HAW HAMBURG

BACHELORTHESIS
Franek Stark

Monokulare visuelle Odometrie auf einem autonomen Miniaturschiff

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Franek Stark

Monokulare visuelle Odometrie auf einem autonomen Miniaturschiff

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

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 02. Januar 2020

Franek Stark

Thema der Arbeit

Monokulare visuelle Odometrie auf einem autonomen Miniaturschiff

Stichworte

Visuelle Odometrie, IMU, Miniatur, Schiff, Autonome Systeme

Kurzzusammenfassung

Diese Arbeit befasst sich mit einem monokularen visuellen Odomometrie-Verfahren für autonome Schiffe. In dieser Arbeit ist der Kontext ein autonomes Miniaturschiff. Das Verfahren nimmt eine Inertial-Measurement-Unit (IMU) zu Hilfe. In der Arbeit wird das verwendete Verfahren erklärt. Um das Verfahren zu testen, wird ein Softwareentwurf vorgestellt, der das Verfahren auf dem Miniaturschiff implementiert. Bei ersten Tests hat sich heraus gestellt, dass das verwendete Verfahren teilweise noch sehr experimentell ist. Daher wurden in dieser Arbeit Anpassungen und Erweiterungen, teilweise unter Zuhilfenahme des Autors des Verfahrens, gemacht. Das Verfahren konnte unter diesen Anpassungen auf die Qualität der ermittelten Positionen und der dafür benötigte Rechenzeit getestet werden. Die Rechenleistung auf dem Miniaturschiff ist ausreichend. Die Bestimmung der Bewegungsrichtung funktioniert mit einer Genauigkeit von bis zu 10°. Die Bestimmung der Bewegungslänge funktioniert hingegen nicht. Diese Arbeit versucht die Gründe dafür zu analysieren. Im Ausblick wird eine Verbesserung vorgeschlagen.

Franek Stark

Title of Thesis

Monocular visual odometry on an autonomous miniature ship

Keywords

Visual odometry, IMU, Miniature, Ship, Autonomous systems

Abstract

This thesis concerns a monocular visual odomometry method for autonomous ships. In this work the context is an autonomous miniature ship. The method uses an Inertial-Measurement-Unit (IMU). The used method is explained. To test the method a software design is presented which implements the method on the miniature ship. During first tests it turned out that the used method is partly still very experimental. Therefore, in this thesis adjustments and extensions were made, partly with the help of the author of the method. The procedure could be tested under these modifications regarding the quality of the determined positions and the required computing time. The computing power on the miniature ship is sufficient. The determination of the direction of movement works with an accuracy of up to 10°. The determination of the length of movement, however, does not work. This work tries to analyse the reasons for this. In the outlook an improvement is proposed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

//CSTI /* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */.

autosys „Reseach lab for autonomous systems, intelligent sensors, smart mobility concepts, machine learning and embedded programming“.

BA Bundle-Adjustment.

HAW Hochschule für Angewandte Wissenschaften.

IMU Inertial-Measurement-Unit.

LK Lucas-Kanade.

LM Levenberg-Marquard.

LS least-Squares.

MAE mittlere absolute Fehler.

MLESAC Maximum-Likelihood-Estimation-Random-Sample-Consensus.

MSAC M-Estimator-Sample-Consensus.

NLLSQ Non-Linear-Least-Squares.

RANSAC Random-Sample-Consensus.

SLAM Simultaneous-Localization-and-Mapping.

VINS Visual-Inertial System.

VO Visuelle Odometrie.

vSLAM Visual-Simultaneous-Localization-and-Mapping.

1 Einleitung

An der Hochschule für Angewandte Wissenschaften (HAW) entwickelt das Forschungslabor „Reseach lab for autonomous systems, intelligent sensors, smart mobility concepts, machine learning and embedded programming“ (autosys) [Pareigis u. a., 2019] unter anderem miniatur-autonome Fahrzeuge. Es besteht eine Kooperation mit dem „Miniatur Wunderland“ [Miniatur Wunderland, 2019b] in Hamburg. Dort bewegen sich zentral gesteuert Modellfahrzeuge wie Autos, Züge und Flugzeuge im Maßstab von etwa 1:87 durch die Modellanlage. Neben der Modelllandschaft mit Straßen und Bahnstrecken, existiert ein 80 m^2 großes, 25.000 L fassendes Echtwasserbecken mit 5 cm Tidenhub [Miniatur Wunderland, 2019a]. Aufgrund von Problemen mit der über dem Wasserbecken installierten Ortungsanlage, werden die Schiffe derzeit per Hand gesteuert. Für die Miniatur-autonomen Fahrzeuge von autosys stellt das Miniatur Wunderland ein zur Realwelt originalgetreues Testumfeld dar.

1.1 Miniaturschiff

Eines der autosys Fahrzeuge ist das Schiff **MS nHAWigatora** [Burau u. a., 2019]. Dabei handelt es sich um ein Modellschiff im Maßstab 1:100. Die **nHAWigatora** befindet sich zum Zeitpunkt dieser Arbeit in einem sehr frühen Stadium der Entwicklung zu einem autonomen Fahrzeug. Bisher wurde die Schiffsplattform mit Aktorik und Sensorik ausgestattet. Verschiedene Sensoren und deren Anordnung wurden erprobt. Da es sich um ein Modellschiff handelt, wurden bewusst Komponenten im unteren Preissegment ausgewählt. Als Rechenhardware arbeitet derzeit ein Verbund aus einem `Raspberry PI 3B+` und einem `Raspberry PI (4 4GB)`. Zum Einsatz kommt das Robotik-Framework ROS [Stanford Artificial Intelligence Laboratory et al.]. Ausgestattet ist die **nHAWigatora** (siehe Abbildung 1.1) mit folgenden Sensoren:

Abbildung 1.1: Die nHAWigatora im Miniatur Wunderland

Eine Farbkamera *Basler Dart daA1920-30uc* [Basler] mit einer Auflösung von 1920$px \times$ 1080px und einer maximalen Bildrate von 30 fps ist am Heck des Schiffes auf Höhe der Brücke montiert. Um die Datenrate zu verringern, kann die Auflösung durch Binning verringert werden. Die Bildwiederholrate beträgt derzeit je nach eingestelltem Binning ca. 4 fps–20 fps. Aufgrund des langsamen Datenspeicherzugriffes werden bei diesem Sensor bei Aufnahmen je nach eingestelltem Binning, Bildkomprimierung und Dauer der Aufnahme Bildraten von ca. 1 fps–10 fps erreicht.

Ein 2D-LIDAR *Hokuyo URG-04LX-UG01* [Hokuyo] ist am Bug des Schiffes montiert. Dieses tastet mit 10 Hz eine Bereich von 240° ab. Da es sich um ein 2D-LIDAR handelt, ist dieser Bereich auf nur eine Ebene beschränkt. Durch die geringere Datenmenge, können die 10 Hz auch beim Datenschreiben erreicht werden.

Eine Inertial-Measurement-Unit (IMU) *InvenSense MPU-9250* [InvenSense] liefert mit 100 Hz Winkelgeschwindigkeiten und lineare Beschleunigungen des Schiffes. Außerdem beinhaltet diese ein Magnetometer, welches derzeit nicht genutzt wird. Auch hier werden beim Datenschreiben die 100 Hz erreicht.

11 Abstandssensoren *Sharp GP2Y0E03* [Sharp] sind rund um das Schiff montiert und messen die Distanz zu anderen Objekten in einem Bereich von 4 cm–50 cm.

1.2 Problemstellung

Für ein autonomes Fahrzeug ist das Wissen über die aktuelle Position, Ausrichtung und Geschwindigkeit unter anderem zur Regelung notwendig. Dafür kommt die Odometrie zum Einsatz. Die wohl typischste Form ist die **Rad-Odometrie**. Hierbei werden die Radumdrehungen eines Fahrzeuges gezählt. Anhand des Raddurchmessers und gegebenenfalls dem Lenkeinschlag, kann so die zurückgelegte Strecke bestimmt werden. [Aqel u. a., 2016]

Diese Form der Odometrie ist jedoch nur auf Landfahrzeugen mit Rädern sinnvoll einsetzbar. Für Wasserfahrzeuge ist eine Rad-Odometrie nicht verfügbar.

Aqel u. a. [2016] zählen alternative Systeme mit ihren Vor- und Nachteilen auf:

In **Inertialen Systemen** wird mithilfe einer IMU die Position durch Auf-integrieren der Beschleunigungen und Drehgeschwindigkeiten berechnet. Die Sensordaten der IMU müssen zur Positionsbestimmung doppelt integriert werden. Dies führt schnell zu hohen Ungenauigkeiten, da die Messfehler durch Integration aufsummiert werden. Diese können durch Genauere und entsprechend Teurere IMUs verhindert werden. Ein Vorteil ist dann, dass die Messung rein passiv und ohne Kontakt und den damit verbundenen Interferenzen zur Umwelt erfolgt. [Aqel u. a., 2016]

Lasersensoren messen die Entfernung zu statischen Objekten der Umwelt und ermittelt so die zurückgelegte Strecke. Die absolute Messung bestimmt den Abstand zur Umwelt. Um eine genaue Positionsbestimmung zu ermöglichen sind mehrere Sensoren, bis hin zum LIDAR nötig. Dies führt wiederum zu hohen Kosten. Daneben haben Abstandssensoren eine maximale Reichweite. [Aqel u. a., 2016]

Bei **GPS**-Systemen bietet eine absolute Position durch Verwendung von GPS-Satelliten. Dies ist eine absolute Positionsbestimmung frei von aufsummierten Fehlern. Allerdings sind solche Lösungen abhängig von der Verfügbarkeit der GPS-Satelliten. [Aqel u. a., 2016]

Neben den genannten Verfahren gibt es die **Visuelle Odometrie (VO)**. Hier werden die Kamerabilder der, auf einem Fahrzeug montierten, Kamera verarbeitet und die Bewegungsvektoren zwischen zwei Kamerabildern aufaddiert. Damit ähnelt dieses Verfahren der **Rad-Odometrie**. Es besitzt jedoch den Vorteil wie **Inertiale Systeme**, passiv zu sein. Darüber hinaus enthalten Bilder viele Informationen und erzielen höhere Sichtweiten als **Lasersensoren**. Eine Kamera ist daneben vergleichsweise günstig. [Aqel u. a., 2016]

Mithilfe des LIDARs auf der nHAWigatora konnten bereits Karten der Umgebung erzeugt werden. Es kommt das Simultaneous-Localization-and-Mapping (SLAM)-Verfahren von Kohlbrecher u. a. [2011] (Hector-SLAM) zum Einsatz. Durch das Rollen und Stampfen ist die Qualität an einigen Stellen sehr unterschiedlich. Daneben treten an ebenen oder sehr glatten Küstenzonen und an Stellen in Mitten des Wasserbeckens, an denen sich nicht genügend Features in Reichweite des LIDARs befinden, Probleme mit der Positionierung auf: Es wird keine oder eine falsche Bewegung erkannt.

Im Miniatur Wunderland ist keine GPS-Positionierung möglich. Wie oben beschrieben existiert darüber hinaus derzeit kein externes Tracking. Somit ist eine externe Positionierung nicht möglich.
Auf dem Schiff kommt eine vergleichsweise ungenaue IMU zum Einsatz. Diese eignet sich, aufgrund der aufsummierten Ungenauigkeiten nicht als alleinige Odometrie.

Neben den Einschränkungen in der Wahl der Odometrie existieren auf einem autonomen Miniaturschiff weitere Einschränkungen und Besonderheiten. Das Schiff rollt und stampft. Dies beeinflusst die Verfahren als Störgröße. Darüber hinaus ist die verfügbare Rechenleistung an Bord eingeschränkt. Daneben handelt es sich bei Schiffen um ein System mit einer komplexen Fahrdynamik. Es kann kein einfaches Bewegungsmodell auf Basis der Steuerbefehle berechnet werden.

1.3 Ziel dieser Arbeit

Aufgrund der genannten Punkte bietet sich für die **nHAWigatora** eine kamerabasierte Lösung an. Dieses Problem wurde auch schon in anderen Arbeiten behandelt. So stellen Terzakis u. a. [2017] eine Visuelle Odometrie (VO) für autonome Schiffe in Realitätsgröße vor. Das Verfahren ist eingeschränkt für Schiffe, die sich in Küstennähe bewegen. Mithilfe der sichtbaren Küste wird der zurückgelegte Weg ermittelt. Im Miniatur Wunderland trifft dies auf ideale Bedingungen. Die Küstenlandschaft ist detailreich und an jeder Stelle

kann das Schiff Teile der Küste sehen. Neben den Kamerabildern basiert das Verfahren auf den Orientierungen aus einer IMU. Auch diese ist an Bord der **nHAWigatora** wie oben beschrieben vorhanden.

Das Ziel dieser Arbeit ist es, das von Terzakis u. a. [2017] vorgestellte Verfahren auf der existierenden Schiffsplattform zu implementieren. Dazu muss ein geeigneter Softwareentwurf implementiert werden. Darüber hinaus soll das Verfahren anhand der ermittelten Bewegungen bzw. Positionen und der dafür benötigten Rechenzeit bewertet werden. Mögliche Probleme des Verfahrens sollen identifiziert und Verbesserungen vorgeschlagen werden.

2 Grundlagen

2.1 Lochkameramodell

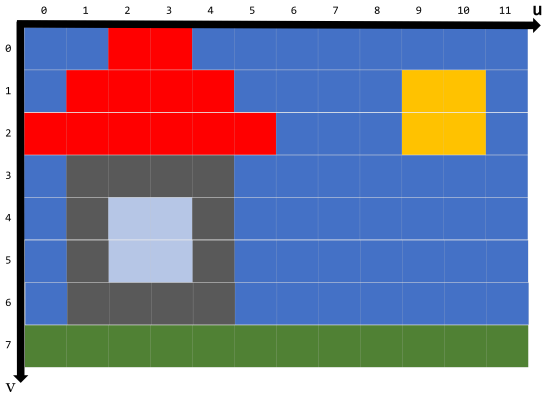


Abbildung 2.1: Die Pixelkoordinaten eines $12x8$ Pixel großen Bildes

Auf Kamerabildern werden Bildpunkte in Pixelkoordinaten beschrieben. Ein Bild mit der Auflösung $12x8$ Pixel besitzt ein Koordinatensystem wie in Abbildung 2.1 dargestellt. Zur mathematischen Betrachtung erkannter Objekte im Bild reicht das Pixelkoordinatensystem nicht aus. Im Folgenden wird beschrieben, wie mithilfe der sogenannten Kameramatrix Bildpunkte aus dem 2D-Pixelkoordinatensystem im 3D-Koordinatensystem der Kamera beschrieben werden können.

Es existieren je nach Kameratyp verschiedene mathematische Modelle [Hata und Savarese, 2017, Seite 14]. Im Folgenden soll in das Standardmodell, welches auch für die Kamera, die in dieser Arbeit genutzt wird, (nach [Hata und Savarese, 2017] und [OpenCV, 2014]) eingeführt werden.

Dabei handelt es sich um das **Lochkameramodell**. In einer Lochkamera werden die Strahlen der Umwelt nur am Loch in die Kamera gelassen. Dies ermöglicht eine Projektion

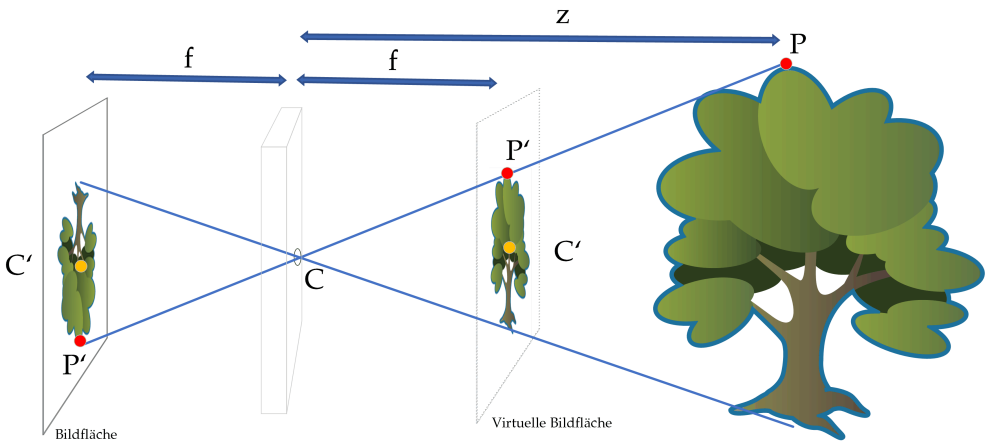


Abbildung 2.2: Lochkameramodell

der Umwelt auf die Bildfläche der Kamera. Die Abbildung 2.2 zeigt das Modell der Lochkamera. Das Loch befindet sich am Punkt C. Die **Bildfläche** liegt mit der Brennweite f vom Loch C entfernt. Es ist ebenfalls die **virtuelle Bildfläche** eingezeichnet. Diese ist identisch zur **Bildfläche** und dient später dem einfacheren Verständnis. Ein Punkt P aus der Umwelt wird an den Punkt P' in den Bildflächen projiziert. Die Umwelt ist ein dreidimensionaler Raum, während die **Bildfläche** eine Ebene ist. Dies ist eine $\mathbb{R}_3 \rightarrow \mathbb{R}_2$ Abbildung. Das Loch C wird ebenfalls an die Stelle C' projiziert.

In modernen Kameras ersetzt das Loch aufgrund der besseren Abbildungseigenschaften eine Sammellinse. Diese wird trotzdem mathematisch nach dem Lochkameramodell modelliert. Die Abbildung 2.3 zeigt eine formalere Darstellung der Lochkameraprojektion. In ihr ist zur vereinfachten Darstellung nur die **virtuelle Bildfläche** dargestellt. Der Punkt des Loches C aus Abbildung 2.2 wird hier als Fluchtpunkt F_c bezeichnet. Dieser ist der Ursprung des **Kamerakoordinatensystems** mit den Achsen X_c, Y_c und Z_c. In diesem System liegt auch der sichtbare Punkt P_k:

$$P_k = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.1}$$

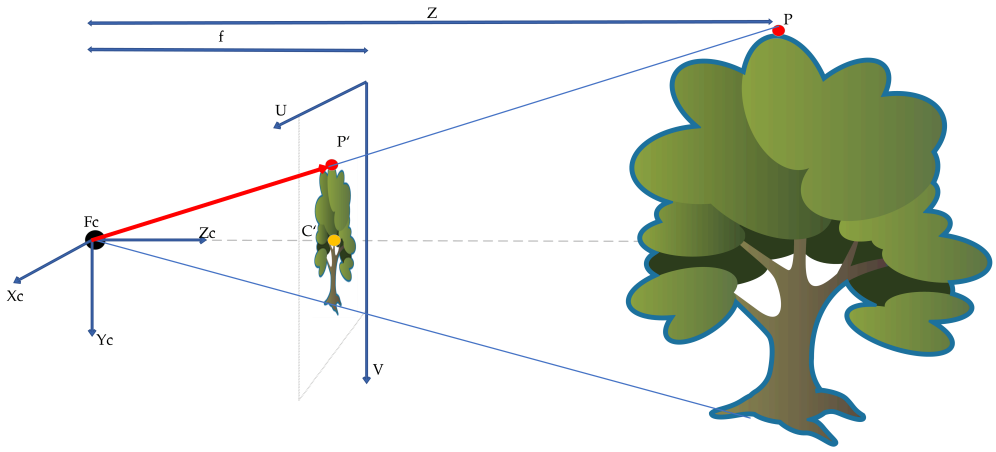


Abbildung 2.3: Formale Betrachtung der Lochkameraprojektion

Der projizierte Punkt hat in **Kamerakoordinaten** die Position P'_k:

$$P'_k = \begin{pmatrix} x \\ y \\ f \end{pmatrix} \tag{2.2}$$

Auf der **virtuellen Bildfläche** bilden die Achsen U und V das oben erklärte 2D **Pixelkoordinatensystem**. Der projizierte Punkt lässt sich in diesem System als P'_p ausdrücken:

$$P'_p = \begin{pmatrix} u \\ v \end{pmatrix} \tag{2.3}$$

Der Punkt C' lässt sich in diesem Koordinatensystem folgendermaßen ausdrücken.

$$C'_p = \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{2.4}$$

Über den Strahlensatz ergibt sich folgender Zusammenhang:

$$\begin{aligned} \frac{f}{x} &= \frac{Z}{X} \\ \frac{f}{y} &= \frac{Z}{Y} \end{aligned} \tag{2.5}$$

Die Projektion lässt sich von **Kamerakoordinaten** in **Pixelkoordinaten** umrechnen:

$$WP'_p = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} s_x x + c_x \\ s_y y + c_y \end{pmatrix} \tag{2.6}$$

Die Faktoren s_x und s_y rechnen **Kamerakoordinaten** in **Pixelkoordinaten** um. Jede Dimension besitzt ihren eigenen Faktor, da Kameras nicht immer quadratische Pixel aufweisen [Hata und Savarese, 2017, Seite 6].

Aus der Gleichung 2.5 und 2.6 lässt sich folgender Zusammenhang von Projektion in **Pixelkoordinaten** und 3D-Punkt in **Kamerakoordinaten** aufstellen:

$$P'_p = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} s_x f \frac{X}{Z} + c_x \\ s_y f \frac{Y}{Z} + c_y \end{pmatrix} \tag{2.7}$$

Mithilfe von homogenen Koordinaten lässt sich die Projektion $P \to P'_h$ als Matrixmultiplikation ausdrücken:

$$ZP'_h = Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = Z \begin{pmatrix} f\frac{X}{Z} + c_x \\ f\frac{Y}{Z} + c_y \\ 1 \end{pmatrix} = \begin{pmatrix} fX + c_x Z \\ fY + c_y Z \\ Z \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = KP \tag{2.8}$$

Hierbei bezeichnet P'_h den Punkt P' in homogenen Koordinaten. Zu beachten ist, dass die Richtungen des roten Vektors vom Ursprung F_c zum Punkt P' durch die homogenen Koordinaten P'_h angegeben werden. Dabei wird $s_x f$ zu f_x und $s_y f$ zu f_y zusammengefasst. Die Matrix K wird als intrinsische **Kameramatrix** bezeichnet:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

Mithilfe dieser Zusammenhänge können Pixelkoordinaten auch in Kamerakoordinaten zurückprojiziert werden:

$$ZP'_h = KP \tag{2.10}$$

$$\Leftrightarrow P = Z K^{-1} P'_h \tag{2.11}$$

$$\Leftrightarrow P = Z \begin{pmatrix} \frac{1}{f} & 0 & -\frac{c_x}{f} \\ 0 & \frac{1}{f} & -\frac{c_y}{f} \\ 0 & 0 & 1 \end{pmatrix} P'_h \tag{2.12}$$

$$\Leftrightarrow P \sim \begin{pmatrix} \frac{1}{f} & 0 & -\frac{c_x}{f} \\ 0 & \frac{1}{f} & -\frac{c_y}{f} \\ 0 & 0 & 1 \end{pmatrix} P'_h \tag{2.13}$$

$$\tag{2.14}$$

Im Kontext der Abbildung 2.3 kann die Richtung des roten Vektors vom Ursprung F_c zum Punkt P' in homogenen Kamerakoordinaten mithilfe der Pixelkoordinaten berechnet werden. Dieser Vektor wird **Kameraprojektion** eines Punktes oder **Punktprojektion** genannt.)

Die Kameramatrix lässt sich aus Kamerabildern bestimmen. Dies wird Kamerakalibrierung genannt. In einem späteren Teil dieser Arbeit wird das hier verwendete Kalibrierungsverfahren erwähnt.

2.2 Kameraentzerrung

Neben den Parametern in Abschnitt 2.1 erklärten Kameramatrix, existieren weitere Kameraspezifische Modellparameter in einer Kamera. Aufgrund der Abbildungsfehler einer Kamera ergeben sich Verzeichnungen in einem aufgenommenen Bild. Durch die physikalische Beschaffenheit eines Objektivs kommt es zu einer radialen Verzeichnung. Je weiter vom Mittelpunkt der Linse entfernt, desto stärker werden die Lichtstrahlen gebrochen. Abbildung 2.4 zeigt die Verzeichnungsarten einer Kamera mit Objektiv. Liegt die Bildebene bzw. der Bildsensor nicht orthogonal zur Bildachse und somit parallel zum Objektiv entsteht dazu eine tangentiale Verzeichnung. Das Bild wird geschert. [Hata und Savarese, 2017, Seite 4]

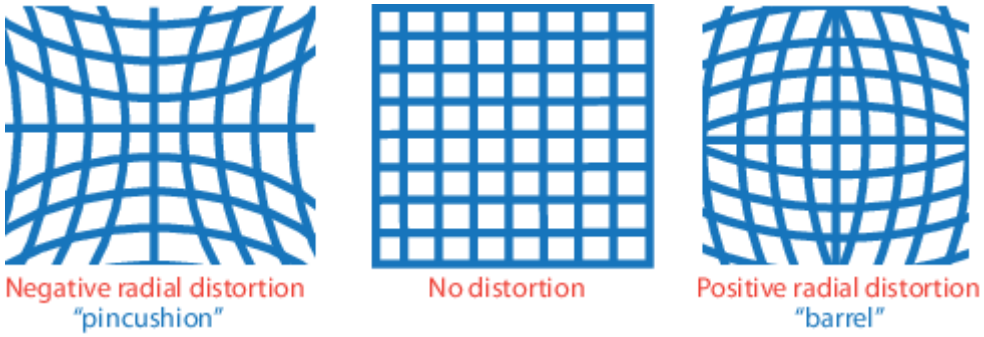


Abbildung 2.4: Die verschiedenen radialen Linsenverzeichnungen [MathWorks, Inc, 2019]

Die in dieser Arbeit genutzten Implementation aus dem Framework ROS nutzt die Implementation aus OpenCV [James Bowman, 2017]. Dabei wird die Verzeichnung durch drei radiale Koeffizienten k_1, k_2, k_3 und zwei tangentiale Koeffizienten p_1, p_2 modelliert. Folgende Funktion bildet von einem Bildpunkt (x', y') auf einen verzerrten Bildpunkt (x'', y'') ab:

$$\begin{aligned} x'' &= x'(1 + k_1 r^2 + k_2 r^4 + k^3 r^6) + 2t_1 x' y' + t_2(r^2 + 2x'^2) \\ y'' &= y'(1 + k_1 r^2 + k_2 r^4 + k^3 r^6) + 2t_2 x' y' + t_1(r^2 + 2y'^2) \end{aligned} \tag{2.15}$$

Wobei $r^2 = x'^2 + y'^2$.
Mithilfe obiger Gleichung kann bei bekannten Koeffizienten, zwischen den Bildpunkten umgerechnet werden. [OpenCV, 2014]

Auch auf die Bestimmung der Verzeichnungskoeffizienten wird in einem späteren Kapitel eingegangen.

2.3 Harris-Corner-Detector

Ecken haben sich als gute Features herausgestellt.Ein Punkt auf einer Kante hebt sich nur orthogonal zur Kante gut von seiner Umgebung ab. In der Richtung der Kante lässt sich kein großer Intensitätsunterschied der Pixel feststellen. Eine Ecke befindet sich an dem Schnittpunkt zweier Kanten. So hebt sich eine Ecke in alle Richtungen von ihrer Umgebung ab. [Sinha, 2016]

Der Harris-Corner-Detector [Harris u. a., 1988] findet in einem Schwarzweißbild Ecken. Die Idee ist auf dem Bild ein kleines Fenster zu definieren. Der Harris-Corner-Detector betrachtet die Summe der Pixelintensitäten innerhalb des Fensters. [Harris u. a., 1988]

Anhand der Intensitätsänderung bei Bewegung des Fensters können Ecken gefunden werden. Wie in Abbildung 2.5 zu sehen, ändern sich die Intensitätssumme bei einer Ecke in alle Richtungen stark. Bei einer Kante hingehen ändert sich diese nur in der Richtung orthogonal zur Kante. In einem Bereich ohne Ecken oder Kanten ergeben sich keine großen Änderungen in alle Richtungen. Mathematisch lässt sich diese Intensitätsänderung folgendermaßen ausdrücken:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{2.16}$$

Dabei ist $E(u,v)$ der Intensitätsunterschied zwischen dem Fenster an dessen ursprünglichen Position x,y und dem um u,v verschobenen Fenster. Die Summe $\sum_{x,y}$ summiert alle Pixel eines Bildes. $I(x,y)$ ist die Intensität des Pixels an der Position x,y. Die Fenster-Funktion ist $w(x,y)$. In der einfachsten Variante ist diese für Werte innerhalb des Fensters 1 und für Werte außerhalb des Fensters 0. [Harris u. a., 1988]

Ecken befinden sich an den Stellen x,y, an denen die Funktion $E(u,v)$ für kleine Verschiebungen u,v maximiert wird. Um Gleichung 2.16 zu vereinfachen, wird die Taylor-Approximation erster Ordnung genutzt. Diese approximiert den Term $I(x+u,y+v)$ der Gleichung an der Stelle x,y mithilfe dessen eigener Ableitung u,v ausreichend genug:

$$\sum_{x,y} w(x,y)[I(x+u,y+v)-I(x,y)]^2 \approx \sum_{x,y} w(x,y)[I(x,y)+u\cdot\frac{I(x,y)}{\delta x}+v\cdot\frac{I(x,y)}{\delta y}-I(x,y)]^2 \tag{2.17}$$

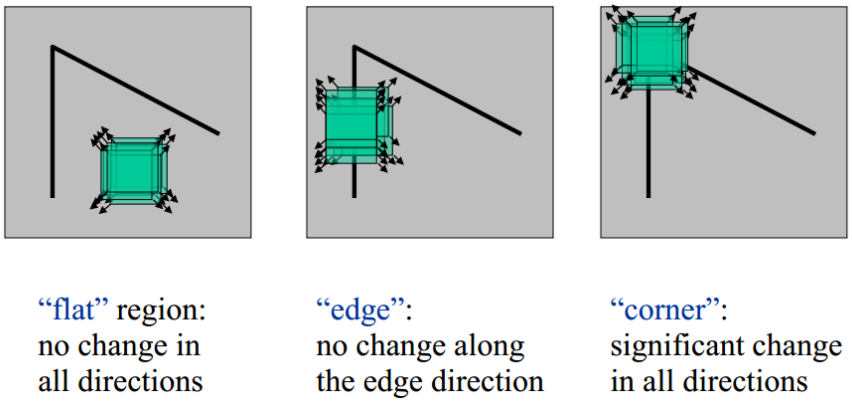


Abbildung 2.5: Das Fenster des Harris-Corner-Detectors in verschiedenen Bildbereichen [Yang, 2018]

Daraus ergibt sich dann:

$$
\begin{aligned}
&\sum_{x,y} w(x,y)[I(x,y) + u \cdot \frac{I(x,y)}{\delta x} + v \cdot \frac{I(x,y)}{\delta y} - I(x,y)]^2 \\
=&\sum_{x,y} w(x,y)[u \cdot \frac{I(x,y)}{\delta x} + v \cdot \frac{I(x,y)}{\delta y}]^2 \\
=&\sum_{x,y} w(x,y) u^2 \left(\frac{I(x,y)}{\delta x}\right)^2 + 2uv \frac{I(x,y)}{\delta x}\frac{I(x,y)}{\delta y} + v^2 \left(\frac{I(x,y)}{\delta y}\right)^2 \\
=&\begin{bmatrix} u & v \end{bmatrix} \sum_{x,y} w(x,y) \begin{bmatrix} \left(\frac{I(x,y)}{\delta x}\right)^2 & \frac{I(x,y)}{\delta x}\frac{I(x,y)}{\delta y} \\ \frac{I(x,y)}{\delta x}\frac{I(x,y)}{\delta y} & \left(\frac{I(x,y)}{\delta y}\right)^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}
\tag{2.18}
$$

[Harris u. a., 1988]

Zur Vereinfachung der Darstellung wird die Matrix M definiert:

$$
M = \begin{bmatrix} \left(\frac{I(x,y)}{\delta x}\right)^2 & \frac{I(x,y)}{\delta x}\frac{I(x,y)}{\delta y} \\ \frac{I(x,y)}{\delta x}\frac{I(x,y)}{\delta y} & \left(\frac{I(x,y)}{\delta y}\right)^2 \end{bmatrix} \tag{2.19}
$$

Für die ursprüngliche Funktion aus Gleichung 2.16 ergibt sich so für kleine Verschiebungen u, v die nachfolgende Approximation:

$$
E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.20}
$$

Über die Eigenwerte λ_1, λ_2der Matrix M lassen sich Rückschlüsse über die Qualität der Ecke an der jeweiligen Stelle ziehen. Der erste Eigenwert λ_1 gibt die Intensitätsänderung in der Richtung der stärksten Intensitätsänderung an. Der zweite Eigenwert λ_2 gibt die Intensitätsänderung der Richtung orthogonal zur stärksten Intensitätsänderung an. [Sánchez u. a., 2018] Daraus lassen sich verschiedene Fälle ableiten:

- Sind beide Eigenwerte niedrig, handelt es sich um eine flache Region.
- Ist ein Eigenwert groß und der andere klein, handelt es sich um eine Kante.
- Sind beide Eigenwerte groß, handelt es sich um eine Ecke.

[Harris u. a., 1988]

Um dies zu bewerten, führt der Harrison Corner-Detector den Corner-Response R ein:

$$R = det(M) - k(trace(M))^2 \tag{2.21}$$

$$\tag{2.22}$$

$$det(M) = \lambda_1 \lambda_2$$
$$trace(M) = \lambda_1 + \lambda_2$$

Der Parameter k wird empirisch ermittelt und liegt im Bereich $k = 0.05 - 0.06$. Der Harris-Corner-Detector berechnet die Gradienten $\frac{I(x,y)}{\delta x}$ und $\frac{I(x,y)}{\delta x}$ über das gesamte Bild. Danach wird für jedes Pixel der Corner-Responce R berechnet. Bildpunkte auf Kanten haben einen negativen Wert für R, Bildpunkte auf Ebenen einen kleinen Wert für $|R|$. Bildpunkte auf Ecken haben einen positiven Wert für R. Anhand von R werden die gefundenen Ecken ihrer Qualität nach sortiert. Entweder werden dann alle Ecken über einem Schwellwert oder die benötigte Anzahl an Ecken für die weitere Verarbeitung genutzt. [Harris u. a., 1988]

2.4 Lucas-Kanade-Feature-Tracker

Der Lucas-Kanade (LK)-Feature-Tracker [Lucas u. a., 1981] findet ein, in einem Bild gefundenes, Feature in einem anderen Bild. Dabei basiert der Tracker auf der Annahme, dass die Position des Features sich nur wenig verändert hat. Die Bilder sind schwarzweiß mit verschiedenen Intensitäten. Die Intensitäten eines Features sollten sich zwischen zwei nachfolgenden Bildern nicht zu stark verändern. [Rojas, 2010]

Lucas-Kanade Feature Tracker

Das Ziel des LK Feature-Trackers ist es, eine Bildbewegung d zu ermitteln, für die folgende Fehlerfunktion minimiert wird:

$$\epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y}^{u_y+w_y} = u_y - w_y(I(x,y) - J(x+d_x, y+d_y))^2 \tag{2.23}$$

Dabei ist $d = d_x, d_y$ die Bildbewegung. Der Intensitätswert $I(x,y)$ gibt den Bildpunkt an Stelle x, y im ersten Bild an. Der zweite Intensitätswert $J(x,y)$ ist der Bildpunkt an Stelle x, y im zweiten Bild. Die Größen w_x und w_y geben das zu betrachtenden Fenster um das Feature an. Diese Kostenfunktion betrachtet ein Fenster um ein Feature an der Stelle u_x, u_y in beiden Bildern. Je exakter die Bildbewegung zwischen den Bildern gewählt wird, desto minimaler wird der Unterschied der Intensitätswerte innerhalb der Fenster. Entscheidend ist dabei die Wahl des Fensters. Es sollte gelten $d_x <= w_x$ und $d_y <= w_y$. Um große Bewegungen zu erkennen, werden große Fenster benötigt. Große Fenster haben allerdings den Nachteil, dass Feature-Informationen verloren gehen können. [Tarasenko und Park, 2016]

Pyramidal-Feature-Tracking

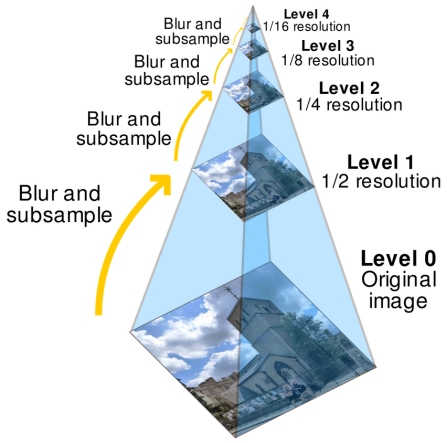


Abbildung 2.6: Eine Bildpyramide mit fünf Ebenen [Branch und Stewart, 2018, Seite 2]

Um Features sowohl bei großen als auch bei kleinen Bewegungen tracken zu können, werden aus den Bildern Bildpyramiden mit einer definierten Anzahl an Ebenen erzeugt. Abbildung 2.6 zeigt eine Beispielbildpyramide. Die Ebene $L = 0$ beinhaltet dabei das

originale Bild. In jeder weiteren Ebene wird die Auflösung verringert. [Bouguet u. a., 2001]

Der Bildpunkt I einer Ebene L an der Position x, y berechnet sich wie folgt aus der vorherigen Ebene $L-1$:

$$\begin{aligned} I^L(x,y) = & \frac{1}{4} I^{L-1}(2x, 2y) + \\ & \frac{1}{8}(I^{L-1}(2x-1, 2y) + I^{L-1}(2x+1, 2y) + \\ & I^{L-1}(2x, 2y-1) + I^{L-1}(2x, 2y+1)) + \\ & \frac{1}{16}(I^{L-1}(2x-1, 2y-1) + I^{L-1}(2x+1, 2y+1) + \\ & I^{L-1}(2x-1, 2y+1) + I^{L-1}(2x+1, 2y+1)) \end{aligned} \tag{2.24}$$

[Bouguet u. a., 2001]

Der LK-Pyramidal-Feature-Tracker beginnt mit der Suche des Features auf der tiefsten Ebene der Bildpyramide. Die daraus erhaltene Bildbewegung d^L wird auf die nächst höhere Bildebene umgerechnet. Von der Position aus wird auf dieser Ebene die Bildbewegung berechnet. Dies wird bis zu Ebene $L = 0$ wiederholt. Bei allen Bildebenen bleibt die Fenstergröße gleich groß. Als insgesamte Bildgeschwindigkeit ergibt sich dann:

$$d = \sum_{L=0}^{Lm} 2^L d^L \tag{2.25}$$

Dabei bezeichnet Lm die tiefste Ebene. [Bouguet u. a., 2001]

Bei einer maximal zu händelnden Bildbewegung von d_{max}, die eine einzelne Bildebene betrachten kann, ergibt sich so bei einer Pyramide der Tiefe Lm eine maximal zu händelnde Bildbewegung von:

$$d_{maxsum} = (2^{Lm+1} - 1) d_{max} \tag{2.26}$$

Hierdurch können auch bei relativ kleinen Fenstergrößen hohe Bildbewegungen abgedeckt werden. [Bouguet u. a., 2001]

(Zur Lösung der nicht linearen Gleichungen 2.23 in Verbindung mit Bildpyramiden siehe [Bouguet u. a., 2001])

2.5 MLESAC

Im maschinellen Sehen und anderen Bereichen wird meist die least-Squares (LS)-Methode oder ein anderes Ausgleichsverfahren genutzt, um aus mehreren Messpunkten die Modellkurve zu ermitteln. Einzelne Ausreißer, durch z. B. verrauschte Bilder oder falscher Erkennung von Features, beeinflussen dies negativ. Bei Maximum-Likelihood-Estimation-Random-Sample-Consensus (MLESAC) [Torr und Zisserman, 2000] handelt es sich um einen iterativen Algorithmus. Dieser wird eingesetzt, um aus mehreren Messungen nur die Messungen zu berücksichtigen, bei denen es sich um keine Ausreißer handelt. Diese Menge wird *Consensus-Set* genannt. Dabei handelt es sich um eine Erweiterung zu Random-Sample-Consensus (RANSAC). [Fischler und Bolles, 1981]

RANSAC

Der RANSAC-Algorithmus arbeitet wie folgt:

1. Zufällig eine minimal zum Lösen benötige Anzahl an Datenpunkten auswählen und das Modell lösen.
2. Zähle alle Datenpunkte, die dieses Modell bestätigen würden und füge diese zum Consensus-Set hinzu.
3. Wenn es sich um das Modell mit den meisten bestätigenden Punkten handelt, speichere dieses und die zugehörigen Punkte.
4. Wiederhole dies für N Schritte.

