

**Department of Physics and Astronomy
University of Heidelberg**

Bachelor Thesis in Physics
submitted by

Jan Baumgärnter

born in Heidelberg (Germany)

2019

Stützung der Relativgeschwindigkeit einer SUAU zum Untegrund

This Bachelor Thesis has been carried out by Jan Baumgärtner at the
ZITI Institute in Heidelberg
under the supervision of
Prof. Essameddin Badreddin

Abstract

The aim of the following bachelor thesis is to generate a velocity measurement of a SUAV (Small Unmanned Aerial Vehicle). An algorithm is proposed and its error evaluated. This resulted in a coarse numeric approximation as well as a model of the error dynamic, both of which were tested in a simulation. Although no measurements were conducted due to time constraints, a thorough investigation of the required hardware was conducted and the error of the measurement quantified. The work presented here is not only of interest to computer engineering students, but also physicists and anybody who is interested in nonlinear error propagation and analysis or image information extraction.

Abstrakt

Ziel der vorliegenden Bachelorarbeit war die optische Messung der Geschwindigkeit eines SUAV (*Small Unmanned Aerial Vehicle*). Hierzu wird ein Algorithmus vorgestellt und eine ausführliche Fehleruntersuchung durchgeführt. Das Ergebnis ist eine grobe numerische Abschätzung, sowie eine Quantifizierung der systematischen Fehlerdynamik. Beides wird durch eine Simulation überprüft. Obwohl keine Messungen durchgeführt werden konnten, werden auf die notwendigen Hardwarekomponenten eingegangen und die Messfehler der Sensorik bestimmt. Relevant ist diese Bachelorarbeit für alle Studenten der Technischen Informatik sowie für Physiker oder andere, die sich für nichtlineare Fehlerbetrachtungen oder der Informationsextraktion aus Bildern interessieren.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Problemstellung	2
2 Stand der Technik	5
2.1 Generierung der optischen Daten	5
2.2 Bestimmung der Geschwindigkeit	6
3 Theoretische Grundlagen	9
3.1 Shi-Tomasi Corner Detection	9
3.2 Lukas Kanade Algorithmus	11
3.3 Geschwindigkeitsbestimmung	14
3.3.1 Fehlerberechnung	17
3.3.2 Klassische Numerische Störungsanalyse	23
3.4 Vorfilterung valider Punkte	25
4 Implementierung	35
4.1 Hardware	36
4.1.1 Orientierungs- und Winkelgeschwindigkeitsfehler	38
4.1.2 Optischer Fehler	40
4.1.3 Umrechnung von Pixeln auf Meter	41
4.1.4 Fehler der Höhenmessung	42
4.2 Kalibrierung der Kamera	45
4.3 Feature Detektion	48

4.4	Feature Tracking	51
4.5	Raspberry Pi Companion Computer	52
5	Ergebnisse	57
5.1	Simulation	57
5.1.1	Bestimmung der Geschwindigkeit	58
5.1.2	Aussortieren ungeeigneter Features	71
6	Diskussion und Fazit	77
6.1	Fazit	79
7	Verbesserungsvorschläge und Zukunftsausblick	81
7.1	Inhaltlich aufbauende Verbesserungen	81
7.2	Verbesserungen des Ansatzes selbst	84
8	Anhang	95
8.1	Rotatorischer Anteil der Bildbewegung	95
8.2	Spektralnorm der Pseudoinversen	97
8.3	Erwartungswerte	98
8.4	In der SUAV verwendete Bauteile	99
8.5	Setup und Aufbau der SUAV	99
8.5.1	Pixhawk Setup	99
8.5.2	Raspberry Pi setup	103
8.5.3	Hardware Aufbau	103
8.5.4	Starten des Hexacopters	106
8.5.5	Landen des Hexacopters	107

