglichen Form ein Verfahren, das die Vilder 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 Erscheinungsbild eines Ortes ermöglicht. Seönst 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 haben sich bereits viele Arbeiten damit beschäftigt, ihn als Grundlage einer optischen Navigation zu verwenden. Allerdings sind hierfür noch einige Anpassungen nötig. Diese Arbeit beschäftigt sich mit den notwendigen Schritten, um den SequenceSLAM für eine Live-Anwendung vorzubereiten.

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

2.1.1 Aufnahme der Bildsequenzen

Die Lokalisierung basiert auf dem Vergleich einer Bildsequenz mit einer Datenbank. Daher muss die Route, die später gefahren werden soll, 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 unpräziser. Gleichzeitig darf der Abstand zwischen den Bildern auch nicht zu groß sein. 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 Aufnahmefrequenz 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 ist jedoch zu beachten, dass die Bildfrequenz sowohl für den Referenzvektor, als auch für den Vergleichsvektor gleich gewählt sein muss, 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:

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

Für den Algorithmus ist das Umwandlungsverfahren jedoch nicht relevant. Es gilt nur zu beachten, dass sowohl bei den Referenzbildern der Datenbank, als auch bei den

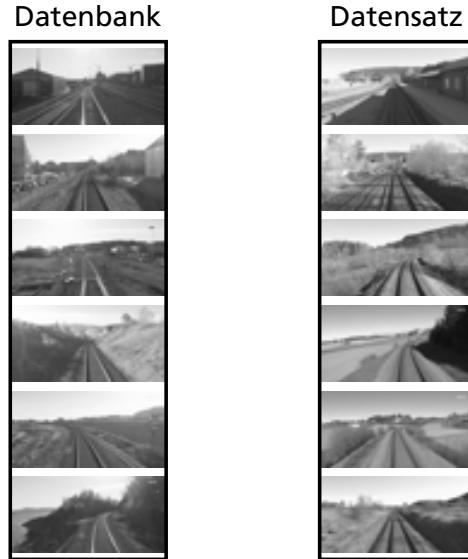


Abbildung 1: Beispielhafter Ausschnitt der beiden Vektoren für den SequenceSLAM. Links ist der Datenbank-Vektor, rechts der danach aufgenommene Vergleichsdatensatz. Die Bilder sind aus dem sogenannten Nordland-Datensatz (Näheres in Unterabschnitt 3.2.1)

Vergleichsbildern aus dem zweiten Datensatz die gleiche Methode verwendet wird, um die resultierenden Ähnlichkeitswerte nicht zu verfälschen.

Anschließend wird das Bild auf eine Größe von 64x32 Pixeln skaliert. Dieser Wert wurde von Milford und Wyeth in ihrer ursprünglichen Veröffentlichung verwendet. In weiteren Studien wurde zudem gezeigt, dass höhere Auflösungen keine nennenswerten Verbesserungen der Ergebnisse zur Folge hatten [2]. Im Gegenteil konnten sogar mit einer Auflösung von nur 32 Bildpixeln noch beeindruckend gute Ergebnisse erzielt werden. Aufgrund der sehr geringen Bildauflösung wird die Rechenzeit für die Berechnung der absoluten Summe der Grauwertdifferenzen extrem verringert. Zudem ist selbst bei großen Datenbanken der Speicherbedarf sehr gering.

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

$$D_{i,n} = c_1 * (D_i - c_2) \quad (2)$$

Hierbei gilt:

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

$$c_2 = D_{min}$$

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 einzigartiger wird.

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 drone-racing Video. Die Größenrelationen sind für die Visualisierung 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.

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

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

Außer dieser Formel wurden keine weiterführenden Filteroperationen für die Ähnlichkeitsbestimmung verwendet. Die Art des Vergleichs ist jedoch laut Milford nicht relevant für den Algorithmus. Wichtig ist nur, dass ein numerischer Wert bestimmt wird, der die Ähnlichkeit zwischen den beiden Bildern beschreibt [1]. Die Gründe hierfür werden später nochmals beschrieben.

Anschließend wird jedes einzelne 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 im Folgenden zur besseren Visualisierung als Grauwertbild dargestellt.

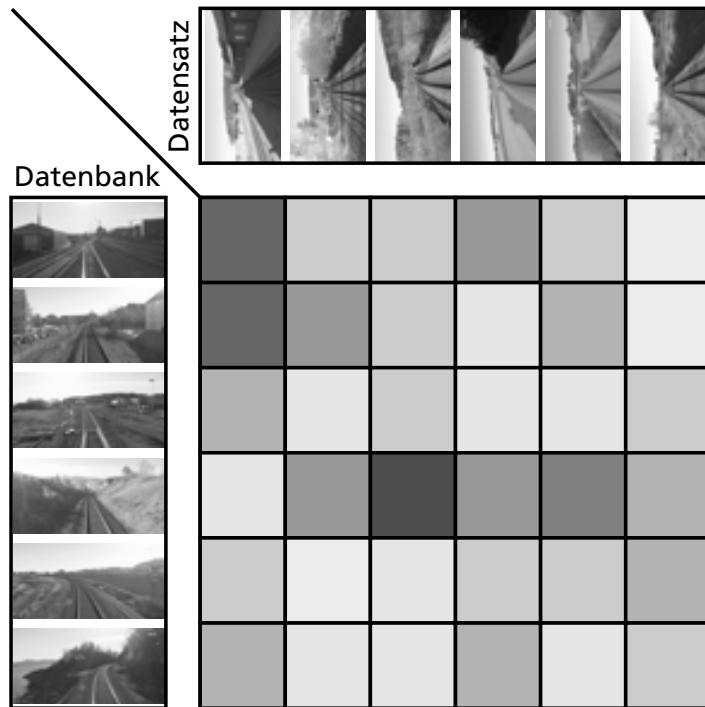


Abbildung 3: Schematische Darstellung der Differenzenmatrix aus den vorher gezeigten Datenvektoren. Je dunkler das Rasterfeld, desto ähnlicher sind sich die Bilder und desto wahrscheinlicher ein gefundener Match

Ab hier könnte bereits die Lokalisierung gestartet werden. Da jedoch zwischen zwei verschiedenen Aufnahmen Unterschiede auftreten können, wurde von Milford noch ein Zwischenschritt eingefügt, um trotzdem die ähnlichsten Bilder zu finden. Dabei handelt es sich um ein Verfahren zur lokalen Kontrastverstärkung. Jedes Matrixelement wird dabei mit seinen direkten Nachbarn aus der gleichen Spalte verglichen und mithilfe des Mittelwerts \bar{D}_i und der Standardabweichung σ_i normalisiert (siehe Gleichung 4). Die so ermittelten Werte werden anschließend positionsgetreu in eine neue Matrix geschrieben.

$$D_i = \frac{\bar{D}_i - D_i}{\sigma_i} \quad (4)$$

Abschließend findet die eigentliche Lokalisierung statt. Wie bereits erwähnt wird hierfür nach einer passenden Sequenz in der Datenbank gesucht. Da die Bilder chronolo-