[Fischler und Bolles, 1981]

Die Bestätigung des Modells wird dabei durch eine Fehlerfunktion abgebildet. Alle Punkte, deren Fehler zum Modell unter einer Fehlerschranke liege, bestätigen dieses Modell. Das Lösen des Modells ist z. B. das lineare Berechnen eines Richtungsvektors aus getrackten Features. Die Wahl für N entscheidet über die Wahrscheinlichkeit, dass ein Consensus-Set gefunden wird, in dem keine Ausreißer existieren. Nachdem das beste Consensus-Set gefunden wurde, wird aus allen unterstützenden Datenpunkten das Modell neu berechnet. [Derpanis, 2010]

MLESAC

Das Finden von Datenpunkten, die das Modell bestätigen, entspricht in RANSAC dem Minimieren der Kostenfunktion C:

$$C = \sum_i \rho(e_i^2) \qquad \text{mit } \rho(e^2) = \begin{cases} 0 & e^2 < T^2 \\ \text{konstant} & e^2 \geq T^2. \end{cases} \tag{2.27}$$

Der Wert e^2 bezeichnet den Fehler eines Punktes i zum Modell und T^2 bezeichnet die Fehlerschranke. [Torr und Zisserman, 2000]

Bei zu hoher Fehlerschranke liefern viele Datenpunkte keinen Fehler zum Modell. So können Ausreißer schlecht erkannt werden. [Zuliani, 2011]

MLESAC erweitert RANSAC. Hier wird nicht wie in RANSAC nur die Anzahl der modellunterstützenden Datenpunkte als Indikator für das beste Modell berücksichtigt. MLESAC bewertet die Modelle anhand der Wahrscheinlichkeiten, dass die unterstützenden Datenpunkte keine Ausreißer sind. Diese Wahrscheinlichkeiten berechnen sich aus der Fehlerfunktion. Im Vergleich zu RANSAC minimiert M-Estimator-Sample-Consensus (MSAC) diese Kostenfunktion: [Torr und Zisserman, 2000]

$$C = \sum_i \rho_2(e_i^2) \qquad \text{mit } \rho_2(e^2) = \begin{cases} e^2 & e^2 < T^2 \\ T^2 & e^2 \geq T^2. \end{cases} \tag{2.28}$$

Auch wenn Datenpunkte unterhalb der Fehlerschranke liegen, wird ihr Fehler weiterhin zur Wahl des besten Modells berücksichtigt. Datenpunkte oberhalb der Fehlerschranke gehen weiterhin mit einem konstanten Wert ein. Im Detail ist MLESAC eine Erweiterung zu MSAC, die Modelle anhand einer Mischfunktion, bestehend aus den Wahrscheinlichkeiten aller Datenpunkte für dieses Modell und den Gegenwahrscheinlichkeiten der Punkte gegen das Modell, bewertet[1]. [Tordoff und Murray, 2005]

[1] Mehr dazu in Torr und Zisserman [2000].

2.6 Notation

In diesem Abschnitt soll in die in dieser Arbeit gewählte Notation eingeführt werden:

Koordinatensysteme

Die Odometrie bezieht sich relativ zum Weltkoordinatensystem. Jede Kameraposition, an der ein Bild erzeugt wird, besitzt ein eigenes Kamerakoordinatensystem. Der Ursprung liegt dabei am Fluchtpunkt[2] der jeweiligen Kameraposition. Das Koordinatensystem ist außerdem zum Weltkoordinatensystem entsprechend der Kameraausrichtung verdreht. In Kamerakoordinatensystemen zeigt die x-Achse nach rechts, die y-Achse nach unten und die z-Achse nach vorne. Zur Vereinfachung entspricht innerhalb der VO-Formeln die Ausrichtung der Achsen des Weltkoordinatensystems dem Kamerakoordinatensystem.

Rotationen

Alle Rotationen sind als Rotationsmatrix angegeben. Im Folgenden werden diese als Transformation zwischen zueinander verdrehten Koordinatensystemen interpretiert. Die Matrix R_b^a beschreibt die Rotation von dem Koordinatensystem a in das Koordinatensystem b. Eine Multiplikation mit einem Vektor v im Koordinatensystem b transformiert den Vektor in das Koordinatensystem a.

$$v_a = R_b^a v_b \tag{2.29}$$

Frame

Ein Frame F_t wird zu einem Zeitpunkt t erzeugt. Ein Frame beinhaltet das Kamerabild zu diesem Zeitpunkt. Außerdem beinhaltet er die Position P_t und aktuelle Drehung der Kamera R_t^w. Die Rotationsmatrix beschreibt dabei die Rotation des Kamerakoordinatensystems im Weltkoordinatensystem zu diesem Zeitpunkt. Die Position ist die Translation des Kamerakoordinatensystems im Weltkoordinatensystem.

[2] Siehe F_c in Abschnitt 2.1.

Feature

Ein 3D-Feature im Weltkoordinatensystem wird als $M = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ angegeben. Dieses Feature kann in den Kamerabildern unterschiedlichster Frames sichtbar sein. Im Folgenden heißt es, „Ein Feature ist im Frame sichtbar“. Die Projektion des Features im jeweiligen Frame wird als „Feature-Projektion eines Features in einem Frame“ bezeichnet. Diese wird als homogener Vektor vom Kameraursprung auf die Projektionsfläche beschrieben. (Siehe auch Abschnitt 2.1) Mathematisch wird diese Projektion so ausgedrückt:

$$m_t^i \tag{2.30}$$

Dabei bezeichnet i den Index des 3D-Features, dessen Projektion m beschreibt. Die Variable t gibt den Zeitpunkt des Frames an, in dem diese Feature-Projektion erkannt wurde. Sofern nicht anders angegeben, wird der Vektor im Koordinatensystem der zugehörigen Kameraposition zum Zeitpunkt t ausgedrückt.

Die Tiefe Z_t^i beschreibt die unbekannte Entfernung des 3D-Features zur Kamera[3]. Es gilt für das 3D-Feature im Kamerakoordinatesystem dieser Kamera:

$$M^i = Z_t^i \cdot m_t^i \tag{2.31}$$

Feature-Korrespondenz

Eine Feature-Korrespondenz werden die Feature-Projektionen desselben 3D-Features in den Kamerabildern zweier oder mehrerer Frames genannt. Mathematisch enthält die Menge $F_{t,t'}$ alle Feature-Korrespondenzen der 3D-Features, die in den Frames F_t und $F_{t'}$ sichtbar sind.

Der Term

$$i \in F_{t,t'} \tag{2.32}$$

bezeichnet alle 3D-Feature M_i, die in beiden Frames als jeweilige Feature-Projektion m_t^i und m_t^i sichtbar sind.

[3]Siehe Z in Abschnitt 2.1.

Zur Vereinfachungen könne Gleichungen generalisiert für nur eine Feature-Korrespondenz betrachtet werden, wobei dann der hochgestellte Index i weggelassen wird.

Bewegungsvektoren

Zwischen zwei direkt aufeinander folgende Frames an unterschiedlichen Positionen liegt ein Positionsunterschied. Dieser wird im Folgenden als Bewegungsvektor $b_{t,t+1}$ bezeichnet:

$$b_{t,t+1} = P_t - P_{t+1} \tag{2.33}$$

Da im Verfahren nur die Bewegungsvektoren zwischen zwei folgenden Frames gespeichert werden, setzen die Bewegungsvektoren über mehrere Frames sich aus diesen zusammen. Der Vektor $b_{t,t'}$ ist der zusammengesetzte Bewegungsvektor zwischen den Frames zum Zeitpunkt t und t' in Weltkoordinaten.

$$b_{t,t'} = \sum_{l=t}^{t'-1} b_{l,l+1} \tag{2.34}$$

3 Stand der Technik

Basierend auf den Arbeiten von Aqel u. a. [2016]; He u. a. [2019]; Shan u. a. [2016] soll in diesem Kapitel der aktuelle Stand der Technik betrachtet werden.

VO wird in vielen Systemen eingesetzt. Dabei unterscheiden diese sich in ihren Anforderungen und der daraus genutzten Technologie zur Positionsbestimmung aus Bildern.

vSLAM

Ein wichtiger Unterschied besteht dabei zwischen **Visual-Simultaneous-Localization-and-Mapping (vSLAM)** und reiner **VO**. Während die reine VO, mit der Rad-Odometrie vergleichbar, nur die Transformationen zwischen jeweils zwei Kamerabildern aufsummiert, setzt vSLAM das SLAM-Verfahren visuell um. Dabei wird zur Positionierung eine Karte aus den Kamerabildern erzeugt. Dies erfordert mehr Rechenleistung, ermöglicht dann jedoch eine globale Lokalisation. [He u. a., 2019]

Man unterscheidet dabei auch in **lokale** und **globale** Verfahren. Globale VO betrachtet in der Optimierung und je nach Verfahren auch in der Bewegungsermittlung alle bisherigen Transformationen und erkannte Features. Dies ist sehr aufwendig, erzielt aber sehr genaue Ergebnisse. Lokale VO betrachtet nur eine festgelegte Anzahl der letzten Transformationen und Feature. [He u. a., 2019]

Bei vSLAM handelt es sich, wenn eine vollständige Karte aufgebaut wird, um globale Verfahren. [He u. a., 2019]

Feature-basierte und direkte Ansätze

Es werden Ansätze basierend auf einzelnen **Features** und **direkte** Ansätze unterschieden. Direkte Ansätze arbeiten direkt auf den Pixeln des ganzen oder auf Teilen des Bildes. Dies erspart aufwendiges Erkennen und Zuordnen oder Tracken von Features und verhindert so

Fehler durch optisches Rauschen. Dafür sind direkte Verfahren, da diese das gesamte Bild betrachten, anfälliger für Abbildungsfehler im Bild. Außerdem wirken sich Veränderungen der Bilder durch geänderte Beleuchtung negativ aus. [He u. a., 2019]

Um beide Vorteile zu nutzen, existieren **hybride** Verfahren. Solche erkennen nur in einigen Kamerabildern Features und nutzen dazwischen direkte Verfahren. [Shan u. a., 2016; He u. a., 2019; Aqel u. a., 2016]

Workflow

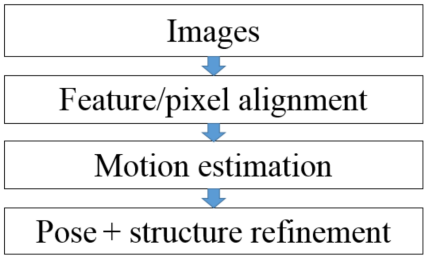


Abbildung 3.1: Genereller Ablaufs in einem VO-System [Shan u. a., 2016, Seite 1]

Abbildung 3.1 zeigt den typischen Ablauf, in dem sich alle VO-Verfahren ähneln. In aufeinander folgenden Bildern werden Features oder Pixel zugeordnet. Daraus ergibt sich eine zurückgelegte Bewegung und Rotation der Kamera zwischen zwei Bildern. In einem weiteren Schritt werden alle oder einige der Positionen optimiert. [Shan u. a., 2016]

Das Optimieren (Refinement) geschieht mithilfe einer Kostenfunktion, die alle Kamerapositionen, oft auch Kameraausrichtungen und bei vSLAM auch die 3D-Positionen der Features in der Karte als Parameter beinhaltet. [Shan u. a., 2016; He u. a., 2019]

3D-Vision

Dreidimensionale Bilder können durch **Stereoskopie** erzeugt werden. Dazu werden gleichzeitig durch zwei Kameras zwei Bilder gemacht. Der Abstand der Optiken ist bekannt, sodass auf den Bildern erkannte Features bzw. Objekte sofort trianguliert werden können. Die so erhaltene Tiefeninformation ermöglicht ein absolutes skalieren der Bewegungen zwischen zwei Positionen. [Shan u. a., 2016], [Aqel u. a., 2016]

Dazu müssen allerdings die Features oder Objekte zugeordnet werden können. Auch das weitere Tracken oder Zuordnen zu weiteren Kamerabildern muss auf jeweils beiden Bildern erfolgen. [Shan u. a., 2016]

Darüber hinaus können Features nur effektiv trianguliert werden, wenn die Entfernung zu ihnen nicht viel größer, als der Abstand der beiden Kameraoptiken ist. Andernfalls gelten die Regeln der **monokularen** Odometrie und die Stereobilder sind unnötig. [Aqel u. a., 2016; Shan u. a., 2016]

Darüber hinaus ist die Kalibrierung eines Stereokameraaufbaus sehr aufwendig und fehleranfällig. Daneben werden doppelte Daten erzeugt, mehr physischer Platz ist auf dem Fahrzeug nötig und der Preis des Aufbaus steigt. [Aqel u. a., 2016]

Neben der Nutzung von Stereoskopie existieren **Tiefenkameras**. Die Pixel enthalten neben der Farbinformation noch eine Tiefeninformation. Ein begrenzender Faktor ist hier derzeit die Reichweite und Qualität bei kompakter Bauform. [He u. a., 2019]

Monokulare VO

Die meisten Arbeiten im Bereich der VO verwenden daher einen **monokularen** Ansatz mit einer einzelnen Kamera [Aqel u. a., 2016]. Und nutzen Features [He u. a., 2019].

Dabei wird zwischen **2D-2D**- und **3D-2D**-Verfahren unterschieden. In 3D-2D-Verfahren werden neue 2D-Feature-Projektionen eines Features aus zwei aufeinanderfolgenden Bildern trianguliert. So erhält man die Position des 3D-Features. Weitere 2D-Projektionen des 3D-Features in folgenden Bildern werden diesen triangulierten 3D-Features zugeordnet. Aus diesen Zusammenhängen kann dann die Bewegung zwischen den Bildern errechnet werden. [Scaramuzza und Fraundorfer, 2011]

3D-2D-Verfahren sind meistens auch vSLAM-Verfahren. Die triangulierten 3D-Features bilden die vSLAM-Karte. [He u. a., 2019]

Im 2D-2D-Verfahren wird ausschließlich mit den Projektionen auf den Kamerabildern gearbeitet. So fällt die aufwendige Triangulation, die vor allem bei verrauschten Projektionen schlechte Ergebnisse erzielt, weg. Dafür können keine Karten aus den Positionen der 3D-Features erzeugt werden. Viele Arbeiten nutzten ein 2D-2D-Verfahren, um die Rotation und Translation zwischen den ersten beiden oder allen Kamerabildern anhand

zugeordneten 2D-Feature-Projektionen zu ermitteln. Anhand dessen können dann die ersten 3D-Features trianguliert werden. [Scaramuzza und Fraundorfer, 2011]

Um Ausreißer zu erkennen, werden mehr als die benötigen Korrespondenzen erkannt und diese Algorithmen mit RANSAC kombiniert. [Shan u. a., 2016; He u. a., 2019; Aqel u. a., 2016]

Die häufigste und bekannteste Form des Pose-And-Structure-Refinements ist in diesen Verfahren das Bundle-Adjustment (BA). Bei der BA wird der **Reprojektionsfehler** in Abhängigkeit der Kameramatrix, Kameraposition, Kamerausrichtung und der Position der triangulieren 3D-Features über alle oder die letzten N Kamerapositionen verbessert. Der Reprojektionsfehler beschreibt die Abweichung zwischen der, mithilfe der Kameraposition, Kameraausrichtung und Kameramatrix berechneten, 2D-Projektionen eines triangulierten 3D-Features und der wirklich aus dem Kamerabild aufgenommenen Projektionen des zugehörigen echten 3D-Features. Hierbei handelt es sich um ein Non-Linear-Least-Squares (NLLSQ)-Problem. [Shan u. a., 2016; He u. a., 2019]

Skalierung monokularer VO

Bei monokularer Odometrie besteht das Problem, dass die Skalierung der Bewegungen zwischen den Bildern nicht bekannt ist. [Shan u. a., 2016; He u. a., 2019; Aqel u. a., 2016]

Viele Arbeiten erkennen daher nur oder einige Features auf dem Boden, wissen den Abstand der Kamera zum Boden und können so die Skalierung berechnen. Dies erfordert einen ebenen Boden und beschränkt die Kamerabewegung auf diese Ebene. [Aqel u. a., 2016]

Andere Verfahren nutzen gemessene Längen bekannter Objekte, die im Kamerabild zu erkennen sind. Anhand dieser kann die Bewegung skaliert werden. [Aqel u. a., 2016]

In Arbeiten, in denen keine weiteren Einschränkungen vorgenommen werden können, wird auf eine absolute Skalierung innerhalb der VO verzichtet. Stattdessen wird die erste erkannte Bewegung zwischen den ersten beiden Bildern als absolut angenommen und alle Folgenden anhand dieser skaliert. In der Folge existiert eine Odometrie, die einen unbekannten Skalierungsfaktor besitzt. Dieser kann mithilfe anderer Sensoren ermittelt werden. Zum Beispiel existieren Szenarien, in denen GPS-Positionen oder Rad-Odometrie manchmal vorhanden sind, in diesen Phasen kann der Skalierungsfaktor berechnet werden. [Shan u. a., 2016]

Matching und Tracking

Es unterscheiden sich die auf Features basierenden Verfahren beim Zuordnen der projizierten Features auf verschiedenen Kamerabildern in **Tracking**- und **Matching**-Verfahren. Beim Matching werden die Features auf allen Kamerabildern separat erkannt und dann jeweils einander zugeordnet. [Shan u. a., 2016]

Beim Tracking werden die Features nur einmal neu erkannt und basierend auf der bekannten Position im folgenden Kamerabild gesucht. Vor allem bei kontinuierlichen Bewegungen mit wenig Veränderungen zwischen den Bildern werden hiermit gute Ergebnisse mit weniger Rechenaufwand als beim Feature-Matching erzielt. [Shan u. a., 2016]

Weitere Sensoren

Um die Ergebnisse der VO zu verbessern, haben einige Autoren mit dem Einsatz von anderen Sensoren zur Ergänzung des Kamerasystems experimentiert. Hervorzuheben ist dabei die IMU. Hieraus hat sich eine neue Kategorie von VO ergeben. Ein **Visual-Inertial System (VINS)** kombiniert herkömmliche VO und die Daten aus einer IMUs. [He u. a., 2019; Shan u. a., 2016]

Einige Arbeiten entwickeln ein Bewegungsmodell ihres Fahrzeuges und fusionieren die IMU mit der VO um absolut skalierte Bewegungen zu erhalten. [Shan u. a., 2016]

Maschine-Learning

Des Weiteren existieren Ansätze, in denen Teile oder die gesamte VO als neuronales Netz umgesetzt sind. Es existieren Arbeiten, in denen eine absolute Skalierung der monokularen VO so bestimmt wird. [He u. a., 2019]

3.1 Wahl des Verfahrens

Auf dem autonomen Miniaturschiff herrschen besondere Anforderungen an die VO. Da es sich um ein Modellschiff handelt, ist der Platz sehr eingeschränkt. Dies wirkt sich direkt auf die vorhandene Rechenleistung aus.

Unabhängig von der Größe existieren auf Wasserfahrzeugen weitere Besonderheiten und Einschränkungen: Die Bodenebene besteht aus Wasser. Dies bewegt sich dynamisch und unabhängig von der Bewegung des Schiffes. Somit kann die Bodenebene nicht zur Bewegungserkennung des Schiffes verwendet werden. [Terzakis u. a., 2017]

Das von Terzakis u. a. [2017] vorgestellte Verfahren ist für ein autonomes Wasserfahrzeug im Küstengebiet ausgelegt. Das Verfahren basiert auf monokularen, auf Features basierender VO. Aufgrund der Bodenfläche aus Wasser bleibt als Ort zur Erkennung der Features nur die Küste. Das Miniaturwunderland mit seiner detailreichen Modellbaulandschaft besitzt ideale Voraussetzungen für das Erkennen von guten Features.

Terzakis u. a. [2017] nennt neben dem Vorhandensein von Wasser, die große Tiefe der Szenerie als Problem von VO auf Wasserfahrzeugen. Abbildung 3.2 zeigt, dass bei einer größeren Entfernung eines Features sich die Ungenauigkeit dessen Projektion auf dem Kamerabild viel größer auswirkt, als bei einem nahen Feature. Soll ein Feature mithilfe seiner Projektionen auf zwei Kamerabildern trianguliert werden, wirkt sich der Effekt negativ aus. Die Fläche, in der das Feature aufgrund der Ungenauigkeit liegen könnte, vergrößert sich mit der Entfernung. Dies würde eine vSLAM negativ beeinflussen. Daher verwendet dieses Verfahren keine 3D-Features. Es handelt sich um ein reines 2D-2D-Verfahren, welches nur auf den 2D-Projektionen der 3D-Features arbeitet. Die Optimierung der Translationen erfolgt dabei lokal auf den letzten N Translationen und zugehörigen 2D-Projektionen. Dies erfordert, als weiteres Argument, zudem weniger Rechenleistung.

Um Ausreißer zu erkennen, kommt hier MLESAC, eine Variante von RANSAC, zum Einsatz.

Die reinen Bewegungen des Schiffes sind kontinuierlich und langsam. Daher bietet sich ein Feature-Tracking-Verfahren an. Die nahegelegenen Features legen bei gleicher Ge-

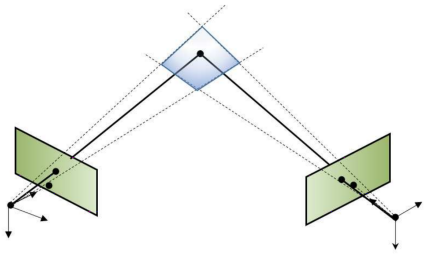


Abbildung 3.2: Fläche in der die Triangulation liegen kann [Terzakis u. a., 2017, Seite 2]

schwindigkeit eine weitere Strecke auf den Bildern der Kamera zurück, als die weit entfernten. Um auch sowohl weit entfernte, als auch nähere tracken zu können, kommt der LK-Pyramidal-Feature-Tracker zum Einsatz. Auch dies erspart Rechenleistung anstelle eines Matching-Verfahrens.

Da auf der Bodenfläche keine Features getrackt werden können und die Höhe der Kamera zum Boden durch das Schaukeln variiert, kann hierüber keine absolute Skalierung berechnet werden. Ohne Bewegungsmodell des Schiffes kann auch hieraus keine Skalierung berechnet werden. Das Verfahren verzichtet auf eine absolute Skalierung innerhalb der VO. Vielmehr ist es konzipiert, um die GPS-Odometrie bei Ausfall für Teilstrecken zu ersetzten. In den Phasen, in denen das GPS funktioniert, kann der Skalierungsfaktor ermittelt oder angepasst werden. Terzakis u. a. [2017]
Wie oben beschrieben wird auf dem Miniaturschiff zusätzlich eine LIDAR-Odometrie eingesetzt. Diese kann, wenn verfügbar, eingesetzt werden, um die Skalierung zu ermitteln.

Als weiteres Problem auf einem Schiff existiert, wie schon erwähnt, das Rollen und Stampfen. Die Kamera ist, um einen guten Sichtwinkel zu erhalten, an der Brücke am Heck des Schiffes erhöht montiert. Durch das Stampfen des Schiffes erfährt sie einen Hub, durch das Rollen durchläuft sie eine halbkreisförmige Bewegung. Um den Algorithmus zu vereinfachen, benutzt Terzakis u. a. [2017] die Orientierungsdaten der IMU. So muss nur die Translation, nicht aber die Rotation zwischen den Bildern, ermittelt werden.

4 VO-Verfahren

Im Folgenden wird in das bereits kurz beschriebene Verfahren von Terzakis u. a. [2017] eingeführt. Im Wesentlichen besteht auch dieses Verfahren aus den typischen Schritten **Feature-Alignment**, **Motion-Estimation** und **Pose-Refinement**. Als Features kommen Harris-Corners zum Einsatz. Getrackt werden diese mit dem LK-Pyramidal-Feature-Tracker. Die Motion -Estimation wird in Abschnitt 4.1 und das Refinement in Abschnitt 4.3 näher erläutert. Wie in der Diskussion zur Wahl dieses Verfahren in Abschnitt 3.1 beschrieben, arbeitet dieses Verfahren nur auf den getrackten 2D-Feature-Projektionen. Es werden keine 3D Positionen der getrackten Features trianguliert. Dazu ist dieses Verfahren auf die Orientierungsinformationen einer IMU angewiesen.

Da das Refinement bei ersten Tests auf die im Paper beschriebene Art nicht funktioniert hat, wurden unter Zuhilfenahme des Autors einige Änderungen am Refinement vorgenommen. Diese werden in Abschnitt 4.3.3 erklärt.

Daneben wurden in dieser Arbeit noch eigene Erweiterungen und Anpassungen vorgenommen. Auf diese wird in Abschnitt 4.4 eingegangen.

4.1 Baseline-Estimation

Dieser Teil berechnet aus zwei folgenden Frames einen Richtungsvektor. Dieser gibt die Richtung an, in die sich die Kamera vom vorherigen Frame zum nachfolgenden Frame bewegt hat.

4.1.1 Gleichung

Betrachtet werden zwei nachfolgende Frames. Dies beinhalten eine Anzahl von Features, welche in beiden Frames erkannt worden sind. Folgende Formel drückt die Feature-Projektion eines 3D-Features M^i in einem Frame zum Zeitpunkt $t-1$ in den

Kamerakoordinaten eines nachfolgenden Frames zum Zeitpunkt t aus:

$$m_t^i = \frac{(R_t^w)^T R_{t-1}^w \left[Z_{t-1} m_{t-1}^i - (R_{t-1}^w)^T b_{t-1,t}\right]}{(\,0\;\;0\;\;1\,)\,(R_t^w)^T R_{t-1}^w \left[Z_{t-1} m_{t-1}^i - (R_{t-1}^w)^T b_{t-1,t}\right]} \tag{4.1}$$

Zur Vereinfachung werden nun zwei Frames zu den Zeitpunkten $t = 1$ und $t = 2$

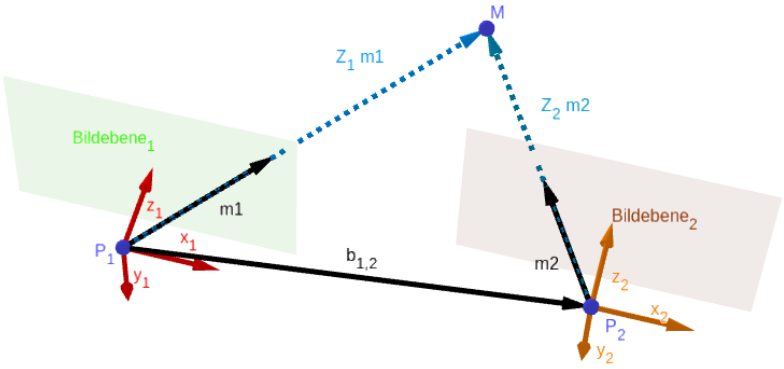


Abbildung 4.1: Die Feature-Projektionen m_1 und m_2 (unrotiert) eines 3D-Features M in zwei Frames

betrachtet. Außerdem wird der Bewegungsvektor im Kamerakoordinatensystem des ersten Frames ausgedrückt. Weiterhin wird ein beliebiges Feature ausgewählt und der Index i im Folgenden der Übersicht halber weggelassen. Aus Gleichung 4.1 ergibt sich dann:

$$m_2 = \frac{(R_2^1)^T (Z_1 m_1 - b_{1,2})}{(\,0\;\;0\;\;1\,)\,(R_2^1)^T (Z_1 m_1 - b_{1,2})} \tag{4.2}$$

Die Rotationsmatrix R_2^1 transformiert Punkte vom zweiten Kamerakoordiantensystem ins Erste:

$$R_2^1 = (R_1^w)^T R_2^w \tag{4.3}$$

Die Rotationsmatrix zwischen den beiden Frames ist aus der IMU bekannt. Daher können die Kamerakoordinaten der Features im zweiten Frame in das Koordinatensystem des ersten Frames transformiert werden:

$$(m_2)' = \frac{R_2^1 m_2}{(\,0\;\;0\;\;1\,)\,R_2^1 m_2} \tag{4.4}$$

Durch Einsetzen von Gleichung 4.4 in 4.2 entfällt die Rotationsmatrix:

$$(m_2)' = \frac{(Z_1 m_1 - b_{1,2})}{(\,0\;\;0\;\;1\,)\,(Z_1 m_1 - b_{1,2})} \tag{4.5}$$

Abbildung 4.1 illustriert diesen Zusammenhang für den Fall, dass alle Vektoren ins Koordinatensystem der ersten Kamera rotiert sind.

Wie in Anhang A.1 gezeigt, lässt sich daraus folgende Gleichung herleiten:

$$0 = -b_x \cdot (y_2' - y_1) + b_y \cdot (x_2' - x_1) + b_z \cdot (x_2' \cdot (y_2' - y_1) - y_2' \cdot (x_2' - x_1)) \tag{4.6}$$

Aus allen Feature-Korrespondenzen zwischen den beiden Frames ergibt sich aus dieser Gleichung ein überbestimmtes Gleichungssystem. Jede Feature-Korrespondenz nimmt dabei eine Zeile ein.

4.1.2 Lösung des Gleichungssystems

Das Gleichungssystem kann mit der LS-Methode gelöst werden. Bei dem Gleichungssystem handelt es sich um ein homogenes Gleichungssystem mit der trivialen Lösung $b = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$. Um ein homogenes Gleichungssystem nach der LS-Methode nicht mit der trivialen Lösung zu lösen, eignet sich zum Beispiel die **Singular value decomposition**. Dabei wird das Gleichungssystem unter der Bedingung $|b| = 1$ gelöst. Wenn eine Lösung $b \neq 0$ existiert, ist $\lambda \cdot b, \lambda \in \mathbb{R}_0$ auch eine Lösung. Es wird also nur das Verhältnis der zu lösenden Parameter gelöst. (vgl. und mehr dazu in [Inkilä, 2005])

Im Kontext der Kamerabewegung ist nur der Richtungsvektor der Bewegung bekannt. Insbesondere der Fall, dass keine Bewegung vorliegt, wird aufgrund dieser Bedingung hier nicht erkannt. Der Fall, dass $\lambda < 0$ ist, wird in Abschnitt 4.4.2 näher betrachtet.

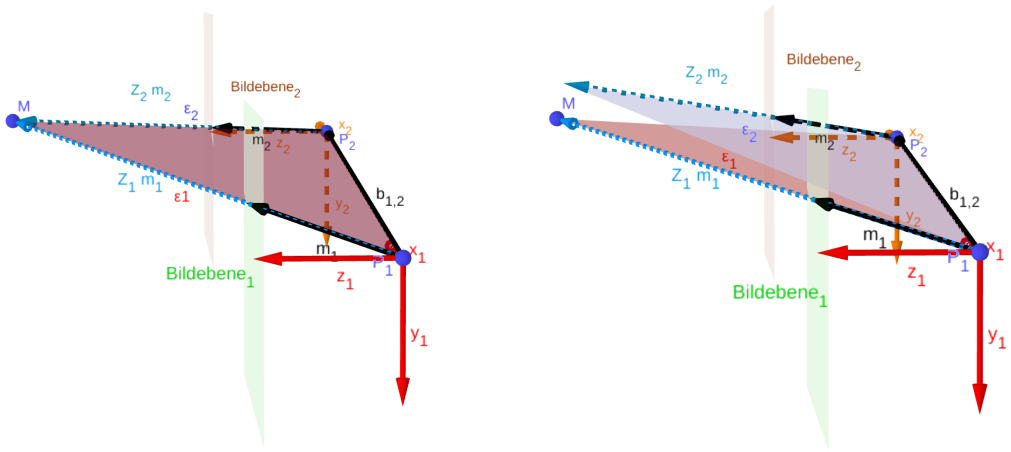
Der ermittelte Bewegungsvektor muss dann noch in Weltkoordinaten transformiert werden:

$$b_{1,2}^w = R_1^w b_{1,2} \tag{4.7}$$

Dabei ist $b_{1,2}$ der ermittelte Bewegungsvektor in den Kamerakoordinaten des ersten Frames und $b_{1,2}^w$ derselbe in Weltkoordinaten.

4.2 Outlier-Detection

Damit Ausreißer, die hier durch fehlerhaftes Tracking der Features entstehen können, in der Baseline-Estimation nicht das Ergebnis Verfälschen, wird der MLESAC-Algorithmus verwendet.



a) Die Feature-Projektionen passen zum Bewegungsvektor

b) Die Feature-Projektionen passen nicht zum Bewegungsvektor

Abbildung 4.2: Darstellung der aufgespannten Ebenen ϵ_1 und ϵ_2 für Vektoren im Kamerakoordinatensystem des Frames F_1

Die zu maximierende Bewertungsfunktion leitet sich aus den Feature-Korrespondenzen zwei nachfolgender Frames her. Die Projektionen m_1 und m_2' spannen mit dem Richtungsvektor b die Ebenen ϵ_1 und ϵ_2 auf. Bei falsch ausgerichtetem Richtungsvektor b liegt m_2' nicht mehr in der von b und m_1 aufgespannten Ebene ϵ_1. Im Idealfall handelt es sich dabei um dieselbe Ebene. Durch Ausreißer entsteht eine Fehlausrichtung der Ebenen. (Siehe Abbildung 4.2.) Der Unterschied der Ebenen kann bestimmt werden, in dem die beiden Feature-Projektionen m_1 und m_2' auf die Ebene orthogonal zum Richtungsvektor b projiziert werden.

Folgende Gewinnfunktion wird in Anhang A.2 hergeleitet:

$$p = cos(phi) = \frac{(m_2')^T(I_3 - bb^T)m_1}{\sqrt{||m_1||^2 - (m_1^T b)^2}\sqrt{||m_2'||^2 - ((m_2')^T b)^2}} \tag{4.8}$$

Auch hier wird analog zur Baseline-Estimation angenommen, dass der Bewegungsvektor b und die Feature-Projektion m_2 in das Kamerakoordinatensystem des Frames F_1 unrotiert sind. (Deswegen m_2')

Funktion 4.8 hat für den perfekten Fall, wenn die Ebenen aufeinander liegen ($e_1 = e_2$), die Lösung $p = 1$. Für einen Unterschied von 90° ist $d = 0$. Winkel größer als 90° werden mit negativen Werten bestraft.

Im Paper empfehlen Terzakis u. a. [2017] eine MLESAC-Grenze zwischen $cos(7°)$ und $cos(3°)$ empfohlen. Ab dieser Grenze wird eine Feature-Korrespondenz als Ausreißer betrachtet.

4.3 Iterative-Refinement

Das Refinement optimiert die aktuelle Position über die letzten N Bewegungsvektoren. Dabei werden die Vektoren außerdem zueinander skaliert.

Dazu definiert Terzakis u. a. [2017] ein Sliding-Window über die letzten N Frames. Für jeden neuen Frame wird so eine Kostenfunktion über diesen und alle im Fenster enthaltenen Frames berechnet. Diese bestraft, ähnlich zur Gewinnfunktion, in 4.2 die Fehlausrichtung der jeweiligen Kameraprojektionen eines Features.

4.3.1 Kostenfunktion

Für die Kostenfunktion wird der gleiche Ansatz zur Erkennung der Ausreißer in Abschnitt 4.2 benutzt. Da eine Kosten- und keine Gewinnfunktion benötigt wird, berechnet sich diese Fehlausrichtung hier aber aus dem Skalarprodukt zwischen der Normalen der Ebene ϵ_1 und der Projektion m_2':

$$C = \left((m_2')^T (b_{2,1} \times m_1)\right)^2 \tag{4.9}$$

Liegen m_2' und ϵ_1 in einer Ebene, sind die Normalen der Ebene e_1 und m_2' orthogonal zueinander und es ist $C = 0$. Bei einer Fehlausrichtung von b verändert sich der Winkel ϕ zwischen der Normalen von e_1 und m_2'. Die Kosten C verändern sich gemäß der quadrierten Kosinusfunktion. Der Winkel verändert sich auch, bei fehlerhaftem Feature-Tracking und damit fehlerhaften Feature-Projektionen m_1 bzw. m_2'. Diese werden jedoch schon durch das in Abschnitt 4.2 erklärte Verfahren aussortiert. Wenn die Anzahl der

betrachtenden Frames N größer als zwei ist, kann durch die Kostenfunktion ebenfalls der Längenfehler vom Bewegungsvektor b im Vergleich zu den bisherigen Bewegungsvektoren bewertet werden.

Wie in Anhang A.3 ergibt sich nach Terzakis u. a. [2017] die Kostenfunktion für den Frame um Zeitpunkt $t = l$ über die letzten N Frames:

$$C = \sum_{j=0}^{N-1} \sum_{k=j+1}^{N} \sum_{i \in F_{l-j,l-k}} \left((m_{l-k}^{i})^{T} (R_{l-k}^{w})^{T} \left[\frac{b_{l-k,l-j}}{||b_{l-k,l-j}||} \right] \times R_{l-j}^{w} m_{l-j}^{i} \right)^{2} \tag{4.10}$$

(Ab dieser Gleichung sind die Feature-Projektionen wieder, wie in Abschnitt 2.6 angegeben, im jeweiligen Kamerakoordinatensystem ihres Frames und die Bewegungsvektoren in Weltkoordinaten angegeben.)