Kapitel 1

Einleitung

In vielen urbanen Gebieten, wie beispielsweise Hochhäuserschluchten oder Häusern, kommt es bei GPS-Signalen zu Multipathing oder Verschattung. In Abwesenheit eines GPS-Signals kann die Geschwindigkeit einer SUAV (*Small Unmanned Aerial Vehicle*) nur über die numerische Integration ihrer Beschleunigungen berechnet werden. Dies führt allerdings zu einem im Laufe der Zeit immer größer werdenden Fehler, welcher sich einerseits aus dem statistischen Rauschen der Sensoren, andererseits auch aus einem systematischen Anteil, zusammensetzt. Dabei ist der systematische Teil durch die fehlerbehaftete Korrektur der Erdbeschleunigung g zu erklären. Ziel dieser Arbeit ist es nun, diesen Drift periodisch durch optische sowie Höhenmessungen zu korrigieren. Die bisher erprobten Ansätze, sowie ihre Stärken und Schwächen, sind im folgenden Kapitel dargelegt. Anschließend folgt im Kapitel **Theoretische Grundlagen** eine Herleitung des hier verwendeten Ansatzes, welcher einer stochastischen und numerischen Fehleranalyse unterzogen wird. Das Kapitel **Implementierung** setzt sich mit der verwendeten Hardware auseinander, dabei wird die Messgenauigkeit der Sensoren quantifiziert und das Zeitverhalten des Algorithmus bewertet. Im Kapitel **Ergebnisse** folgt schließlich eine Simulation des Ansatzes, welcher im Kapitel **Diskussion** weiter betrachtet wird. Zuletzt wird noch ein Fazit gezogen.

und Vorschläge zu Verbesserungen oder alternativen Ansätzen dargelegt.

1.1 Problemstellung

Ohne eine direkte Messung durch andere Sensoren, als die in der Regel vorhandene IMU (Inertial Measurement Unit), kann für die Bestimmung der Geschwindigkeit v_k einer SUAV zum Zeitpunkt k nur ein numerisches Integrationsverfahren verwendet werden. Unter vorläufiger Missachtung der Rotation gilt:

$$v_k = \sum_{i=0}^{k-1} a_i t_i. \quad (1.1)$$

Daraus folgt nach gaußscher Fehlerfortpflanzung:

$$\Delta v_k = \sqrt{\sum_{i=0}^{k-1} \Delta a_i^2 t_i^2}. \quad (1.2)$$

Dabei sind a_i und t_i die Beschleunigungen und Zeitunterschiede zum Messzeitpunkt i. Unter der Annahme, dass der statistische Fehler und der Zeitunterschied konstant bleiben, gilt:

$$\Delta v_k = \sqrt{\sum_{i=0}^{k-1} \Delta a^2 t^2} = \sqrt{k-1} \Delta at \quad (1.3)$$

Wie man sieht, wird der Fehler im Laufe der Zeit größer und sein Wert kann mathematisch durch eine eindimensionale Irrfahrt [13] beschrieben werden. In Abbildung 1.1 ist eine solche Irrfahrt gemessen worden. Während der Messung befand sich die SUAV ruhend auf einer geraden Oberfläche.