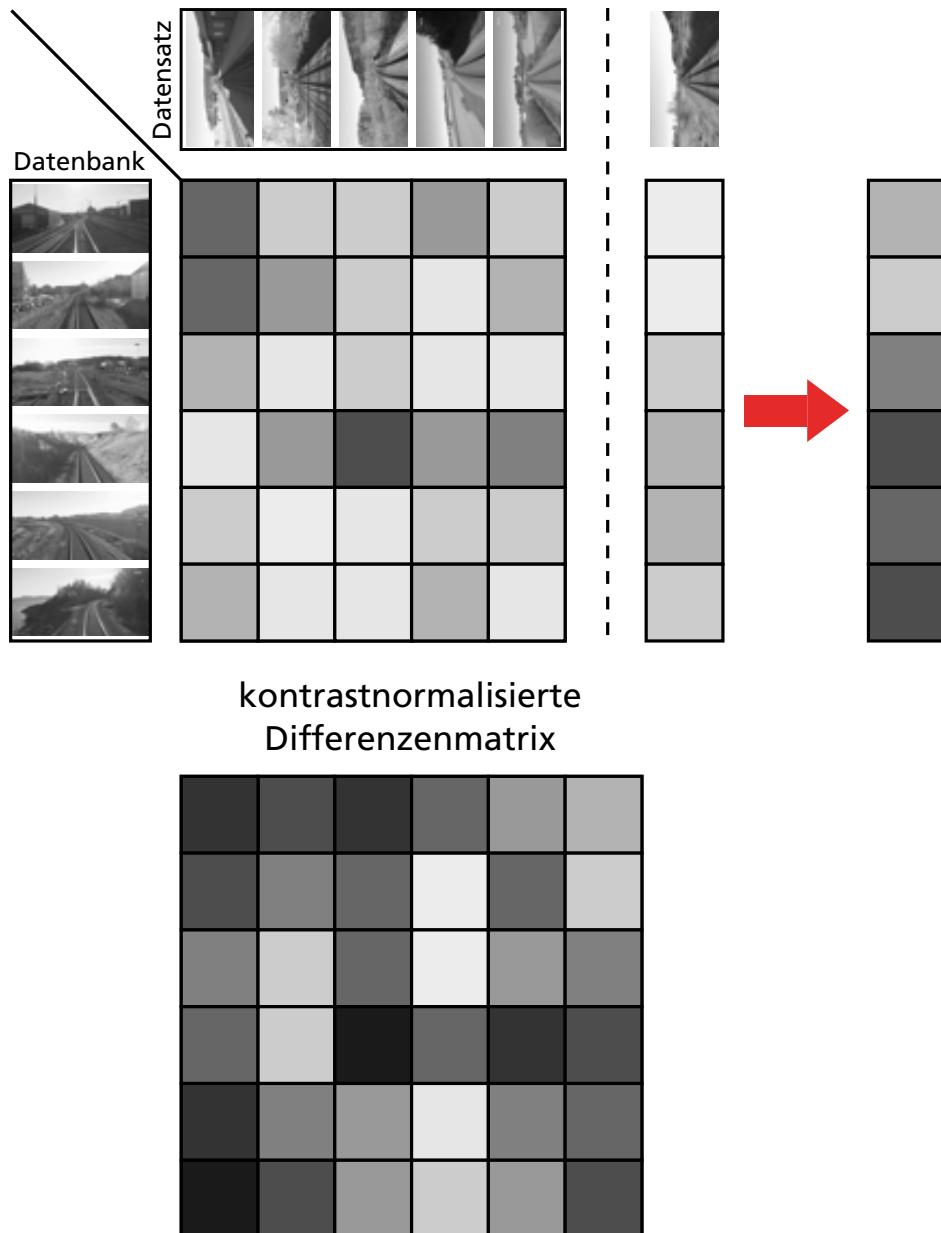


Abbildung 4: Kontrastnormalisierung der Ergebnismatrix. Hier ist ein beispielhafter Ausschnitt dargestellt, wie er in der Differenzenmatrix vorkommen könnte. Jede Spalte der Matrix wird nun einzeln kontrastnormalisiert und anschließend wieder positionsgetreu in die neue Matrix eingetragen.

gisch 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.

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.

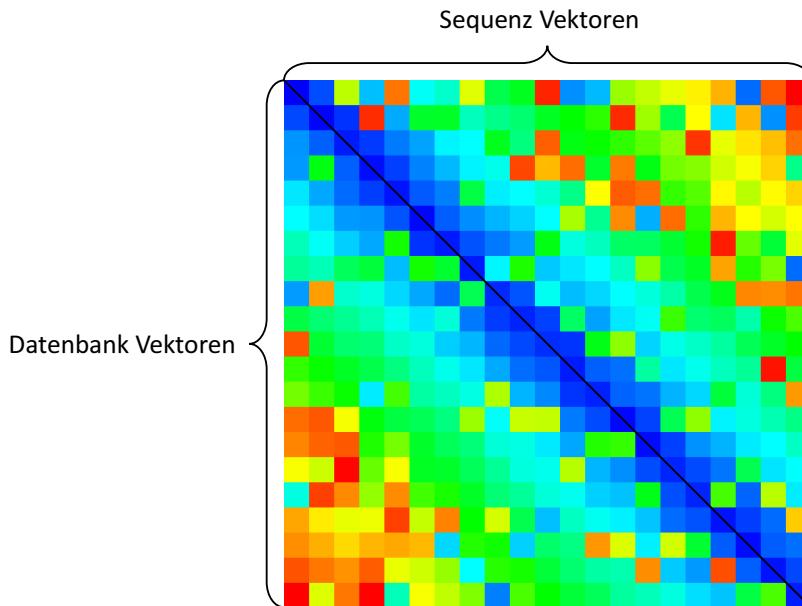


Abbildung 5: Schematische Darstellung der Sequenzfindung. Die Differenzenmatrix wird hier in Falschfarben dargestellt (blau entspricht einer sehr hohen Ähnlichkeit, rot einer sehr geringen). Die Diagonale ist hierbei die beste Sequenz.

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 das aufaddierte Ergebnis 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 das Fahrzeug 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
<ul style="list-style-type: none"> Der SequenceSLAM ist extrem robust gegenüber Helligkeitsänderungen 	<ul style="list-style-type: none"> Es handelt sich um ein rein passives Lokalisierungsverfahren, es können also keine Bewegungsinformationen anhand der Bilder ermittelt werden.
<ul style="list-style-type: none"> Das Matching-Verfahren ist aufgrund des Grauwertvergleichs und der geringen Bildauflösung relativ schnell 	<ul style="list-style-type: none"> Es ist nur teilautonom, da die Route vorher einmal aufgenommen werden muss.
<ul style="list-style-type: none"> Der Speicherbedarf (und damit die matching-Zeit) hängt linear von der Größe der Datenbank ab und ändert sich nicht während der Laufzeit 	<ul style="list-style-type: none"> Der Blickwinkel und andere Kamera-einstellungen 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. Gleichzeitig werden global sehr gute Ergebnisse, die allerdings keine großen Unterschiede zu den sie umliegenden Orten aufweisen, dadurch entfernt. Dieses Verfahren ist nochmal grafisch in der nachfolgenden Abbildung dargestellt.

Allerdings werden durch diesen Schritt auch einige falsche Positionen als mögliche Lokalisierungen gewertet. Anhand der Sequenz können jedoch einzelne Ausreißer herausgefiltert werden (siehe Abbildung 5). Im optimalen Fall gibt es nur eine Sequenz von Bildern in der Differenzenmatrix. Damit dieser Fall sichergestellt werden kann, wird die Güte der Lokalisierung bestimmt mithilfe der zweitbesten Sequenz bestimmt. Ist diese zu schlecht, wird der Ort als „false positive“ deklariert und verworfen.