Der aus den beiden Bewegungsvektoren zusammengesetzte Bewegungsvektor wird normiert, da für die Winkelberechnung mittels des Skalarproduktes die Vektoren normiert sein müssen. Dies ist von Terzakis u. a. [2017] so nicht beschrieben. Dort wird der Teil in den eckigen Klammern als $P_{l-k} - P_{l-j}$ angegeben. Ohne diese Normierung funktioniert das Skalarprodukt jedoch nicht. Eine minimale Lösung wäre dann $P_{l-k} = P_{l-j}$

4.3.2 Optimierung mithilfe der Kostenfunktion

Die Kostenfunktion ist eine nicht lineare Funktion in Abhängigkeit der Bewegungsvektoren. Beim Minimieren der Kostenfunktion ergibt sich ein NLLSQ-Problem. Zur Lösung eignet sich zum Beispiel der Levenberg-Marquard (LM)-Algorithmus. Dabei werden die Bewegungsvektoren iterativ angepasst, um die Kostenfunktion zu minimieren. (Vgl. und mehr dazu in [Gavin, 2019].)

4.3.3 Anpassung der Kostenfukntion

In ersten Tests hat sich gezeigt, dass im Fall der zusammengesetzten Bewegungsvektoren aus Gleichung 4.10 es, trotz Normierung, in einigen Fällen in ein Minimum führt, einzelne Bewegungsvektoren auf die Länge 0 zu optimieren. Generell hat sich das Refinement sehr unstabil gezeigt. Es gab starke Ausreißer bei den Bewegungsvektoren. (Dazu später mehr in Abschnitt 7.3.5)

Um dem entgegenzuwirken, wurde mithilfe des Autors des Papers eine neue Parametrisierung der Bewegungsvektoren für die Kostenfunktion eingeführt. Der Lösungsraum wird dadurch auf die sinnvollen Lösungen eingeschränkt. Dies geschieht unter der Annahme, dass durch die Baseline-Estimation aus Abschnitt 4.1 eine bereits näherungsweise korrekte Richtung berechnet wurde. Es wird ein einzelner Bewegungsvektor b in seinen Skalar- n und Richtungsanteil u aufgeteilt:

$$b = n \cdot u \tag{4.11}$$

Da b bisher nur als Richtungsvektor mit der Länge $||b|| = 1$ behandelt wurde, ist zunächst $n = 1$.

Optimierung der Skalierung Um das Ausreißen der Skalierung ins Negative oder Unendliche und das Optimieren auf die Länge 0 zu verhindern, wird angenommen, dass das Schiff eine Mindest- und Maximalgeschwindigkeit besitzt, mit der es sich im Falle einer Bewegung fortbewegen kann. So kann die Skalierung durch eine logistische Funktion nach oben und unten begrenzt werden:

$$n = min + \frac{max - min}{1 + e^{-t}} \tag{4.12}$$

Dabei stellt min die untere und max die obere Grenze dar. Die logistische Funktion eignet sich als differenzierbare Funktion gut, um den Parameterraum einzugrenzen. Der Parameter t wird durch das iterative Verfahren während der Optimierung angepasst. Die Grenzen müssen für das Schiff experimentell ermittelt werden. Da es sich insgesamt nur um eine relative Skalierung handelt, stellt $\frac{max}{min}$ den maximalen Längenunterschied dar, mit dem sich die Kamera bzw. das Schiff bewegt.

Optimierung der Richtung In diesem Teil soll nur die Richtung des Vektors verändert werden. Dazu wird der Vektor um den Ursprung gedreht.

Um den Vektor in dem gewünschten Bereich zu rotieren, genügt jeweils eine Periode der Sinus- bzw. Kosinusfunktion. Folgende Funktionen bilden die reellen Zahlen jeweils auf

nur eine Periode ab [Terzakis u. a., 2018]:

$$s(x) = \frac{2x}{1+x^2} \tag{4.13}$$

$$c(x) = \frac{1-x^2}{1+x^2} \tag{4.14}$$

Die Rotation wird durch die zusammengesetzte Rotationsmatrix R_v parametrisiert.

$$R_v = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1-a^2}{1+a^2} & -\frac{2a}{1+a^2} \\ 0 & \frac{2a}{1+a^2} & \frac{1-a^2}{1+a^2} \end{pmatrix} \begin{pmatrix} \frac{1-b^2}{1+b^2} & 0 & \frac{2b}{1+b^2} \\ 0 & 1 & 0 \\ -\frac{2b}{1+b^2} & 0 & \frac{1-b^2}{1+b^2} \end{pmatrix} \begin{pmatrix} \frac{1-c^2}{1+c^2} & -\frac{2c}{1+c^2} & 0 \\ \frac{2c}{1+c^2} & \frac{1-c^2}{1+c^2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.15}$$

Der neue Translationsvektor u' berechnet sich dann durch Rotation von u mit R_v:

$$u' = R_v * u \tag{4.16}$$

Im Kontext der einzelnen Rotationsmatrizen bedeutet die Konvergenz der Funktion $s(x)$ gegen 0 bzw. der Funktion $c(x)$ gegen -1, dass diese gegen eine Drehung um 180° konvergieren, aber nie ganz erreichen. Dies verhindert in der iterativen Optimierung das einfache Invertieren des Vektors.

Mehr zu dieser Parametrisierung findet sich in Anhang A.4.

4.4 Anpassungen und Erweiterungen

Im Rahmen dieser Arbeit wurden weitere Anpassungen und Erweiterungen des Verfahrens getätigt. Im Folgenden sollen diese erklärt werden.

4.4.1 IMU im Feature-Tracking

Wie in Abschnitt 2.3 beschrieben, sucht das Tracking an der Stelle der alten Position eines Features nach dem verschobenen Feature. Besonders bei starkem Schaukeln des Schiffes und damit der Kamera oder bei schnellen Rotationen ändern sich diese Positionen sehr schnell sehr weit. Um das Tracking zu erleichtern, wird die Orientierung der Kamera

zum Zeitpunkt des vorherigen Frames und des neuen Frames berücksichtigt. Dazu wird die Rotation zwischen dem alten Frames F_t und dem neuen Frame F_{t+1}berechnet:

$$R_{t+1}^{t} = (R_t^w)^T R_{t+1}^w \tag{4.17}$$

Mithilfe dieser Rotation können die Kamerakoordinaten der Features des vorherigen Frames an die neuen Positionen transformiert werden:

$$m_{t+1}^{i} = (R_{t+1}^{t})^T m_t^i \tag{4.18}$$

An dieser Position befänden sich jeweils die Features, wenn nur die Rotation und keine Translation zwischen dem vorherigen und dem neuen Frame läge. Mit den zurück in Pixelkoordinaten projizierten neuen Positionen wird der Tracker initialisiert, um ein besseres Tracking zu erzielen.

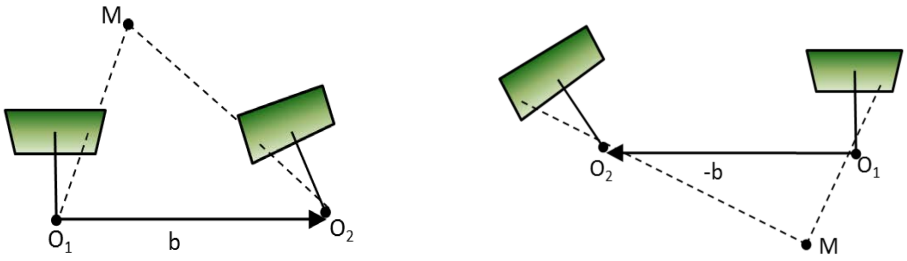
4.4.2 Negative Richtungen

Der Autor erwähnt in seinem Paper das Problem der negativen Richtungsvektoren zwischen zwei Frames. Wie in Abschnitt 4.1 beschrieben existieren für die Gleichung alle möglichen Skalierungen der Lösung als Lösung. Die impliziert auch die negativen. Somit existieren für den Richtungsvektor zwei Lösungen. Abbildung 4.3 zeigt beide Möglichkeiten. In der Lösung mit negativem Richtungsvektor ist die Feature-Tiefe Z negativ:

$$M = -Z * m \tag{4.19}$$

Die Features liegen also „hinter“ den Kamerabildflächen.

Um die richtige Lösung des Richtungsvektors zu erhalten, schlägt der Autor vor, die 3D-Positionen der, in beiden Frames erkannten, Features (Feature-Korrespondenzen) über Triangulation zu rekonstruieren. Aus der jeweiligen Triangulation eines Features kann dann für jede der beiden Projektionen auf den Kamerabildern der beiden Frames der Wert für Z bestimmt werden. Zur Triangulation wird der Richtungsvektor einmal negiert und einmal normal in die Triangulation einbezogen. So ergeben sich zwei Mengen von Triangulationen. Die Menge mit weniger negativen Werten für Z gewinnt und wird als korrekte Lösung für den Richtungsvektor angenommen. Für den Fall von überwiegend negativen Werten für das jeweilige Z der Feature-Projektionen, muss der ermittelte Richtungsvektor negiert werden. [Terzakis, 2013, Seite 12 f.]



a) Positive Richtung **b)** Negative Richtung

Abbildung 4.3: Mögliche Richtungsvektoren bei gleicher Feature-Korrespondenz [Terzakis, 2013]

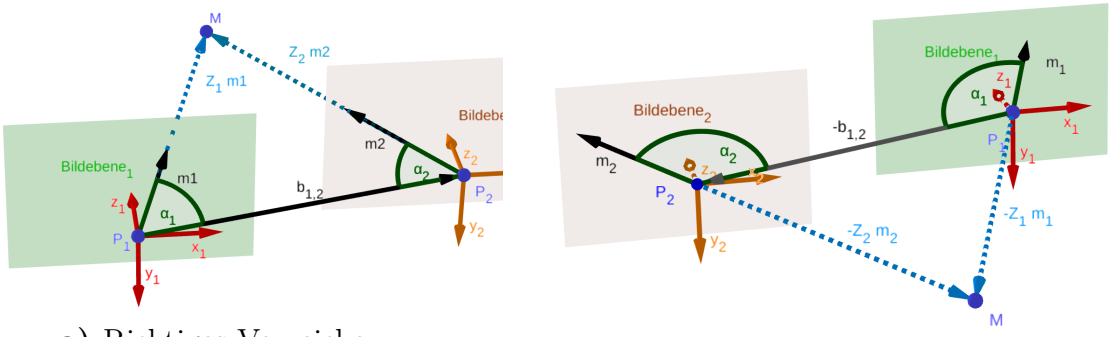
Wie in der Diskussion zu diesem Verfahren erwähnt, ist der Vorteil dieses VO-Verfahrens der Verzicht auf Triangulationen. Daher wird in dieser Arbeit der Fall einer negativen Richtung anders berechnet. Es genügt das Vorzeichen beider Z zu ermitteln. Also ob das Feature bei dem erkannten Bewegungsvektor b „hinter“ oder „vor“ den Kamerabildflächen liegt. Abbildung 4.4 zeigt für beide Varianten den Winkel zwischen den erkannten Feature-Projektionen m_1 und m_2 und dem Vektor b:

$$\alpha_1 = \frac{m_1^T R_w^1 b}{||m_1^T R_w^1 b||} \tag{4.20}$$

$$\alpha_2 = \frac{m_2^T R_w^2 (-b)}{||m_2^T R_w^2 (-b)||} \tag{4.21}$$

Im korrekten Fall beträgt die Summe beider Winkel eine Summe kleiner $cos(180°)$. Im falschen Fall mit negativer Richtung ist die Summe größer. Diese einfach zu errechnende Winkelsumme wird so für alle Feature-Korrespondenzen für den in der Baseline-Estimation berechneten Bewegungsvektor bestimmt. Sind einzelne Features sehr weit entfernt, nähert sich die Summe immer weiter der $cos(180°)$. Ist die Feature-Projekion verrauscht, kann hier fälschlicherweise die $cos(180°)$ überschritten werden. Diese Feature-Korrespondenz würde dann die falsche Richtung voraussagen. Um weit entfernten Features weniger Gewichtung zu schenken als nahen, wird über alle Feature-Korrespondenzen folgende Fehlerfunktion berechnet:

$$E = \sum_{i \in F_{j,k}} cos \left[0.5 \left[cos^{-1} \left(\frac{(m_1^i)^T R_w^1 b}{||(m_1^i)^T R_w^1 b||} \right) + cos^{-1} \left(\frac{(m_2^i)^T R_w^2 (-b)}{||(m_2^i)^T R_w^2 (-b)||} \right) \right] \right] \tag{4.22}$$



a) Richtiges Vorzeichen

b) Falsches Vorzeichen

Abbildung 4.4: Winkel zwischen dem Richtungsvektor und den (unrotierten) Feature-Projektionen für beide Bewegungsvektorrichtungen

Die Kosinusfunktion mit halber Frequenz führt dazu, dass Werte im Bereich 180° eine niedrige Gewichtung in der Fehlersumme erhalten. Des Weiteren werden Feature-Korrespondenzen mit einer Winkelsumme im Bereich $cos(180° - 1°) \geq (\alpha_1 + \alpha_2) \leq cos(180° + 1°)$ in der Fehlersumme nicht berücksichtigt. Ist die Fehlersumme positiv, wird die Lösung akzeptiert. Ist diese negativ, wird der Richtungsvektor negiert.

4.4.3 Refinementfenster beschränken

Die Kostenfunktion 4.10 des iterativen Refinements optimiert die letzten N Frames. Dabei werden $N - 1$ Richtungsvektoren und deren Länge optimiert. Bei ersten Tests zeigte sich hierbei jedoch eine Gefahr: Während der Optimierung wird teilweise die Skalierung des Vektors zwischen den Frames $t - N$ und $t - (N - 1)$ stark verändert, um die Kostenfunktion zu minimieren. Diese Änderung geschieht aber ohne Berücksichtigung der Vektoren außerhalb der letzten N Frames. Das Verfahren arbeitet mit einer relativen Skalierung. Anhand der vorherigen Richtungsvektorlängen werden die nachfolgenden Richtungsvektoren skaliert. Das beschriebene Verhalten steht dazu im Widerspruch. Dieses Problem wird im Folgenden als das „Konsistenz-Problem“ bezeichnet.

Um es zu verhindern, wird neben der Fenstergröße N noch das sogenannte Refinemenfenster mit der Größe R eingeführt. Das Fenster mit der Fenstergröße N wird im Folgenden als das Betrachtungsfenster bezeichnet. Die letzten N Frames werden während der Optimierung betrachtet und die letzten R Frames bzw. deren Richtungsvektoren und

Skalierungen verändert. Da zur Optimierung eines Vektors $b_{t-1,t}$ und dessen Skalierung immer die Feature-Korrespondenzen der Frames $b_{t-1} und b_t$ benötigt werden gilt:

$$R < N \tag{4.23}$$

Die Kostenfunktion 4.10 bleibt dieselbe, mit der Einschränkung, dass die Frames außerhalb des Refinementfensters R nicht Teil der zu optimierenden Parametermenge sind, sondern als konstant angenommen werden. Fehlertherme, die ausschließlich aus Bewegungsvektoren außerhalb des Refinementfensters bestehen, sind unnötig während der Optimierung. Deshalb ändert sich eine der Summen: (Kostenfunktion zum Zeitpunkt $t = l$.)

$$C = \sum_{j=0}^{R} \sum_{k=j+1}^{N} \sum_{i \in F_{l-j,l-k}} \left((m_{l-k}^{i})^T (R_{l-k}^{w})^T \left[\frac{b_{l-k,l-j}}{||b_{l-k,l-j}||} \right] \times R_{l-j}^{w} m_{l-j}^{i} \right)^2 \tag{4.24}$$

Das Refinementfenster und das Betrachtungsfenster werden im Folgenden als „Fenster“ und deren Größen als „Fenstergrößen“ zusammengefasst.

4.4.4 Mindestdisparität

Um den Fall, dass zwischen zwei Frames keine Bewegung liegt zu erkennen, wird eine Mindestdisparität zwischen zwei Frames definiert.

An dieser Stelle wird dafür die sog. Disparität d zwischen den Feature-Korrespondenzen über deren Winkelunterschied berechnet:

$$d = \frac{(m_{t-1}^{i})^T R_t^{t-1} m_t^i}{||m_{t-1}^{i}|| \cdot ||m_t^i||} \tag{4.25}$$

Wird diese Disparität über alle erkannten Feature-Paare gemittelt, erhält man ein Maß auf Positionsänderung der Features. Liegt der Wert unter einem Schwellwert, liegt keine Bewegung vor und es muss auf einen neuen Frame gewartet werden, bevor die Baseline-Estimation ausgeführt wird.

5 Softwareumsetzung

5.1 Anforderungen

Um das in bisher erklärte Verfahren auf dem Schiff nutzen zu können, muss ein Softwaresystem implementiert werden. Es existieren neben der Lösung des beschriebenen Algorithmus dabei weitere Anforderungen:

1. Ein Frame beinhaltet mehrere Informationen[1], diese müssen **in einer Datenstruktur effizient gehalten** werden. Da unter anderem Bilder verarbeitet werden, bedeutet effizient nicht nur schnellen Zugriff, sondern auch wenig Kopiervorgänge und keine Redundanzen.

2. Zur Berechnung einer Translation werden zwei Frames benötigt. Die Translation wird immer als Bewegungsvektor zwischen dem aktuellen und dem vorherigen Frame berechnet.

3. In einem neuen Frame werden nicht nur neue Features erkannt, sondern auch die Features aus dem vorherigen Frame getrackt. Der **Zusammenhang der Features über mehrere Frames** muss abgebildet werden.

4. Es arbeiten im Verfahren unterschiedlichste Algorithmen zusammen. Diese benötigen unterschiedlich viel Rechenzeit. Dennoch sollen die Features in jedem neuen Frame weiter getrackt werden, damit keine Bewegungen verloren gehen oder zu schnelle Bewegungen der Kamera das Tracking des Features beeinträchtigen. Die verschiedenen Teile des Systems müssen daher **parallel** arbeiten, ohne dass ein langsamer Teil die anderen Teile blockieren lässt.

5. Wie in Abschnitt 4.1 erklärt, sind nicht-Bewegungen für das Verfahren zunächst nicht zu erkennen. Das System muss Frames, zwischen denen keine Bewegung liegt, erkennen und **aussortieren**. Außerdem hat das Verfahren je nach Rechenleistung

[1] Siehe dazu die Erklärung in Abschnitt 2.6

und Anzahl der Features eine maximale Anzahl an Frames, die in einem Zeitabschnitt verarbeitet werden können. Es muss eine Frequenz einstellbar sein, mit der Frames verarbeitet werden. In beiden Fällen darf dies nicht das Feature-Tracking negativ beeinflussen, weil zum Beispiel Kamerabilder einfach ignoriert würden.

5.2 Softwareentwurf

Im Folgenden wird das Softwaresystem, welches obige Anforderungen genügt und in dieser Arbeit zur Nutzung des Verfahrens implementiert wurde, erklärt. Die Implementation ist in *C++* geschrieben. Zum Einsatz kommt das bereits erwähnte Robotik-Framework *ROS* [Stanford Artificial Intelligence Laboratory et al.], die Bildverarbeitungs-Library *OpenCV* [Bradski, 2000] und zum Lösen von nicht linearen Problem die Library *CERES-Solver* [Agarwal u. a.] in Verbindung mit der Mathematik-Library *Eigen* [Guennebaud u. a., 2010].

5.2.1 Datenstruktur Frame

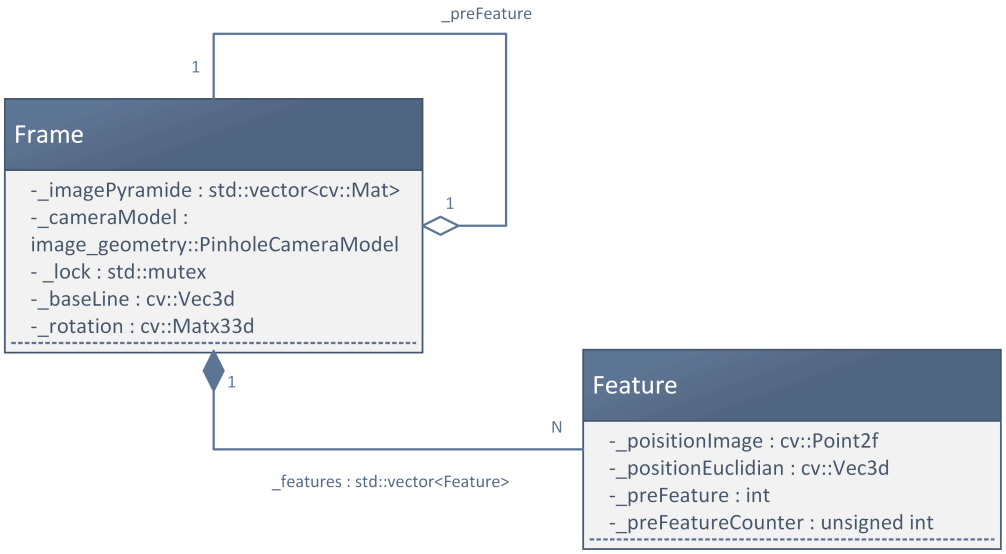


Abbildung 5.1: Klassendiagramm der Klasse `Frame`

.

Abbildung 5.1 zeigt die Klassen, die den Frame implementieren. Die Klasse `Frame` hält das Kamerabild als Bildpyramide in `_imagePyramide`. So muss diese nur bei der

Erzeugung berechnet werden. Neben dem eigentlichen Bild wird im Feld `_cameraModel` die Kameramatrix gehalten.

Neben den Bildinformationen enthält die Klasse `Frame` in `_rotation` die aktuelle Orientierung der Kamera als Rotationsmatrix. Der Bewegungsvektor vom vorhergehenden Frame zu diesem Frame setzt sich aus dem Richtungsvektor `_baseLine` und der Länge `_scale` zusammen.

Um den Zusammenhang zum vorherigen Frame herzustellen, hält die Klasse `Frame` den Zeiger `_prevFrame` auf den vorhergehenden Frame. Gibt es keinen Vorgänger, handelt es sich um einen `nullptr`. Zusätzlich hält `Frame` eine Liste aller in diesem Kamerabild erkannten und getrackten Features. Letztere werden durch die Klasse `Feature` modelliert. Ein Feature besteht aus seiner Position im Bild in Pixelkoordinaten `_positionImage` und seine Projektion in Kamerakoordinaten `_positionProjected`. Da die Pixelkoordinaten in Kamerakoordinaten umgerechnet werden können, herrscht hier eine Redundanz. Diese ist allerdings bewusst eingebaut, da an verschiedensten Stellen beide Arten benötigt werden. So werden mehrfache Umrechnungen vermieden. Handelt es sich um ein Feature, welches auch im vorherigen Frame erkannt wurde und in diesem Frame nur weiter getrackt wurde, steht im Feld `_preFeature` der Index der `_features`-Liste des vorherigen Frames, an dem dieses Feature im vorherigen Frame befindet. Andernfalls ist `_preFeature` auf den Wert -1 gesetzt. Das Feld `_preFeatureCounter` gibt an, in wie vielen vorherigen Frames dieses Feature mindestens schon erkannt wurde.

Abbildung 5.2 zeigt exemplarisch drei aufeinander folgende Frames mit jeweils erkannten und getrackten Features. Im `frame0` wurden drei neue Features erkannt. Im darauf folgenden `frame1` wurden die Features an den Stellen 0 und 2 getrackt und zwei neue Features erkannt. In dessen Nachfolger `frame2` wurden die Features an den Stellen 0, 1 und 3 des `frame1` getrackt und ein neues Feature erkannt. Die Features an Stelle 0 und 1 des `frame2` wurden über alle drei Frames erkannt, daher steht deren `_preFeatureCounter` auf 2 und die Felder `_preFeature` verweisen auf den Vorgänger in `frame1`. An den entsprechenden Vorgängern an Stelle 0 und 1 im `frame1` steht der `_preFeatureCounter` auf 1 und das Feld `_preFeature` verweist auf die jeweiligen Vorgänger im `frame0`. Das Feature an Stelle 2 des Frames `frame2` wurde erstmalig in `frame1` erkannt und getrackt. Entsprechend steht der `_preFeatureCounter` auf 1. Das Feature an Stelle 3 im `frame2` wurde in diesem Frame erstmalig erkannt. Der `_preFeatureCounter` steht entsprechend auf 0 und das Feld `_preFeature` auf -1.

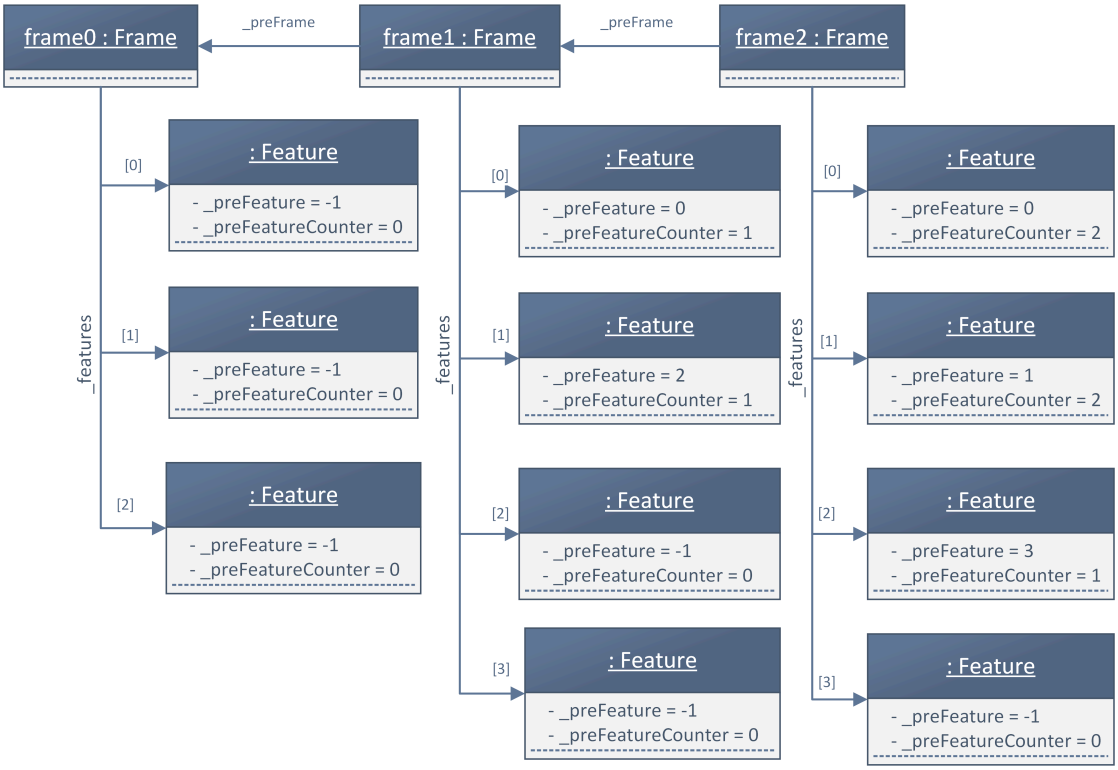


Abbildung 5.2: Objektdiagramm für drei aufeinander folgende Frames.

Der Zeitstempel `_timeStamp` in `Frame` dient der späteren Zuordnung der Positionen auch nach verzögerter Berechnung. Um das parallele Arbeiten zu synchronisieren, verfügen Objekte der Klasse `Frame` über einen Mutex `_lock`.

5.2.2 Pipeline

Die verschiedenen Teile des Verfahrens sollen parallel arbeiten. Da dabei für jeden Frame eine feste Reihenfolge eingehalten wird, bietet sich die Architektur einer Pipeline an. Die Frames werden dabei durch die Pipeline gereicht. Eine Pipeline-Stufe kann intern zusätzlich einen Frame-Puffer besitzen. Dies wird für Pipeline-Stufen, die auf mehr als einem Frame arbeiten, benötigt. Das Grundgerüst für eine Pipeline-Stufe implementiert die in Abbildung 5.3 dargestellte abstrakte Basisklasse `PipelineStage`. Von dieser erben die einzelnen Pipeline-Stufen. Im Feld `_precursorStage` hält die `PipelineStage` einen Zeiger auf ihren Vorgänger. Daneben besitzt jede `PipelineStage`

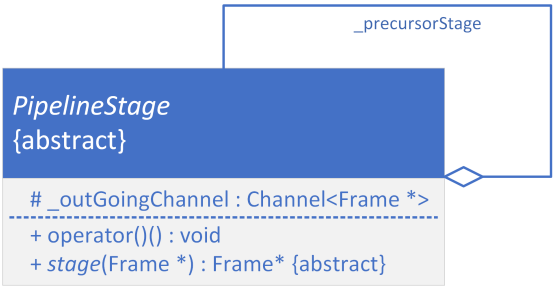


Abbildung 5.3: Klassendiagramm der abstrakten Klasse `PipelineStage`.

einen `_outGoingChannel`. Hierbei handelt es sich um eine threadsichere Warteschlange mit einer maximalen Kapazität. Ist diese erreicht, blockiert der Schreibzugriff.

Die einzelnen Pipelines-Stufen erben von `PipelineStage` und implementieren die abstrakte Methode `stage(Frame*): Frame*`.

Beim Start des Systems wird für jede Pipeline-Stufe ein Thread erzeugt, dem die jeweilige Stufe übergeben wird. Die *C++*-Implementation für Threads führt dann den `operator()` aus. Die Pipeline-Logik ist von diesem Operator implementiert. Das Listing 5.1 zeigt die Implementation des Operators.
Während der Ausführung des Systems wird der anliegende `Frame` aus dem `_outGoingChannel` der `_precursorStage` entnommen. Mit dem neuen `_Frame` als Parameter wird die abstrakte Methode `Frame * stage(Frame * newFrame)` aufgerufen. Diese wird von den konkreten Pipeline-Stufen mit dem jeweiligen verhalten implementiert. Der Methodenaufruf gibt den Zeiger auf einen `Frame` zurück. Sofern es sich um keine `nullptr` handelt, wird dieser in den eigenen `_outGoingChannel` übergebenen. Da Pipeline-Stufen

Listing 5.1: `PipelineStage::operator()`

```
1     while (ros::ok()) {
2         Frame *ingoingFrame =
3             _precursorStage->_outGoingChannel.dequeue();
4         Frame *outgoingFrame =
5             stage(ingoingFrame);
6         if (outgoingFrame != nullptr) {
7           _outGoingChannel.enqueue(outgoingFrame);
8         }
9     }
```

intern mehrere Frames puffern können, handelt es sich bei dem zurückgebenden Frame um den ältesten dieser Frames.

Der interne Puffer, ist nicht mit der Warteschlange `_outGoingChannel` zu verwechseln. Erstere wird, wie oben angesprochen, benötigt, da Teile des Algorithmus und damit einige Pipeline-Stufen auf mehreren Frames arbeiten und ist von den erbenden Pipeline-Stufen selbst zu implementieren. Letztere ermöglicht es den einzelnen Pipeline-Stufen auch dann parallel voneinander zu arbeiten, wenn in einer Pipeline-Stufe ein Stau entsteht.

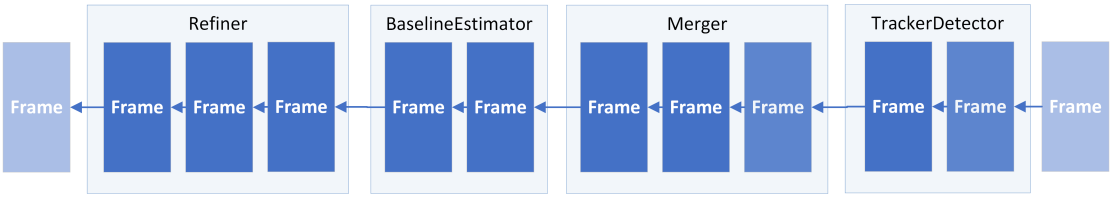


Abbildung 5.4: Blockdiagramm der Pipeline mit leeren Warteschlangen

Die Größe der Warteschlangen zwischen den Stufen ist so zu wählen, dass alle Frames warten können, falls die jeweils nachfolgende Pipeline-Stufe zwischenzeitig einen niedrigeren Durchsatz als die jeweils vorhergehende erreicht. Die Architektur ermöglicht außerdem eine klare Unterteilung von Zuständigkeitsbereichen der Frames auf die Pipeline-Stufen. Ein Frame gehört immer zu der Stufe, in der er sich gerade zur Verarbeitung oder in deren `_outGoingChannel` er sich gerade befindet. Eine Pipeline-Stufe darf nur auf Frames in ihrem Zuständigkeitsbereich lesende oder schreibende Operationen durchführen.

Um unnötige Kopiervorgänge zu vermeiden, werden im gesamten System nur Zeiger auf die Frames übergeben. Ein Exemplar der Klasse `Frame` wird mit dem Erreichen des Kamerabildes erzeugt. Erst nachdem der Zeiger auf dieses Exemplar die letzte Pipeline-Stufe wieder verlassen hat, wird dieses Exemplar zerstört.

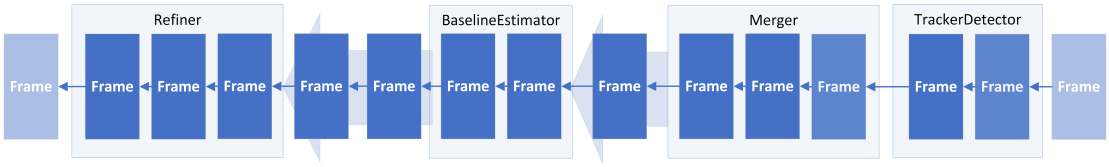


Abbildung 5.5: Blockdiagramm der Pipeline mit gefüllten Warteschlangen

Abbildung 5.4 zeigt die Pipeline-Stufen im Fall, dass die Warteschlangen zwischen den Stufen leer sind. Die Pipeline besteht aus vier Pipeline-Stufen. Die Abbildung zeigt eine

Reihe von nachfolgenden Frames. Der Pfeil verweist auf den jeweiligen Vorgänger. 10 der 12 Frames werden von den Pipeline-Stufen verarbeitet. Dabei arbeitet jede Stufe auf einer unterschiedlichen Anzahl von Frames. Der `Refiner` optimiert z. B. $N = 3$ Frames.

Im Gegensatz zur Abbildung 5.4, in der die Pipeline-Stufen synchron arbeiten bzw. den gleichen Durchsatz haben, zeigt Abbildung 5.5 den Einsatz der Warteschlangen zwischen den Stufen. Zwischen `Merger` und `BaselineEstimator` liegt ein Frame in der Warteschlange. Zwischen den Stufen `BaselineEstimator` und `Refiner` liegen zwei Frames in der Warteschlange. Zwischen den Stufen `TrackerDetector` und `Merger` ist die Warteschlange leer. Das bedeutet entweder, dass die Stufe `Merger` mindestens den gleichen Durchsatz wie die Stufe `TrackerDetector` hat oder, dass die Größe der Warteschlange auf 0 gesetzt wurde und die Stufe `TrackerDetector` blockiert, weil die Stufe `Merger` arbeitet oder ihrerseits blockiert ist.

5.2.3 Latenzverringerung

Durch den Einsatz einer Pipeline kommt es zu einer Latenz zwischen der Erzeugung eines Frames und der Verarbeitung seines Bewegungsvektors zur Gesamtposition am Ende der Pipeline. Je nach Wahl der Anzahl zu betrachtenden Frames N im `Refiner` beträgt die Latenz mindestens $2 + 3 + 2 + N = 7 + N$ Frames. Besonders negativ wirkt sich das auf die Positionsbestimmung aus, wenn das Schiff bzw. die Kamera still steht. In diesem Fall hält der `Merger` die Pipeline an. Es werden von den Pipeline-Stufen nach dem `Merger` die Warteschlangen zwischen ihnen abgearbeitet und die Stufen blockieren. Damit liegt die zuletzt berechnete Position $7 + N$ Frames hinter der letzten Bewegung.

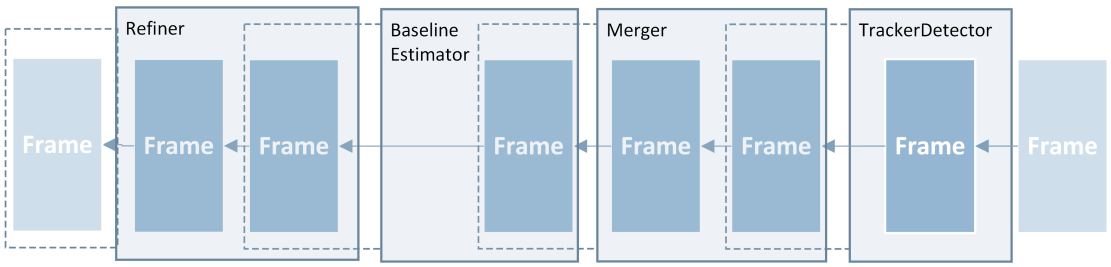


Abbildung 5.6: Blockdiagramm der Pipeline mit der Erlaubnis auf dem vorherigen Frame zu lesen.

Eine Erweiterung des Lesbarkeitsbereiches einer Pipeline-Stufe verringert die Latenz. Eine Pipeline-Stufe darf neben den Frames in ihrem Zuständigkeitsbereich auch auf deren

unmittelbaren Vorgängern lesende Operationen ausführen. Dabei ist allerdings nicht gewährleistet, dass sich die Daten im Vorgänger nicht geändert haben. Für Vorgänger, die ohnehin schon im Zuständigkeitsbereich der Pipeline-Stufe liegen, ist dies unerheblich. Für den ältesten gehaltenen Frame erweitert dies jedoch den lesbaren Bereich.

In Abbildung 5.6 wird dargestellt, wie sich damit der lesende Bereich der Pipeline-Stufen erweitert. In jeder Stufe wird der jeweils letzte Frames nur gehalten, um lesend auf diesen zuzugreifen. Dargestellt durch die gestrichelte Linie. Mit dieser Erweiterung der Pipeline können Frames früher weiter gereicht werden. Die Latenz reduziert sich so von $7 + N$ auf $3 + N$ Frames. Den Pipeline-übergreifenden Zugriff synchronisiert der oben erwähnte Mutex `_lock` in der Klasse `Frame`. Zur Übersichtlichkeit wird in dieser Abbildung angenommen, dass die Warteschlangen zwischen den Pipeline-Stufen leer sind. Der `Refiner` optimiert in diesem Beispiel $N = 3$ Frames.