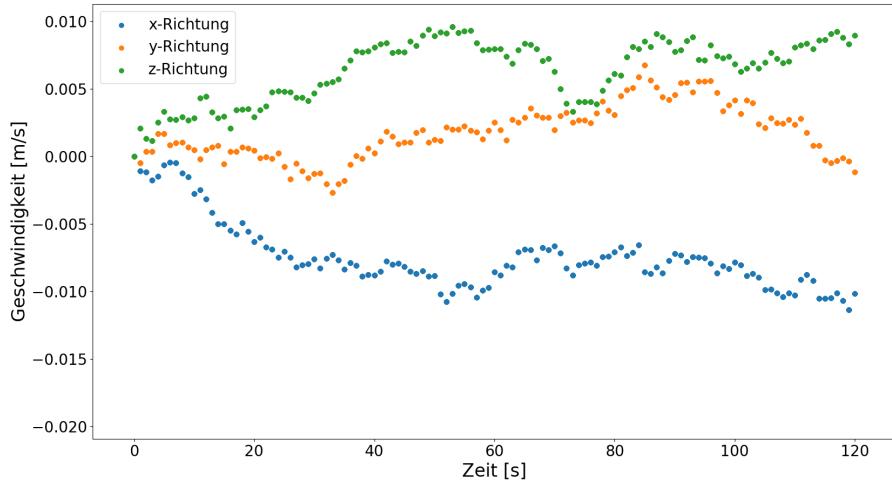


Abbildung 1.1: Drift Messung einer ruhenden SUAV

Zu bemerken ist, dass der unten rechts aufgelistete Zeitoffset eine Folge der internen Zeitmessung des im Kapitel 4.1 beschrieben Flightcontrollers ist, welcher stets nach oben zählt. Die Abweichung scheint hier zunächst klein. Da die tatsächliche Geschwindigkeit null ist, handelt es sich allerdings um eine untere Abschätzung des Fehlers, in der Realität kommt erschwerend dazu das die Startgeschwindigkeit v_0 von der IMU prinzipiell nicht gemessen werden kann. Um diesen falschen Wert zu korrigieren, ist eine äußere Stützung gesucht, welche die Geschwindigkeit der SUAV anhand optischer Daten bestimmt.

Kapitel 2

Stand der Technik

Das im vorangegangenen Kapitel skizzierte Problem kann über die Extraktion von 3D Informationen aus einem 2D Bild gelöst werden. Hierzu wurden im Laufe der Jahre verschiedene Ansätze vorgeschlagen. Im Folgenden soll getrennt auf die Bestimmung der Geschwindigkeit aus den optischen Daten und die Generierung der optischen Daten eingegangen werden. Optische Daten bezeichnen dabei aus Bildern extrahierte Informationen.

2.1 Generierung der optischen Daten

Es gab seit den 90er Jahren zahlreiche Versuche, fehlende GPS Informationen durch optische Daten zu kompensieren [23]. Eine einfache Implementierung ist die Verwendung bekannter Marker aus denen Informationen über Winkel und Abstand extrahiert werden können [17].

Diese Implementierung ist aber in ihrer Einsatzmöglichkeit stark limitiert, da künstliche Marker im Navigationsraum angebracht werden müssen. Aus diesem Grund werden in dieser, wie auch in anderen Arbeiten natürliche Features aus der Landschaft extrahiert [18] [11].

Hier wird vor allem die sogenannte *Corner Detection* verwendet, da die genaue Position der dort verwendeten Features unter geringem Rechenaufwand exakt bestimmt werden kann. Als Teil dieser Arbeit wird Shi-Tomasi Corner Detection [20] verwendet, da diese auch schon auf anderen SUAV mit großem Erfolg eingesetzt wurde [11] [12] [23]. Populär sind auch die sogenannten SIFT Features (Scale-Invariant Feature Transform), welche rotations- und skaleninvariant sind. Diese zusätzlichen Eigenschaften der SIFT Deskriptoren werden für das Verfolgen von Features nicht benötigt und erhöhen lediglich den Rechenaufwand[23]. Sogenannte FAST Features, sind wiederum weniger aufwändig zu generieren, wären jedoch deutlich anfälliger gegenüber Störungen [23].

Um die Position des Features zu verfolgen existieren einige Algorithmen, welche traditionell auch dazu verwendet werden, den sogenannten *sparse optical flow* [4] zu bestimmen. In dieser Arbeit wird der Lucas Kanade Algorithmus verwendet, da dieser auf einzelne Features angewandt werden kann und robust gegenüber Störungen ist [4] [2] (genaueres in Kapitel 3.2).