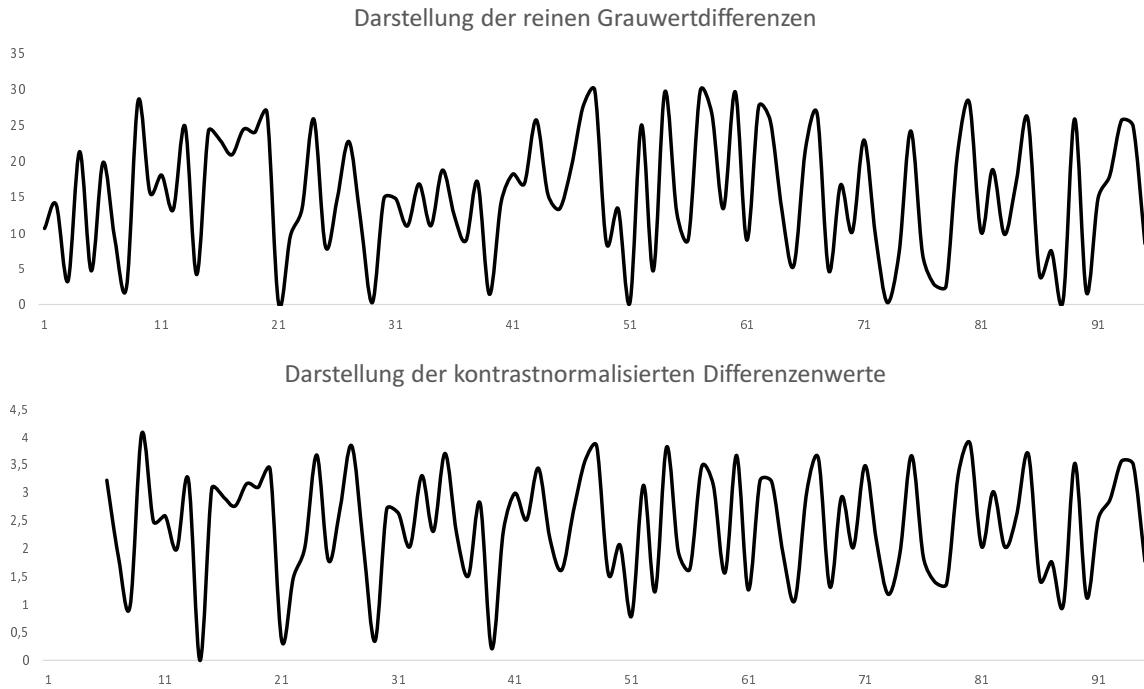


Abbildung 6: Darstellung der Ähnlichkeitswerte eines Differenzenvektors als Graph. Die Ähnlichkeitswerte werden hierbei auf die Bildindizes aufgetragen. Oben ist der Ähnlichkeitsvektor vor der Kontrastverstärkung dargestellt, unten danach. Der grüne Punkt ist dabei der lokal beste Match, die roten Punkte die global besten Matches.

Die 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 [3]). 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 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 Laufzeit des Algorithmus ständig angepasst und durch den Differenzenvektor des aktuellen Frames mit allen Datenbankbildern ergänzt.

3.1.1 Dynamische Matrixerzeugung

Wie beim originalen SequenceSLAM-Algorithmus muss die Datenbank einmal komplett aufgenommen werden. Im Gegensatz zum Original wird der Vergleichsdatensatz jedoch 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 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 Vektor in die entsprechende Spalte der Matrix geschrieben. Dadurch wird die Rechenzeit deutlich verkürzt, da statt einer $N \times M$ Matrix nur noch ein Vektor pro Durchlauf berechnet werden muss.

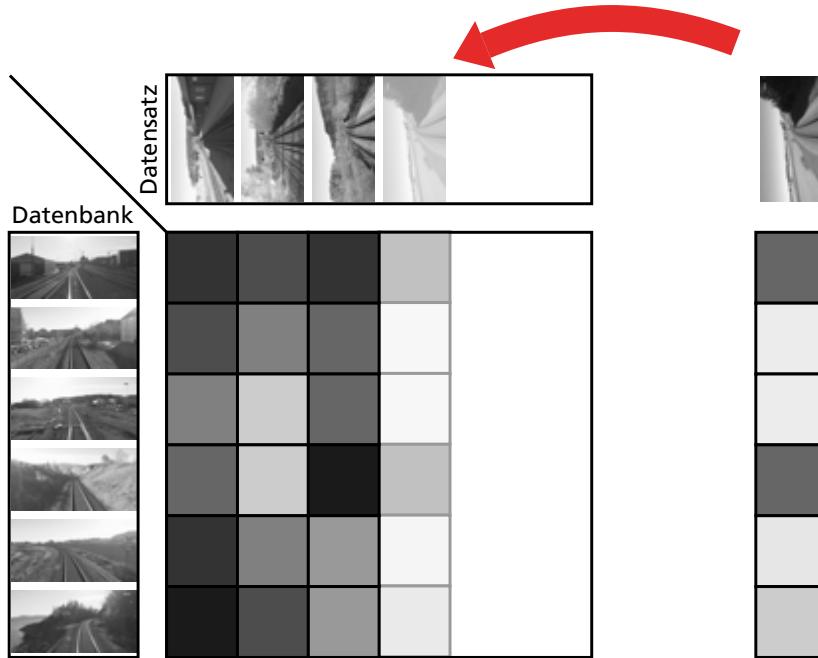


Abbildung 7: Darstellung der dynamischen Matrixerzeugung. Der Vektor wird direkt nach der Bildaufnahme berechnet und normalisiert. Die Differenzenmatrix wird während der Laufzeit nach und nach erstellt.

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. Daher wurde über verschiedene Methoden nachgedacht, welche eine solche Synchronisation der beiden Aufnahmen ermöglicht.

3.1.2 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 relativ hohen Aufnahmeraten und bekannten Geschwindigkeiten während der Aufnahme der Datenbank und des Datensatzes, ist die zeitliche Synchronisation jedoch 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 relativ 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 getriggert.

Dieses Verfahren liefert zwar keine ground-truth-Daten, es ist jedoch in den meisten Fällen ausreichend, um die Aufnahmepositionen miteinander zu synchronisieren, sofern 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 wäre 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, die den gesamten Aufbau unnötig kompliziert machen würde. Die vorherigen Methoden würden mit der bordeigenen Sensorik funktionieren.

Die Vor- und Nachteile dieser Methoden 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.

Von einer externen Synchronisation wird komplett abgeraten, da die benötigte Sensorik und deren Einbindung extrem hohen Aufwand und zusätzliche Kosten bedeuten würde. Zudem werden optische Navigationsverfahren gerade in Gegenden benötigt, wo solch eine externe Lokalisierung nicht möglich ist.

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

Art der Synchronisation	Startplatz-unabhängig	Geschwindigkeits-unabhängig	mit bordeigener Sensorik
zeitliche Synchronisation	X	X	✓
odmetrische Synchronisation	X	✓	✓
externe Synchronisation	✓	✓	X

3.2 Dynamische Ermittlung der Sequenzlänge

Wie bereits erwähnt, dokumentierten Milford und Wyeth in ihrer Arbeit, dass mithilfe des SequenceSLAM eine Matching-Quote von ca. 33% erreicht werden konnte, ohne dass dabei falsche Lokalisierungen stattfanden. Obwohl dieses Ergebnis in Hinblick auf die großen Unterschiede zwischen den Aufnahmen erstaunlich war, wird für eine erfolgreiche Navigation eine deutlich höhere Lokalisierungsrate benötigt.

Die entscheidenden Parameter des SequenceSLAM, die die Lokalisierungsergebnisse am meisten beeinflussen, sind hierbei die Länge der verwendeten Sequenz und das Vergleichsverfahren. Hierzu haben bereits Sünderhauf et al. in ihrer Arbeit ausführliche Experimente mit verschiedenen Parametereinstellungen durchgeführt [3].

Dabei fanden sie heraus, dass eine längere Sequenz auch eine höhere Lokalisierungsrate bedeutete. Dafür wurden die Lokalisierungen ungenauer, wodurch auch einige falsche Ergebnisse gefunden wurden. Zudem wurden die Ergebnisse ab einer Länge von mehr als 100 Bildern wieder schlechter, da nun die Sequenzen zu unspezifisch waren. Ein weiterer Negativpunkt war die deutlich erhöhte Rechenzeit im Vergleich zu den kleineren Sequenzen.