5.2.4 Pipeline-Stufen

Pipeline-Stufe	Funktion
`TrackerDetector`	Tracking und Erkennen von Features
`Merger`	Aussortieren von Frames ohne Änderungen und Sicherstellen einer Maximalfrequenz
`BaselineEstimator`	Berechnung des Richtungsvektors und Erkennen von Ausreißern
`Refiner`	Optimierung der Richtungen und relative Skalierung

Tabelle 5.1: Pipeline-Stufen mit ihren Funktionen

Das System besteht aus den in Tabelle 5.1 aufgeführten Pipeline-Stufen. Diese sind der Reihenfolge nach angeordnet. Der `Merger` befindet sich hinter dem `TrackeDetector`. So wird wie gefordert eine Frame-Frequenz ermöglicht, ohne das Tracking zu beeinflussen. Neben den eigentlichen Pipeline-Stufen existieren spezielle Pipeline-Ränder. An diesen werden die neuen Frames erzeugt und an die erste Stufe übergeben bzw. die fertig bearbeiteten und nicht mehr benötigten Frames entfernt. Außerdem wird ein spezielles Konstrukt benötigt, um die ermittelten Positionen an verschiedenen Stellen aus dem System auszugeben.

Im Folgenden werden die Pipeline-Stufen einzeln erläutert.

TrackerDetector

Der `TrackerDetector` ist für das Erkennen neuer Features und das Tracking von bereits erkannten Features zuständig. Zunächst werden die, im vorherigen Frame bereits erkannten Features, mithilfe des LK-Pyramiden-Verfahrens unter Zuhilfenahme des Rotationsunterschieds im Bild des neuen Frames getrackt. Diese Features werden mit der zugehörigen Beziehung in die Datenstruktur `Frame` eingetragen. Danach werden von dieser Pipeline-Stufe mit dem Harris-Corner-Detector neue Features im Bild des neuen Frames berechnet.

Um eine möglichst gleichmäßige Verteilung der Features zu erzielen, kann ein Mindestabstand zwischen zwei Features festgelegt werden. Dieser Abstand wird prozentual zur Bildgröße festgelegt, um verschiedenste Bildgrößen gleichermaßen zu behandeln. Der Mindestabstand wird sowohl beim Erkennen neuer, als auch beim Tracking schon bekannter Features berücksichtigt. Bei einer Rückwärtsfahrt der Kamera verschwinden mehrere Features an einem Punkt. Wird der Mindestabstand unterschritten, fasst der `TrackerDetector` diese zu einem Feature zusammen.

Neben dem Mindestabstand lässt sich ein Bereich definieren, in dem keine Features erkannt oder getrackt werden sollen. Im Kontext des Schiffes kann so das Tracking der im Kamerabild sichtbaren Schiffsaufbauten verhindert werden. Diese Features würden sich fälschlicherweise nie bewegen und die Bewegungserkennung so falsch beeinflussen.

Merger

Die Pipeline-Stufe `Merger` fasst zwei Frames zusammen. Wie in den Anforderungen gefordert, kann der Ausgangsdurchsatz dieser Pipeline-Stufe auf zwei Arten eingeschränkt werden. Zum einen, muss zwischen den Frames die, in Abschnitt 4.4.4 erklärte, Feature-Mindestdisparität liegen. Zum anderen soll der Ausgangsdurchsatz durch eine Maximalfrequenz begrenzt werden. Erreicht ihn ein neuer Frame, vergleicht er den neuen Frame mit dessen Vorgänger. Sind beide Voraussetzungen noch nicht erfüllt, hält er den neuen Frame intern. Dieser bekommt im Folgenden den Namen mittlerer Frame. Es gelangt in diesem Fall kein neuer Frame in die `_outGoingQueue` des `Merger`. Erreichen den `Merger` ein Frame, fasst dieser diesen neuen Frame mit dem mittleren zusammen. Dazu wird der Mittlere mit dem Neuen ersetzt. Es wird sichergestellt, dass im neuen Frame in den Datenstrukturen `Feature` die Felder `_preFeature` und `_preFeatureCounter` aus dem

jeweils zugehörigen `Feature` des Mittleren Frames übernommen werden. Außerdem muss im neuen `Frame` der Zeiger auf seinen Vorgänger auf den ursprünglich gehaltenen `Frame` umgelegt werden. Da der Mittlere Frame nun von keinem anderen mehr referenziert sein kann, gibt der `Merger` den Speicher frei. Der Neue und ursprünglich alte Frame werden verglichen. Treffen nun beide obige Bedingungen zu, wird der neue Frame weitergegeben. Ist eine der beiden Bedingungen nicht erfüllt, rückt der neue Frame an die Stelle des mittleren.

BaselineEstimator

Diese Pipeline-Stufe berechnet den Richtungsvektor zwischen zwei Frames. Betrachtet wird der letzte Frame F_{alt}. Erreicht ein neuer Frame F_{neu} diese Pipeline-Stufe, werden alle in beiden Frames erkannten Features ermittelt. Hieraus errechnet sich gemäß den Verfahren aus Abschnitt 4.1 und 4.2 die Richtung des Bewegungsvektors von F_{neu} zu F_{alt}. In der Datenstruktur `Frame` des F_{neu} wird das Feld `_baseLine` entsprechend gesetzt. Das Feld `_scale` wird auf 1 gesetzt. Außerdem werden im Frame F_{neu} bei den erkannten Ausreißern das Feld `_preFeature` auf -1 und das Feld `_preFeatureCounter` auf 0 gesetzt.

Refiner

Diese Pipeline-Stufe soll die Positionsschätzung gemäß dem Verfahren aus Abschnitt 4.3 durch die Optimierung der Skalierungen und Richtungen der Bewegungsvektoren zwischen den letzten N Frames optimieren. Beim Erzeugen der Pipeline-Stufe `Refiner` kann Betrachtungs- N bzw. Refinementfenstergröße R festgelegt werden. Dabei wird ein interner Puffer mit der Größe N angelegt, um die nötige Anzahl der letzten Frames zu halten.

Trifft ein neuer Frame ein, wird dieser im internen Puffer abgelegt und der älteste aus diesem entfernt und in die ausgehende Warteschlange gelegt. Das Refinement wird dann auf die N Frames im Puffer angewendet.

Durch die automatische Differenzierungsfunktion des Ceres-Solvers können die Iterationsschritte effizient berechnet werden. Derzeit können auf bis zu $N = 6$ und $R = 5$ Frames optimiert werden.

5.2.5 Pipeline-Ränder

Um die Frames aus den Kamerabildern und IMU-Messungen zu erzeugen, wird zu Beginn der Pipeline eine weitere Stufe benötigt. Die erste Stufe kapselt die Pipeline gegenüber dem ROS-Schiffs-System. Sie erhält die synchronisierten Kamera- und IMU-Orientierungen, erzeugt die Bildpyramide und einen Frame mit passendem Zeitstempel.

Am Ende der Pipeline werden die verarbeiteten Frames aus der `_outgoingQueue` der Pipeline-Stufe `Refiner` entnommen. Dafür existiert eine eigene Pipeline-Stufe `PipelineEnd`. An dieser Stelle werden die Frames auch zerstört. Die Pipeline-Stufe wartet vor dem Zerstören eines Frames jedoch auf seinen Nachfolger, da mit diesem ein Pointer auf den zu zerstörenden Frame existiert. Sonst wäre die Bedingung, dass eine Pipeline-Stufe auf dem Vorgänger der Frames in ihrem Zuständigkeitsbereich lesen darf, missachtet.

Um die durch das System ermittelte Position der Kamera bzw. des Schiffes zu erhalten, müssen die einzelnen Bewegungsvektoren zwischen den Frames aufaddiert werden. Es eignen sich verschiedene Stellen der Pipeline um die Bewegungsvektoren aufzusummieren. Die beste Positionsschätzung wird durch das Aufsummieren der Bewegungsvektoren am Pipeline-Ende erreicht. Die Schnellste, aber Ungenauste durch das aufsummieren der Bewegungsvektoren nach der Stufe `BaselineEstimator`. Dazu existieren an diesen Pipeline-Stufen zusätzliche Warteschlangen. In diese wird jeweils ein Tripel, bestehend aus Zeitstempel, bisheriger Bewegungsvektor und Orientierung des letzten Frames, abgelegt. Ein weiterer Thread arbeitet jeweils diese Warteschlangen ab und summiert jeweils die Bewegungsvektoren zu den an den verschiedenen Pipeline-Stufen geschätzten Positionen auf. Die Stelle des Systems kapselt den Ausgang des VO-Systems.

5.3 Integration in das Schiffssystem

Durch den Einsatz von ROS ist dieses VO-System als sogenannter ROS-Node umgesetzt. ROS setzt die Kommunikation über das Publish-Subscribe-Pattern um. Die benötigten Sensordaten erreichen mit passendem Zeitstempel das System und die berechnete Position kann in das System veröffentlicht werden.

5.3.1 Koordinatentransformation

Da die VO im Kamerakoordinatensystem arbeitet, müssen die IMU-Daten am Eingang und die berechneten Positionsdaten entsprechend transformiert werden. In dieser Arbeit beziehen sich die Rotationsmatrizen der einzelnen Kamerapositionen auf das Kameraweltkoordinatensystem w. Dieses ist jedoch in Kamerakoordinaten. Das Schiff arbeitet nach ROS-Konvention im Schiffskoordinatensystem s. Die Matrix T_w^s transformiert Punkte vom Kameraweltkoordinatensystem in das Schiffskoordinatensystem:

$$T_w^s = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \tag{5.1}$$

Die durch das Verfahren berechneten Positionen P_w werden daher wie folgt aus dem Kameraweltkoordinatensystem in das Schiffskoordinatensystem transformiert:

$$P_s = T_w^s * P_w \tag{5.2}$$

Die IMU-Orientierungsmatrix kann als Matrix R_s^s, die vom Schiffskoordinatensystem ins Schiffskoordinatensystem transformiert, betrachtet werden. Um die Orientierungsmatrix im Kameraweltkoordinatensystem auszudrücken, wird folgende Rechnung durchgeführt:

$$R_w^w = (T_w^s)^T * R_s^s * T_w^s \tag{5.3}$$

5.3.2 Kamera-Kalibrierung

Um die Kameramatrix und die Koeffizienten der Linsenverzeichnung zu ermitteln, wird die in ROS und OpenCV implementierte Methode von Zhang [2000] genutzt. Dabei wurde mithilfe unterschiedlich großer Schachbrettmuster die Kalibrierung durchgeführt.

5.3.3 IMU-Kalibrierung

Die IMU wurde in der Arbeit von Schnirpel [2019] kalibriert. Jedoch hat es sich herausgestellt, dass nach einiger Zeit oder nach mechanischen Belastungen, die Offsets der Daten neu kalibriert werden müssen. Dazu werden 100 Sekunden lang die Daten der einzelnen

Achsen aufgenommen und der daraus resultierende Mittelwert als neuer Offset gespeichert. Dieser wird dann von den Sensorwerten abgezogen. Um aus den Beschleunigungen und Winkelgeschwindigkeiten die Orientierung zu berechnen, kommt der in ROS bereits umgesetzte Komplementärfilter [Valenti u. a., 2015] zum Einsatz.

6 Evaluationskonzept

In den folgenden Kapiteln wird das angepasste VO-Verfahren evaluiert. Neben der Bewertung der Trajektorie wird die Einsatzfähigkeit auf der Hardware des autonomen Schiffes anhand der benötigten Rechenzeit bewertet. Dabei werden für die Tests unter den verschiedenen Parametern des Verfahrens wiederholt, um deren Auswirkung auf die Qualität zu ermitteln.

Zunächst werden in diesem Kapitel die Bewertungsmetriken erläutert. Dann wird auf die Testumgebungen, Testfahrten und Parameter eingegangen. Im nächsten Kapitel werden die Ergebnisse präsentiert, analysiert und bewertet.

6.1 Bewertungsansatz

Im Folgenden werden verschiedenen Metriken eingeführt. Anhand dieser wird zum einen die ermittelte Trajektorie bewertet. Zum anderen bewerten diese Außenfaktoren, welche die VO nicht beeinflussen kann. Diese könnten aber Auswirkungen auf die Qualität des Verfahrens haben. Um die relative Skalierung getrennt von den Richtungen zu bewerten, werden diese getrennt betrachtet. Die gesamt Trajektorie setzt sich aus den einzelnen Bewegungsvektoren zusammen.

6.1.1 Richtungsvektoren

Die einzelnen Bewegungsvektoren werden mit Ground-truth verglichen. Ein einzelner durch die VO ermittelter Bewegungsvektor $\vec{b}^t_{vo}$ beschreibt die Bewegung zwischen zwei Frames zu den Zeitpunkten t und $t-1$. Anhand der Zeitstempel der Frames kann aus den zugehörigen Ground-truth-Positionen der passende Ground-truth-Vektor $\vec{b}^t_{gt}$ berechnet werden.
Zur Bewertung wird der Winkel zwischen den Vektoren berechnet. Ein perfekter Vektor

hätte dann einen Unterschied von 0°. Aus der Trajektorie einer Testfahrt wird der mittlere absolute Fehler (MAE) und die Standardabweichung berechnet. Diese Bewertung betrachtet somit nur die normierten Vektoren und erfolgt unabhängig von der Skalierung zu vorherigen Bewegungsvektoren.

6.1.2 Skalierungen

Um die relative Skalierung zu bewerten, wird der sogennante Skalierungsfaktor λ zwischen den Bewegungsvektoren der VO und Ground-truth für jedes Vektorpaar berechnet:

$$\lambda_t = \frac{||\vec{b}_{gt}^t||}{||\vec{b}_{vo}^t||} \tag{6.1}$$

Als Fehlermaß wird die Skalierungsabweichung als Verhältnis berechnet. Diese berechnet sich für einen einzelnen Vektor wie folgt:

$$C = \frac{\left| ||\vec{b}_{vo} * \lambda_{\text{gesamt}}|| - ||\vec{b}_{gt}|| \right|}{||\vec{b}_{gt}||} \tag{6.2}$$

Dabei ist λ_{gesamt} ein Skalierungsfaktor, der auf alle Vektoren einer Testfahrt angewendet wird. Dazu wird der Median aller Skalierungsfaktoren berechnet. So wird der best möglichste Skalierungsfaktor betrachtet. Dieser Fehler bewertet so also die Längenabweichung als Anteil der Ground-truth-Vektorlänge. Auch hierfür werden MAE und Standardabweichung bestimmt.

6.1.3 Positionsabweichung

Die absolute Positionsabweichung entspricht den aufsummierten Fehlern der Bewegungsvektoren. Hier kann also die absolute Positionsdrift erkannt werden. Um auch in der absoluten Positionsabweichung die Fehler durch falsche Skalierung getrennt von dem Fehler durch falsche Vektorrichtungen zu ermitteln, werden zwei Positionsabweichungen berechnet:
In der ersten Variante werden die einzelnen Bewegungsvektoren mit den jeweils aus dem Ground-truth berechneten Skalierungsfaktoren skaliert und summiert. Für den Zeitpunkt

N ergibt sich dann eine Abweichung d_r:

$$d_r(N) = \sum_{t=0}^{N} \lambda_t \cdot \vec{b}_{vo}^{t} - S_{gt}^{N} \tag{6.3}$$

S_{gt}^{t} ist die Ground-truth Position zum Zeitpunkt t.

Diese Variante bewertet somit nur die Positionsabweichung, die durch falsche Vektorrichtungen entsteht. Im Gegensatz zum Richtungsfehler betrachtet dieser Fehler aber die Komposition aller Vektoren. In der zweiten Variante werden alle Bewegungsvektoren, wie oben, mit dem selben Faktor skaliert. Daraus ergibt sich die Abweichung d_a:

$$d_r(N) = \lambda_{\text{gesamt}} \cdot \sum_{t=0}^{N} \vec{b}_{vo}^{t} - S_{gt}^{N} \tag{6.4}$$

Auch hier wird λ_{gesamt} aus dem Median aller Skalierungsfaktoren bestimmt. Diese Variante berücksichtigt also auch die Skalierungen.

Daneben eignet diese sich auch in Szenarien, in denen kein Ground-truth vorhanden ist. Das Schiff kann seinen eigenen Startpunkt wieder anfahren und so der Ringschluss geprüft werden.

Wenn die Bewegungsvektoren am jeweiligen Ground-truth-Vektor skaliert werden, wird die sich ergebene Trajektorie aus den Vektoren im Folgenden auch „perfekt“ skaliert gennant.

6.1.4 Aufwand

Der Aufwand der VO wird durch die benötigte Berechnungszeit bestimmt. Dabei werden für jeden einzelnen Frame zwei Werte gemessen. Der Wert „Baseline-Estimation“ gibt die benötigte Zeit eines Frames für das Feature-Detection, Tracking und die Baseline-Estimtation an. Im System befindet sich zwisschen den Pipeline-Stufen `TrackerDetector` und `BaselineEstimator` noch die Piepeline-Stufe `Merger`. Da im `Merger`, wie oben beschrieben, die Frames ersetzt werden, wird die Wartezeit im `Merger` nicht in diese Berechnung einbezogen. Der Berechnungsaufwand jedoch schon. Der Wert „Refiner“ gibt die Rechenzeit eines Frames in der Pipelinestufe `Refiner` an. Dabei wird für jeden Frame die Zeit nur einmal, nämlich wenn dieser der neueste Frame im `Refiner` ist, berechnet. Die Pipeline-Latenz wird durch diese Messungen nicht betrachtet.

6.1.5 Schaukeln

Um den Einfluss des Rollens und Stampfens auf die Qualität der VO zu ermitteln, werden die Orientierungsdaten fouriertranformiert. So lassen sich die Frequenzanteile des Schaukelns und deren Stärke der Achsen ablesen und vergleichen.

6.1.6 Rotationen

Um bei möglichen Fehlern der VO den Rotationsfehler aus der IMU zu berücksichtigen, kann bei vorhandenem Ground-truth n die Abweichung bestimmt werden. Dazu wird die Abweichung der Orientierung auf der z-Achsen in Grad berechnet.

6.2 Szenarien und Tests

Um möglichst vergleichbare Werte für die verschiedenen Parameter des Verfahrens zu erhalten, wurden während der Testfahrten die Sensordaten nur aufgezeichnet. ROS bietet einen Mechanismus, um diese während der Auswertung wieder in das System einzuspielen.

6.2.1 Labortests

Abbildung 6.1: Testaufbau im Labor mit Wagen im Vordergrund

Für die meisten der oben beschriebenen Metriken ist ein Ground-truth notwendig. Das Labor /* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */ (//CSTI) [von Luck u. a., 2019] der HAW besitzt eine etwa 4 m mal 4 m große Trackinganlage. Diese bestimmt die absolute Position und Orientierung von sogenannten Tracking-Markern. Für die Testversuche wurde ein Marker direkt an der Kamera befestigt und dessen Position als Ground-truth angenommen. Um genügend gute Features zu erkennen, wurden verschiedene Objekte und Schachbrettmuster in diversen Größen im Bereich der Anlage positioniert. In Abbildung 6.1 ist der Aufbau zu erkennen.

Es wurden in der Anlage zwei Testfahrten aufgezeichnet. Zunächst eine ruhige Fahrt, während der die Schiffsplattform auf einem Wagen geschoben wurde. Der Wagen ist im Vordergrund der Abbildung zu erkennen. Diese Testfahrt dauert 180 s und es wurden Kamerabilder mit etwa 6.5 fps aufgenommen.

Als Zweites wurde eine Fahrt mit Schaukeln aufgezeichnet. Um das natürliche Schiff-schaukeln zu simulieren wurde die Plattform durch die Anlage getragen. Diese Testfahrt dauert 100 s und es wurden Kamerabilder mit etwa 6.5 fps aufgenommen.

6.2.2 Miniatur Wunderland

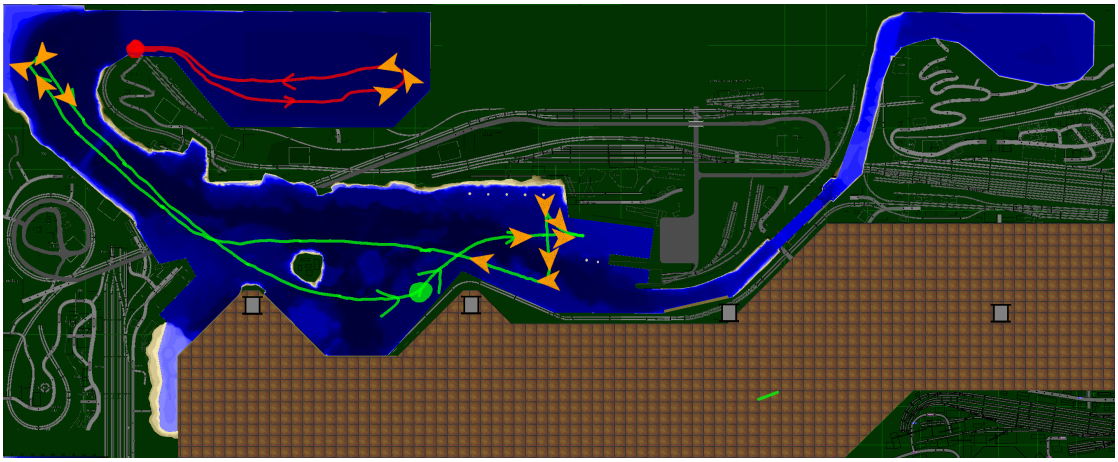


Abbildung 6.2: Skizze der Testfahrten im Miniatur Wunderland

Zum Zeitpunkt dieser Arbeit ist die Schiffs-Trackinganlage im Miniatur Wunderland ausgefallen. Daher existieren hier nur Testfahrten ohne Ground-truth. Um dennoch die ermittelte Trajektorie zu bewerten, wurden zwei Testfahrten aufgezeichnet, bei denen das Schiff seine Startposition wieder erreicht. Abbildung 6.2 skizziert die Route dieser Fahrten.

In Rot die Fahrt im Schattenhafen, in Grün die Fahrt durch den Ausstellungsbereich. In Orange die Schiffsausrichtungen.

Die erste Testfahrt (rot) führt durch den sogenannten Schattenhafen. Dies ist einer von den Besuchern nicht sichtbarer Bereich des Wasserbeckens an dem die Schiffe liegen, wenn sie nicht im Besucherbereich fahren. Hier ist daher mit keinem negativen Einfluss durch Besucher im Kamerabild zu rechnen. Diese Testfahrt dauert 500 s und es wurden Kamerabilder mit etwa 3.5 fps aufgenommen.

Die zweite Fahrt führt durch den für die Zuschauer sichtbaren Bereich des Wasserbeckens. Darüber hinaus befinden sich hier neben den Besuchern andere bewegte Objekte, wie andere Schiffe oder Autos und Züge an Land im Bild. Diese Testfahrt dauert 1050 sec und es wurden Kamerabilder mit ebenfalls 3.5 fps aufgenommen.

Um die ermittelte Trajektorie vergleichen zu können und einen durchschnittlichen Skalierungsfaktor berechnen zu können, wird auf diesen Testfahrten zusätlich das LIDAR-SLAM Hector-SLAM von Kohlbrecher u. a. [2011] eingesetzt.

6.2.3 Parameter

Im Folgenden sollen die Parameter, deren Einfluss auf das Verfahren getestet werden soll, kurz aufgelistet werden. Außerdem können, um den Rahmen dieser Arbeit nicht zu übersteigen, einige Parameter nicht berücksichtigt werden und sind fix. Auch diese werden kurz erwähnt.

Im Haris-Corner-Detector wird der Corner-Response-Schwellwert, ab dem Features vom Verfahren benutzt werden untersucht. Hier und im LK-Tracker wird außerdem der Mindestabstand, den die Features haben müssen, untersucht. Für die Fenstergröße des Haris-Corner-Detector wird der Standardwert 3 und für den Parameter k der Standardwert 0.04 benutzt.

In [Bouguet u. a., 2001] zeigt der Autor, dass bei einer Pyramidentiefe von 3 schon ausreichend große Bewegungen abdeckt und ein relativ kleines Fenster[1] ermöglicht. Deshalb werden hier die OpenCV-Standardwerte genutzt. Die Pyramidenhöhe beträgt 3 und die Fenstergröße $21x21$.

[1] Hier ist das in Abschnitt 2.4 erklärte Tracker-Fenster gemeint.

Die Tests werden unter einer `Meger`-Frequenz von 1 Hz durchgeführt. Als Mindestdisparität wurde $cos(1°)$ festgelegt.

Für die Ausreißer-Erkennung wird die MLESAC-Grenze untersucht.

Beim Refinement werden die Betrachtungs- und Refinementfenstergröße unersucht. Außerdem werden die Längengrenzen der erweiterten Parametrisierung der Kostenfunktion betrachtet. Eine Auswirkung auf die Rechendauer hat die Anzahl der Iterationen und das Konvergenzkriterium des NLLSQ-Lösungsalorithmus.

Um neben der Wahl des Betrachtungs- und Refinementfensters die Auswirkungen der Erweiterungen aus Abschnitt 4.4 zu Testen, wird das Verfahren mit und ohne diese getestet.

Um besser differenzieren zu können, wird das Verfahren einmal nur mit Baseline-Estimation und einmal mit Baseline-Estimation und Refinement getestet.

Der Aufwand wird durch verschieden Faktoren beeinflusst. Dazu gehören im Wesentlichen das Tracking, die Fenstergrößen im Refinement, die maximale Anzahl an Iterationen im Refinement und die Konvergenzkriterien[2] des Refinements. Hier soll nur der Einfluss des Refinements über die Refinement- und Betrachtungsfenstergröße, der Anzahl der Refinement-Iterationen und der Konvergenzkriterien auf die Laufzeit des Verfahrens getestet werden.

Bei allen Testfahrten beträgt das KameraBinning $(x, y) = (2, 2)$. Damit beträgt die Auflösung 540×960 Pixel.

6.2.4 Testdurchführung

Die Tests wurden mit den aufgenommenen Daten auf dem Testsystem durchgeführt. Außer für die Zeitmessungen handelt es sich um ein `Ubuntu-18.04-System` auf einem `Intel Core i7-8650U` mit `16 GB` Ram. Um möglichst aussagekräftige Ergebnisse zu erhalten, wurden in allen Tests mindestens fünf Durchläufe pro Test gemacht. Auf die Abweichung bei mehreren Testläufen wird in Abschnitt 7.6 eingegangen.

[2] Konvergenzkriterien sind dabei *Funktionstoleranz*, *Gradientoleranz* und *Parametertoleranz*. Liegt eine der Toleranzen unter dem vorgegebenen Wert, nimmt das iterative Lösungsverfahren die nicht lineare Kostenfunktion als ausreichend gelöstes Problem an. (Mehr dazu in [Agarwal u. a.])

Unter Veränderung der Parameter wurde für die Testfahrten im Labor versucht, das beste Ergebnis zu erzielen. Mit diesen Einstellungen wurden dann die Fahrten im Miniatur Wunderland getestet.

Um den Aufwand zu bestimmen, wurde das System unter höchsten Compilerverbesserungen und ohne Debugbilder-Ausgaben übersetzt. Daraufhin wurde es auf dem `Raspberry PI 4 - 4GB` mit `Raspbian-Buster` gestartet. Während der Ausführung wurden manuell keine anderen Programme auf dem Pi ausgeführt.

Es kamen unter den verwendeten Bibliotheken die Versionen `ROS-Melodic-Morenia`, `OpenCV 4.1.0`, `ceres-solver-1.14.0` und `eigen-lib-3.3.4` zum Einsatz.

7 Evaluationsergebnisse, Analyse und Beurteilung

In den folgenden Abschnitten werden die Testergebnisse für die Tests ausgewertet.

Zunächst werden die Ergebnisse für die nicht vom Verfahren beeinflussbaren Faktoren beschrieben. Im Anschluss folgen die Ergebnisse der Untersuchung der beschriebenen Parameter auf den Labortestdaten. Dabei werden die einzelnen Verfahrensteile getrennt untersucht und die Auffälligkeiten analysiert. Darauf folgend werden die Ergebnisse für die, unter den ermittelten Parametern, durchgeführten Tests auf den Daten aus dem Miniatur Wunderland dargestellt und analysiert.

Abschließend folgen die Ergebnisse der Aufwands- und Abweichungsuntersuchung.

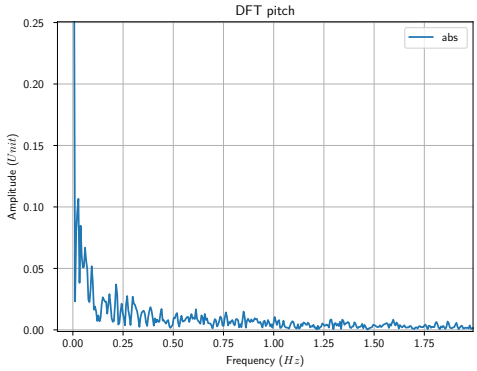
Während der Darstellung und Analyse findet auch die Bewertung statt.

7.1 Schaukeln

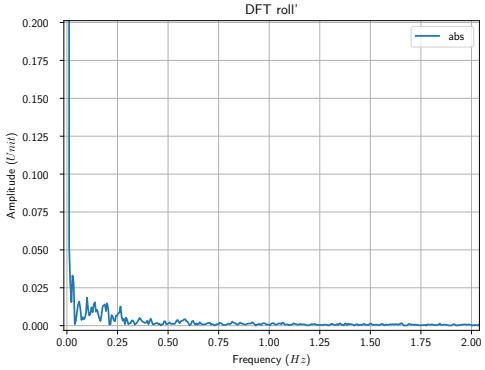
In der Abbildung 7.1 sind die Fouriertransformationen der Schaukelachsen für die verschiedenen Testfahrten dargestellt. Die Abbildungen wurden alle auf die sichtbaren Teile der Kurven skaliert. Die Laborfahrt ohne Schaukeln weist für beide Achsen (Abbildung 7.1a und Abbildung 7.1b), außer dem Offset-Anteil, keine besonders hohe Amplitude für eine Frequenz auf. Im Gegensatz dazu zeigt Abbildung 7.1c eine deutliche Amplitude bei 0.25 Hz für die Stampfachse der Laborfahrt mit simuliertem Schaukeln auf. Durch das Tragen sind dazu noch weitere niedrige Frequenzanteile erhöht. Ähnliches zeigt Abbildung 7.1d für die Rollachse dieser Fahrt. Im Bereich 0 Hz–0.25 Hz treten starke Amplituden auf. Im Vergleich dazu zeigt Abbildung 7.1e keine nennenswerte periodische Bewegung für die Stampfachse. Abbildung 7.1f zeigt eine Rollfrequenz von 0.5 Hz. Die Amplitude ist aber etwa um den Faktor 5 geringer, als die Rollbewegung der Laborfahrt.

Schwingungen mit dieser Amplitude existieren in der Laborfahrt bis zu einer Frequenz von 0.75 Hz.

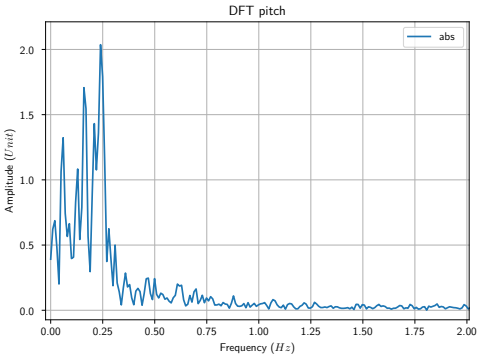
Das Stampfen des Schiffes ist in der reellen Umgebung also nur sehr gering. Rollen tut es leicht mit 0.5 Hz. Im Vergleich dazu ist das simulierte Schaukeln auf beiden Achsen zu hoch. Außerdem wurden durch das Tragen nicht nur eine, sondern mehrere Schwingungen verursacht. Im späteren Vergleich der Qualität des Verfahrens sind die Fahrten im Miniatur Wunderland also eher der Laborfahrt ohne Schaukeln zuzuordnen.



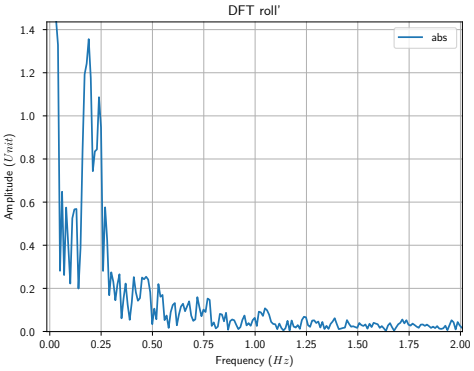
a) Stampfen der Laborfahrt ohne Schaukeln



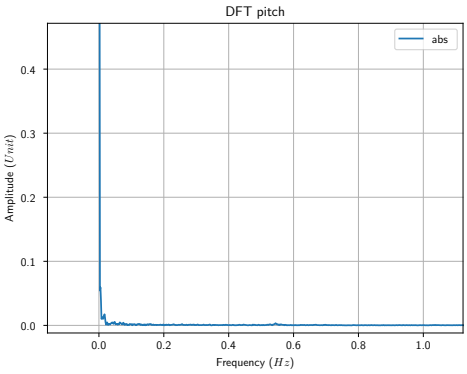
b) Rollen der Laborfahrt ohne Schaukeln



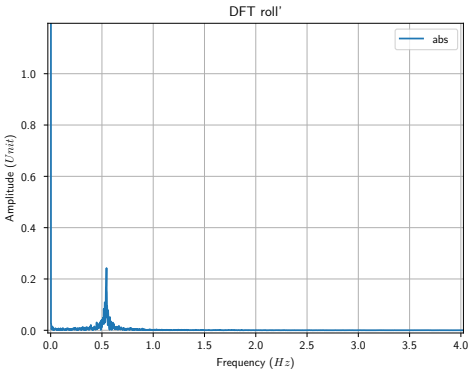
c) Stampfen der Laborfahrt mit Schaukeln



d) Rollen der Laborfahrt mit Schaukeln



e) Stampfen der Fahrt im Schattenhafen des Miniatur Wunderlandes



f) Rollen der Fahrt im Schattenhafen des Miniatur Wunderlandes

Abbildung 7.1: Fouriertransformierte Roll- und Stampfachse bei den verschiedenen Testfahrten

7.2 Rotationsdrift

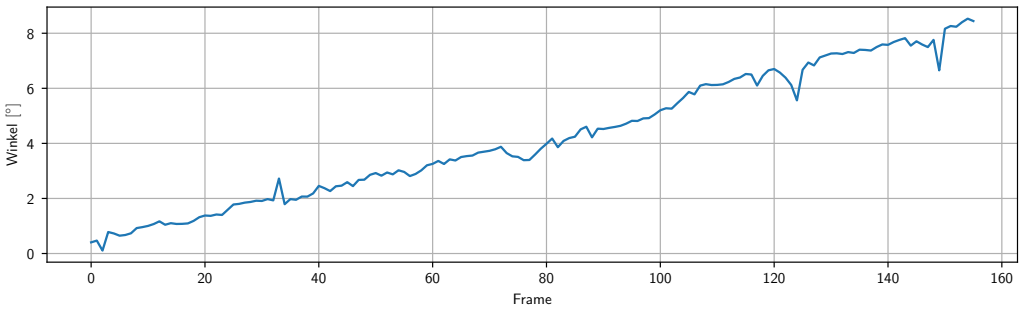


Abbildung 7.2: Yaw-Abweichung der IMU

In den Labortests wurde eine Yaw-Drift festgestellt. Abbildung 7.2 zeigt die Abweichung der IMU-Orientierungswerte und dem Ground-truth für die Fahrt ohne Schaukeln. Nach 150 Frames beträgt die Drift etwa 8°. Die Anzahl von 150 Frames entspricht bei einer `Merger`-Frequenz von 1 Hz eine Dauer von 150 sec. Die Drift verhält sich unabhängig von den gefahrenen Kurven linear. Daher kann von dem Plot ungefähr 3 deg /min abgelesen werden.

Da das Verfahren immer nur auf den letzten N Frames arbeitet, sollte sich diese Drift auf die lokalen Bewegungsvektoren nicht besonders negativ auswirken. Auf eine gesamte Trajektorie einer längeren Fahrt kann die Drift jedoch deutlich sichtbar werden.

7.3 Labortests

7.3.1 Features

Feature-Qualität

Abbildung 7.3 zeigt den durchschnittlichen Winkelfehler bei unterschiedlichem Haris-Corner-Response. In der OpenCV-Implemenation bezieht sich dieser Schwellwert prozentual auf die Corner-Response des besten Feature [OpenCV, 2019]. Neben dem durchschnittlichen Winkelfehler über alle Bewegungsvektoren dieser Fahrt ist die Standardabweichung der Fehler geplottet. Der geringste Fehler liegt etwa bei 5 %. Wie in Abschnitt 2.3 erklärt, werden die neu erkannten Features nach dem Corner-Response R sortiert. Das

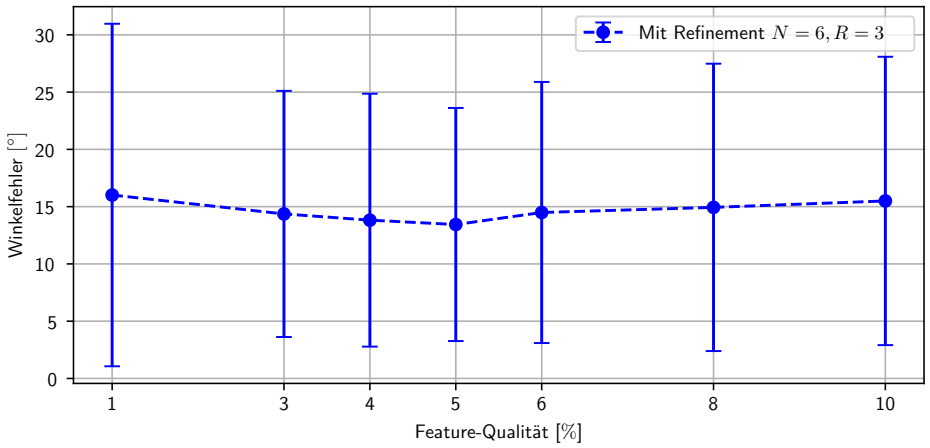


Abbildung 7.3: Durchschnittlicher Winkelfehler der Fahrt mit Schaukeln bei unterschiedlicher Feature-Anzahl

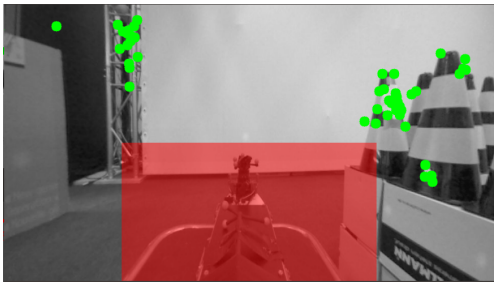
Verfahren entnimmt die Features mit dem Corner-Response über dem Schwellwert. Ist der Schwellwert zu niedrig, werden keine geeigneten Features im Verfahren verwendet und verschlechtern die Ergebnisse. Bei zu hohem Schwellwert sind zu wenige Features vorhanden.

Es wurde so der Wert 5 % ermittelt. Dieser wird für alle weiteren Tests verwendet. Abbildung 7.7 zeigt, dass bei diesem Wert im Schnitt 80–100 Features getrackt werden.