2.2 Bestimmung der Geschwindigkeit

Eine weitverbreitete Methode zur Geschwindigkeitsbestimmung ist die zeitliche Differenzierung des Lochkameramodells [4]. Die Bildkoordinaten (x, y) erhält man dabei aus dem Weltpunkt (X, Y, Z) mithilfe der folgenden Gleichungen, sowie der Brennweite f :

$$\begin{aligned} x &= f \frac{X}{Z} \\ y &= f \frac{Y}{Z} \end{aligned} \tag{2.1}$$

Differenzieren nach der Zeit liefert:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = f \begin{bmatrix} \frac{1}{Z} & 0 & -\frac{X}{Z^2} \\ 0 & \frac{1}{Z} & -\frac{Y}{Z^2} \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} \quad (2.2)$$

Dabei wird davon ausgegangen, dass die Änderung der Position eines Weltpunktes nur durch Eigenbewegung der Kamera erzeugt wird. Da diese Matrix des LGS (Lineares Gleichungssystem) jedoch nicht invertierbar ist, wird in [4] die sogenannte Nahfeldnäherung verwendet. Der Näherung nach befinden sich alle Bildpunkte nahe am Zentrum ($\mu = \nu = 0$), wodurch die Gleichung folgendermaßen vereinfacht wird:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} f \frac{v_x}{Z} \\ f \frac{v_y}{Z} \end{bmatrix} + \begin{bmatrix} f \omega_y \\ -f \omega_x \end{bmatrix}. \quad (2.3)$$

Insgesamt hat dieser Ansatz mehrere Nachteile:

Erstens ist es nötig, jedem Punkt eine Tiefe zuzuordnen und selbst wenn ein Höhensor zur Verfügung stünde, wäre η_z für jeden Punkt verschieden, sobald die Drohne in Schräglage geraten würde.

Zusätzlich sorgt die oben getroffene Annahme dafür, dass die Z-Komponente der Geschwindigkeit nicht mehr bestimmbar ist.

In [18] wurde in einer weiteren Arbeit die Differenz der Featurepositionen δx verwendet, um die translativen Geschwindigkeiten v zu berechnen:

$$\delta x = \frac{v}{d} \sin \beta - \omega \quad (2.4)$$

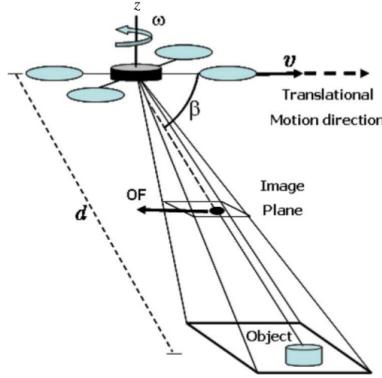


Abbildung 2.1: Variablen Definition für (2.4) aus [18]

Die Annahme hierbei ist, dass Drehungen nur um die z-Achse erfolgen, was für den Richtungswechsel eines Multicopters nicht erfüllt ist. Entsprechend wird auch diese Arbeit für ungeeignet gehalten.

In [15] wird *Epipolargeometrie* verwendet, dabei wird der namensgebende Epipol zwischen 2 Bildern gesucht. Über die Transformationsvektoren einzelner Bildpunkte in Relation zu besagtem Epipol kann die Bewegung einer Kamera rekonstruiert werden. Unter Anwesenheit eigenbewegter Punkte schlägt dieses Verfahren jedoch schnell fehl und ist deswegen ungeeignet.

In [12] verwenden die Autoren Homographien (siehe Kapitel 3.3). Es wird dabei die Annahme getroffen, dass sich alle Punkte in einer Ebene befinden und die Winkelgeschwindigkeit messbar ist, da ihre Bewegung um die Winkelgeschwindigkeit bereinigt wird. Die Entfernung zur Ebene wird über ihren Abstand und den Normalvektor ausgedrückt. Von diesem wird angenommen, dass er parallel zum Normalvektor der Ebene ist. In [11] gehen die Autoren noch einen Schritt weiter und beschreiben eine Möglichkeit, Punkte auszusortieren, welche nicht zur korrekten Ebene gehören. Besagte Sortierung funktioniert dabei nur, wenn die auszusortierenden Punkte stark in der Unterzahl sind. Beide Veröffentlichungen sind die Grundlage dieser Arbeit.

Kapitel 3

Theoretische Grundlagen

In diesem Kapitel werden die Mathematischen Grundlagen der verwendeten Methoden und Algorithmen gelegt. Diese werden dabei entweder direkt hergeleitet oder zumindest motiviert mit einer entsprechenden Referenz zu einer ausführlicheren Quelle.