Um die Vorteile beider Sequenzlängen 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. Ist es dahingegen zu schlecht, 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 nicht allzu sehr vergrößert wird.

Um zu überprüfen, ob dieses Verfahren eine Verbesserung zum originalen SequenceS-LAM darstellte, wurden anschließend Versuche durchgeführt. Dabei waren folgende Faktoren entscheidend:

- Können mehr Orte korrekt lokalisiert werden?
- Werden mehr „false positives“ lokalisiert?
- Erhöht sich die Laufzeit des Algorithmus?

3.2.1 Versuchsbedingungen

Zum Testen wurde der sogenannte "Nordland-Datensatz" verwendet. 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 auf der Seite <http://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/> 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. Diese bereits synchronisierten und vorverarbeiteten Bilder wurden von ihm für weitere Tests ebenfalls online hochgeladen (<https://svn.openslam.org/data/svn/openseqslam/trunk/datasets/>).

Wie bereits erwähnt eigneten sich diese Datensätze, um die Robustheit gegenüber jahreszeitlichen, aber auch gegen Helligkeitsschwankungen zu testen. So sind die Bilder im Winter aufgrund des Schnees deutlich heller, als zum Beispiel im Herbst. Allgemein

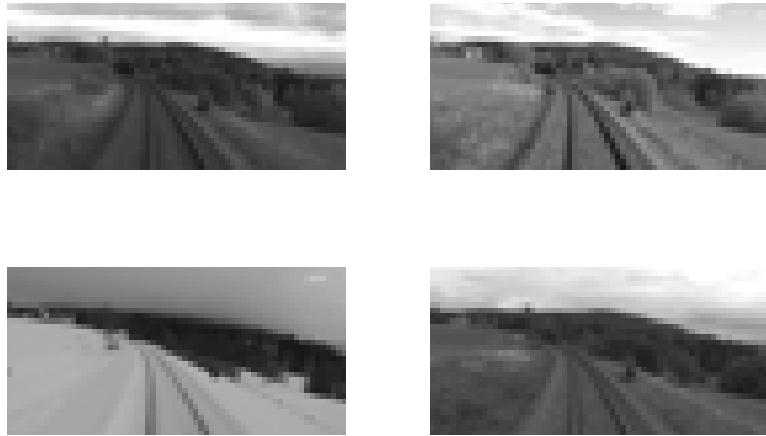


Abbildung 8: einige Beispielbilder aus dem Nordland-Datensatz. Jedes Bild zeigt den gleichen Ort zu unterschiedlichen Jahreszeiten: Oben links im Herbst, Oben rechts im Frühling, unten links im Winter und unten rechts im Sommer.

sind die gleichen Orte in den vier Datenbanken teilweise so unterschiedlich, dass ein reiner Grauwertvergleich beinahe keine Ähnlichkeit mehr ermitteln kann.

Für den Test wurden jeweils 350 Bilder aus den Datensätzen „Sommer“ und „Winter“ verwendet. Der Abstand zwischen den einzelnen Bildern lag bei 100 Datenbankbildern. Der Algorithmus wurde dabei mit c++ geschrieben und auf einem 32 Bit SUSE Enterprise-Linux-System angewendet. Für die Tests war bereits die live-Berechnung der Differenzenmatrix implementiert worden. Der erste Durchlauf fand ohne die dynamische Anpassung der Sequenzlänge statt, der zweite Versuch mit ihr. Ansonsten wurden keine Änderungen am Algorithmus vorgenommen. Gemessen wurde neben der Laufzeit auch, wie viele Bilder lokalisiert werden konnten und wie viele nicht. Bei den lokalisierten Orten wurde zudem noch zwischen korrekten und falschen Ergebnissen unterschieden. Ein Ergebnis galt als falsch, wenn der gefundene Datenbankindex mehr als eine Stelle zum gesuchten Index verschoben war.

3.2.2 Versuchsergebnisse

Beim ersten Durchlauf ohne Anpassung der Sequenzlänge wurde imemr über eine Sequenz von 10 Bilder gematcht. Dabei konnten insgesamt 137 Bilder aus dem Winter-Datensatz dem Sommer-Datensatz korrekt zugeordnet werden, wobei keine falschen Lokalisierungen auftraten. Insgesamt wurden somit 213 Datenbankbilder nicht lokalisiert. Damit lag die Matching-Quote bei 39,1%, was ungefähr den Ergebnissen aus dem originalen SequenzSLAM-Paper entsprach. Die Gesamlaufzeit vom Starten des Algorithmus bis zum Überprüfen des letzten Bildes und der Ausgabe des Ergebnisses lag bei ca. 5,5 Sekunden.

Tabelle 3: Matchingergebnisse mit und ohne dynamischer Sequenzlänge

	korrekte Matches/ inkorrekte Matches	Durchschnittliche Laufzeit pro Bild
Durchlauf ohne dynamische Se- quenzanpassung	137/0	15,7ms
Durchlauf mit dynamischer Se- quenzanpassung	271/0	23,7ms

Im Gegensatz dazu wurden beim zweiten Durchlauf mit der dynamischen Sequenzanpassung 271 Bilder korrekt lokalisiert, wobei ebenfalls keine falschen Lokalisierungen vorkamen. 79 Bilder konnten dementsprechend nicht zugeordnet werden. Somit lag die Matching-Quote bei 77,4%. Die Gesamtaufzeit des Algorithmus wurde hier mit 8,3 Sekunden gemessen.

Anhand des durchgeföhrten Experiments kann gezeigt werden, dass die dynamische Sequenzanpassung die Lokalisierungsergebnisse deutlich verbessert. Es können beinahe doppelt so viele Matches gefunden werden, wobei die Rate an falschen Matches nicht verschlechtert wird. Allerdings wurde die Laufzeit um ca. 3 Sekunden erhöht, was beinahe 60% mehr Laufzeit entsprach. Pro gematchtem Bild war diese trotzdem noch sehr gut im Vergleich zu anderen Algorithmen.

3.3 Sequenzbildung

Ein weiterer Aspekt, der die Live-Anwendung des Algorithmus behinderte, war die mathematische Berechnung der matching-Geraden. Diese hatten ihren Ausgangspunkt immer die Anzahl Bilder einer Sequenz in der Vergangenheit. Das bedeutet, dass nicht das aktuelle Bild, sondern ein schon längst vergangenes lokalisiert wird.

$$\vec{g}_i = (n + v * i) + (m - 10 + i) * D_{max} \quad (5)$$

Dadurch können jedoch ungenaue Ergebnisse entstehen, was anhand folgender Abbildung verdeutlicht werden soll. Um solchen Fehlern vorzubeugen, wurde der Startpunkt, ähnlich wie in [4] auf das aktuelle Bild gelegt und die Gerade von dort in die Vergangenheit gebildet. Dadurch ergibt sich folgende Formel für die jeweiligen Gera-

den.

$$\vec{g}_i = (n - v * i) + (m - i) * D_{max} \quad (6)$$

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 Anzahl der Bilder in der Sequenz. Auf diese Weise wird ein Vektor mit den Indizes der Differenzenwerte in der Matrix berechnet. Da diese jedoch immer natürliche Zahlen sein müssen, werden die ermittelten Ergebnisse noch zur 0 hin gerundet. 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.