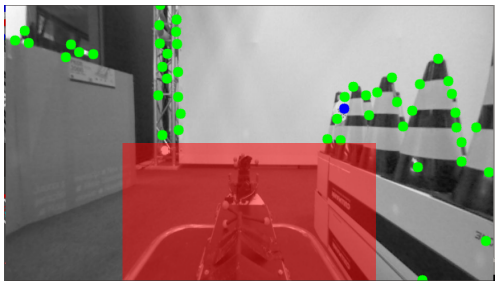
Feature-Verteilung

Abbildung 7.4 zeigt die getrackten Features bei zwei verschiedenen Werten für den Mindestabstand der Features in zwei Situationen der Fahrt ohne Schaukeln. Der Mindestabstand bezieht sich dabei prozentual auf die diagonale der Kamerabilder. Bei zu geringem Abstand häufen sich die Features an bestimmten Bereichen, weil dort die höchste Corner-Response erreicht wird. Dafür werden dann Features die nur einen wenig geringeren Corner-Response haben, dafür aber weit weg von einer Anhäufung liegen, „überschattet“ und nicht getrackt. Dies führt zu einer schlechten Feature-Verteilung.

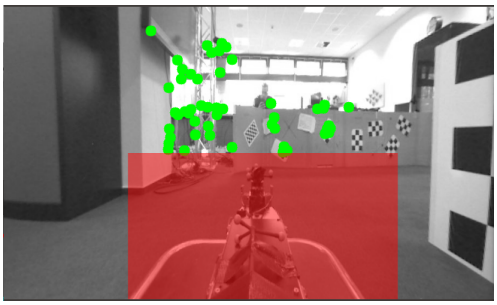
Abbildung 7.5 zeigt, dass eine schlechte Feature-Verteilung den Fehler erhöht. Bei einem Mindestabstand von 1 % wird der beste Wert erreicht. Zu hohe Mindestabstände führen



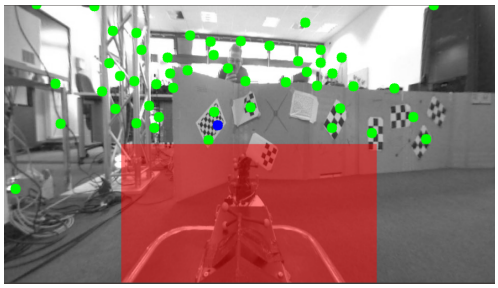
a) Situation 1 mit Feature-Abstand 0.05 %



b) Situation 1 mit Feature-Abstand 0.2 %



c) Situation 2 mit Feature-Abstand 0.05 %



d) Situation 2 mit Feature-Abstand 0.2 %

Abbildung 7.4: Zwei Situationen mit unterschiedlicher Feature-Verteilung

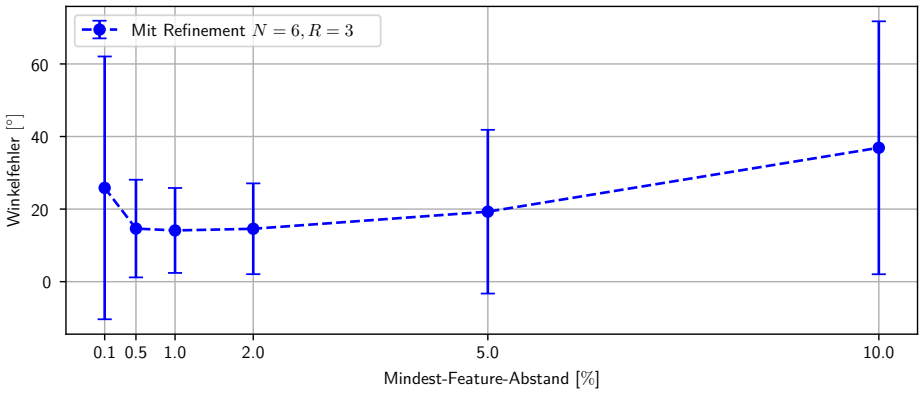


Abbildung 7.5: Durchschnittlicher Winkelfehler bei unterschiedlichem Feature-Mindestabstand für die Fahrt mit Schaukeln

dazu, dass zu wenig Features getrackt werden können und verschlechtern ebenfalls den Wert. Im allen anderen Tests wird daher ein Mindestabstand von 1 % genutzt.

7.3.2 Baseline-Estimation

Fahrt ohne Schaukeln

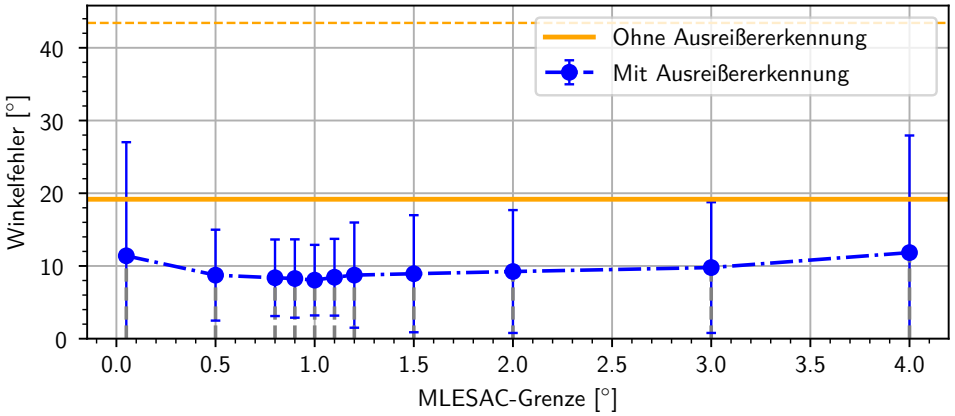


Abbildung 7.6: Durchschnittlicher Winkelfehler und Standardabweichung der Richtungsvektoren bei unterschiedlichen MLESAC-Grenzen in der Laborfahrt ohne Schaukeln

Abbildung 7.6 zeigt den durchschnittlichen Winkelfehler der, nur durch die Baseline-Estimation berechneten, Richtungsvektoren bei verschiedenen MLESAC-Grenzen. Die orangefarbene Gerade zeigt den Fehler, wenn auf MLESAC verzichtet und die Richtung aus allen Feature-Korrespondenzen, ohne die Berücksichtigungen von Ausreißern, berechnet wird. Der durchschnittliche Fehler beträgt dann ca. 19°. Dabei treten jedoch, wie die Standardabweichung zeigt, starke Schwankungen auf. Die orangefarbene, gestrichelte Gerade zeigt die Standardabweichung über die einzelnen Vektoren der Testfahrt-Trajektorie. Demnach beträgt diese etwa 22°.

Wie oben erwähnt, empfiehlt der Autor des Verfahrens eine MLESAC-Grenze von 7° bis 13°. In dieser Testfahrt befindet sich das Minimum bei 1°. Hier beträgt der durchschnittliche Fehler noch ca. 9.1°. Die Standardabweichung beträgt ca. 5°. Bei niedrigeren Grenzen als 1° steigt der durchschnittliche Fehler wieder. Hier ist das Consensus-Set zu klein. Wie in Abbildung 7.7 zu erkennen, werden zu viele Feature aussortiert, die wichtige Bewegungsinformationen enthalten.

Abbildung 7.8 zeigt die ermittelte Trajektorie bei verschiedenen MLESAC-Grenzen. Ohne die Ausreißererkennung und bei hoher MLESAC-Grenze sind starke Richtungsfehler

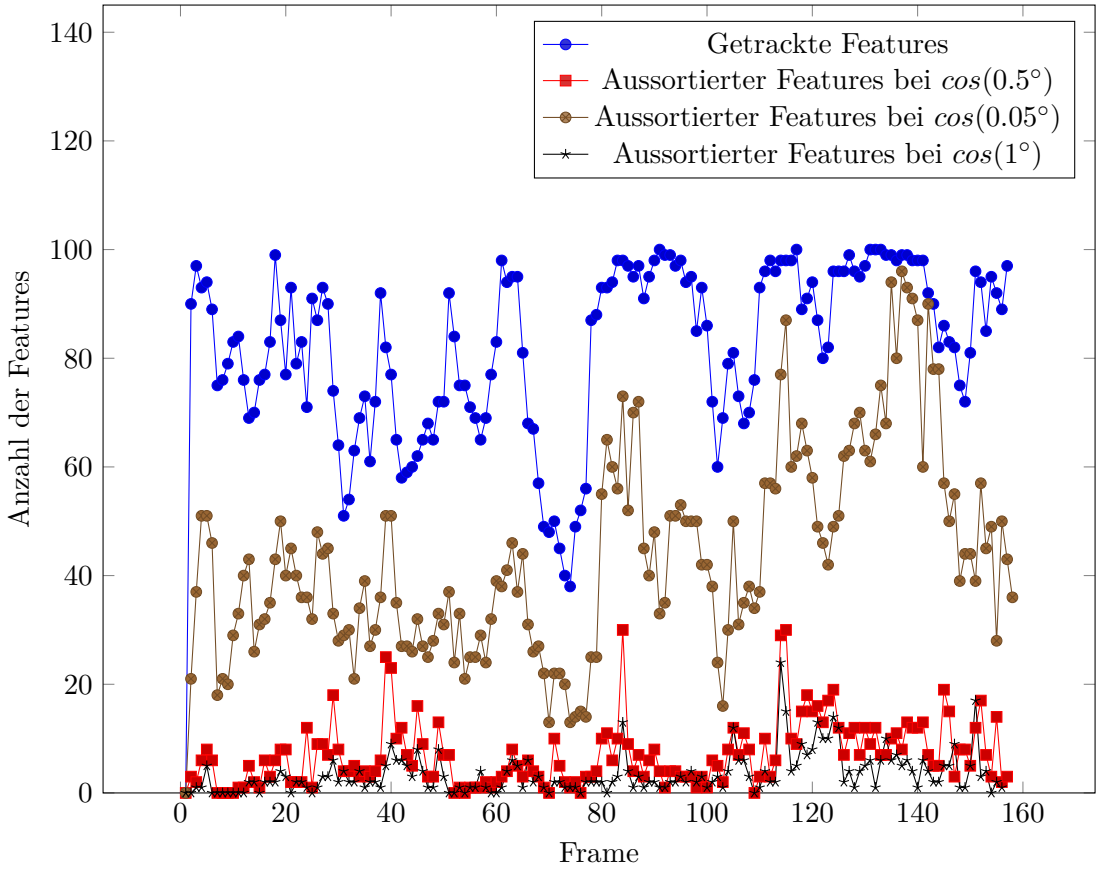


Abbildung 7.7: Die Anzahl der aussortierten Features im Vergleich mit der Anzahl der getrackten Features bei unterschiedlichen MLESAC-Grenzen in der Laborfahrt ohne Schaukeln

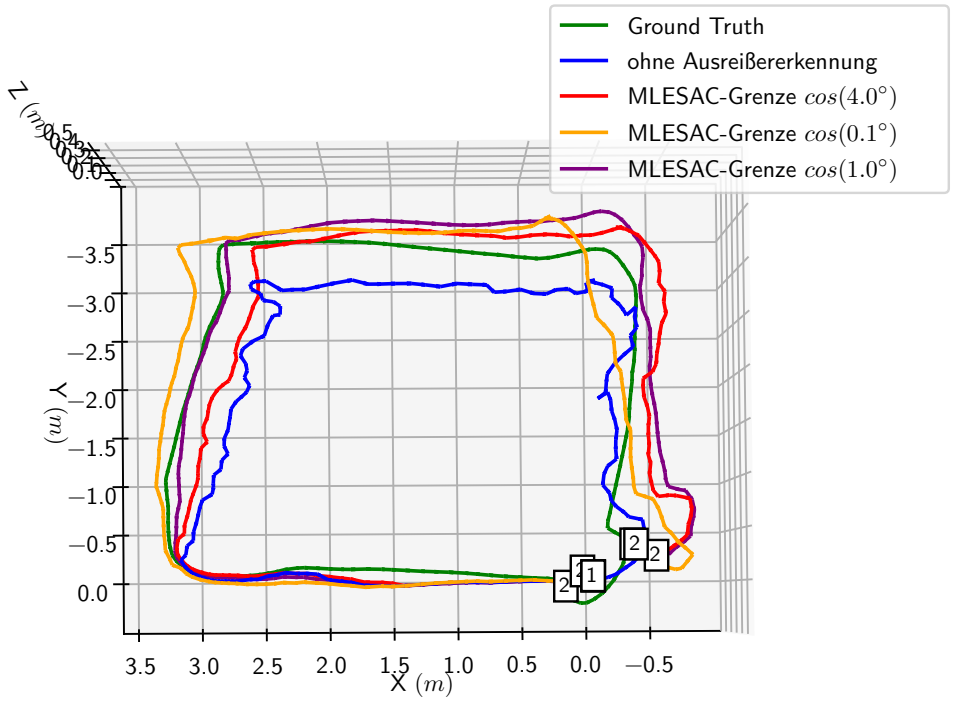


Abbildung 7.8: Durch die Baseline-Estimation ermittelten, anhand des Ground-truth perfekt skalierte, Trajektorie der Laborfahrt ohne Schaukel bei verschiedenen MLESAC-Grenzenn

erkennbar. Bei zu strenger Grenze gehen insbesondere im Bereich von Kurven Bewegungsdetails verloren. Durch eine zu strenge MLESAC-Grenze werden hier wichtige Bewegungsinformationen aussortiert.

Für die weiteren Tests dieser Fahrt wird eine Grenze von 1° genutzt.

Fahrt mit Schaukeln

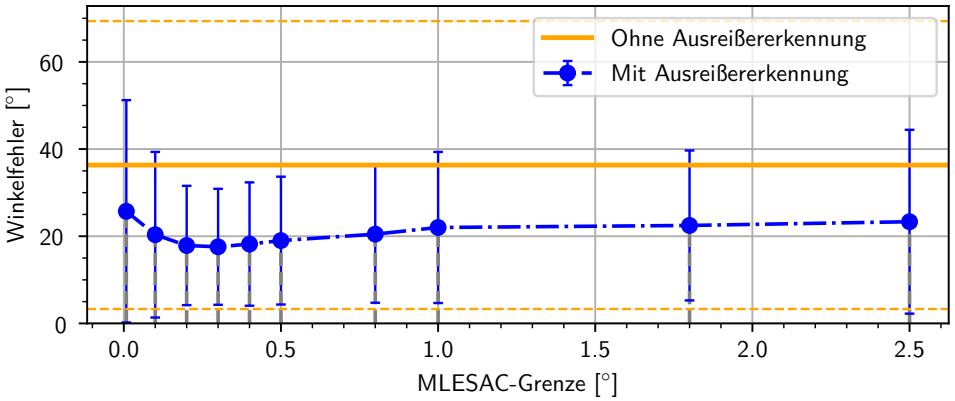


Abbildung 7.9: Durchschnittlicher Winkelfehler und Standardabweichung der Richtungsvektoren bei unterschiedlichen MLESAC-Grenzen der Fahrt mit Schaukeln. Nur unter Einsatz der Baseline-Estimation.

Abbildung 7.9 zeigt den durchschnittlichen Winkelfehler und die Standardabweichung für die Fahrt mit Schaukeln. Ohne Ausreißererkennung ist der durchschnittliche Fehler mit 36° höher als in der Fahrt ohne Schaukeln. Auch die Standardabweichung ist mit etwas 32° höher. Durch den Einsatz von MLESAC verringert sich der durchschnittliche Fehler auf einem Bereich von etwa 18° bis 23°. Auch die Standardabweichung verringert sich auf etwa 20°. Das Minimum befindet sich etwa bei 0.3°.

	Durchnittlicher Winkelfehler
Normal	22.01°
Ohne Negative-Depth-Erkennung	29.30°
Ohne Orientierungsdaten im Tracking	24.95°

Tabelle 7.1: Durchschnittlicher Winkelfehler der Fahrt mit Schaukeln ohne Anpassung bei einer MLESAC-Grenze von 1°

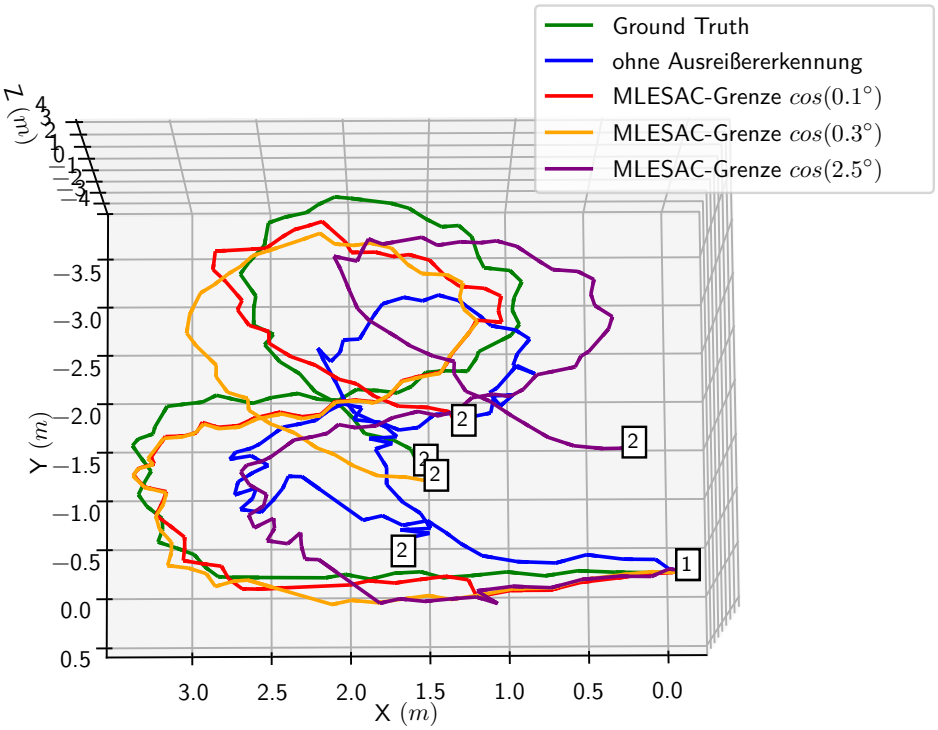


Abbildung 7.10: Durch die Baseline-Estimation ermittelte, anhand des Ground-truth perfekt skalierte, Trajektorie der Fahrt mit Schaukeln bei verschiedenen MLESAC-Grenzen

Abbildung 7.10 zeigt die ermittelte Trajektorie mit aus Ground-truth für jeden Vektor berechneter „perfekter“ Länge bei verschiedenen Grenzen. Es lassen sich die durch das Schaukeln entstandenen Bewegungen erkennen.

Tabelle 7.1 vergleicht den durchschnittlichen Winkelfehler ohne die Anpassungen aus Abschnitt 4.4. Das Nutzen der Rotationsmatrix im Tracking senkt den Fehler um etwa 3.0°. Das Erkennen von negativen Richtungen um etwa 7.3°.

Im Vergleich zur Fahrt ohne Schaukeln, sind die Winkelfehler für die Fahrt mit Schaukeln um etwa 10° höher. Unter Berücksichtigung des viel stärkeren Schaukelns ist dieser Unterschied erklärbar.

7.3.3 Refinement der Vektorrichtungen

Fahrt ohne Schaukeln

Für diese Testfahrt wird im Folgenden die MLESAC-Grenze von 1° verwendet. Abbildung

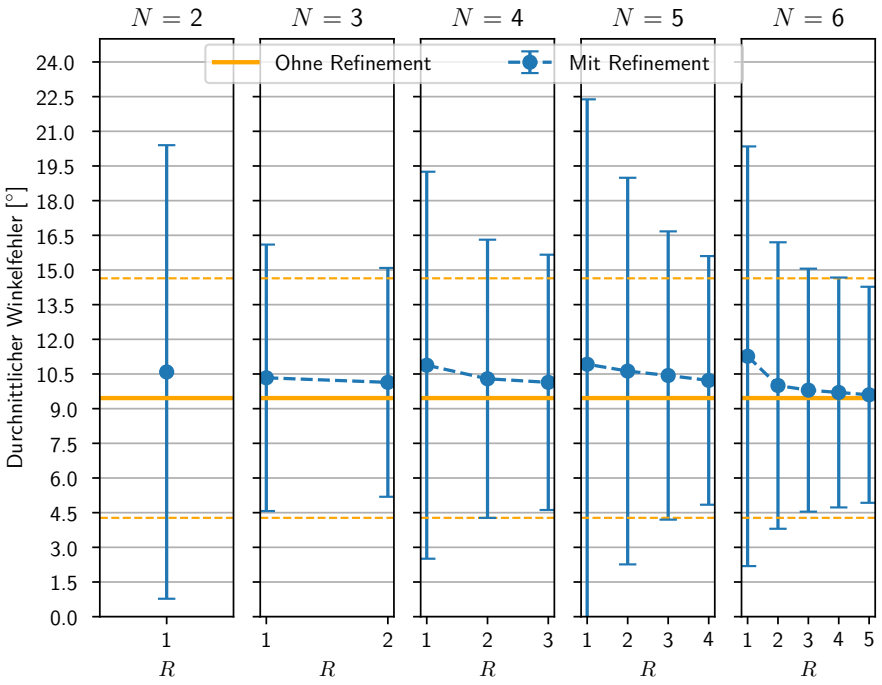


Abbildung 7.11: Durchschnittlicher Winkelfehler der Fahrt ohne Schaukeln bei verschiedener Fenstergrößen

7.11 zeigt den Durchschnittswert und die Standardabweichung der Winkelfehler der Fahrt ohne Schaukeln für verschiedene Fenstergrößen. Eine Fenstergrößen N nimmt dabei einen eigenen Subplot ein. Auf der y-Achse sind die verschiedenen Refinementfenstergrößen R aufgetragen. Es wurde das Refinement für alle in der Implementation möglichen Stufen durchlaufen. Zum Vergleich zeigt die orangefarbene Gerade den Fehler ohne Refinement. Neben den Durchnittswerten (MAE) sind ebenfalls die Standardabweichungen geplottet.

Der durchschnittliche Winkelfehler wird durch Refinement bei dieser Fahrt nicht verbessert. Das Refinement hat sogar für eine minimale Verschlechterung des Wertes gesorgt. Allerdings lässt sich gut erkennen, dass dieser Verschlechterung mit zunehmender Anzahl

an refineten Frames abnimmt. Für die maximale Größe $N = 6, R = 5$ wurde eine minimale Verbesserung der Standardabweichung erzielt.

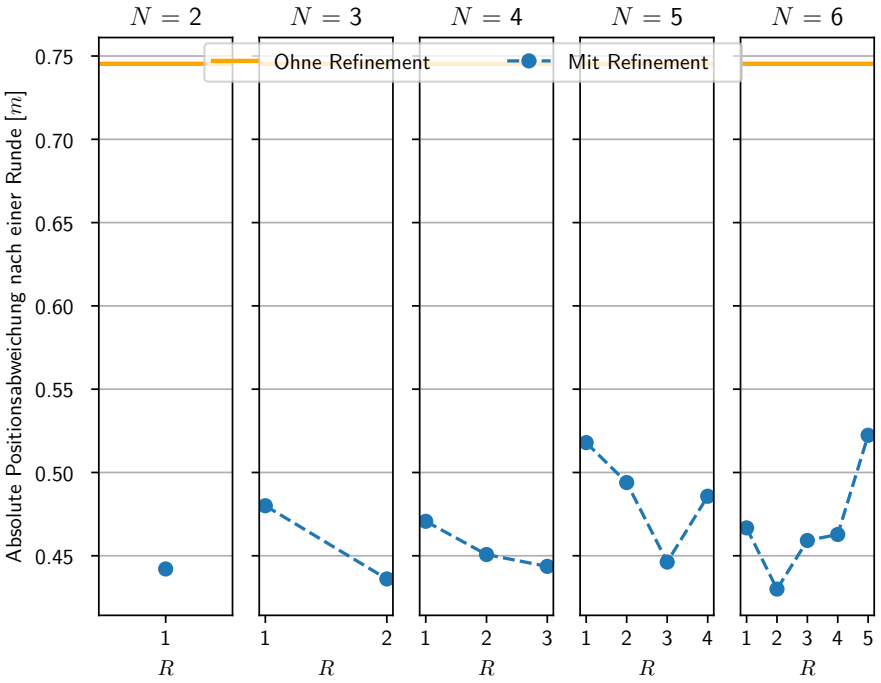


Abbildung 7.12: Positionsfehler der, mit den perfekt skalierten Vektoren, zusammengesetzte Trajektorie am Ende der Fahrt ohne Schaukeln bei verschiedener Fenstergrößen

Die Positionsabweichung, welche sich, unter Annahme der korrekten Längen (perfekt Skaliert), ergibt, zeigt Abbildung 7.12. Durch das Refinement wird insgesamt eine Verbesserung von 0.74 m auf 0.45 m erzielt. In diesem Plot zeigt sich darüber hinaus, dass die Wahl großer Fenster N in Kombination mit großen Refinementfenstern R diesen Wert negativ beeinflusst. Dieses Problem wurde oben bereits in Zusammenhang mit der Vektorlängenberechnung als das „Konsistenz-Problem“ bezeichnet. Bei der ungefähr 15 m langen Testfahrt entspricht eine Verbesserung von ungefähr 0.3 m jedoch nur etwa 0.3 %.

Abbildung 7.13 zeigt die Trajektorien dieser Testfahrt unter Annahme der korrekten Vektorlängen. Im Bereich der engen 90° Kurven verbessert das Refinement die Vektoren nicht optimal. Die Geraden und die weite Kurve am Ende der Fahrt hingegen scheinen verbessert worden zu sein.

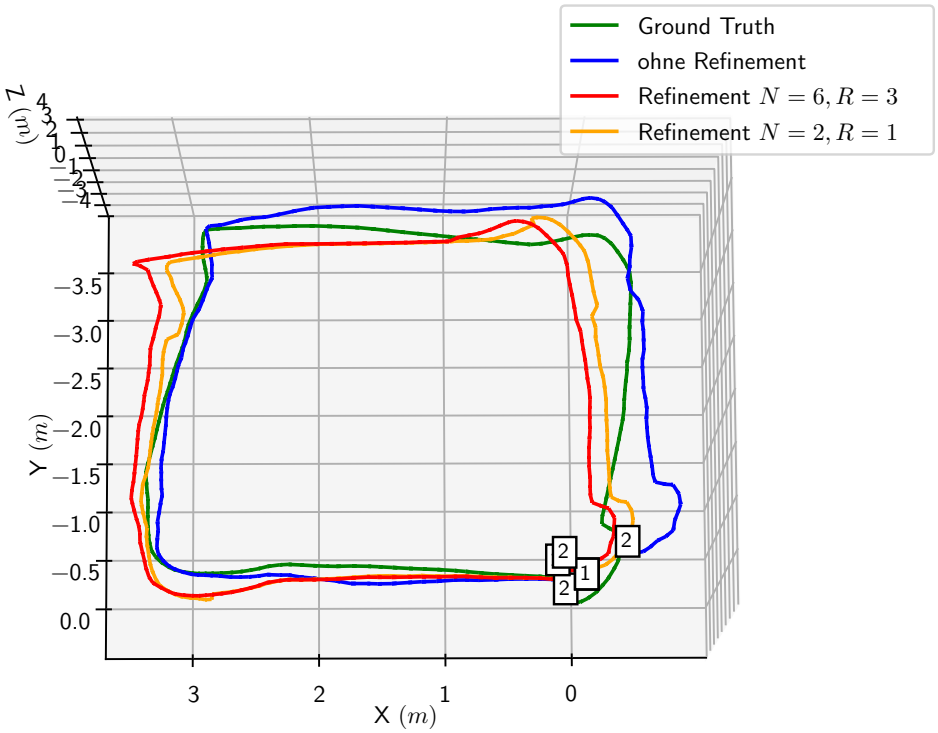


Abbildung 7.13: Trajektorie der Fahrt ohne Schaukeln bei verschiedener Fenstergrößen mit am Ground-truth perfekt skalierten Vektoren

Dies zeigt sich ebenfalls in Abbildung 7.14. Hier sind die Winkelfehler der einzelnen Richtungsvektoren in Kombination mit dem Yaw-Winkel der Kamera geplottet. Die Peeks treten im Bereich von starken Änderungen von Yaw auf. Obwohl hier in der Kurve ohne Refinement keine Peeks sind. Sonst liegen die Fehler der Refinten Vektoren unter den der Fehler ohne Refinement. Hierfür kommen mehrere Gründe in Verdacht. Zum einen zeigt Abbildung 7.7, dass im Bereich der Kurven die Anzahl der getrackten Features sinken und die Zahl der Ausreißer steigt. Oben zeigte sich bereits, dass ein zu geringer Feature Consensus-Set sich negativ auf das Refinement auswirkt.

Zusätzlich sei aber zu erwähnen, dass das Refinement versucht die Trajektorie, bestehend aus mehreren Vektoren, zu verbessern. Diese zeigt, gemessen an der Positionsabweichung der gesamten Trajektorie, eine Verbesserung.

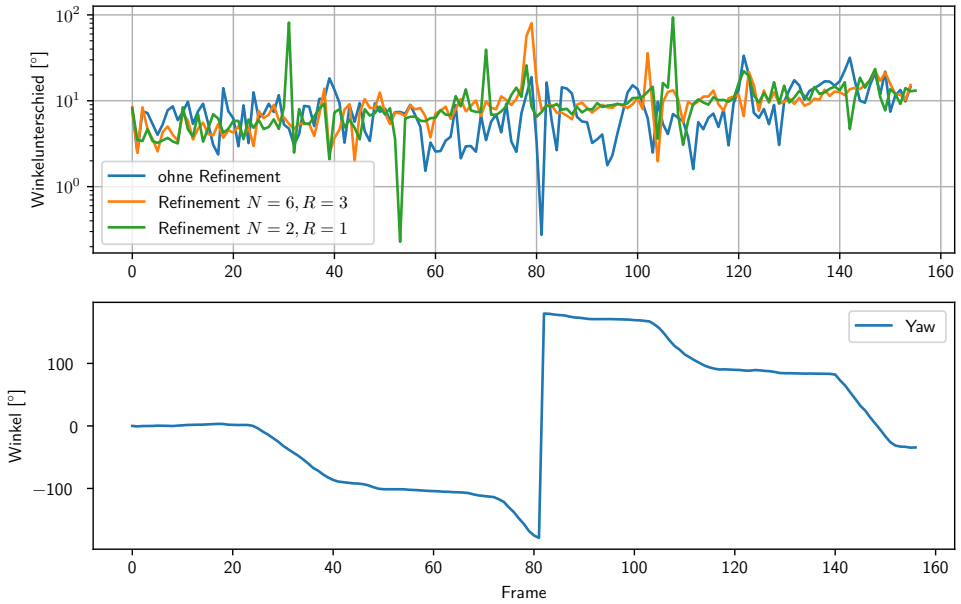


Abbildung 7.14: Winkelfehler der Frames in Vergleich mit der Yaw-Orientierung aus Ground-truth für die Fahrt ohne Schaukeln

Fahrt mit Schaukeln

Bei dieser Testfahrt hat sich gezeigt, dass wie oben erwähnt, die Baseline-Estimation für die MLESAC-Grenze 0.03 die niedrigsten Fehler hat. Für das Refinement gilt dies jedoch bei dieser Fahrt nicht. Abbildung 7.15 zeigt den durchschnittlichen Winkelfehler exemplarisch für zwei Fenstergrößen bei unterschiedlicher Wahl der MLESAC-Grenze. Den besten Fehler hat auch hier, wie bei der Fahrt ohne Schaukeln, die Grenze 1.0°. Das Refinement betrachtet nicht nur die Feature-Korrespondenz von zwei aufeinander folgenden Frames. Bei Korrespondenzen über mehrere Frames wirkt sich eine niedrige Grenze stärker auf die Anzahl der Korrespondenzen aus, weil in jedem Frame mehr Features aussortiert werden. Damit fehlen Korrespondenzen, die zwar für zwei aufeinander folgende Frames nicht optimal zur Bewegung, aber möglicherweise über mehrere Frames, passen. Ein größeres Consensus-Sets liefert hier bessere Ergebnisse. Bei höheren Fenstergrößen ist der Fehler, der durch Ausreißer entsteht, geringer, wie der Vergleich der beiden Kurven zeigt. Es muss bei der Wahl der MLESAC-Grenze also entschieden werden, ob das Refinement genutzt wird oder nicht. Für die weiteren Tests wird die Grenze 1° genutzt.

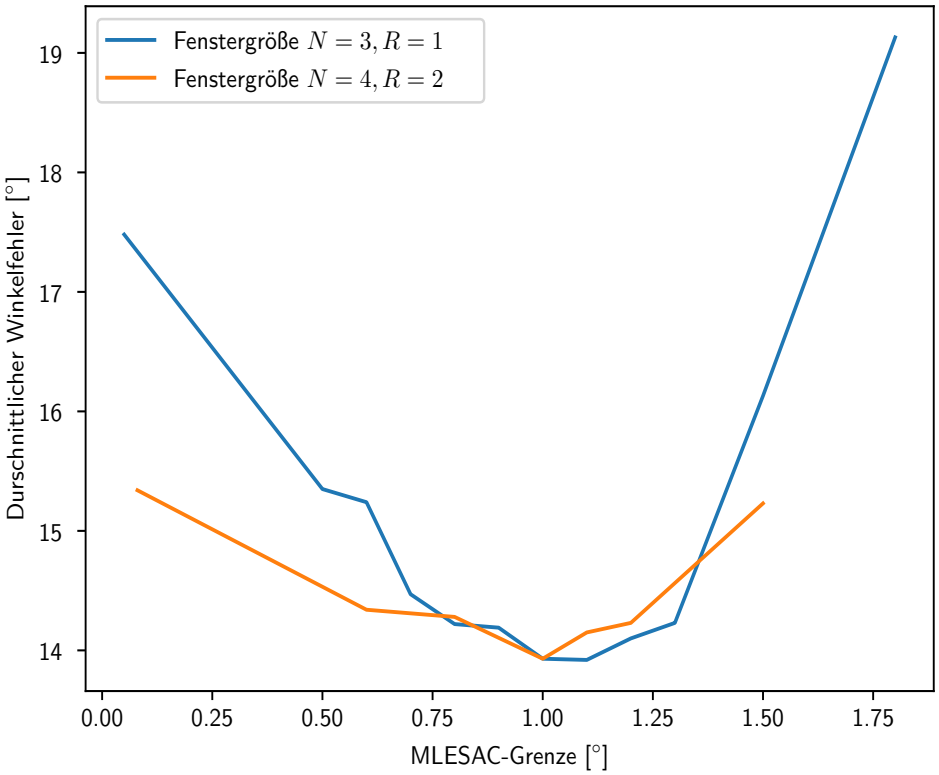


Abbildung 7.15: Durchschnittlicher Winkelfehler in der Laborfahrt mit Schaukeln bei verschiedenen MLESAC-Grenzen unter Nutzung des Refinements

Abbildung 7.16 stellt für die Fahrt mit Schaukeln den durchschnittlichen Winkelfehler und dessen Standardabweichung bei verschiedenen Fenstergrößen dar. Für diese Testfahrt konnte durch das Refinement der Fehler und auch die Standardabweichung verringert werden. Die Vektorrichtungen sind also insgesamt besser.

Der Fehler wird mit der Größe des Refinementfensters und des Betrachtungsfensters kleiner. Innerhalb eines Betrachtungsfenster-Plots nimmt die Steigung der Kurve mit zunehmender Refinementfenstergröße ab. Zwischen den beiden letzten Refinementfenstergrößen $R = N - 2$ und $R = N - 1$ ist die Fehlerverbesserung nur noch sehr gering.

Eine wesentliche Optimierung des Winkelfehlers wird schon bei $N = 2, R = 1$ erreicht. Alle größeren Refinementfenster tragen nur noch minimal zur Verbesserung des Fehlers bei.

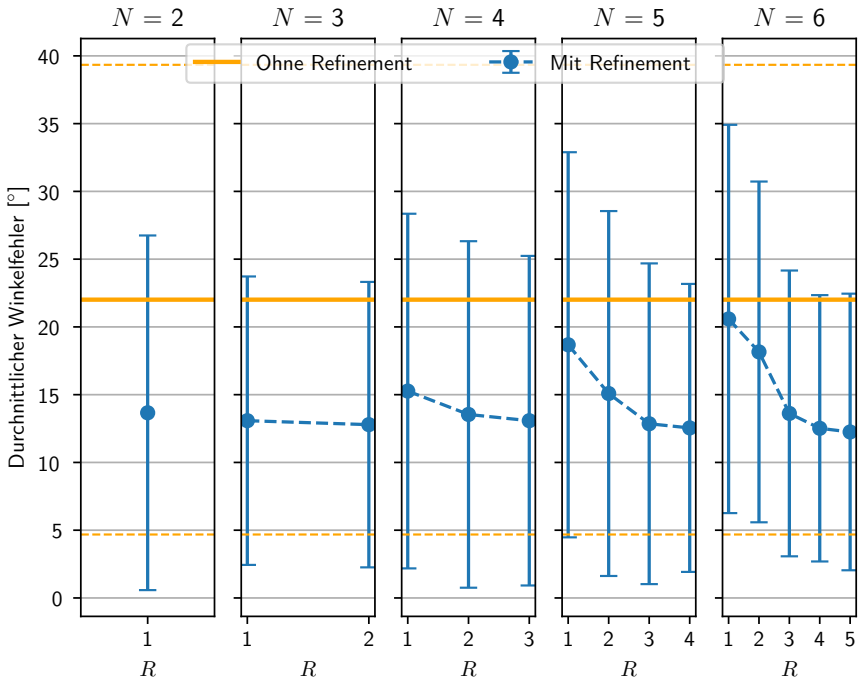


Abbildung 7.16: Plot des durchschnittlichen Winkelfehlers der Fahrt mit Schaukeln bei verschiedenen Betrachtungs- und Refinementfenstergrößen.