3.1 Shi-Tomasi Corner Detection

Um möglichst viele Informationen aus der Bildbewegung zu ziehen, ist es von größter Wichtigkeit, Regionen des Bildes zu finden, die sich unter einer Positionsänderung besonders stark verändern.

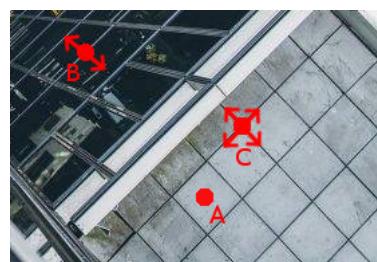


Abbildung 3.1: Flächen, Ecken und Kanten

Abbildung 3.1 veranschaulicht dies. Es ist nur schwer möglich, die Ver-

schiebung von auf Flächen befindlichen Punkten (Punkt A) zu detektieren. Bei Kanten (Punkt B) ist es immerhin möglich eine Bewegung orthogonal zur Achse zu detektieren. Bei einer Ecke (Punkt C) jedoch ist es möglich die volle Translation zu bestimmen. Man spricht bei solchen Regionen deswegen oft auch von Ecken und bei der Suche von Ecken-Erkennung oder *Corner Detection*.

Mathematisch wird der Intensitätsunterschied zweier Regionen eines schwarz-weißen Bildes $I(x, y)$ häufig durch die quadratische Abweichung $E(u, v)$ der Pixel beschrieben. Von dieser wird gefordert, dass sie zwischen dem um u, v bewegten Bild und dem Ursprünglichen möglichst groß ist.

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2. \quad (3.1)$$

Dabei beschreibt $w(x, y)$ das betrachtete Fenster über eine Stufenfunktion, welche innerhalb des Fensters 1 und außerhalb 0 ist. Nach [6] lässt sich E wie folgt beschreiben:

$$E(u, v) \approx [u \ v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad (3.2)$$

mit den partiellen Ableitungen I_x und I_y in jeweils x- und y-Richtung gilt für M :

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}. \quad (3.3)$$

Shi-Tomasi Corner Detection [20] wählt nun jene Features aus, für die

$$R = \min(\lambda_1, \lambda_2) \quad (3.4)$$

groß genug ist, wobei λ_1 und λ_2 die Eigenwerte der Matrix M sind.

3.2 Lukas Kanade Algorithmus

Nachdem Punkte gefunden wurden, deren Translation gut detektiert werden kann, müssen diese Änderungen auch gemessen werden. Anders ausgedrückt, es wird ein Algorithmus benötigt, welcher die Translation $d = (d_x, d_y)$ des Punktes bestimmen kann. Die Annahme in [2] ist, dass sich das Bild nur mittels einer affinen Matrix A transformiert:

$$A = \begin{bmatrix} 1 + d_{xx} & d_{xy} \\ d_{yx} & 1 + d_{yy} \end{bmatrix}. \quad (3.5)$$

Eine affine Transformation ist dabei eine Transformation, die folgende Eigenschaften erfüllt:

1. Geraden werden wieder auf Geraden abgebildet [9]
2. Parallelle Geraden werden auf parallele Geraden abgebildet [9]
3. Die Längenverhältnisse zwischen je 3 Punkten auf einer Gerade bleiben unverändert [9]

Für die hier betrachtete Anwendung folgt aus diesen Annahmen, dass es sich bei der Umwelt um starre Körper handeln muss und die Lichtverhältnisse näherungsweise konstant bleiben müssen. Ist $J(x, y)$ das neue Bild, dann ist eine Abbildungsmatrix A sowie ein Verschiebungsvektor d gesucht, welche die quadratische Abweichung zwischen einer Region (ω_x, ω_y) um einen Punkt \mathbf{u} im alten und neuen Bild minimiert [2].

$$\epsilon(\mathbf{d}, \mathbf{A}) = \epsilon(d_x, d_y, d_{xx}, d_{xy}, d_{yx}, d_{yy}) := \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} (I(\mathbf{x}+\mathbf{u}) - J(\mathbf{Ax}+\mathbf{d}+\mathbf{u}))^2 \quad (3.6)$$