3.4 Umgang mit sich selbst wiederholenden Routen

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

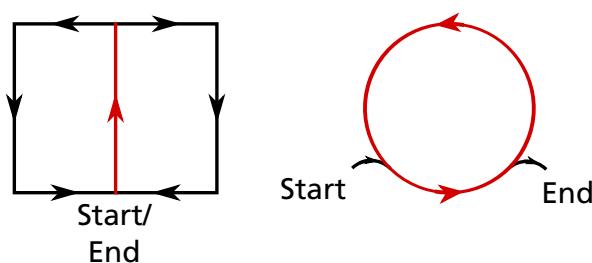


Abbildung 9: Beispielhafte Darstellung zweier Routen, bei denen Streckenabschnitte mehrfach nachgefahren werden. Beim linken Beispiel wird der mittlere Streckenabschnitt mehrfach passiert, beim rechten wird der gesamte Kreis mehrfach umfahren, bevor das Fahrzeug beim Endpunkt ankommt.

Der ursprüngliche SequenceSLAM kann mit solchen mehrfach vorkommenden Streckenabschnitten nicht umgehen. Sind zwei gleiche Orte in der Datenbank vorhanden

können dementsprechend auch zwei gleich gute Ergebnisse beim Matching gefunden werden. Während eines der beiden Ergebnisse als bester Match gewertet wird, wird das andere als zweitbeste Match deklariert. Die anschließende Überprüfung würde daher einen Wert nahe 1 ergeben, wodurch die Lokalisierung nicht gewertet wird.

Zur besseren Visualisierung wurde die Fahrt der linken Route in Abbildung 10 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.



Abbildung 10: Aufteilung der linken Route in einen Zeitstrahl. Die sich wiederholenden Bereiche sind grün markiert.

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, wie oft er bereits den jeweiligen Abschnitt passiert hat. Nur 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 11: 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 eines bestimmten 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 12: Zeitstrahl der Strecke mit rot markiertem Suchbereich.

Für diese Arbeit wurde die zweite Methode verwendet. Dies liegt vor allem daran, dass dabei keine aufwendige Bearbeitung der Datenbank vor dem ersten Durchlauf nötig

ist. Daher funktioniert der Algorithmus auch relativ schnell mit jeder beliebigen Route mit sich wiederholenden Streckenabschnitten. 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. Aufgrund dessen, dass das Fahrzeug einer fest vorgegebenen Route folgen soll, kann die nächste Position in der Datenbank relativ genau eingegrenzt werden. Um jedoch Abweichungen von der erwarteten Position abzufangen wird dieser Bereich vergrößert. Zudem werden noch Vergleichssequenzen benötigt, um die Güte des Matchings zu berechnen.

Der Suchbereich darf jedoch auch zu groß sein, da sonst die Vorteile der Eingrenzung wieder verloren gehen. Genaugenommen darf die Anzahl der Bilder innerhalb der Grenzen nicht größer sein als der minimale Abstand zwischen zwei sich wiederholenden Orten in der Datenbank.

Der Matchingvorgang wird dabei nicht verändert. Zuerst wird das beste Ergebnis im Bereich gesucht und danach der Quotient mit dem zweitbesten Ergebnis ermittelt. 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 Routenabschnitte, die ähnlich aussehen wie der eigentliche Ort, an dem sich der Rover befindet, nicht mehr miteinbezogen und verfälschen so nicht mehr die Lokalisierungsergebnisse. Ein weiterer Vorteil ist die nun verringerte Rechenzeit. Anstatt über die gesamte Datenbank den korrekten Ort zu lokalisieren, ist der Abschnitt der Datenbank nun immer gleich klein. Somit kann gerade bei sehr großen Datenbanken von 1000 Bildern oder mehr die Laufzeit deutlich verringert werden.

3.5 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 eliminieren. Trotzdem wäre es von Vorteil, wenn der Himmel keinen so starken Einfluss mehr auf die Lokalisierungsergebnisse hat.

Dafür haben Pepperel et al. das sogenannte „sky blackening“ eingeführt [5]. 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.

3.5.1 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 [6].

$$C = -1,16 * R + 0,363 * G + 1,43 * B - 82,3 \quad (7)$$

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. Die Pixelwerte liegen dabei in einem Bereich von $-378,1$ bis $374,915$.

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 13b). Anschließend folgt die Segmentation des Himmels.

3.5.2 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 alten Grauwerte beibehalten. Die Binärschwelle der Maske wird dabei automatisiert und angepasst an jedes einzelne Bild bestimmt. Das wohl bekannteste Verfahren

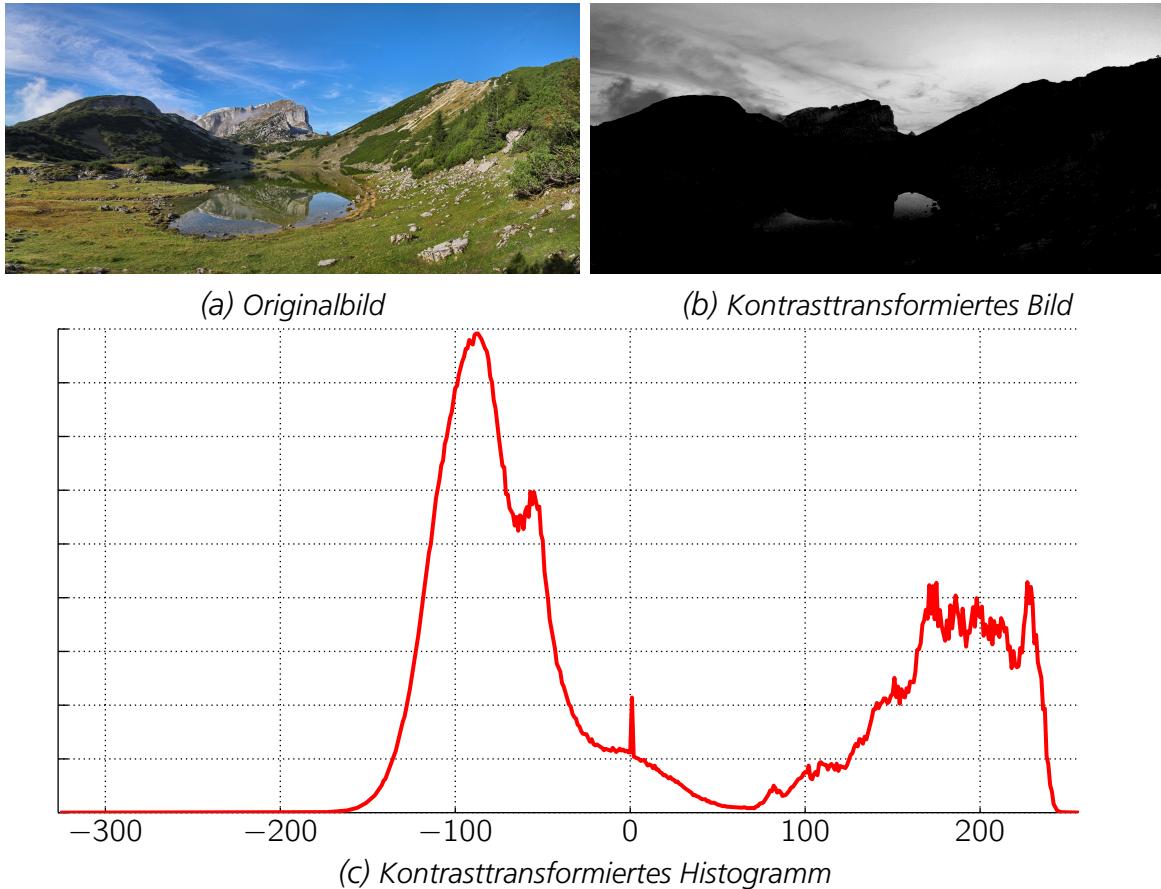


Abbildung 13: Kontrasttransformation einer Landschaftsaufnahme. (a) zeigt das Originalbild, (b) das transformierte Bild und (c) das Histogramm des Transformationsbildes. Das Bild stammt von <https://commons.wikimedia.org/w/index.php?curid=1262156> (fotografiert und hochgeladen von Karsten Dörre)