Aufällig ist auch, wie schon in der Fahrt ohne Schaukeln zu erkennen, dass der Fehler in für die Refinemenfenstertgröße $R = 1$ in dem Betrachtungsfenster $N = 2$ im Vergleich zu den größeren Betrachtungsfenstern mit $N > 3$, der minimalste ist. Dies zeigt sich ebenfalls bei $R = 2$ und $R = 4$. Den besten Fehlerwert haben die Refinementfenstergrößen, desto weniger zusätzliche Vektoren im Betrachtungsfenster mit in die Kostenfunktion eingerechnet werden. Eine mögliche Erklärung ist auch hier der Aufbau der Kostenfunktion. Bei einer hohen Differenz von R und N werden Frames in die Berechnung der Kostenfunktion mit einbezogen, können aber nicht verändert werden. Befindet sich ein fehlerhafter Vektor außerhalb des Refinementfensters kann dieser also nicht korrigiert werden. Abbildung 7.17 zeigt dies am Beispiel von $N = 5$ und $R = 1$. Die gestrichelten Vektoren sind außerhalb des Refinementfensters aber innerhalb des Betrachtungsfensters. In der Kostenfunktion werden zusammengesetzte Vektoren betrachtet. Für dieses Beispiel setzt sich folgende Kostenfunktion, zur Vereinfachung nur für eine Feature-Korrespondenz m, zusammen:

$$C = \left(m_{t-4}^{T} (R_{t-4}^{w})^{T} \left[\frac{b_{t-4,t}}{||b_{t-4,t}||} \right] \times R_{t}^{w} m_{t} \right)^{2} \tag{7.1}$$

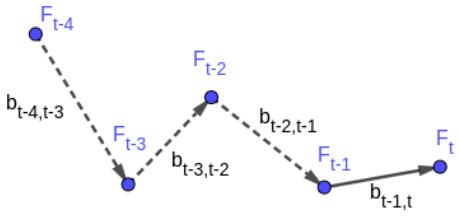


Abbildung 7.17: Zusammengesetzter Bewegungsvektor bei $N = 5, R = 1$

Dabei ist $b_{t-4,t} = b_{t-4,t-3} + b_{t-3,t-2} + b_{t-2,t-1} + b_{t-1,t}$ der zusammengesetzte Vektor. Während der Optimierung kann der Fehler aufgrund der Fenstergrößen nur unter Anpassung des Vektors $b_{t-1,t}$ verringert werden. Sind einer oder mehrere der anderen Vektoren fehlerhaft, wird auch deren Fehler über den Vektor $b_{t-1,t}$ verringert. Dadurch muss dieser Vektor jedoch falsch angepasst werden, was zu dem sichtbaren Winkelfehler im Vergleich mit Ground-truth führt. Dieses Problem wird hier das „Betrachtungsfenster-Problem“ genannt.

Die in Abbildung 7.18 dargestellte Positionsabweichung nach der Fahrt bestätigt diese Erkenntnisse. Zu beachten ist, dass, wie in der Fahrt ohne Schaukeln, die einzelnen Vektoren mithilfe des Ground-truth korrekt skaliert worden sind. Die Fehler liegen also auch hier alleine falsche Vektorrichtungen zugrunde. Insgesamt kann die Abweichung von etwa $0.6m$ ohne Refinement auf etwa $0.4m$ bei starkem Refinement verringert werden. Da sich in diesem Fehlermaß die Richtungsfehler indirekt aufsummieren, zeigen sich das Konsistenz- und auch das Betrachtungsfensterproblem hier stärker. Diese führen sogar dazu, dass bei ungünstiger Wahl der Fenstergrößen der Fehler durch das Refinement vergrößert wird. Es muss also bei der Wahl der Fenstergrößen ein Verhältnis gewählt werden, welches weder das eine noch das andere Problem hervorruft und so die Trajektorie am besten verbessert. Für diesen Fehler liegt die Größe bei $R = N - 2$ für die Betrachtungsfenster $N > 4$. Bei der Wahl der Betrachtungsfenstergröße ist die erforderliche Winkelgenauigkeit zu beachten. Die Verbesserung bei Betrachtungsfenstern ab der Größe $N = 3$ sind nur noch sehr gering.

Abbildung 7.19 zeigt die Trajektorien der Fahrt mit Schaukeln für die Fenstergrößen im Vergleich mit Ground-truth und der Trajektorie ohne Refinement.

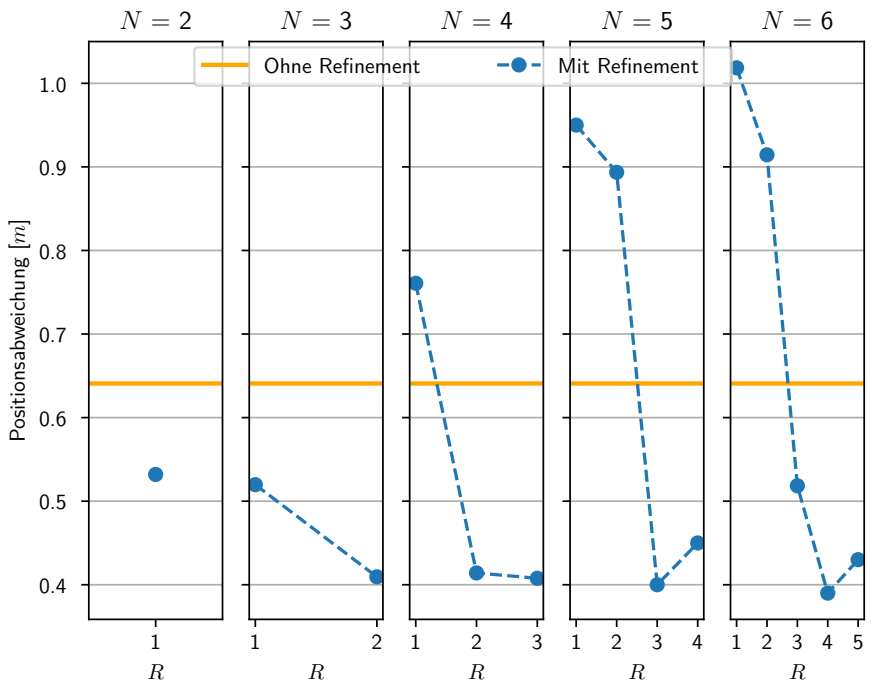


Abbildung 7.18: Plot der absoluten Positionsabweichung nach der Fahrt mit Schaukeln unter Berechnung der perfekten Vektorlängen aus Ground-truth bei verschiedenen Fenstergrößen

Durch das Refinement kann also bei dieser Testfahrt der durchschnittliche Winkelfehler um fast 10° verbessert werden. Damit wird ein ähnliches Niveau wie für die Fahrt ohne Schaukeln erreicht.

7.3.4 Refinement von Ausreißern

In der analysierten Testfahrt ohne Schaukeln waren die Richtungsvektoren so genau, dass das Refinement für die einzelnen Vektorrichtungen für keine Verbesserung gesorgt hat. Um dennoch die Beeinflussung der Vektorrichtungen für diese Fahrt zu testen und genauer zu analysieren, wurden in den ermittelten Vektorrichtungen künstlich vereinzelnd Ausreißer erzeugt.

Tabelle 7.2 zeigt den durchschnittlichen Winkelfehler bei verschiedenen Fenstergrößen. Ohne Refinement beträgt der Fehler etwa 17.40°. Zum Vergleich wurde das Refinement

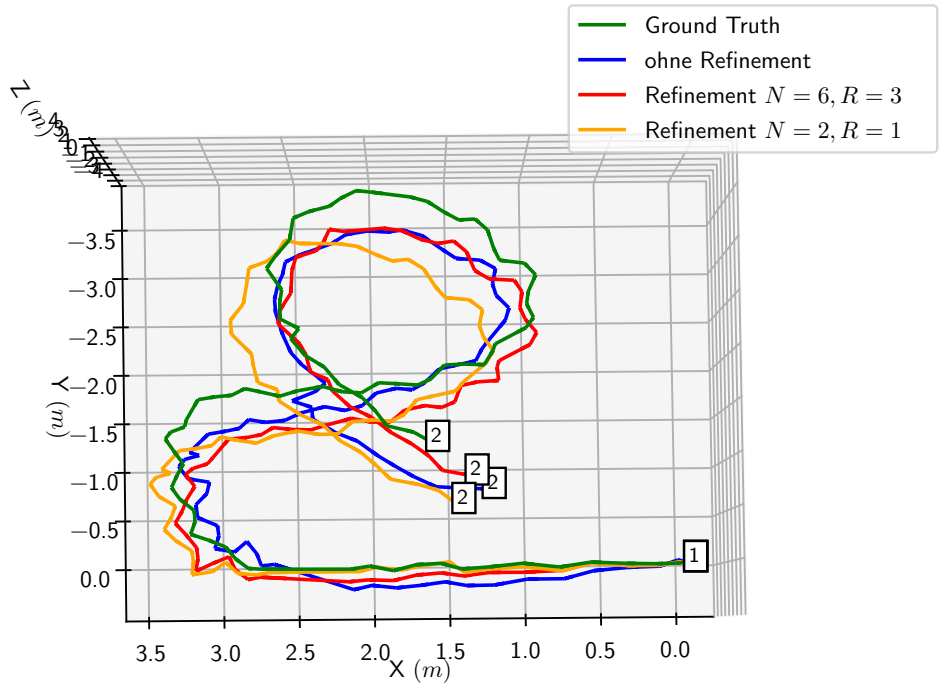


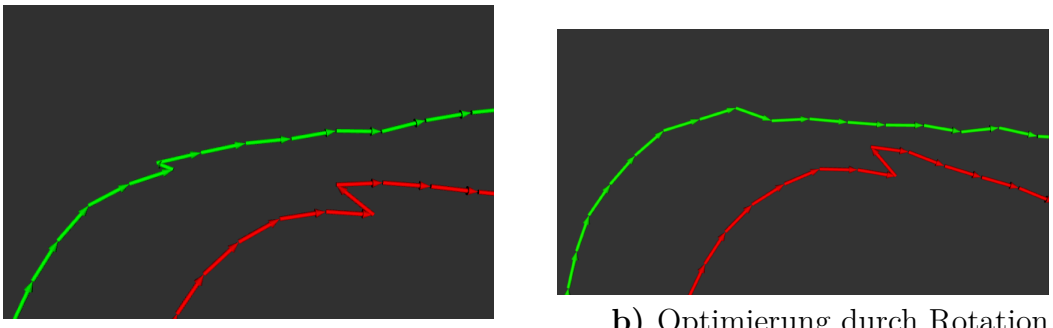
Abbildung 7.19: Trajektorien der Laborfahrt mit Schaukeln bei verschiedenen Refinefenstergrößen für die perfekt skalierten Längen

Betrachtungsfenster	**Refinementfenster**	**Durschnittlicher Winkelfehler**	
		mit Längenoptimierung	ohne Längenoptimierung
ohne Refinement		17.40°	
$N = 2$	$R = 1$	16.79°	16.81°
$N = 3$	$R = 1$	15.16°	14.07°
$N = 3$	$R = 2$	15.9°	12.47°

Tabelle 7.2: Winkelfehler für eine Testfahrt mit künstlichen Richtungsausreißern bei verschiedenen Refinementfenstergrößen

einmal normal auf den Daten ausgeführt und in einem anderen Durchlauf des Refinements wurden die Längenparameter aus der Menge der zu optimierenden Parameter entfernt. Damit gehen die Längenparameter nicht in den Gradienten mit ein und werden während des Refinements nicht verändert.

Für das Betrachtungsfenster $N = 2$ ist eine leichte Verbesserung des Fehlers erzielt worden. Da bei dieser Fenstergröße nur ein Richtungsvektor betrachtet wird, ist hier keine Längenoptimierung möglich. Im Vergleich der Testläufe mit und ohne Längenoptimierung unterscheidet sich der Fehler daher auch nur um einen kleinen Wert, der durch das normale Rauschen erklärt werden kann. Für die weiteren Fenstergrößen fällt auf, dass ohne Längenoptimierung bessere Wert erreicht werden.



a) Optimierung durch Länge

b) Optimierung durch Rotation

Abbildung 7.20: Optimierte Version (grün) eines fehl-ausgerichteten Vektors (rot)

Abbildung 7.20 zeigt einen der künstlich falschen Vektoren (rot) und dessen optimierte Version (grün) im Fall $R = 2, N = 3$. Die Abbildng 7.20b zeigt dies für das Refinement mit deaktivierter Längenoptimierung, 7.20a für aktivierte Längenoptimierung. Bei aktivierter Längenoptimierung wird der falsche Vektor hinabskaliert und seine Auswirkung auf die Trajektorie somit verringert. In 7.20b wird der Vektor wie vorgesehen durch Rotation korrigiert. Eine Richtungskorrektur durch Längenanpassung wirkt sich zudem negativ auf alle folgenden Vektorlängen aus. Da dann diese Länge als Referenz für die nachfolgenden Vektoren gilt.

In Tabelle 7.2 tritt dieses Problem in dem rot markierten Test auf. Im orange markierten Test tritt das Problem bei etwa der Hälfte der Vektoren auf. Nur in den grünen Tests tritt dieses Problem nicht auf. Dass das Problem mit bei $N = 3, R = 2$ mit aktivierter Längenoptimierung größer als bei $N = 3, R = 1$ ist, deutet auch hier auf das Konsistenz-Problem hin.

Für die Testfahrten ohne künstlich erzeugte Ausreißer ist für die Winkelfehler kaum ein Unterschied zwischen Refinements mit und ohne Längenoptimierung festzustellen. Dennoch ist festzustellen, dass dieses Problem existiert und sich möglicherweise negativ auf die, im Weiteren untersuchte, relative Skalierung auswirkt.

7.3.5 Refinement der Längen

Fahrt ohne Schaukeln

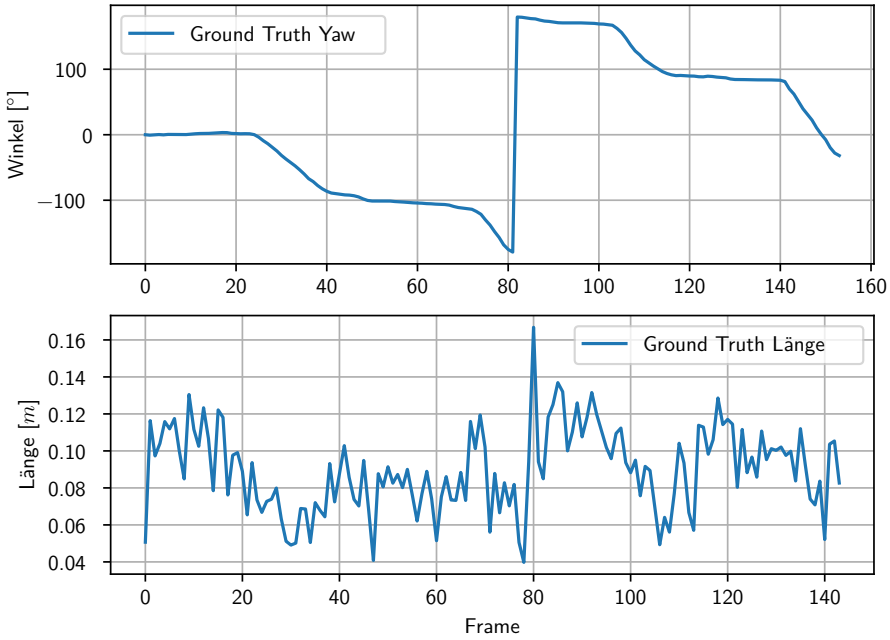


Abbildung 7.21: Ground-truth der Vektorlängen und Yaw-Orientierung der Frames in der Fahrt ohne Schaukeln

Abbildung 7.21 zeigt im unteren Plot die Vektorlängen aus Ground-truth. Zu Beginn der Fahrt wird das Schiff beschleunigt. Hier erhöht sich die Vektorlänge von 0.05 m maximal auf 0.13 m. Der obere Plot zeigt die Yaw-Winkel. Bei großen Änderungen von Yaw wird eine Kurve durchfahren. Insgesamt lassen sich so die vier Kurven dieser Fahrt, wie in

Kurve	**Frame**
Kurve 1	24-40
Kurve 2	67-90
Kurve 3	103-116
Kurve 4	140-152

Tabelle 7.3: Kurven der Fahrt ohne Schaukeln

Tabelle 7.3 dargestellt verordnen. Verglichen mit den Vektorlängen zeigt sich, dass im

Bereich der Kurven die Vektoren kürzer sind. In den Kurven musste die Fahrt verlangsamt werden. In der dritten Kurve musste die Fahrt aufgrund des geringen Kurvenradius fast gestoppt werden, um die Kurve zu durchfahren. Danach wird die Fahrt wieder stark beschleunigt, so ergibt sich hier eine Vektorverlängerung von 0.04 m auf 0.17 m. Auch im Bereich der anderen Kurven und Geraden wurde die Fahrt verlangsamt, so ergibt sich hier über Zeit maximal eine Vektorlängenänderung von 0.17 m auf 0.05 m. Insgesamt ergibt sich so ein maximaler Längenzuwachs um etwa den Faktor 4.25 und eine Längenverkürzung um etwa den Faktor 3.4.

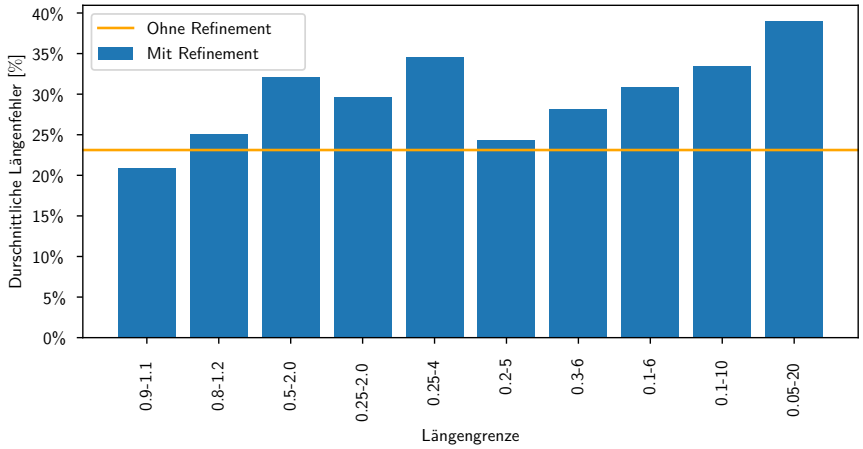


Abbildung 7.22: Plot des durchschnittlichen Längenfehlers in Abhängigkeit der Längengrenzen für die Fahrt ohne Schaukeln (Fenstergröße $N = 6, R = 3$)

Abbildung 7.22 zeigt den durchschnittlichen Längenfehler für verschiedene, im Refinement eingestellte, Längengrenzen. Die Fenstergrößen sind dabei $N = 6, R = 3$. Diese Fenstergröße wurde gewählt als Kompromiss zwischen Konsistenz- und Betrachtungsfenster-Problem. Die orangefarbene Gerade gibt wieder den Fehler ohne Refinement an. Dieser beträgt für den Längenfehler etwa 22 %. Im Vergleich besser als ohne Refinement ist dabei laut dem Längenfehler aber nur der Bereich 0.9–1.1. In diesem Fall wird dem Refinement kaum Spielraum gegeben die Längen anzupassen. Wie zu erwarten, wird daher auch nur eine leichte Verbesserung erzielt. Eine logische Konsequenz wäre, dass weitere Grenzen die Längenskalierung verbessern würden. Das ist jedoch nicht der Fall. Ab der Grenze 0.2–5 sind die Grenzen weit genug auseinander, um die oben berechneten Faktoren zu ermöglichen. Der erste Vektor in diesem Verfahren erhält die Länge 1.0. Mit den oben

berechneten Faktoren ergeben sich so ausgehend von 1.0 maximal eine Länge von 4.25 und minimal eine Länge von 0.29. Der Längenfehler ist bei dieser Grenzen zwar im Vergleich zu anderen Grenzen geringer. Dennoch ist keine Optimierung der Längen erreicht worden. Mit Erweiterung der Längengrenzen steigen dann auch weiterhin die Fehler. Dies ist das in Abschnitt 4.3.3 genannte Problem, weshalb die Längengrenzen benötigt werden.

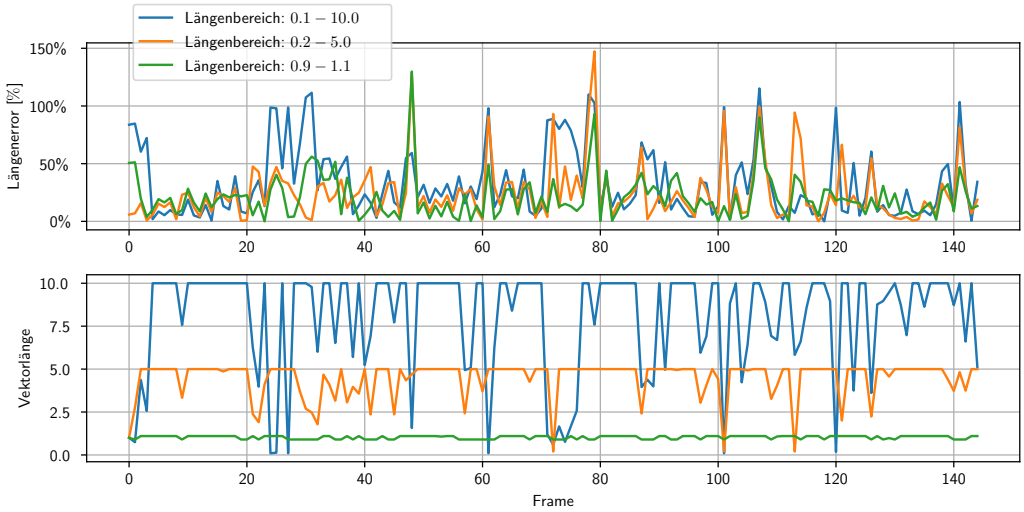


Abbildung 7.23: Skalierungsfehler und die optimierten Vektorlängen bei verschiedenen Längenbereichen in der Fahrt ohne Schaukeln (Fenstergröße $N = 6, R = 3$)

Der Plot in Abbildung 7.23 zeigt die, durch das Refinement berechneten, Längen für drei ausgewählte Längenbereiche. Zunächst beginnen alle Kurven bei der relativen Länge 1.0. Daraufhin erfolgt die Beschleunigung des Schiffes. Auffällig ist, das bereits nach wenigen Frames die Längen auf das Maximum gesetzt werden. Bei der grünen Kurve resultiert dies daraus, dass die Grenze zu niedrig angesetzt ist. Die orangefarbene Kurve erreicht auch ihren höchst möglichen Wert von 5.0. Die blaue Kurve erreicht zunächst den Wert 4.25 und dann auch ihr Maximum von 10.0. Offensichtlich übersteuern die Längen zu Beginn auf ihr Maximum. Sobald das Maximum erreicht ist, können, wie oben erklärt, aufgrund der Wahl des Refinementfensters von $R = 3$ im Vergleich zum Betrachtungsfenster von $N = 6$ nur noch kürzere Längen korrekt gesetzt werden. Im Vergleich mit der Ground-truth-Länge zeigt sich im Verlauf zwar die richtige Tendenz, allerdings reißen gerade im Bereich der Kurven bei Verlangsamung der Fahrt die Längen

auf den minimal möglichen Wert aus. Dieses „Übersteuern“ äußert sich zu Beginn einer Kurve in einem hohen Fehler.

Wenn sich die Länge der Vektoren stärker verändert, als dies durch die Grenzen ermöglicht wird, kann der Vektor nicht optimal angepasst werden. Die Kostenfunktion des Refinements bewertet nur den relativen Unterschied der Vektoren im Fenster N. Wird es durch die Wahl $R = N - 1$ dem Verfahren ermöglicht alle Vektoren im betrachteten Fenster zu ändern, können einfach alle anderen Vektoren innerhalb des Fensters so weit verkleinert oder vergrößert werden, dass das Verhältnis wieder stimmt. Dies führt dann allerdings zu einer Inkonsistenz zu den Vektoren außerhalb des Fensters. Diese äußerst sich als der sichtbare Skalierungssprung.

Ingesammt deuten die Beobachtungen auf ein generelles Problem der Kostenfunktion hin. Die Einschränkung der Vektorlängen geschieht durch absolute Werte. Die Skalierung der Vektoren erfolgt jedoch relativ zueinander. Haben im Betrachtungsfenster alle bisherigen Vektoren die maximale Länge oder eine Länge nah dem maximalen Wert und es tritt ein längerer Vektor auf, kann dieser nicht korrekt skaliert werden. Auch dies kann bei zu hohen Refinementfenstern dazu führen, dass Vektoren fälschlicherweise wieder verkürzt werden, um das Verhältnis aufrechtzuerhalten. Durch das oben erwähnte Übersteuern der Längen kommt es aber schnell zu diesem Effekt.

Abbildung 7.24 zeigt den durchschnittlichen Längenfehler bei verschiedenen Fenstergrößen. Die Längengrenze ist bei 0.2–5.0. Offensichtlich verbessert sich dieser Fehlerwert durch den Einsatz des Refinements bei keiner der Fenstergrößen. Die orangefarbene Gerade zeigt wieder den Fehler ohne den Einsatz des Refinements. Es wird sichtbar, dass das in Abschnitt 4.4.3 angesprochene Konsistenz-Problem auftritt. Bei im Verhältnis zu N hohen Werten von R verschlechtert sich die Skalierung der Vektoren. Es treten höhere Fehler auf, weil während der Optimierung die Skalierung der Vektoren außerhalb des Fensters nicht berücksichtigt werden. Bei im Verhältnis zu N kleinen Werten für R wird das Refinement gezwungen, das Verhältnis der zu optimieren Vektoren zu den restlichen Vektoren aufrechtzuerhalten.

Das Betrachtungsfenster der Größe $N = 3$ beinhaltet zu wenig Bewegungsvektoren, um konsistent zu bleiben. Im Betrachtungsfenester $N = 2$ erfolgt keine Längenänderung, da hier nur ein Bewegungsvektor betrachet wird.

Im Vergleich dazu zeigt die Abbildungen 7.25 die gleichen Fehlerwerte für den Längenbereich 0.25–2.0. Aufgrund der zu sehr eingeschränkten Längen ist der minimale

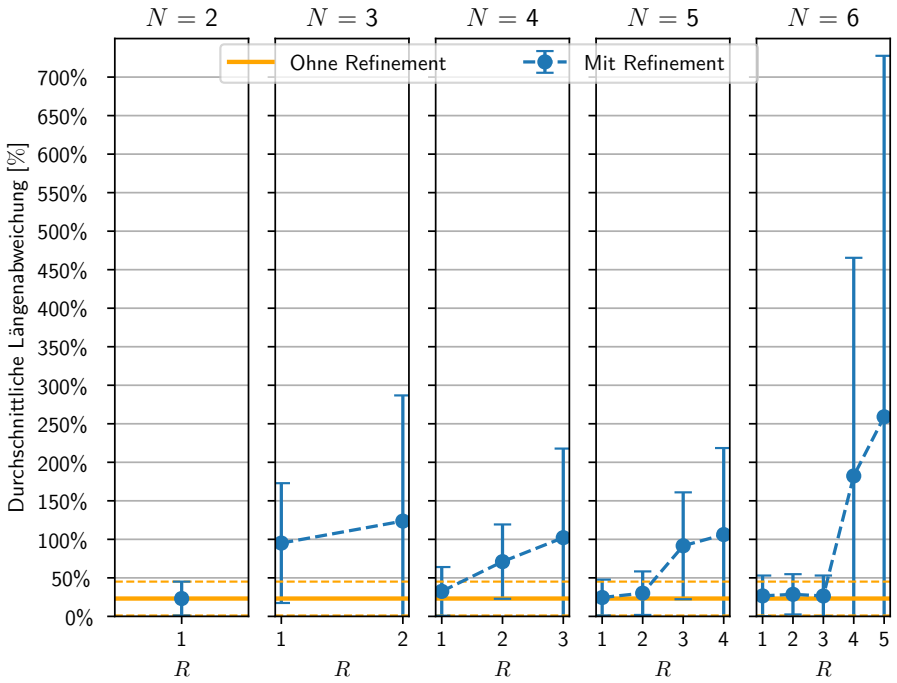


Abbildung 7.24: Durchschnittlicher Längenfehler bei verschiedenen Fenstergrößen der Fahrt ohne Schaukeln (Längengrenze = 0.2–5.0)

Längenfehler höher als für den Längenbereich 0.2–5.0. Dafür sind für hohe Refinementfenster die Übersteuerungen nicht so stark möglich. Für $N = 6, R = 4$ sind die Fehler so deutlich niedriger als für den Längenbereich 0.2–5.0.

Das Problem der Inkonsistenz zu den Vektoren außerhalb des Betrachtungsfensters bei ungünstiger Wahl des Refinementfensters zeigt sich in Abbildung 7.26. Hier sind für den Längenbereich 0.25–4.0 die Fehler und Längen geplottet. Im Fall des kleinsten Refinementfensters $R = 1$ sind die kleinsten Fehler zu erkennen. Zwar hält die Kurve mit dem Refinementfenster $R = 3$ auch Konsistenz zu den vorherigen Vektoren, darf aber mehr Vektoren verändern. Dies führt an einigen Stellen zu größeren Fehlern (Konsistenz-Problem). Im Fall $R = 1$ dagegen ist das Refinement gezwungen alleine den neuesten Vektor aufgrund der vorherigen zu skalieren. An einigen wenigen Stellen führt dies zu etwas höheren Längenfehlern, da hier eine Anpassung einer vorherigen Vektorlänge möglicherweise nötig gewesen wäre.

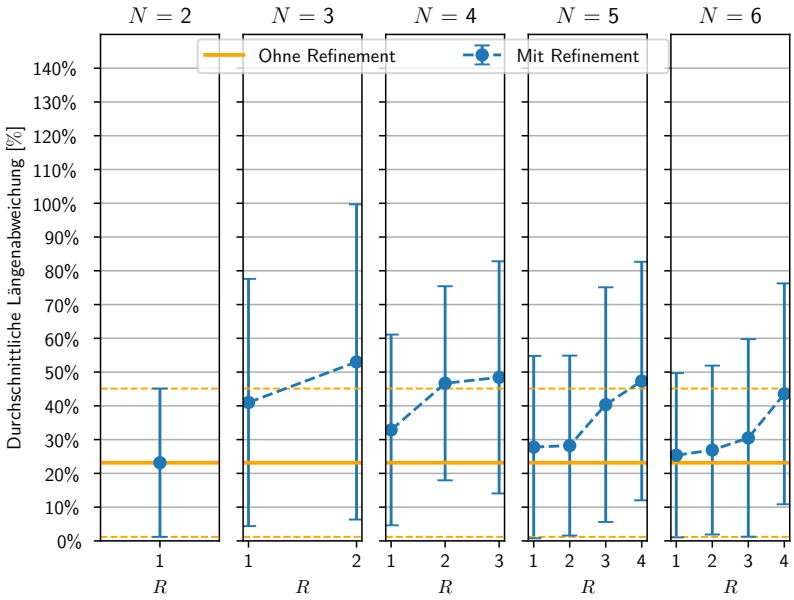


Abbildung 7.25: Durchschnittlicher Längenfehler der Fahrt ohne Schaukeln bei verschiedener Fenstergrößen (Längengrenze = 0.25–4.0)

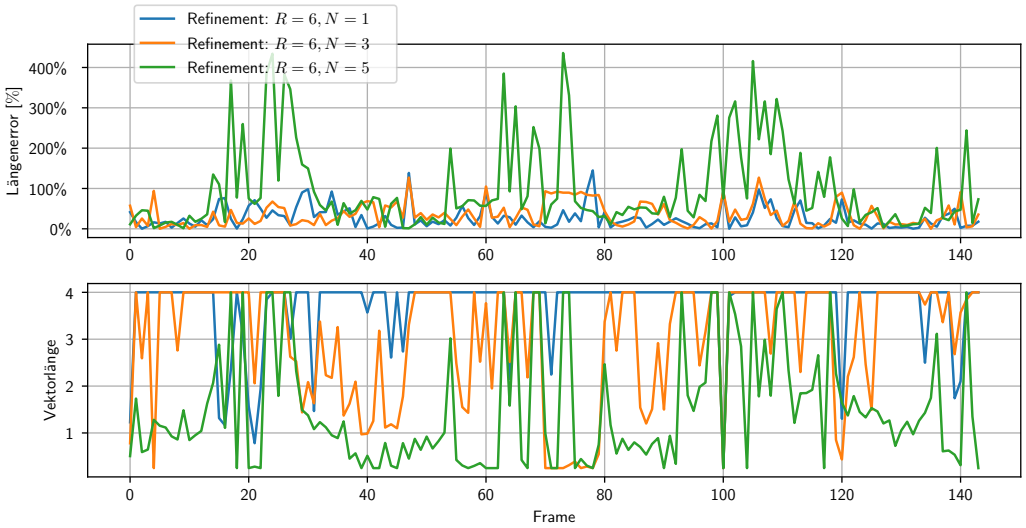


Abbildung 7.26: Längenfehler und die optimierten Vektorlängen bei verschiedenen Fenstergrößen in der Fahrt ohne Schaukeln. (Längengrenze 0.2–4.0)

Der Graph für die Refinementfenster $N = 6, R = 5$ zeigt das vorhergesagte Verhalten. An vielen Stellen ändert das Refinement hier die Länge aller vorherigen Vektoren im Fenster und verliert so die Konsistenz zu den Vektorlängen außerhalb des Fensters. Gut erkennbar im Bereich vor dem Frame F_{80}. Die Länge ist vorher schon am Maximum und es folgt, wie im Ground-truth-Plot (Abbildung 7.21) eine Beschleunigung. Um das Längenverhältnis dennoch aufrechtzuerhalten, verkleinert das Refinement die Vektoren vor F_{80}. Darauf folgt der Peak bei F_{80}.

Ein Problem, welches zu falschen Längen führt, liegt in der Kostenfunktion, welche Längen und Richtungsfehler gleichzeitig betrachtet. Die Kostenfunktion des Refinements optimiert die Längen über zusammengesetzte Vektoren. Die Abbildung 7.27 zeigt beispielhaft

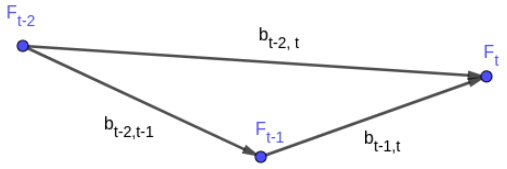


Abbildung 7.27: Zusammengesetzter Vektor innerhalb der Kostenfunktion des Refinements bei $N = 2, R = 1$

zwei Vektoren und dessen zusammengesetzten Vektor. Für das Refinement mit den Fenstergrößen $N = 2, R = 1$ würde sich dann folgende zu optimierende Kostenfunktion ergeben: (Der Einfachheit halber nur für eine Feature-Korrespondenz m)

$$C = \left(m_{t-2}^T (R_{t-2}^w)^T \left[\frac{b_{t-2,t}}{||b_{t-2,t}||} \right] \times R_t^w m_t \right)^2 \tag{7.2}$$

Der interessante Teil ist dabei der zusammengesetzte Vektor $b_{t-2,t}$. Dieser ist aufgrund des Skalarproduktes in der Kostenfunktion normiert. Aufgeteilt in den Richtungsvektor u und die Länge n schreibt sich dieser zusammengesetzte und normierte Vektor so:

$$\frac{b_{t-2,t}}{||b_{t-2,t}||} = \frac{b_{t-2,t-1} + b_{t-1,t}}{||b_{t-2,t-1} + b_{t-1,t}||} = \frac{n_{t-2,t-1} u_{t-2,t-1} + n_{t-1,t} u_{t-1,t}}{n_{t-2,t-1} + n_{t-1,t}} \tag{7.3}$$

Fährt das Schiff mehrere Frames in die gleiche Richtung, gilt $u_{t-2,t-1} = u_{t-1,t-0}$. Daraus ergibt sich folgendes:

$$\frac{n_{t-2,t-1}u_{t-2,t-1} + n_{t-1,t}u_{t-2,t-1}}{n_{t-2,t-1} + n_{t-1,t}} = \frac{(n_{t-2,t-1} + n_{t-1,t})u_{t-1,t}}{n_{t-2,t-1} + n_{t-1,t}} = u_{t-1,t} \tag{7.4}$$

Die Vektorlängen eliminieren sich. Auch wenn sich die Richtungsvektoren nur wenig unterscheiden, geht kaum Skalierungsinformation in die Kostenfunktion mit ein. Dieses Problem wird im Folgenden das „Gleiche-Vektor-Problem“ genannt. Insgesamt spricht das Problem dafür, große Betrachtungsfenster zu nutzen, aber ein sehr kleines Refinementfenster, damit keine gleichen Bewegungen sich zusammensetzten können. Dies ist eine Beobachtung, die bereits oben gemacht wurde und bisher nur mit dem Konsistenz-Problem in Verbindung gebracht wurde.

In Bezug auf die Testfahrt reißen die Längen direkt zu Beginn aus. Am Anfang ist das Betrachtungsfenster noch nicht auf der gewünschten Größe, außerdem ist die Bewegung laut Abbildung 7.8 sehr geradlinig. Somit setzt sich hier die Kostenfunktion nur aus gleichen oder sehr ähnlichen Richtungsvektoren zusammen. Falsche Skalierungen haben so keinen Einfluss auf die Kosten. Möglicherweise führt so das Gleiche-Vektor-Problem zu dem beobachteten Übersteuern zu Beginn der Testfahrt.

Fahrt mit Schaukeln

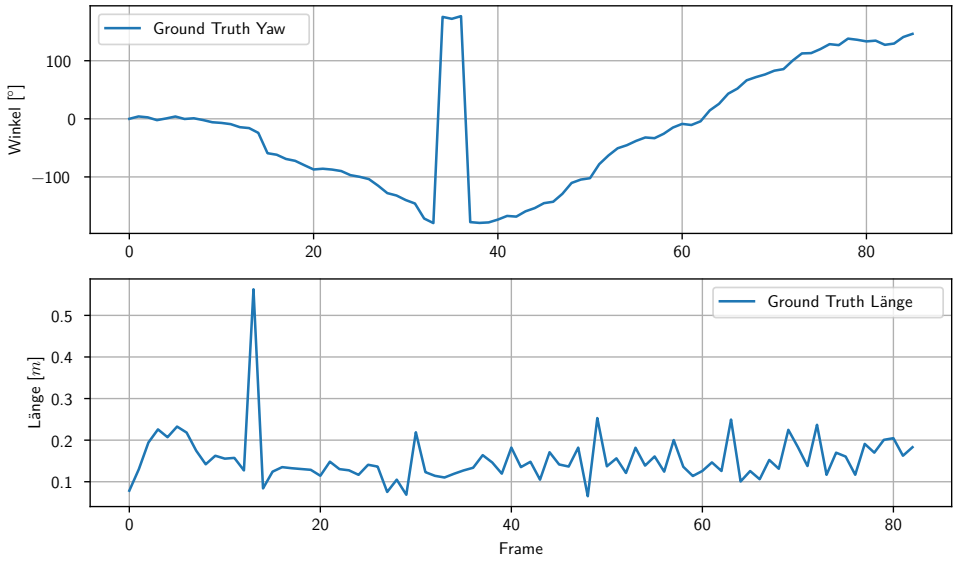


Abbildung 7.28: Ground-truth der Vektorlängen und Yaw-Orientierung für die Fahrt mit Schaukeln

Abbildung 7.28 zeigt im unteren Plot die Vektorlängen aus dem Ground-truth. Im Vergleich zur Fahrt ohne Schaukeln fällt hier auf, dass die Vektoren generell länger und die Fahrt somit schneller war. Ingesamt liegen die Längen im Bereich von ca. 0.06 m–0.25 m. Auffällig ist zu Beginn der Fahrt ein Ausreißer auf etwa 0.56 m. Auch der Plot des Yaw-Winkels zeigt hier einen Sprung. An dieser Steller kam es zu einem Verlust des Trackings. Da die Plattform mangels Wasser während der Fahrt mit Schaukeln getragen werden musste und der Beginn der Fahrt am Rande der Trackinganlage liegt, konnte dies hier nicht vermieden werden. Im Folgenden kann also ein Längenausreißer an dieser Stelle ignoriert werden. Darüber hinaus spielt dieser Fehler in alle Tests mit hinein, sodass ein Vergleich dennoch möglich ist. Weil die Plattform in dieser Testfahrt getragen und nicht auf dem Wagen geschoben wurde, konnten die Kurven wesentlich schneller durchfahren werden. Ohne diesen Ausreißer ändert sich die Länge hier maximal etwa mit dem Faktor 4.2 nach oben bzw. unten.