Das Minimum ist dadurch charakterisiert, dass alle partiellen Ableitungen dort null sind. Gesucht ist also die Nullstelle aller partiellen Ableitungen von

ϵ . Diese lässt sich nach [2] als Lösung des LGS (Lineares Gleichungssystem)

$$Gd = b \quad (3.7)$$

bestimmen mit:

$$G = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x^2 & I_x I_y & x I_x^2 & y I_x^2 & x I_x I_y & y I_x I_y \\ I_x I_y & I_y^2 & x I_x I_y & y I_x I_y & x I_y^2 & y I_y^2 \\ x I_x^2 & x I_x I_y & x^2 I_x^2 & x y I_x^2 & x^2 I_x I_y & x y I_x I_y \\ y I_x^2 & y I_x I_y & x y I_x^2 & y^2 I_x^2 & x y I_x I_y & y^2 I_x I_y \\ x I_x I_y & x I_y^2 & x^2 I_x I_y & x y I_x I_y & x^2 I_y^2 & x y I_y^2 \\ y I_x I_y & y I_y^2 & x y I_x I_y & y^2 I_x I_y & x y I_y^2 & y^2 I_y^2 \end{bmatrix} \quad (3.8)$$

$$b = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x \delta I \\ I_y \delta I \\ x I_x \delta I \\ y I_x \delta I \\ x I_y \delta I \\ y I_y \delta I \end{bmatrix}. \quad (3.9)$$

Bei I_x, I_y handelt es sich um die partiellen Ableitungen des ursprünglichen Bildes I . δI ist eine Hilfsfunktion, welche durch $\delta I := I(x) - J(x)$ definiert ist. Dementsprechend gilt $d = G^{-1}b$, wobei d , wie oben beschrieben, die Verschiebung eines Features ist.

Es sollte erwähnt werden, dass in den meisten Implementierungen mehrere Iterationen des Algorithmus durchgeführt werden. Außerdem werden zwei zusätzliche Faktoren λ und δ eingeführt, welche Kontrast- und Helligkeitsänderungen codieren und den Algorithmus so robuster gegenüber Beleuchtungsänderungen machen.

Es stellt sich nun die Frage, wie die oben beschriebene Fenstergröße zu di-

mensionieren ist. Kleine Fenster sind von Vorteil, um die Details des Bildes zu erhalten (die quadratische Addition kann als eine Art Glättung verstanden werden). Um größere Bewegungen zu registrieren, sind wiederum größere Fenster nötig.

Die Schwierigkeit besteht folglich darin, einen Kompromiss zwischen Genauigkeit und Robustheit zu finden. Eine mögliche Lösung dieses Problems bietet die pyramidische Implementierung des Lucas Kanade Algorithmus. Die exakten Formeln sind in [2] nachzulesen, hier soll nur die generelle Idee umrissen werden:

Es wird zunächst eine grobe, herunterskalierte Version des Bildes betrachtet, auf welchem mit hoher Fenstergröße der Vektor d bestimmt wird. Bei jeder neuen Iteration, hier Ebene genannt, wird die Auflösung vergrößert, während die Fenstergröße verringert wird. Dabei wird die vorherige Lösung bereits auf das Bild angewendet, womit immer kleinere Residuen abgefangen werden können.