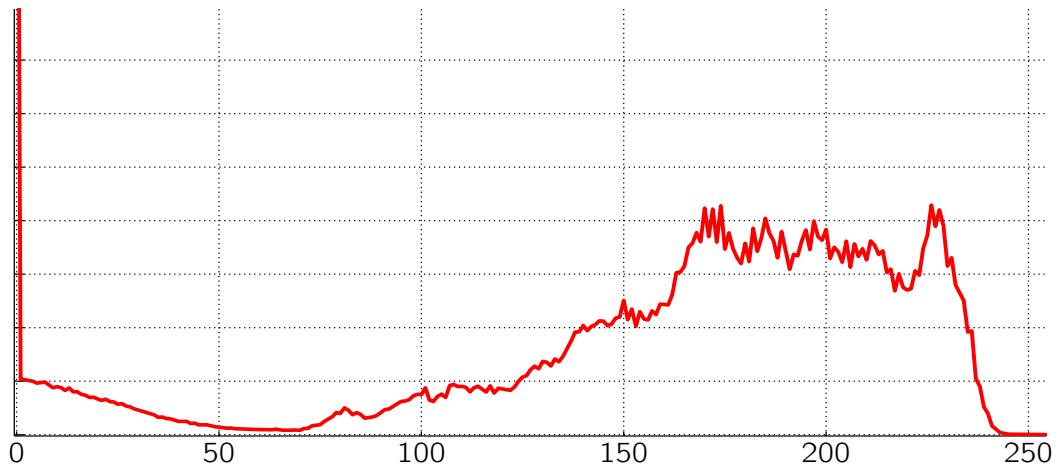


Abbildung 14: Das Histogramm nach der Transformation auf den 8-Bit Grauwertbereich. Alle Pixel, deren Werte vorher kleiner als 0, sind nun genau auf der Nullachse. Daher verlässt das Histogramm an dieser Stelle den Wertebereich.

zur Berechnung des Schwellwerts ist die sogenannte Otsu Methode [7], die im folgenden kurz beschrieben werden soll.

Anschließend wird ein Schwellwert in das Histogramm gelegt, wodurch das Histogramm in zwei Klassen unterteilt wird. Für jede der beiden Klassen wird anschließend die gesamte Wahrscheinlichkeit, dass ein Pixel in dieser Klasse auftritt, und deren mittlerer Grauwert berechnet.

$$P_0(t) = \sum_{i=0}^t p_i \text{ und } P_1(t) = \sum_{i=t+1}^{255} p_i = 1 - P_0(t) \quad (8)$$

$$\bar{g}_0 = \frac{1}{P_0(t)} \sum_{i=0}^t p_i * i \quad (9)$$

$$\bar{g}_1 = \frac{1}{P_1(t)} \sum_{i=t+1}^{255} p_i * i \quad (10)$$

Dabei ist t der aktuelle Schwellwert, P_0 und P_1 sind die aufaddierten Häufigkeiten der beiden Klassen und $p(i)$ ist die Häufigkeit eines einzelnen Grauwerts. Das Ziel der Otsu Methode ist nun herauszufinden, bei welchem Schwellwert t die Varianz innerhalb der Klassen minimal ist.

$$\sigma_i^2(t) = P_0(t)\sigma_0^2(t) + P_1(t)\sigma_1^2(t) \quad (11)$$

Bei σ_0 und σ_1 handelt es sich um die Varianzen innerhalb der beiden Klassen.

$$\sigma_0^2 = \frac{1}{P_0} \sum_{i=0}^t (i - \bar{g}_0)^2 * p_i \quad (12)$$

$$\sigma_1^2 = \frac{1}{P_1} \sum_{i=t+1}^{255} (i - \bar{g}_1)^2 * p_i \quad (13)$$

Otsu hat jedoch gezeigt, dass die Stelle, an der die Varianz innerhalb der Klassen minimal ist, gleichzeitig auch die Stelle ist, an der die Varianz zwischen den beiden Klassen maximal ist. Da die Varianz zwischen den Klassen deutlich einfacher zu berechnen ist und auch programmatisch leichter umgesetzt werden kann, wird für die Otsu Methode meistens nur noch diese berechnet und nach dem maximalen Wert gesucht.

$$\begin{aligned} \sigma_z^2(t) &= P_0(\bar{g}_0 - \bar{g}_{ges})^2 + P_1(\bar{g}_1 - \bar{g}_{ges})^2 \\ &= P_0 P_1 (\bar{g}_0 - \bar{g}_1)^2 \end{aligned} \quad (14)$$

Aufgrund seiner Einfachheit und den sehr guten Ergebnissen, wird die Otsu Methode sehr oft für automatisierte Schwellwertdetektion eingesetzt. Der Nachteil besteht

jedoch darin, dass sie nur bei bimodalen Grauwertverteilungen angewendet werden kann. Sollten jedoch nur sehr kleine Bereiche des Bildes dem Himmel oder dem Boden angehören, wird die vorherige Verteilung der Grauwerte annähernd unimodal. Daher wurde anstatt der Otsu Methode die sogenannte valley emphasis Methode angewendet. Hierbei sind alle Rechenschritte bis zu Gleichung 14 gleich. Bei der Berechnung der maximalen Varianz zwischen den Klassen wird jedoch noch eine Gewichtung W hinzugefügt [8].

$$W = (1 - p_t) \quad (15)$$

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.

$$\sigma_z^2 = W * P_0 P_1 (\bar{g}_0 - \bar{g}_1)^2 \quad (16)$$

Wie zuvor wird nun nach dem Schwellwert gesucht, bei dem die gewichtete Varianz zwischen den Klassen maximal wird. Dieser wird anschließend zur Berechnung der Bildmaske verwendet. Dieses Verfahren kann jedoch nur bei Tagesaufnahmen. 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, kann es ebenso nicht angewendet werden, da ansonsten Bildinformationen unnötig verloren gehen.

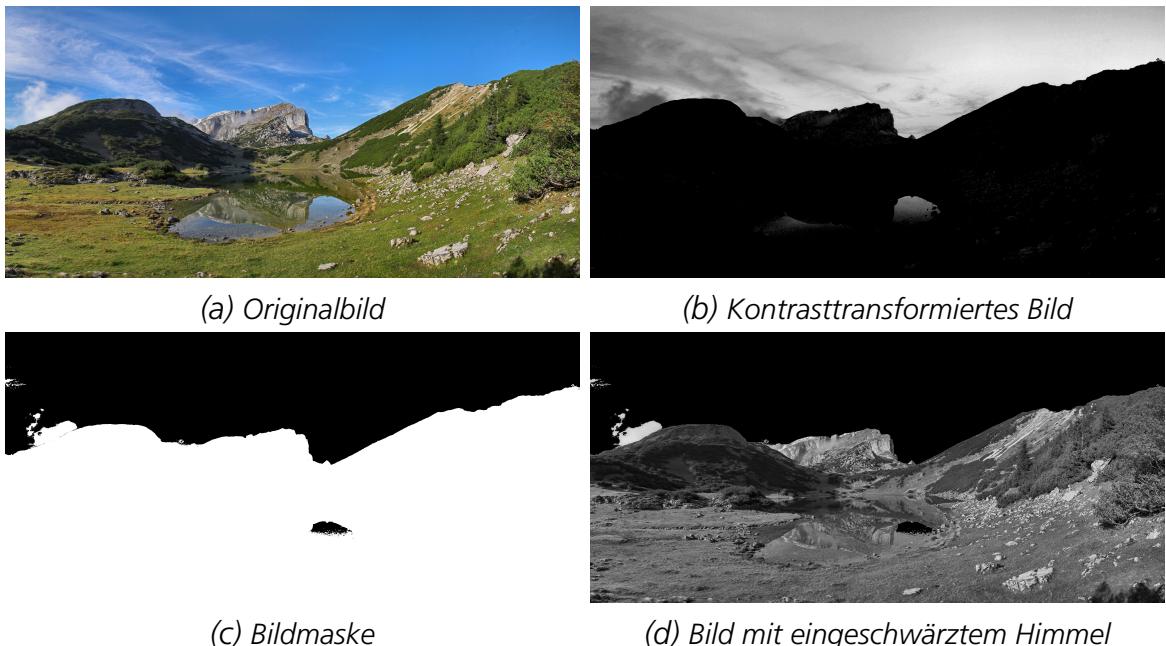


Abbildung 15: Die vier Transformationsschritte vom Originalbild (a) bis zum Grauwertbild mit eingeschwärztem Himmel (b)