Abbildung 7.29 zeigt die durchschnittlichen Längenfehler bei verschiedenen Längenbereichen für diese Fahrt mit den Fenstergrößen $N = 6, R = 3$. Es zeigt sich ein ähnliches

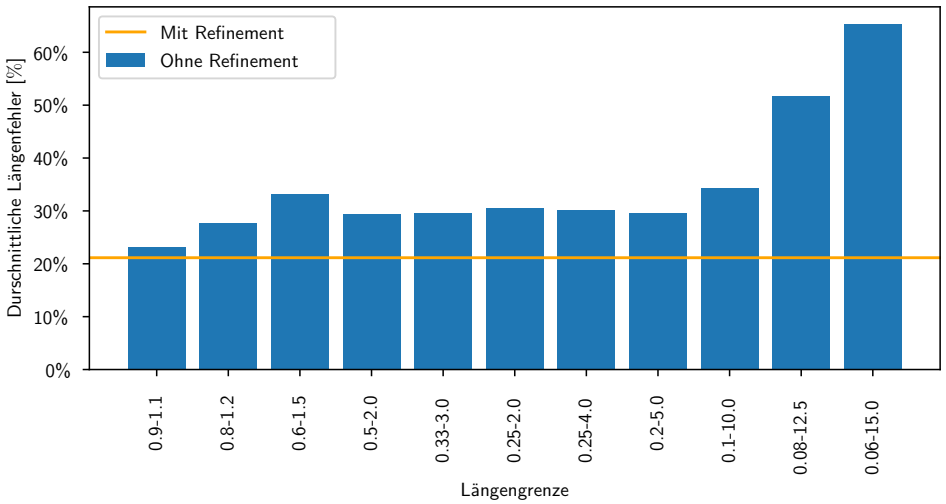


Abbildung 7.29: Durchschnittlicher Längenfehler bei verschiedenen Längengrenzen in der Fahrt mit Schaukeln (Fenstergrößen: $N = 6, R = 3$)

Bild, wie bei der Fahrt mit Schaukeln. Hier gibt es sogar keinen Längenbereich, der die Skalierung verbessert.

Abbildung 7.30 zeigt die Längenabweichung für diese Fahrt bei verschienen Fenstergrößen für die Längengrenze 0.2–5.0. Nach den obigen Berechnungen liegen die Vektorlängen in diesem Bereich. Auch hier ist keine Verbesserung durch das Refinement erreicht worden. Es zeigt sich jedoch auch hier, was schon in der Fahrt ohne Schaukeln erkannt wurde: Die Längenoptimierung ist mit maximalen Refinementfenstern schlechter, als mit kleineren. Auch hier zeigt sich so das Konsistenz-Problem zu den Vektoren außerhalb des Betrachtungsfensters. Im Vergleich zu den Längenfehlern der Fahrt ohne Schaukeln sind die Fehler beim Konsistenz-Problem nicht so hoch. Möglicherweise als Hinweis auf das Gerade-Vektor-Problem, welches bei starkem Schaukeln nicht so stark auftreten sollte.

Insgesamt bringt die relative Skalierung trotz der Anpassungen keine Verbesserung zu den unskalierten Vektoren. Selbst wenn eine besonders gleichmäßige Fahrt mit besonders exakt gewählten Längengrenzen den Längenfehler verbessert, ist dies eine starke Einschränkung für die Bewegung des Schiffes. Allein das Anhalten und Anfahren ergeben schon einen großen Längenbereich. Die Untersuchungen haben gezeigt, dass das Verfahren tendenziell die Vektorlänge in die richtige Richtung ändert. Dies geschieht aber viel zu ungenau.

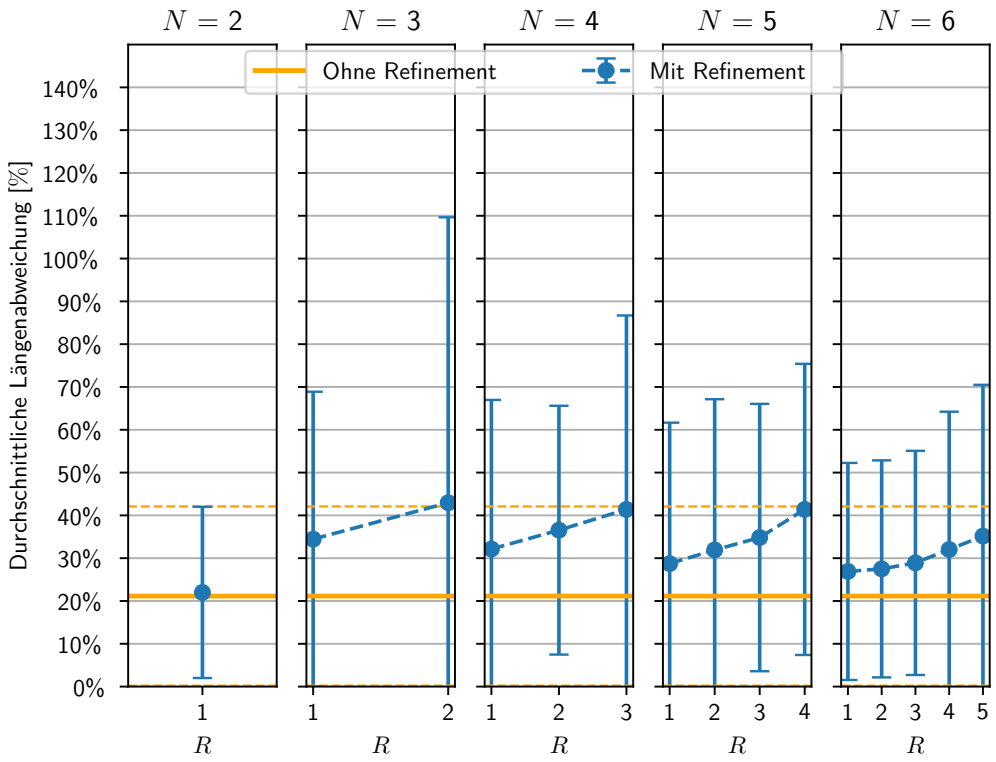


Abbildung 7.30: Plot der durchschnittlichen Längenabweichung der Fahrt mit Schaukeln für verschiedene Fenstergrößen (Längengrenze: 0.2–5.0)

Abschließend sei zu bemerken, dass die Wahl des Verhältnisses zwischen Refinement- und Betrachtungsfenstergröße für die Vektorrichtungen und Vektorlängen im Widerspruch steht. Die Vektorrichtungen werden am besten bei etwa $R = N - 2$ optimiert. Die Vektorlängen erfordern ein kleines Refinementfenster. Hier müsste also ein gutes Verhältnis gefunden werden.

7.4 Miniatur Wunderland

Die Testfahrten im Miniatur Wunderland wurden mit einer Längengrenze im Bereich 0.2–5.0, den Fenstergrößen $R = 3, N = 6$ und einer MELSAC-Grenze von 1° durchgeführt, da für diese Werte in den Testfahrten im Labor die besten Ergebnisse sowohl für Vektorrichtungen als auch Vektorlängen erzielt worden sind.

7.4.1 Features

Feature Die Abbildung 7.31 zeigt, dass bei gleichen Einstellungen, wie für die Labor-

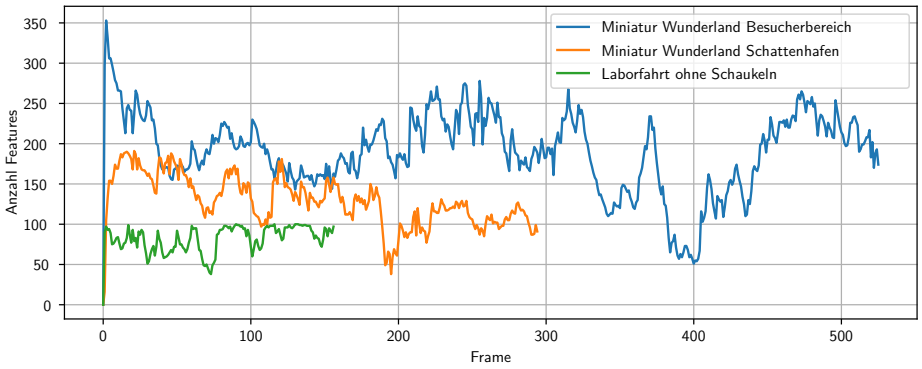


Abbildung 7.31: Vergleich der Anzahl der getrackten Features für die Testfahrten

testfahrten, deutlich mehr Feature erkannt und getrackt wurden, als in den Labortestfahrten. Dies unterstützt die Annahme, dass sich die Modellbauumgebung gut für das Feature-Tracking eignet.

Abbildung 7.32 zeigt die getrackten (grün) und neu erkannten (blau) Features eines Frame der Fahrt im Besucherbereich des Wunderlandes. Im rot maskierten Bereich werden keine Features erkannt. Dieser umfasst nicht nur das sichtbare Schiff, sondern auch Wasserbereiche. Dennoch ist ein Teil des Wassers frei gelassen, sodass hier Features erkannt werden. Im Wasser treten Reflexionen der Decke, der Deckenlampen und anderer Objekte auf, die fälschlicherweise als Features erkannt werden. Um dies zu verhindern, müsste die gesamte Bildhälfte unter dem Horizont maskiert werden. In einigen Passagen würde dies aber dazu führen, dass keine oder viel weniger Features erkannt würden. In der Abbildung ist bereits ein Bereich des Landes von der Maskierung verdeckt.

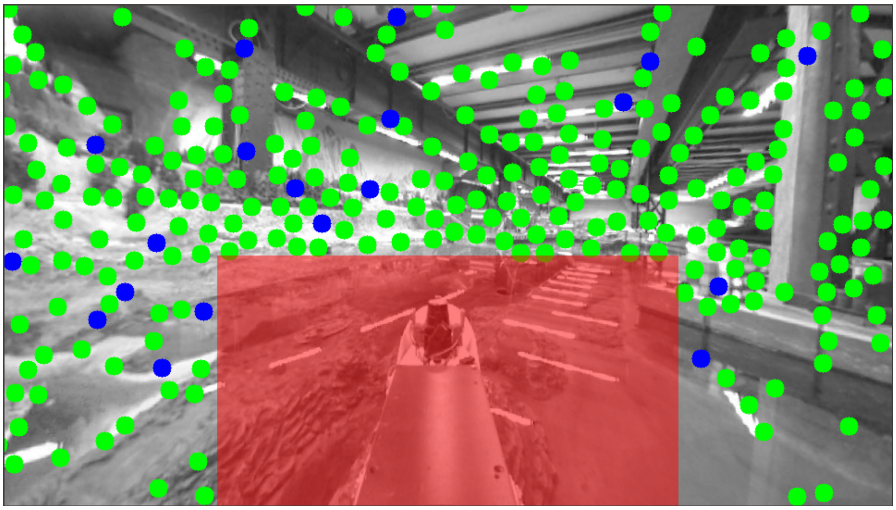


Abbildung 7.32: Kamerabild mit erkannten Features im Miniatur Wunderland

Im Bild aus Abbildung 7.34 zeigt die Kamera in den Besucherbereich. Eine Maskierung des halben Bildes würde hier dazu führen, dass nur Features im Besucherbereich getrackt würden. Da der mittlere Besucher sich bewegt, werden hier Ausreißer erzeugt.

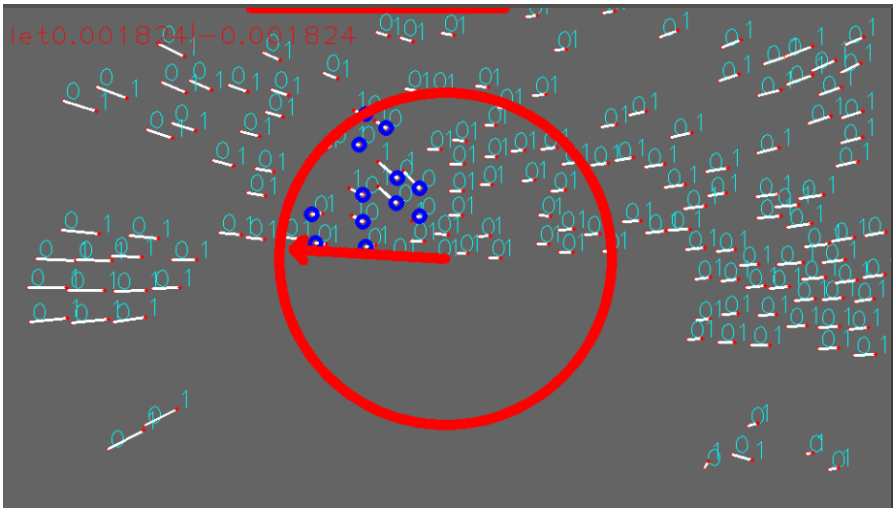


Abbildung 7.33: Ausreißer durch sich bewegende Personen

Abbildung 7.33 zeigt die Ausreißer, die diese Bewegung erzeugt. Da diese blau umkreist sind, wurden sie von MLESAC aussortiert. Solange der Anteil an Ausreißern, die in sich eine konsistente Bewegung ergeben würden, nicht mehr als 50 % beträgt, sollten daher durch die Zuschauer keine Fehler entstehen. Allerdings werden die Ausreißer

in der Pipeline erst im `BaselineEstimator` erkannt. Die Bewegungen im Bild durch dynamische Objekte, wie Besucher, können zunächst auch bei keiner Schiffsbewegung für einen Feature-Positionsunterschied sorgen. Dieser veranlasst den Merger den Frame an den `BaselineEstimator` weiterzureichen. Da, wie bereits erläutert, das Verfahren keine Nicht-Bewegungen erkennen kann, führt dies dennoch zu Fehlern.

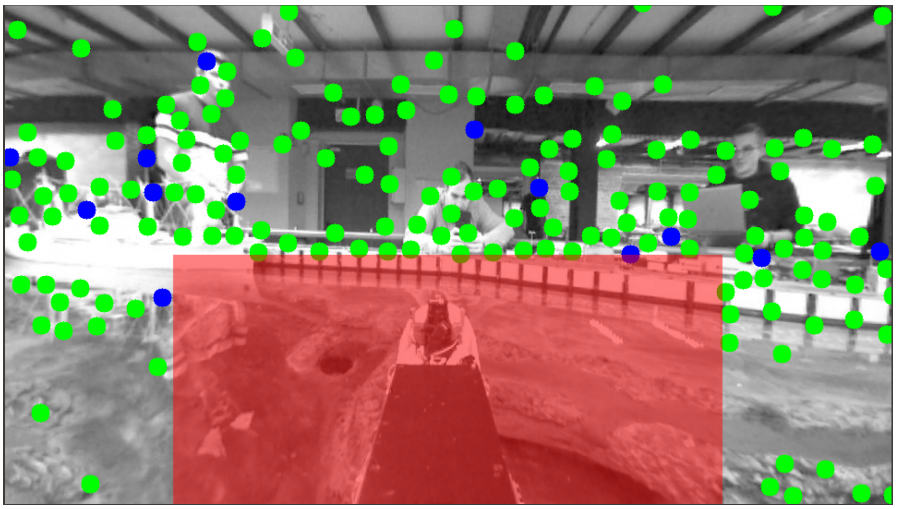


Abbildung 7.34: Kamerabild mit Zuschauern im Bild

7.4.2 Trajektorien

Schattenhafen

Abbildung 7.35 zeigt die Trajektorie der Testfahrt im Schattenhafen. Es sind zum Vergleich die Trajektorie von Hector-Slam und der VO geplottet. Die VO-Trajektorie ist dabei verschieden skaliert. Zum einen wurde die Trajektorie mit dem Skalierungsfaktor des ersten Vektors skaliert. Zum anderen wurde, wie bereits für die anderen Tests, der Skalierungsfaktor aus dem Median aller Skalierungsfaktoren berechnet. Dies bietet die bestmögliche Skalierung. Des Weiteren ist zum direkten Vergleich der Richtungen eine Trajektorie geplottet, in der jeder einzelne Vektor passend zum jeweiligen Hector-Slam Vektor skaliert ist (perfekte Skalierung).

Der Beginn der Trajektorie ist mit ’1’, das Ende mit ’2’ gekennzeichnet. Da es sich um eine Ringfahrt handelt, sollten im besten Fall ’1’ und ’2’ übereinander liegen.

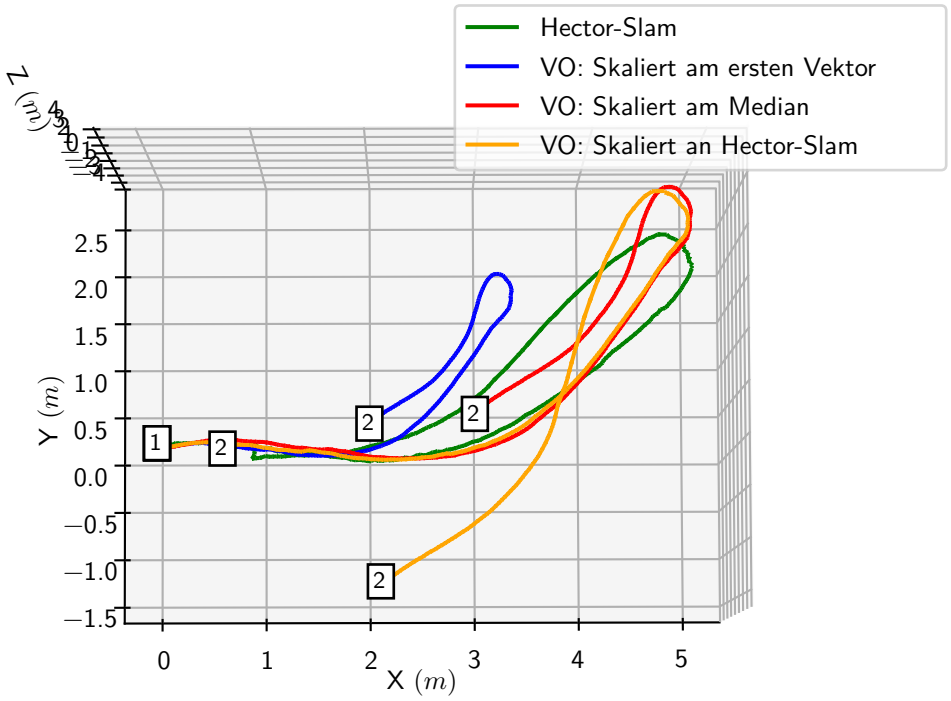


Abbildung 7.35: Trajektorievergleich der Testfahrt im Schattenhafen des Miniatur Wunderlandes

Der Plot des Hectors-Slams zeigt, dass hier die Ausgangsposition fast erreicht wurde. An Stellen, an denen im Lidar-Scan keine Features erkannt werden, wird keine Bewegung erkannt. Dieses Problem tritt im Schattenhafen kaum auf, da mindestens eine Ecke des Schattenhafens zu jeder Zeit erkannt wird. Darüber hinaus erzeugen die liegenden Schiffe viele unterschiedliche Strukturen im Lidar-Scan.

Die am ersten Vektor skalierte Trajektorie ist bereits kurz nach dem Anfang falsch skaliert. Dies kann mehrere Gründe haben. Eventuell war der erste Vektor aus dem Hector-Slam nicht optimal, sodass der Skalierungsfaktor falsch ist. Aufgrund der bisherigen Ergebnisse ist es wahrscheinlicher, dass es im Bereich des Anfanges zu Skalierungssprüngen in der VO kam und zur weiteren Trajektorie keine Skalierungskonsistenz besteht. Die am Median skalierte und die perfekt zu Hector-Slam skalierte Trajektorie zeigen in der ersten Hälfte

der Fahrt eine deutliche Ähnlichkeit mit Hector-Slam. Allerdings kommt es dann im Bereich der Wende am Ende des Beckens zu deutlichen Unterschieden der Bewegung. Nach der Wende scheint das VO-Verfahren die Skalierung verloren zu haben. Der letzte Bereich der Trajektorie ist deutlich kürzer. Beim Vergleich der perfekt an Hector-Slam skalierten VO-Trajektorie und der Trajektorie aus Hector-Slam zeigt sich vor allem im Bereich nach der Wende ein Orientierungsunterschied. Auf dem Plot nicht erkennbar ist, dass die VO hier die gleiche Bewegung erkannt hat, diese aber falsch rotiert ist. Dies weist wieder auf die IMU-Drift hin.

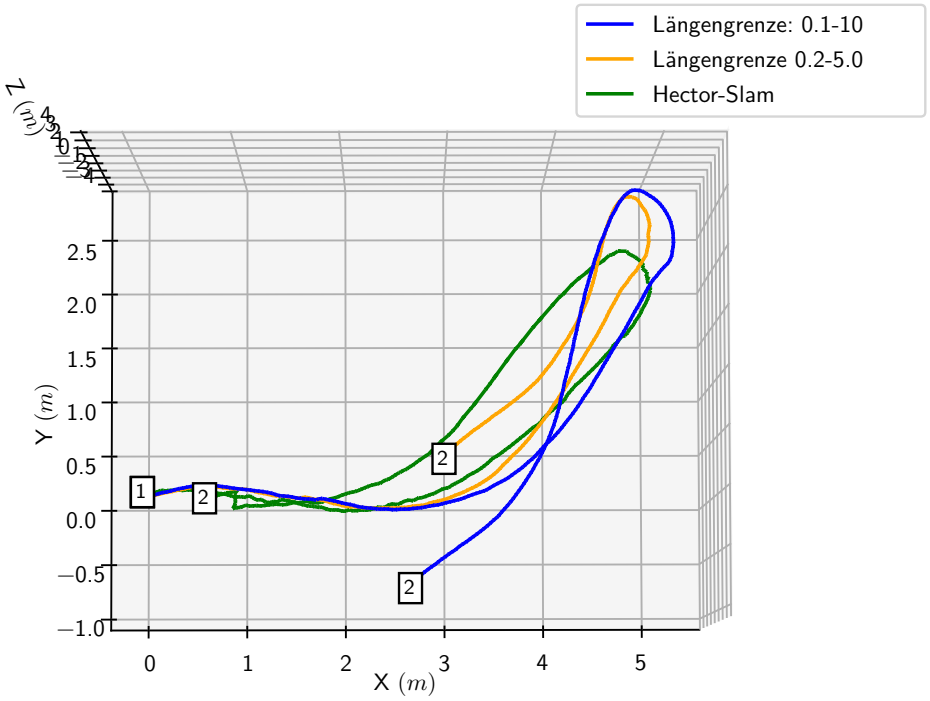


Abbildung 7.36: Trajektorien der Fahrt im Schattenhafen bei unterschiedlichen Längengrenzen

Da in der Wende möglicherweise die Skalierung aufgrund der langsamen Bewegung verloren gegangen sein könnte, wurde ein weiterer Test mit erweiterten Längengrenzen unternommen. Abbildung 7.36 zeigt den Vergleich der Median-skalierten Trajektorien. Die Erweiterung verbessert die Längen nach der Wende. Allerdings sind die Abweichungen vor der Wende größer. Das unterstützt die oben gemachte Beobachtung, dass weite Längengrenzen zu Längensprüngen und mehr Ungenauigkeiten führen.

Austellungsbereich

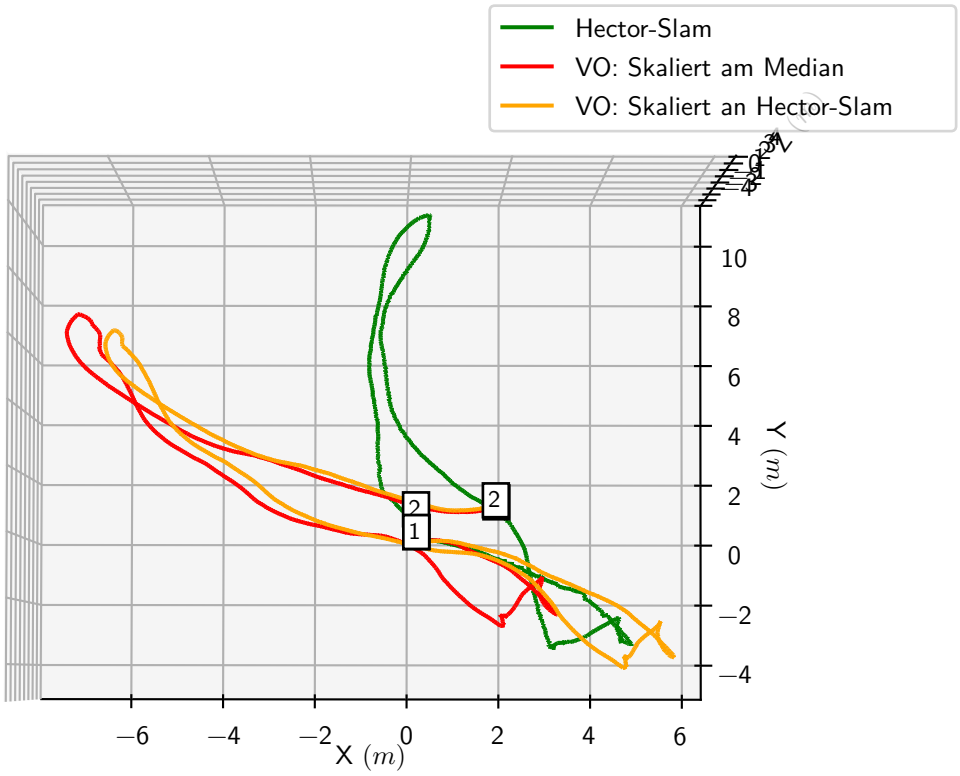


Abbildung 7.37: Trajektorievergleich der Testfahrt im vorderen Bereich des Miniaturwunderlandes

Abbildung 7.37 zeigt die Trajektorien für eine Ringfahrt im Ausstellungsbereich des Wasserbeckens. Auch hierbei handel es sich um eine Ringfahrt, welche von Hector-SLAM auch fast erkannt wird.

Im Bereich $(x, y) = (4, -4)$ (unten rechts) hat das Schiff eine sehr langsame Drehung durchlaufen, bei der es teilweise zum Stehen gekommen ist. Außerdem hat an dieser Stelle die Kamera auf den Besucherbereich gezeigt. Die VO sieht an dieser Stelle deutlich glatter aus.

Insgesamt zeigt sich auch im Vergleich mit der Karte aus Abbildung 6.2 bei dieser Testfahrt wieder die Orientierungsdrift.

Im Vergleich liegt der Positionsunterschied der Median-skalierten Fahrt mit etwa 3 m über den 1 m von Hector-Slam.

Abschließend sei gesagt, dass die VO im Gegensatz zu einem VSLAM, ähnlich wie eine Radodometrie, nicht den Anspruch an einer global korrekten Position hat. Für das Miniatur Wunderland zeigen sich hier, bis auf die falschen Längenskalierungen und die Orientierungsdrift, gute Ergebnisse.

7.5 Aufwand

7.5.1 Refinement-Iterationen

Abbildung 7.38 zeigt den Einfluss der Anzahl der Refinement-Iterationen auf die Berechnungsdauer. Dazu ist zusätzlich der durchschnittliche Winkel- und Längenfehler dargestellt. In den bisherigen Tests wurde die maximale Iterationszahl auf 5000 gesetzt. Nicht für jeden Frame wird jedoch das Maximum genutzt. Oftmals konvergiert das Refinement bereits nach einigen dutzend Iterationen.

Die Fenstergröße ist, um hier den „Worst-Case“ zu betrachten, auf $N = 6, R = 5$.

Die Baseline Estimation benötigt für 5000 Iterationen 0.26 sec. Die Anzahl der Iterationen beeinflusst die Baseline-Estimation nur indirekt über die Gesamtauslastung des Systems. Mit abnehmender Anzahl dieser verringert sich die Rechenzeit leicht bis auf 0.18 sec.

Bei 5000 Iterationen benötigt das Refinement pro Frame durschnittliche 0.79 sec. Dieser Wert verringert sich bei nur einer Iteration auf 0.01 sec. Der durchschnittliche Winkelfehler ist im Bereich über alle Iterationsgrenzen bis auf die normale Abweichung des Verfahrens stabil. Der Längenfehler steigt mit weniger Iterationen leicht an. Dennoch scheint die wesentliche Optimierung innerhalb des ersten Iterationsschrittes erfolgt zu sein. Für alle Weiteren scheint das Refinement nur die Längen weiter zu optimieren.

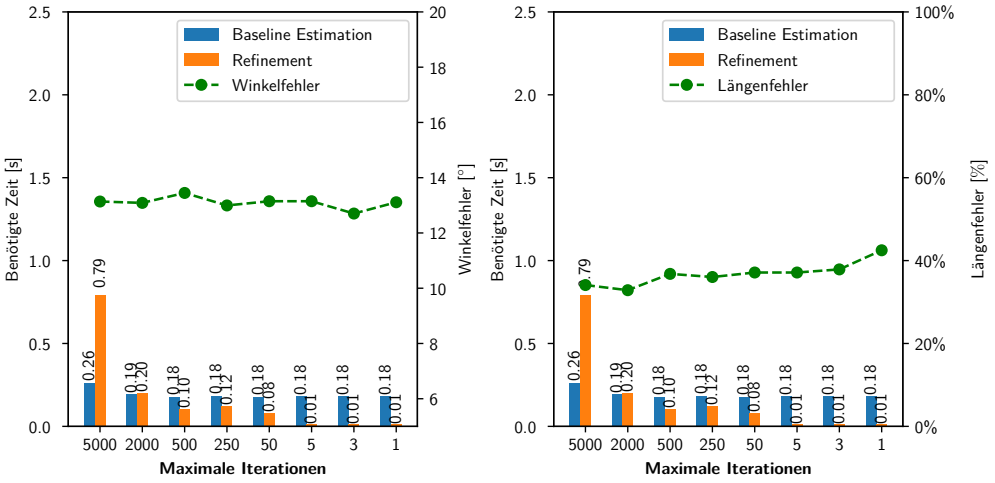


Abbildung 7.38: Durschnittliche Berechnungszeit eines Frames auf dem Raspberry Pi bei unterschiedlicher Anzahl an Refinement-Iterartionen

7.5.2 Konvergenzkriterium

Abbildung 7.39 zeigt die benötigte Zeit und die Fehler für unterschiedlich starke Konvergenzkriterien. Die in der Abbildung dargestellten Tests wurden mit einer maximalen

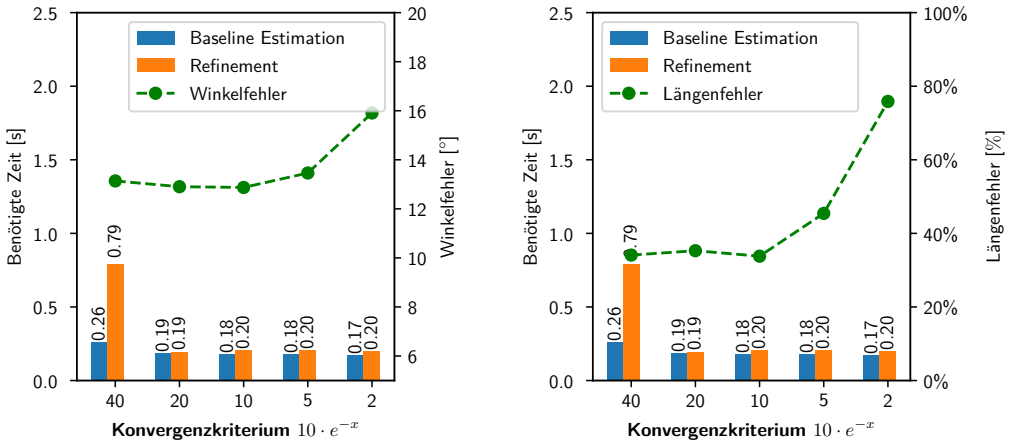


Abbildung 7.39: Durchschnittliche Berechnungszeit eines Frames auf dem Raspberry Pi bei unterschiedlich starken Konvergenzkriterien des Refinements

Iterationszahl von 5000 und einer Fenstergröße von $N = 6, R = 5$ durchgeführt, um nur die Auswirkung dieses Parameters zu ermitteln. Die weniger starken Kriterien bringen das Verfahren früher zum Terminieren. Dies führt genauso, wie bereits untersucht, zu niedrigeren Rechenzeiten. Allerdings ist ab dem Wert $10 \cdot e^{-10}$ eine deutlichere Verschlechterung der Fehler zu erkennen. Das liegt an der Kostenfunktion, diese arbeitet durch die Nutzung des Skalarproduktes in einem Bereich von 0–1. Dementsprechend sollte das Refinement hier auf sehr niedrige Fehler optimieren. Die Rechenzeit verringert sich aufgrund der hohen Iterationszahl nicht weiter. Es existieren einzelne Frames in denen der Fehler nicht minimiert werden kann. An diesen arbeitet das Refinement bis die maximale Iterationszahl erreicht ist.

7.5.3 Fenstergrößen

Das Diagramm in Abbildung 7.40 zeigt die durchschnittliche Berechnungszeit bei verschiedenen Fenstergrößen. Dabei wurden die maximalen Iterationen auf 250 und das Konvergenzkriterium auf $10 \cdot e^{-20}$ gesetzt. Für diese Werte liefert, laut den oberen Tests das Verfahren noch gute Ergebnisse. Die Dauer der Baseline-Estimation bleibt auch hier

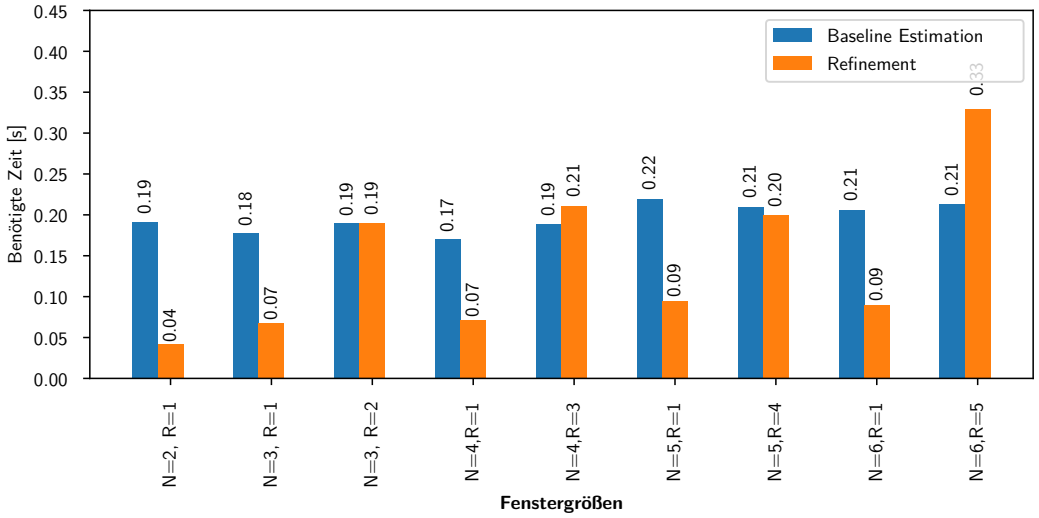


Abbildung 7.40: Durchschnittliche Berechnungszeit eines Frames auf dem Raspberry Pi bei unterschiedlichen Fenstergrößen

wieder im Rahmen der normalen Abweichungen stabil. Die leichten Schwankungen sind durch die Auslastung des `Refiners` zu erklären. Das Diagramm zeigt die Berechnungsdauer für alle implementierten Betrachtungsfenster. Gemessen wurden dann jeweils das kleinste und größte Refinementfenster. So kann der Bereich des jeweiligen Betrachtungsfensters erfasst werden. Die maximalen Refinementfenster mit $R = N - 1$ benötigen aufgrund des größeren Parameterraumes deutlich länger als die Refinementfenster der Größe $R = 1$. Abbildung 7.41 zeigt die Auslastung des RaspberryPis für dieselben Pa-

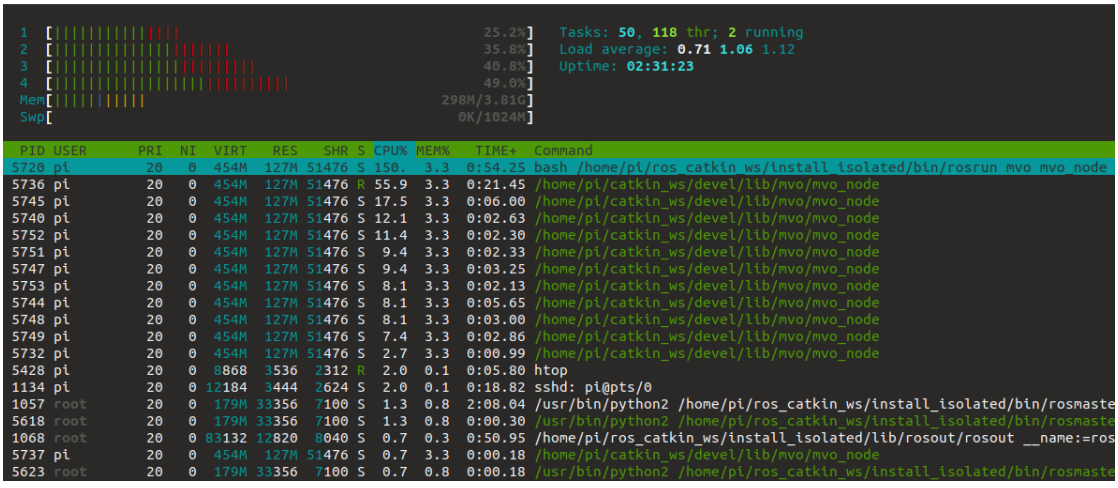


Abbildung 7.41: Prozessorauslastung des RaspberryPIs

rameter wie in diesem Test bei den Fenstergrößen $N = 6, R = 3$. Im Mittel ist der Pi zu 40 % ausgelastet. In den Spitzen erreichte er während des Tests eine Auslastung von etwa 70%. Im Minimum betrug sie etwa 15%. Der Name des Systems ist `mvo_node`. Zu erkennen sind die einzelnen Threads, aus denen das System besteht.

Insgesamt zeigt sich also, dass die Berechnungen auf dem Pi deutlich unter einer halben Sekunde liegen. Und das trotz des kontinuierlichen Trackings bei durchschnittlich 5 fps der Eingangsbilder. Dies würde theoretisch eine höhere Maximalfrequenz der Pipeline nach dem `Merger` von ebenfalls bis zu 5 fps ermöglichen. Nicht berücksichtigt wurde bei diesen Messungen die Pipeline-Latenz. Diese ist besonders bei stehendem Schiff störend. Hier steht, da keine neue Frames aus dem `Merger` kommen, die Pipeline im hinteren Teil still. Damit ist die letzte Position immer veraltet.