Durch das Verfahren wird relativ zuverlässig der Himmel vom restlichen Bild separiert und eingeschwärzt. In obigem Beispiel 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 jedoch nur einen kleinen Anteil des Bildes aus, weswegen ihr Einfluss auf das matching-Ergebnis nicht allzu hoch ist. 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.

Was muss in diesem Kapitel behandelt werden? welche Änderungen sollten Erwähnung finden?

- Anpassung an die Live-Anwendung durch dynamische Matrix-Erzeugung → Nacheinander Differenzenvektoren erzeugen und daraus eine Matrix bilden ✓
- Änderung der Matching-Geraden (statt 10 Bilder in der Vergangenheit zu matchen, wird jetzt das aktuelle Bild gematcht) ✓
- Feste Speicherallokierung für die Matrix (Festlegen der gespeicherten, maximalen Sequenzlänge) ✓
- variable Anpassung der Sequenzlänge (wenn kein Match gefunden wird, wird die Sequenzlänge nach und nach erhöht, um vielleicht doch noch einen Match zu finden) ✓
- Installieren einer Omni-Kamera um die Blickwinkelabhängigkeit zu reduzieren (Wird in biologisch inspirierte Navigation verschoben) ✗
- sky-blackening hinzufügen, um Bilder einander anzulegenen ✓
- Suchbereich in der Matrix nach gefundenem Match einschränken (Risiko falsche Matches zu finden wird deutlich reduziert) ✓
- Ändern der Aufnamefrequenz bei fehlgeschlagener Lokalisierung ✗

4 Biologisch inspirierte Navigation

Für die autonome Navigation eines Fahrzeugs 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.

Normalerweise werden für eine optische Navigation 3D-Informationen oder sehr hoch auflösende 2D-Kameras benötigt. Aufgrund der hohen Komplexität der derart gewonnenen Daten und den sehr aufwendigen Umrechnungen, steigt die Rechenleistung auf ein beinahe nicht mehr zu bewältigendes Pensum an. Dadurch müssen entweder große Computer auf dem Fahrzeug mitgeführt werden, was das Gewicht steigert 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.

Im Lauf der Jahre haben Lebewesen jedoch zuverlässige und gleichzeitig einfache Verfahren zur Lösung dieses Problems entwickelt. Insekten wie die Ameise können beispielsweise Navigationsaufgaben sehr effizient lösen, für die mit modernster Technik Supercomputer benötigt werden würden. Dafür benutzen sie unter anderem optisch gewonnene Informationen. Diese Tatsache ist besonders beeindruckend, wenn man bedenkt, dass die Augen einer Ameise relativ schlecht auflösende Bilder bereitstellen und die Gehirnkapazität im Vergleich zu anderen Lebewesen sehr gering ist. Werden diese Tatsachen auf eine technische Ebene transformiert, bedeutet das eine sichere und zuverlässige Navigation mit niedrig auflösenden Bildern, bei gleichzeitig sehr geringem Speicher- und Rechenbedarf.

Diese Vorteile haben bereits viele Untersuchungen angeregt, deren Ziel eine technische Umsetzung des Navigationsalgoritmus' von Ameisen ist. Nachdem zuerst kurz die Vorgehensweise des biologischen Originals beschrieben wurde, soll die für den SequenceSLAM angepasste Implementierung ausführlich erläutert werden.

4.1 Navigation der Ameise

Ameisen besitzen, wie die meisten anderen Insekten einen dreigeteilten Körper, wobei der Kopf kein eigenes Drehgelenk besitzt. Das bedeutet, dass die Ameise immer in die Richtung blickt, in die sie sich gerade bewegt. Zudem besitzen sie Facetten- oder

sogenannte Komplexaugen, die einen Blickwinkel von beinahe 360° ermöglichen. Die Auflösung dieser Bilder ist jedoch, wie bereits erwähnt, vergleichsweise gering.

Für den ersten Lauf zu einer Futterquelle markieren sich die meisten Ameisenarten die gelaufene Route mit einer Pheromonspur. Jedes Mal, wenn sie anschließend die Strecke wiederholt, folgt sie dieser Spur und merkt sich parallel die Umgebungsbilder. So kann sie auch noch nach mehreren Tagen, wenn längst keine Duftstoffe mehr ihren Weg markieren, diesen sehr präzise anhand der optischen Informationen wiederfinden.

Dabei nutzt sie, dass ihr Kopf, und damit das Bild der Umgebung, immer in die aktuelle Bewegungsrichtung zeigt. Um herauszufinden, in welche Richtung sie nun weiterlaufen muss, dreht sie ihren Körper ein wenig nach links und nach rechts. Anschließend entscheidet sie in welche Richtung das Bild am ähnlichsten mit einem der gemerkten Bilder aus der „Datenbank“ war und setzt ihren Weg dorthin fort.

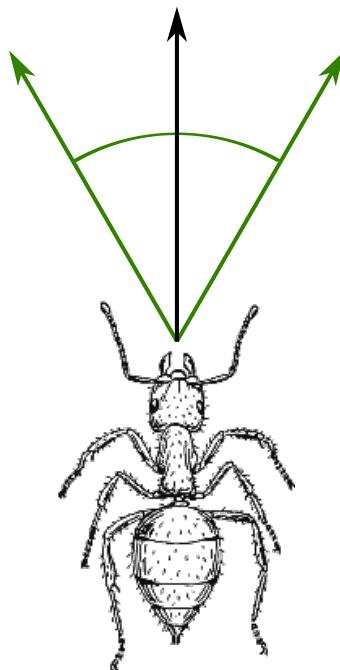


Abbildung 16: Schematische Darstellung der Wegfindung von Ameisen. In diesem Beispiel wird aus drei Richtungen die weitere Bewegungsrichtung der Ameise gewählt.

Ein weiterer Aspekt ist das Erreichen des Endpunktes. So kann es durchaus einmal passieren, dass die Ameise an ihrem Nest oder dem Futterplatz vorbeiläuft. Um dies und eine damit einhergehende Orientierungslosigkeit zu verhindern, läuft sie den Endpunkt einmal aus allen Himmelsrichtungen ab. Dabei merkt sie sich die Umgebung aus allen möglichen Blickwinkeln, um so zum Nest zurückzufinden, sollte sie einmal vorbeigelauft sein [9].

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 perfekt 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, sollten einmal Abweichungen vorkommen. Die genaue Implementierung in den Algorithmus wird im Folgenden Kapitel beschrieben.

4.2 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 das Fahrzeug selbst oder eine Pan-Tilt Einheit unter der Kamera gedreht werden mussten, um die Umgebung aufzunehmen. Das kostet Zeit und macht den Algorithmus undynamisch, da das Fahrzeug an jedem Aufnahmepunkt stehen bleiben müsste, um die weitere Bewegungsrichtung zu ermitteln.