7.6 Abweichungen

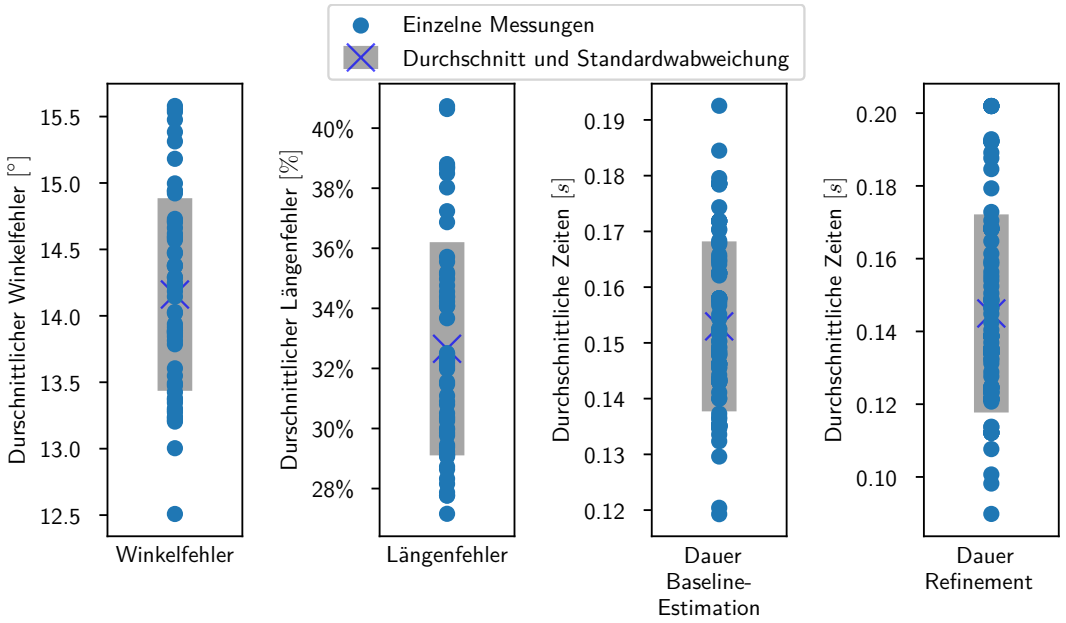


Abbildung 7.42: Abweichungen der verschiedenen Fehlermetriken in der Fahrt mit Schaukeln bei 60 Durchläufen

Um die Aussagekraft der bisherigen Tests zu validieren, wurden 60 Testläufe mit denselben Parametern gemacht. Abbildung 7.42 zeigt die den durchschnittlichen Winkelfehler, den durchschnittlichen Längenfehler und die durchschnittlichen Berechnungsdauern jedes Durchlaufes. Daneben ist der Durchschnittswert dieser Durchschnittswerte und deren Standardabweichung markiert.

Beim Winkelfehler beträgt die Standardabweichung nur etwa 0.7°. Beim Längenfehler ungefähr 3 %. Für die Baseline-Estimation-Dauer 0.014 s und für die Refinement-Dauer 0.03 s. Die Messwerte der Durchläufe werden nach dem Test von Shapiro und Wilk [1965] als Normalverteilt angenommen. Also kann angenommen werden, dass 68.3 % der Messungen im Intervall der gezeigten Standardabweichungen liegen.

8 Fazit und Ausblick

In dieser Arbeit wurde das vorgestellte VO-Verfahren für das Miniaturschiff **MS nHA-Wigatora** implementiert.

Um das Verfahren nutzbar zu machen, wurden, in Absprache mit dem Autor des Verfahrens, Anpassungen vorgenommen. Um das Verfahren zu vervollständigen, wurden darüber hinaus in dieser Arbeit weitere Erweiterungen vorgenommen.

Es wurde eine geeignete Softwarearchitektur entwickelt und implementiert mithilfe der das Verfahren getestet und genutzt werden kann. Außerdem ist das System so aufgebaut, dass es einfach um weitere Teile ergänzt werden kann.

Es wurde gezeigt, unter welchen Parametern das Verfahren am besten funktioniert. Dabei haben die Anpassungen, die zusammen mit dem Autor erarbeitet und die Anpassungen, die alleine entwickelt wurden, sich als nützlich erwiesen.

Die Tests haben gezeigt, dass das Verfahren gute Richtungen liefert. Auch bei sehr unruhiger Fahrt liegen diese durch den Einsatz des Refinements in einem akzeptablen Bereich. Eine Schwachstelle des Verfahrens sind die Längen der zurückgelegten Bewegung. Ohnehin wäre theoretisch nur eine relative Skalierung möglich. Das Skalieren soll implizit über das Refinement erfolgen. Durch den Einsatz des Refinements werden jedoch keine korrekten Längen bestimmt. Es zeigt sich sogar, dass die Längenveränderung zur Verschlechterung der Richtungen führen kann. Es wurden in der Arbeit verschiedene mögliche Gründe für die schlechte Längenbestimmung aufgezeigt. Zur problematischen Längenberechnung zählt auch die Erkennung von nicht-Bewegungen. Diese ist derzeit nur mangelhaft gelöst.

Nebenbei wurde gezeigt, dass die verwendete IMU eine Yaw-Drift aufweist. Diese wirkt sich bei zeitlich langen Trajektorien negativ auf die absolute Position auf.

Die verfügbaren Ressourcen des Schiffes reichen aus, um mit einer maximalen Frequenz Bewegungen aus den Bildern zu ermitteln. Aufgrund der Softwarearchitektur kann mit

temporären Ressourcenengpässen umgegangen werden, ohne dass dabei die Qualität des Verfahrens leidet. Lediglich die Berechnungsdauer verlängert sich. Dennoch existiert eine Latenz zwischen der Aufnahme eines Bildes und der Ausgabe der dazu ermittelten Position.

8.1 Ausblick

Um das Verfahren als VO nutzen zu können, muss das Problem der Vektorlängen gelöst werden. Hierzu existieren zwei Möglichkeiten. Zum einen könnten die Bewegungsrichtungen mit einem Sensor, der Auskunft über die aktuelle Geschwindigkeit gibt, fusioniert werden. Hier würden sich im Kontext des Schiffes Sensoren anbieten, die den Wasserdurchfluss messen. Eine weitere Möglichkeit wäre es, weitere Daten aus der IMU zu beziehen.

Generell könnten also Lösungen auf dem Schiff implementiert werden, in denen das Verfahren nur die Bewegungsrichtung liefert. Anhand anderer Daten würden dann diese Richtungen absolut skaliert.

Dennoch ist es sinnvoll eine relative Skalierung zu implementieren. Dies sollte im Optimalfall zunächst analytisch geschehen. So eine Lösung ist am effizientesten und einfach zu handhaben. Darüber hinaus könnte eine Stufe, die die Skalierung berechnet, einfach in die Pipeline eingebaut werden. Im Refinement sollte die Längenoptimierung aus der aktuellen Kostenfunktion entfernt werden. Die aktuelle Kostenfunktion hat sich für die Optimierung der Länge als nicht geeignet herausgestellt. Das Refinement müsste dann, um die Skalierung weiterhin zu berücksichtigen, um eine weitere Kostenfunktion erweitert werden, die falsche Skalierungen bestraft. Dann sollten die Betrachtungsfenstergrößen für beide Teile getrennt wählbar sein. So kann sowohl das Konsistenz-Problem für die Längenoptimierung, als auch das Betrachtungsfenster-Fehler-Problem umgangen werden. In Anhang B ist eine einfache Formel beschrieben, mit der eine Skalierung analytisch und im Refinement erfolgen könnte. Diese konnte innerhalb dieser Arbeit nicht mehr getestet werden.

Um die Drift durch die IMU zu verhindern, könnte die Orientierung mit in den Parameterraum der Kostenfunktion übernommen werden. Hier wäre es interessant, ob eine Mitoptimierung der Orientierungen auch bessere Vektorrichtungen erzielt.

Um die Latenz an der Pipeline zu verringern, könnte die Beschränkung des Lesebereiches weiter aufgeweicht werden. Damit könnten die Pipeline-Stufen Frames früher weiterreichen, auch wenn noch kein neuer Frame angekommen ist. Dies würde den Vorteil haben, dass bei Stillstand des Schiffes die Pipeline sich „leer“ arbeiten könnte.

Zusätzlich wäre es interessant, den Unterschied zwischen Wasser und Land zu ermitteln, um so Features nur an den richtigen Stellen zu erkennen. Hier wären Lösungen, die das Kamerabild segmentieren, denkbar. Auch würde es im Kontext des Miniatur Wunderlandes Sinn ergeben, damit den Zuschauerbereich zu erkennen.

Literaturverzeichnis

[Agarwal u. a.] Agarwal, Sameer ; Mierle, Keir ; Others: *Ceres Solver*. http://ceres-solver.org

[Aqel u. a. 2016] Aqel, Mohammad O. ; Marhaban, Mohammad H. ; Saripan, M I. ; Ismail, Napsiah B.: Review of visual odometry: types, approaches, challenges, and applications. In: *SpringerPlus* 5 (2016), Nr. 1, S. 1897

[Basler] Basler: *daA1920-30uc - Basler dart*. Basler AG (Veranst.). – URL https://www.baslerweb.com/en/products/cameras/area-scan-cameras/dart/daa1920-30uc-s-mount/. – [Letzter Zugriff: 10.10.2019]

[Bouguet u. a. 2001] Bouguet, Jean-Yves u. a.: Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. In: *Intel Corporation* 5 (2001), Nr. 1-10, S. 4

[Bradski 2000] Bradski, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[Branch und Stewart 2018] Branch, Nicholas ; Stewart, Eric: Applications of Phase-Based Motion Processing, 01 2018

[Burau u. a. 2019] Burau, Henri ; Schnirpel, Thorben ; Stark, Franek: *nHAWigatora miniature ship*. Oct 2019. – URL https://autosys.informatik.haw-hamburg.de/platforms/2019nhawigatora/. – [Letzter Zugriff: 15.12.2019]

[Derpanis 2010] Derpanis, Konstantinos G.: Overview of the RANSAC Algorithm. In: *Image Rochester NY* 4 (2010), Nr. 1, S. 2–3

[Fischler und Bolles 1981] Fischler, Martin A. ; Bolles, Robert C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In: *Commun. ACM* 24 (1981), Juni, Nr. 6, S. 381–395. – URL http://doi.acm.org/10.1145/358669.358692. – ISSN 0001-0782

[Gavin 2019] Gavin, Henri: The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems. (2019). – URL http://people.duke.edu/~hpgavin/ce281/lm.pdf. – [Letzter Zugriff: 23.12.2019]

[Guennebaud u. a. 2010] Guennebaud, Gaël ; Jacob, Benoît u. a.: *Eigen v3*. http://eigen.tuxfamily.org. 2010

[Harris u. a. 1988] Harris, Christopher G. ; Stephens, Mike u. a.: A combined corner and edge detector. In: *Alvey vision conference* Bd. 15 Citeseer (Veranst.), 1988, S. 10–5244

[Hata und Savarese 2017] Hata, Kenji ; Savarese, Silvio: CS231A Course Notes 1: Camera Models. (2017). – URL https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf. – [Letzter Zugriff: 29.12.2019]

[He u. a. 2019] He, Ming ; Zhu, Chaozheng ; Huang, Qian ; Ren, Baosen ; Liu, Jintao: A review of monocular visual odometry. In: *The Visual Computer* (2019), Juni, S. 1. – URL http://dx.doi.org/10.1007/s00371-019-01714-6. – ISSN 1432-2315

[Hokuyo] Hokuyo: *Scanning Laser Range Finer Hokuyo URG-04LX-UG0*. Hokuyo Automatic Co., Ltd. (Veranst.). – URL http://docs.roboparts.de/URG-04LX_UG01/Hokuyo-URG-04LX_UG01_spec.pdf. – [Letzter Zugriff: 10.10.2019]

[Inkilä 2005] Inkilä, Keijo: Homogeneous least squares problem. In: *Photogrammetric Journal of Finland* 19 (2005), Nr. 2, S. 34–42

[InvenSense] InvenSense: *MPU-9250*. InvenSense Inc. (Veranst.). – URL https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. – [Letzter Zugriff: 10.10.2019]

[James Bowman 2017] James Bowman, Vincent R.: *ROS Wiki - Camera calibration*. 2017. – URL http://wiki.ros.org/camera_calibration. – [Letzter Zugriff: 16.12.2019]

[Kohlbrecher u. a. 2011] Kohlbrecher, S. ; Meyer, J. ; Stryk, O. von ; Klingauf, U.: A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)* IEEE (Veranst.), November 2011

[Lucas u. a. 1981] Lucas, Bruce D. ; Kanade, Takeo u. a.: An iterative image registration technique with an application to stereo vision. (1981)

[von Luck u. a. 2019] Luck, Kai von ; Jenke, Philipp ; Draheim, Susanne ; Lehmann, Thomas ; Schorbach, Vera: */* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */*. 2019. – URL https://csti.haw-hamburg.de/. – [Letzter Zugriff: 20.12.2019]

[MathWorks, Inc 2019] MathWorks, Inc: *What Is Camera Calibration?* 2019. – URL https://www.mathworks.com/help/vision/ug/camera-calibration.html. – [Letzter Zugriff: 29.12.2019]

[Miniatur Wunderland 2019a] Miniatur Wunderland: *Echtwasser-Becken*. 2019. – URL https://www.miniatur-wunderland.de/wunderland-entdecken/welten/skandinavien/echtwasser/. – [Letzter Zugriff: 29.12.2019]

[Miniatur Wunderland 2019b] Miniatur Wunderland: *Miniatur Wunderland Hamburg - Modellbahn, Modelleisenbahn Hamburg*. 2019. – URL https://www.miniatur-wunderland.de/. – [Letzter Zugriff: 29.12.2019]

[OpenCV 2014] OpenCV: *OpenCV Documentation - Camera Calibration and 3D Reconstruction*. OpenCV (Veranst.), 2014. – URL https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[OpenCV 2019] OpenCV: *OpenCV Documentation - Open Cv: Feature Detection*. OpenCV (Veranst.), 2019. – URL https://docs.opencv.org/4.1.2/dd/d1a/group__imgproc__feature.html

[Pareigis u. a. 2019] Pareigis, Stephan ; Tiedemann, Tim ; Becke, Martin ; Meisel, Andreas: *autosys*. Oct 2019. – URL https://autosys.informatik.haw-hamburg.de/. – [Letzter Zugriff: 01.11.2019]

[Rojas 2010] Rojas, Raúl: Lucas-kanade in a nutshell. In: *Freie Universit at Berlinn, Dept. of Computer Science, Tech. Rep* (2010)

[Sánchez u. a. 2018] Sánchez, Javier ; Monzón, Nelson ; Salgado De La Nuez, Agustín: An analysis and implementation of the harris corner detector. In: *Image Processing On Line* (2018)

[Scaramuzza und Fraundorfer 2011] Scaramuzza, D. ; Fraundorfer, F.: Visual Odometry [Tutorial]. In: *IEEE Robotics Automation Magazine* 18 (2011), Dec, Nr. 4, S. 80–92

[Schnirpel 2019] Schnirpel, Thorben: Kalibrierung einer IMU. (2019)

[Shan u. a. 2016] Shan, Mo ; Bi, Yingcai ; Qin, Hailong ; Li, Jiaxin ; Gao, Zhi ; Lin, Feng ; Chen, Ben M.: A brief survey of visual odometry for micro aerial vehicles. In: *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society* IEEE (Veranst.), 2016, S. 6049–6054

[Shapiro und Wilk 1965] Shapiro, Samuel S. ; Wilk, Martin B.: An analysis of variance test for normality (complete samples). In: *Biometrika* 52 (1965), Nr. 3/4, S. 591–611

[Sharp] Sharp: *Sharp GP2Y0E03*. Sharp (Veranst.). – URL https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0e02_03_appl_e.pdf. – [Letzter Zugriff: 10.10.2019]

[Sinha 2016] Sinha, Utkarsh: *Features: What are they?* 2016. – URL http://aishack.in/tutorials/features/. – Letzter Zugriff: 25.12.2019]

[Stanford Artificial Intelligence Laboratory et al.] Stanford Artificial Intelligence Laboratory et al.: *Robotic Operating System*. – URL https://www.ros.org

[Tarasenko und Park 2016] Tarasenko, Viacheslav ; Park, Dong-Won: Detection and Tracking over Image Pyramids using Lucas and Kanade Algorithm. In: *International Journal of Applied Engineering Research* 11 (2016), Nr. 9, S. 6117–6120

[Terzakis 2013] Terzakis, George: Relative camera pose recovery and scene reconstruction with the essential matrix in a nutshell / Technical report MIDAS. SNMSE. 2013. TR. 007, Marine and Industrial Dynamic …. 2013. – Forschungsbericht

[Terzakis u. a. 2018] Terzakis, George ; Lourakis, Manolis ; Ait-Boudaoud, Djamel: Modified Rodrigues Parameters: an efficient representation of orientation in 3D vision and graphics. In: *Journal of Mathematical Imaging and Vision* 60 (2018), Nr. 3, S. 422–442

[Terzakis u. a. 2017] Terzakis, George ; Polvara, Riccardo ; Sharma, Sanjay ; Culverhouse, Phil ; Sutton, Robert: Monocular Visual Odometry for an Unmanned Sea-Surface Vehicle. (2017)

[Tordoff und Murray 2005] Tordoff, Ben J. ; Murray, David W.: Guided-MLESAC: Faster Image Transform Estimation by Using Matching Priors. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27 (2005), Oktober, Nr. 10, S. 1523–1535. – URL https://doi.org/10.1109/TPAMI.2005.199. – ISSN 0162-8828

[Torr und Zisserman 2000] Torr, P.H.S. ; Zisserman, A.: MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. In: *Computer Vision and Image Understanding* 78 (2000), Nr. 1, S. 138 – 156. – URL http://www.sciencedirect.com/science/article/pii/S1077314299908329. – ISSN 1077-3142

[Valenti u. a. 2015] Valenti, Roberto ; Dryanovski, Ivan ; Xiao, Jizhong: Keeping a good attitude: A quaternion-based orientation filter for IMUs and MARGs. In: *Sensors* 15 (2015), Nr. 8, S. 19302–19330

[Yang 2018] Yang, Guowei: *Image Warping & Mosaicing Pt. 2.* 2018. – URL https://inst.eecs.berkeley.edu/~cs194-26/fa18/upload/files/proj6B/cs194-26-acg/. – [Letzter Zugriff: 02.12.2019]

[Zhang 2000] Zhang, Zhengyou: A flexible new technique for camera calibration. In: *IEEE Transactions on pattern analysis and machine intelligence* 22 (2000)

[Zuliani 2011] Zuliani, Marco: RANSAC for Dummies. (2011). – URL http://www.cs.tau.ac.il/~turkel/imagepapers/RANSAC4Dummies.pdf. – [Letzter Zugriff: 20.11.2019]

A Formelherleitungen

A.1 Baseline-Estimation

Da es sich bei den Kamerakoordinaten der Features um homogene handelt, sei:

$$(m_2)' = \begin{pmatrix} x_2' \\ y_2' \\ 1 \end{pmatrix} \tag{A.1}$$

und

$$m_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{A.2}$$

und

$$b_{1,2} = \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} \tag{A.3}$$

Unter diesen Annahmen lässt sich Gleichung 4.5 weiter vereinfachen:

$$\begin{pmatrix} x_2' \\ y_2' \\ 1 \end{pmatrix} = \frac{\begin{pmatrix} Z_1 \cdot x_1 - b_x \\ Z_1 \cdot y_1 - b_y \\ Z_1 \cdot 1 - b_z \end{pmatrix}}{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Z_1 \cdot x_1 - b_x \\ Z_1 \cdot y_1 - b_y \\ Z_1 \cdot 1 - b_z \end{pmatrix}} = \frac{\begin{pmatrix} Z_1 \cdot x_1 - b_x \\ Z_1 \cdot y_1 - b_y \\ Z_1 - b_z \end{pmatrix}}{Z_1 - b_1} = \begin{pmatrix} \frac{Z_1 \cdot x_1 - b_x}{Z_1 - b_z} \\ \frac{Z_1 \cdot y_1 - b_y}{Z_1 - b_z} \\ 1 \end{pmatrix} \tag{A.4}$$

$$\begin{aligned} x_2' &= \frac{Z_1 \cdot x_1 - b_x}{Z_1 - b_z} \\ \Leftrightarrow x_2' \cdot (Z_1 - b_z) &= Z_1 \cdot x_1 - b_x \\ \Leftrightarrow Z_1 \cdot x_2' - x_2' \cdot b_z &= Z_1 \cdot x_1 - b_x \\ \Leftrightarrow Z_1 \cdot x_2' - Z_1 \cdot x_1 &= x_2' \cdot b_z - b_x \\ \Leftrightarrow Z_1 \cdot (x_2' - x_1) &= x_2' \cdot b_z - b_x \end{aligned} \tag{A.5}$$

$$
\begin{aligned}
y_2' &= \frac{Z_1 \cdot y_1 - b_y}{Z_1 - b_z} \\
\Leftrightarrow y_2' \cdot (Z_1 - b_z) &= Z_1 \cdot y_1 - b_y \\
\Leftrightarrow Z_1 \cdot y_2' - y_2' \cdot b_z &= Z_1 \cdot y_1 - b_y \\
\Leftrightarrow Z_1 \cdot y_2' - Z_1 \cdot y_1 &= y_2' \cdot b_z - b_y \\
\Leftrightarrow Z_1 \cdot (y_2' - y_1) &= y_2' \cdot b_z - b_y
\end{aligned} \tag{A.6}
$$

Um die letzte Unbekannte Z_1 zu eliminieren, werden die Gleichungen A.5 und A.6 jeweils erweitert:

$$
\begin{aligned}
Z_1 \cdot (x_2' - x_1) &= x_2' \cdot b_z - b_x && | \cdot (y_2' - y_1) \\
\Leftrightarrow Z_1 \cdot (x_2' - x_1) \cdot (y_2' - y_1) &= x_2' \cdot b_z \cdot (y_2' - y_1) - b_x \cdot (y_2' - y_1)
\end{aligned} \tag{A.7}
$$

$$
\begin{aligned}
Z_1 \cdot (y_2' - y_1) &= y_2' \cdot b_z - b_y && | \cdot (x_2' - x_1) \\
\Leftrightarrow Z_1 \cdot (x_2' - x_1) \cdot (y_2' - y_1) &= y_2' \cdot b_z \cdot (x_2' - x_1) - b_y \cdot (x_2' - x_1)
\end{aligned} \tag{A.8}
$$

Dann wird Gleichung A.8 von Gleichung A.7 abgezogen:

$$
\begin{aligned}
|Z_1 \cdot (x_2' - x_1) \cdot (y_2' - y_1) = &\quad x_2' \cdot b_z \cdot (y_2' - y_1) - b_x \cdot (y_2' - y_1)| \\
-|Z_1 \cdot (x_2' - x_1) \cdot (y_2' - y_1) = &\quad y_2' \cdot b_z \cdot (x_2' - x_1) - b_y \cdot (x_2' - x_1)| \\
\hline
= \qquad\qquad 0 = &\quad -b_x \cdot (y_2' - y_1) + b_y \cdot (x_2' - x_1) \\
&+ b_z \cdot (x_2' \cdot (y_2' - y_1) - y_2' \cdot (x_2' - x_1))
\end{aligned} \tag{A.9}
$$

A.2 Outlier-Detection

Ein Vektor u wird per Definition auf die orthogonale Ebene zu einem Vektor v wie folgt projiziert:

$$u' = u - vv^T u \tag{A.10}$$

Daraus ergibt sich für den Richtungsvektor b folgende Projektionsmatrix P:

$$P = I_3 - bb^T \tag{A.11}$$

Mithilfe derer können die zwei Vektoren m_1 und m_2' projiziert werden:

$$\begin{aligned} p_1 &= Pm_1 = (I_3 - bb^T)m_1 \\ p_2 &= Pm_2' = (I_3 - bb^T)m_2 \end{aligned} \tag{A.12}$$

Ausmultipliziert ergeben die rechten Terme der Gleichung A.12 wieder den Rechten Term der Definition A.10

Über die Definition des Winkels zwischen zwei Vektoren kann der Unterschied der Ebenen e_1 und e_2 berechnet werden:

$$cos(\phi) = \frac{p_2^T p_1}{||p_2||||p_1||} \tag{A.13}$$

Mit den Gleichungen A.12 ergibt sich daraus für den Gewinn p folgende Funktion:

$$p = cos(phi) = \frac{(Pm_2')^T Pm_1}{||Pm_2'||||Pm_1||} \tag{A.14}$$

$$= \frac{(m_2')^T P^T Pm_1}{||Pm_2'||||Pm_1||} \quad | \text{ Es gilt: } P^T = T \tag{A.15}$$

$$= \frac{(m_2')^T PPm_1}{||Pm_2'||||Pm_1||} \quad | \text{ Es gilt: } PP = P \tag{A.16}$$

$$= \frac{(m_2')^T Pm_1}{||Pm_2'||||Pm_1||} \tag{A.17}$$

$$= \frac{(m_2')^T (I_3 - bb^T)m_1}{\sqrt{||m_1||^2 - (m_1^T b)^2}\sqrt{||m_2'||^2 - ((m_2')^T b)^2}} \tag{A.18}$$

Im Folgenden ist die Vereinfachung der Norm der Vektorprojektion Pm_1 hergeleitet:

$$
\begin{aligned}
||Pm_1|| =& ||(I_3 - bb^T)m_1|| \\
=& \sqrt{((I_3 - bb^T)m_1)^T(I_3 - bb^T)m_1} \\
=& \sqrt{(I_3m_1 - bb^Tm_1)^T(I_3 - bb^T)m_1} \\
=& \sqrt{((I_3m_1)^T - (bb^Tm_1)^T)(m_1 - bb^Tm_1)} \\
=& \sqrt{(m_1^T - m_1^Tbb^T)(m_1 - bb^Tm_1)} \\
=& \sqrt{(m_1^T - m_1^Tbb^T)m_1 - (m_1^T - m_1^Tbb^T)bb^Tm_1} \\
=& \sqrt{m_1^Tm_1 - m_1^Tbb^Tm_1 - m_1^Tbb^Tm_1 + m_1^Tbb^Tbb^Tm_1} && \mid m_1^Tb \text{ ausklammern} \\
=& \sqrt{m_1^Tm_1 - m_1^Tb(b^Tm_1 - b^Tm_1 + b^Tbb^Tm_1} && \mid \text{Es gilt: } u^Tv = \langle u, v\rangle = uv^T \\
=& \sqrt{m_1^Tm_1 - \langle m_1, b\rangle(\langle b, m_1\rangle + \langle b, m_1\rangle - \langle b, b\rangle\langle b, m_1\rangle)} && \mid \text{Es gilt: } ||b|| = 1 \\
=& \sqrt{m_1^Tm_1 - \langle m_1, b\rangle(\langle b, m_1\rangle + \langle b, m_1\rangle - \langle b, m_1\rangle)} \\
=& \sqrt{m_1^Tm_1 - \langle m_1, b\rangle(\langle b, m_1\rangle)} \\
=& \sqrt{||m_1||^2 - (m_1^Tb)^2} && \text{(A.19)}
\end{aligned}
$$

A.3 Iterative-Refinement

Die Kostenfunktion 4.9 erweitert sich für drei aufeinanderfolgende Frames zu den Zeitpunkten $t = 1$, $t = 2$ und $t = 3$ so:

$$C = \left((m_3')^T \left(\left[\frac{b_{2,3} + b_{1,2}}{||b_{2,3} + b_{1,2}||}\right] \times m_1\right)\right)^2 \tag{A.20}$$

Zur Vereinfachung der Darstellung sind in der oberen Gleichung die Bewegungsvektoren b im Koordinatensystem des Frames F_1 zu betrachten. Die Projektion m_3' ist ebenfalls zur Vereinfachung in das Koordinatensystem des Frames am Zeitpunkt $t = 1$ unrotiert. (Gekennzeichnet durch das $'$.)

Kostenfunktion A.20 berücksichtigt nun nicht mehr die Feature-Projektionen des Frames am Zeitpunkt $t = 2$, obwohl dieser in der zusammengesetzten Translation vorkommt. Daher muss für den Fall $N = 3$ diese Kostenfunktion mit der aus Gleichung 4.9 kombiniert

werden. Für den allgemeinen Fall kann die Kostenfunktion zum Zeitpunkt $t = l$ über die letzten N Frames nach Terzakis u. a. [2017] so definiert werden:

$$C = \sum_{j=0}^{N-1} \sum_{k=j+1}^{N} \sum_{i \in F_{l-j,l-k}} \left((m_{l-k}^{i})^{T} (R_{l-k}^{w})^{T} \left[\frac{b_{l-k,l-j}}{||b_{l-k,l-j}||} \right] \times R_{l-j}^{w} m_{l-j}^{i} \right)^{2} \tag{A.21}$$

A.4 Rotationsparametriesierung

Die Rotationsmatrix für einen Vektor um den Ursprung setzt sich im $\mathbb{R}^3$ aus folgenden Matrizen zusammen:

$$R_v = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.22}$$

Die Rotationsmatrix R_v hängt von den Parametern α, β und γ ab. Dabei dreht *alpha* den Vektor um die x-Achse, β um die y-Achse und γ um die z-Achse. Während der iterativen Optimierung wird der Vektor nicht direkt verändert, sondern indirekt über diese Parameter.

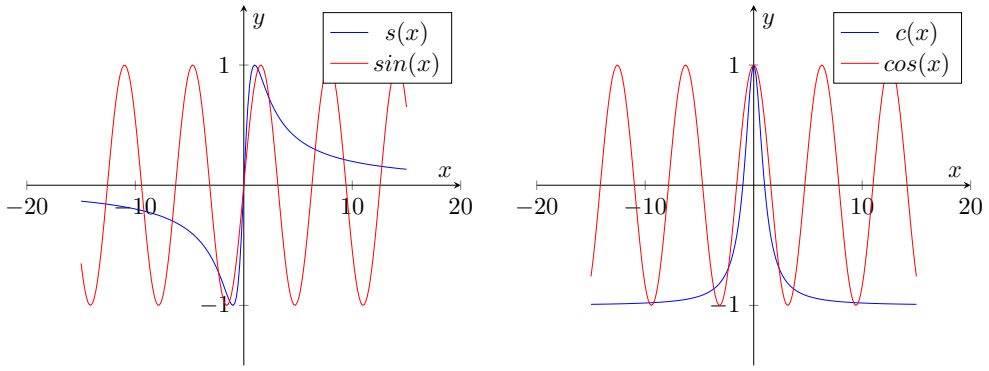
Durch die Verwendung der trigonometrischen Funktionen besteht eine periodische Abbildung von den Parametern auf die Elemente der Rotationsmatrix und damit auf die Ausrichtung des Vektors. Dieselbe Ausrichtung des Vektors kann durch unterschiedliche Wahl der Parameter mehrfach erreicht werden. Außerdem ist eine Invertierung durch eine Drehung um 180° unendlich oft möglich.

Um den Vektor in dem gewünschten Bereich zu rotieren, genügt jeweils eine Periode der Sinus- bzw. Kosinusfunktion. Folgende Funktionen bilden die reellen Zahlen jeweils auf nur eine Periode ab [Terzakis u. a., 2018]:

$$s(x) = \frac{2x}{1+x^2} \tag{A.23}$$

$$c(x) = \frac{1-x^2}{1+x^2} \tag{A.24}$$

Die Funktion $s(x)$ bildet, wie in Abbildung A.1a zu erkennen, von x im Bereich $\pm\infty$ auf eine Periode der Sinusfunktion ab. Sie konvergiert im Unendlichen gegen 0. Die



a) Die Funktion $s(x)$ bildet auf eine Periode von $sin(x)$ ab

b) Die Funktion $c(x)$ bildet auf eine Periode von $cos(x)$ ab

Abbildung A.1: Abbildung der reellen Zahlen auf nur eine Periode der Kosinus- und Sinusfunktion

Funktion $c(x)$ bildet, wie in Abbildung A.1b zu erkennen, von x im Bereich $\pm\infty$ auf eine Periode der Kosinusfunktion ab. Sie konvergiert im Unendlichen gegen -1. Außerdem ergeben die Funktionen $s(x)$ und $c(x)$ quadriert und addiert genauso wie $sin(x)$ und $cos(x)$ die Summe 1. Dies ist erforderlich, damit die Determinante der Matrix, wie für Rotationsmatrizen erforderlich, 1 ergibt.
So ergibt sich eine neue Rotationsmatrix R_v:

$$R_v = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1-a^2}{1+a^2} & -\frac{2a}{1+a^2} \\ 0 & \frac{2a}{1+a^2} & \frac{1-a^2}{1+a^2} \end{pmatrix} \begin{pmatrix} \frac{1-b^2}{1+b^2} & 0 & \frac{2b}{1+b^2} \\ 0 & 1 & 0 \\ -\frac{2b}{1+b^2} & 0 & \frac{1-b^2}{1+b^2} \end{pmatrix} \begin{pmatrix} \frac{1-c^2}{1+c^2} & -\frac{2c}{1+c^2} & 0 \\ \frac{2c}{1+c^2} & \frac{1-c^2}{1+c^2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.25}$$

B Refinement-Ideen

Ein Standardweg, um die relative Länge eines Bewegungsvektors zum Vorgänger zu ermitteln, erfolgt über die Abstandsänderung der triangulierten Features [Scaramuzza und Fraundorfer, 2011]. Da dieses Verfahren extra keine triangulierten Features benötigen soll, wird im Folgenden eine Idee beschrieben, nach der die relative Skalierung berechnet werden könnte.

B.1 Skalierungsformel

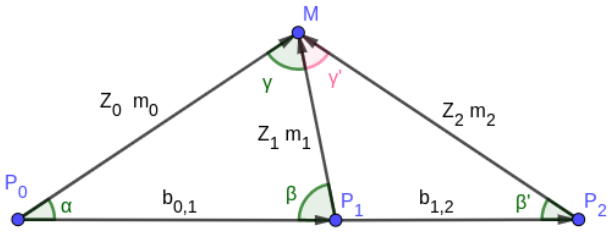


Abbildung B.1: Eine Feature-Korrespondenz über drei Frames in 2D

Mithilfe des Kosinussatzes lässt sich so die Länge von b_1 im Verhältnis zu b_0 berechnen:

$$|b_1| = \frac{|b_0| \cdot \sin(\alpha) \cdot \sin(\gamma')}{\sin(\gamma) \cdot \sin(\beta')} \tag{B.1}$$

Für die Winkel gilt:

$$\alpha = cos^{-1}\left(m_0^T \cdot u_1\right) \tag{B.2}$$

$$\gamma = cos^{-1}\left((-m_0)^T \cdot (-m_1)\right) \tag{B.3}$$

$$\beta' = cos^{-1}\left((-u_2)^T \cdot (m_2)\right) \tag{B.4}$$

$$\gamma' = cos^{-1}\left((-m_1)^T \cdot (-m_2)\right) \tag{B.5}$$

Außerdem gilt:

$$\sin(\cos^{-1}(x)) = \sqrt{1 - x^2} \tag{B.6}$$

und aus Abschnitt 4.3.3:

$$b_t = n_t \cdot \vec{u_t} \tag{B.7}$$

Daraus lässt sich dann folgender Zusammenhang ableiten.

$$n_t = \frac{n_{t-1} \cdot \sqrt{1 - \left[m_{t-2}^T u_{t-1}\right]^2} \cdot \sqrt{1 - \left[(-m_{t-1})^T (-m_t)\right]^2}}{\sqrt{1 - \left[(-m_{t-2})^T (-m_{t-1})\right]^2} \cdot \sqrt{1 - \left[(-u_t)^T m_t\right]^2}} \tag{B.8}$$

Für eine Kostenfunktion wäre folgender Term denkbar:

$$\left(n_{t-1} \cdot \sqrt{1 - \left[m_{t-2}^T u_{t-1}\right]^2} \cdot \sqrt{1 - \left[(-m_{t-1})^T (-m_t)\right]^2} \right.$$
$$\left. -n_t \cdot \sqrt{1 - \left[(-m_{t-2})^T (-m_{t-1})\right]^2} \cdot \sqrt{1 - \left[(-u_t)^T m_t\right]^2} \right)^2 \tag{B.9}$$

B.2 Relative Parametrisierung der Skalierung

Bei vorgeschlagenem Softwareentwurf mit einer Pipeline und zusätzlicher Pipeline-Stufe zur Berechnung der Bewegungsvektorlänge wird gleichzeitig auf den Längen der Bewegungsvektoren von verschiedenen Frames gearbeitet. Da die Skalierung der Bewegungslängen relativ erfolgt, muss auch die Parametrierung der Länge relativ erfolgen. Ein Bewegungsvektor b_t wäre weiterhin:

$$b_t = n_t \cdot u_t \tag{B.10}$$

Allerdings wäre die Länge n_t definiert als:

$$n_t = r_t \cdot r_{t-1} \cdot r_{t-2} \cdot ... \cdot r_0 \tag{B.11}$$

Dabei ist r_t die relative Länge des Bewegungsvektors zu seinem Vorgänger und $r_0 = 1$.

Glossar

Binning Kamera-Binning bezeichnet Bildauflösungsreduktion durch zusammenfassenBbenachberter Pixel. Der Binningfaktor (b_x, b_y) bestimmt den Faktor um den sich die Auslösung in Höhe und Breite verringert.

Feature Merkmal in Sensordaten. Im Kontext von Bildern gut erkennbare Objekte in einem Bild. Diese können sich in anderen Bildern wieder finden oder lassen sich über mehrere nachfolgende Bilder tracken.

Frame Bezeichnet das Kamerabild und die Orientierung der Kamera zu einem bestimmten Zeitpunkt. Zum Kamerabild gehören auch alle Features, die sich in diesem Erkennen lassen (Siehe auch Abschnitt 2.6).

Ground-truth Bezeichet hier die korrekte Position des Schiffes, mit der das Verfahren verglichen wird.

LIDAR Aus dem Englischen (light detection and ranging). Bezeichnet ein Gerät, welches die Entfernung zwischen sich und Objekten in der Umwelt misst. Dazu wird ein Laserstrahl verwendet. Bei einem 2D-Lidar rotiert dieser in einer Ebene und tastet so die Umgebung ab. 3D-Lidar scannen den gesamten Raum.

Odometrie Bezeichnet die Bestimmung des zurückgelegten Weges (und daraus die relative Position) eines Systems aufgrund verschiedener Sensoren.

Rollen Rotation um die Längsachse.

Stampfen Rotation um die Querachse.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Monokulare visuelle Odometrie auf einem autonomen Miniaturschiff

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original