Aufgrund der Empfindlichkeit gegenüber kleinen Verschiebungen der Kameraposition und der Blickwinkeländerung, hat der reine Grauwertvergleich zudem Schwierigkeiten einen guten Match bei verschiedenen Rotationen zu finden. Dies kann nur dann gelingen, wenn nach einer Rotation um einen bestimmten Winkel diese beiden Parameter in der Realität und im Datenbankbild genau gleich sind. Da solche Treffer in einem realen, physikalischen System beinahe unmöglich sind, muss vor der eigentlichen Implementierung der Navigation die Unabhängigkeit des Matchings gegenüber Positions- und Blickwinkelschwankungen sichergestellt werden.

Der Algorithmus ist dabei gegenüber allen drei Rotationen, also yaw, pitch und roll, empfindlich. Zudem haben Verchiebungen entlang der y- und z-Achse des Fahrzeugs großen Einfluss auf das Matching. Verschiebungen entlang der x-Achse des Fahrzeugs sind jedoch 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 im Vergleich zur Größe der aufgenommenen Landschaft sehr groß, können Verschiebungen entlang der x-Achse um die aktuelle Datenbankposition zu Problemen beim Matching führen. Dabei werden unter Umständen komplett andere Szenerien aufgenommen, die keine Ähnlichkeit zu einem der Datenbankbilder besitzen.

Im Gegensatz dazu können kleine Abstände zwischen den Bildern im Vergleich zur Größe der aufgenommenen Landschaft, ändern sich die Bilder bei Verschiebungen nicht so stark. Sollte die Verschiebung größer als der Abstand zwischen zwei Bildern sein, ist der beste Match beim vorherigen oder nachfolgenden Bild.

Für diese Arbeit wird davon ausgegangen, dass die Abstände im Vergleich zur Größe der aufgenommenen Szenerie relativ gering sind. Daher erübrigt sich die Implementierung einer Unabhängigkeit gegenüber Verschiebungen in der x-Achse.

4.2.1 Installation einer Omni-Kamera

Die erste Maßnahme, um die Blickwinkelunabhängigkeit bei der Aufnahme zu erhöhen, war die Verwendung einer Omni-Kamera. Im Gegensatz zu herkömmlichen Kameras können mit ihr 360° Aufnahmen für den Algorithmus bereitgestellt werden. Die Bilder der Kamera werden auf eine „Kugel“ oder einen „Zylinder“ projiziert. Anschließend werden die Bilddaten mithilfe von Umrechnungen der Oberflächen in ein Rechteck in ein 2D-Bild umgewandelt. Die so gewonnenen Bilder werden dann wieder für den SequenceSLAM in einem Bildvektor gespeichert.

Aufgrund dessen, dass nun der komplette Bereich um die Kamera herum aufgenommen wird, wird das Bild komplett unabhängig gegenüber Drehungen um die z-Achse der Kamera. Weisst ein Bild eine yaw-Rotation zum ursprünglichen Bild auf, kann programmatisch eine Drehung des Bildes simuliert werden. Hierfür wird die Breite des Bildes durch 360 geteilt, damit man die Skalierung des Bildes in Pixel pro Grad erhält. Diese wird daraufhin mit der benötigten Rotation multipliziert. Die so berechnete Anzahl an Pixeln wird dann anschließend vom linken oder Rechten Rand abgeschnitten und an den gegenüberliegenden Rand angefügt. Dadurch wird bildlich gesprochen die vorher gewählte Schnittkante um einen bestimmten Winkel verschoben.

Wenn die z-Achse des Fahrzeugs und der Kamera gleich sind, sind die Bilder auch gegen Fahrzeugrotationen auf der Stelle unabhängig. Das bedeutet, dass das Verwenden einer Omni-Kamera die Empfindlichkeit gegenüber yaw-Rotationen komplett entfernt.

Desweiteren wird durch die Omni-Kamera die Abhängigkeit gegenüber Verschiebungen in der y-Achse verringert. Bei einem normalen Bild gehen Bildinformationen bereits bei sehr kleinen Verschiebungen an der einen Seite verloren, während an der anderen Seite welche hinzukommen. Gerade der reine Grauwertvergleich hat dann Schwierigkeiten, da die Positionen der zueinanderpassenden Pixel komplett verschoben sind.

Zwar besteht dieses Problem bei einer Omni-Kamera auch, 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. Das bedeutet, dass sich die Pixel in der Verschiebungsrichtung nicht so stark ändern.

Dadurch können die relativ großen Änderungen entlang der zur Verschiebung orthogonalen Bildachse durch die wenigen Änderungen entlang der Verschiebungsrichtung relativ gut ausgeglichen werden. Zudem kann auch hier das Bild wieder auf der Stelle rotiert werden. Auf diese Weise können bei kleinen Verschiebungen die Pixel wieder an- satzweise an ihre richtige Position geschoben werden. Somit werden die Unterschiede aneinander angeglichen. Anstatt stellenweise sehr große und sehr kleine Unterschiede zu haben, werden die Pixeldifferenzen überall relativ moderat. So können kleine Abstände von der zwischen der aktuellen Position und der ursprünglichen Aufnahmeposition ausgeglichen werden.

Allerdings kann durch eine Omni-Kamera nicht die Verschiebung entlang der z-Achse der Kamera ausgeglichen werden. Auch der Neig- und Schwenkwinkel haben noch einen großen Einfluss auf das Matching-Ergebnis. Bei Bodenfahrzeugen, die in Gebäuden unterwegs sind, sind diese Parameter irrelevant, da solche Verschiebungen nur in einem sehr geringen Maß auftreten. Für andere Anwendungen kann noch ein sogenannte gimbal verwendet werden, der die Kamera immer entlang der xy-Ebene ausrichtet. Dadurch hätten der Neig- und Schwenkwinkel ebenfalls keinen Einfluss mehr auf die Vergleichsergebnisse. Nur die Verschiebung entlang der z-Achse kann nicht hardwaretechnisch kompensiert werden. Da für den SequenceSLAM bisher jedoch nur Boden-Fahrzeuge verwendet wurden, kann dieser Aspekt zunächst vernachlässigt werden.

4.2.2 Einbindung der biologischen Navigation in den SequenceSLAM

Was muss in diesem Kapitel erwähnt werden? Wie ist der Aufbau?

- Navigation von Insekten anhand der Paper von Wolfgang → Was gehört dazu?

- Drehen des Kopfes und Bewegung in die Richtung mit höchster Übereinstimmung aus der Datenbank
 - Durch Verwenden einer Omni-Kamera präzisere Lokalisierung möglich
 - Omni-Kamera erhöht zudem den Abweichungswinkel
- Ermitteln der Rotationswinkel aus dem Panoramabild → Beispiele mit Landmarken an verschiedenen Positionen
 - Vorne und hinten separat matchen
 - Gesamtes Panoramabild drehen und besten match suchen
 - Vor- und Nachteile der beiden Methoden
 - Kurskorrektur on the fly, also ohne das Fahrzeug unnötig abzubrechen. Ausnahme: Abweichung von der Route ist zu groß → stehen bleiben und Fahrtrichtung korrigieren, um zu große Abweichungen zu vermeiden.

5 Versuchsaufbau und -durchführung

6 Ergebnisse und Evaluierung des Verfahrens

7 Ausblick

Quellenverzeichnis

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] E. Pepperell, P. Corke, and M. Milford, "Towards persistent visual navigation using smart," in *Proceedings of Australasian Conference on Robotics and Automation, ARAA*, 2013.
- [6] 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.
- [7] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.
- [8] H.-F. Ng, "Automatic thresholding for defect detection," *Pattern recognition letters*, vol. 27, no. 14, pp. 1644–1649, 2006.
- [9] 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.