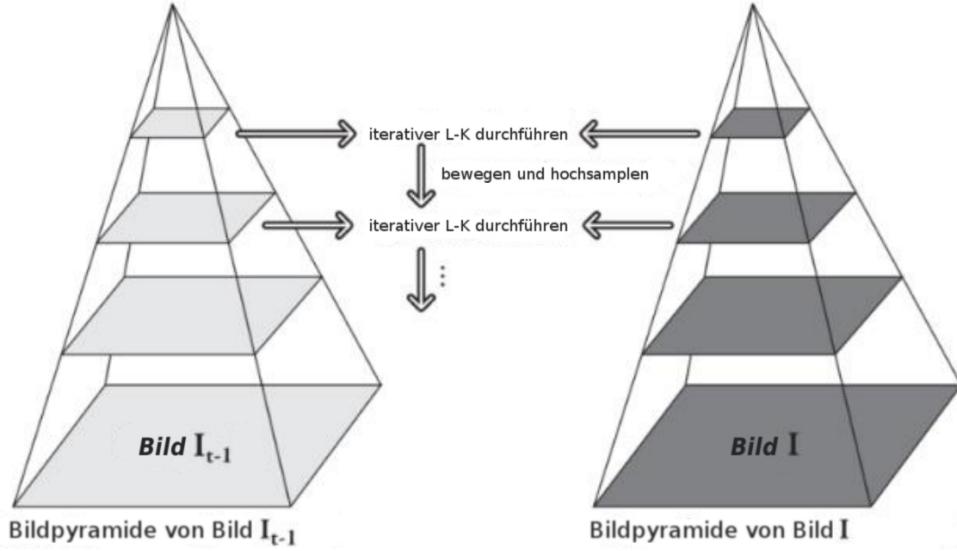


Abbildung 3.2: Pyramidalische Lucas Kanade Implementierung

3.3 Geschwindigkeitsbestimmung

Für die Bestimmung der Geschwindigkeit wird der in [12] vorgestellte Ansatz verwendet. Die kamerainduzierte Änderung der Position eines statischen Punktes i im Dreidimensionalen Raum, lässt sich auf folgende Art beschreiben:

$$\dot{X}_i = \hat{\omega} X_i + v \quad (3.10)$$

Dabei ist ab jetzt, wenn nicht anders beschrieben immer vom Kamerako-inkidenten Bezugssystem die Rede. Hierbei handelt es sich um ein Koordinatensystem, welches sich zu jedem Zeitpunkt an der Position der Kamera befindet, aber starr ist, so dass eine Geschwindigkeit definiert werden kann. $\hat{\omega}$ ist die schiefsymmetrische Matrix, die mit dem Kreuzprodukt des Drehgeschwindigkeitsvektors assoziiert ist ($\hat{\omega}x = \omega \times x \forall x \in \mathbb{R}^3$). Dem System ist jedoch nur die Koordinate x_i in der Bildebene bekannt. Entsprechend

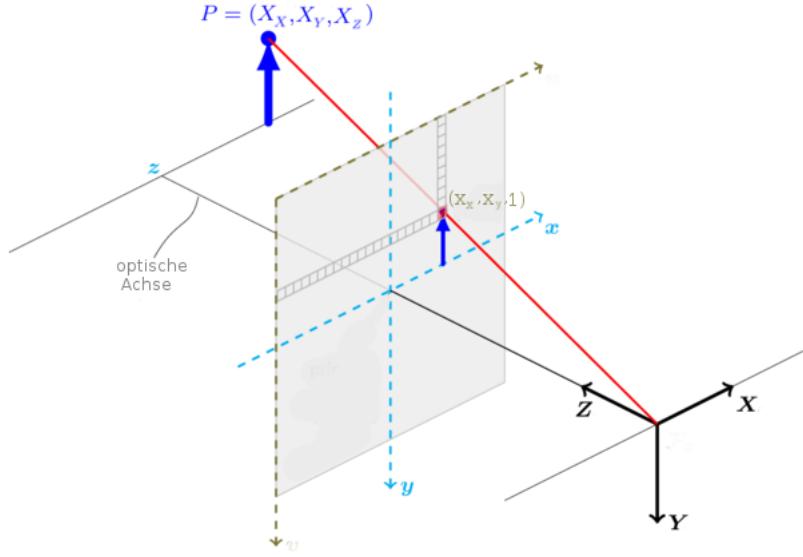


Abbildung 3.3: Kameramodell

Abbildung 3.3 gilt dabei mit der z-Komponente X_{iz} des Punktes X_i :

$$X_i = X_{iz}x_i \text{ und } \dot{X}_i = \dot{X}_{iz}x_i + X_{iz}\dot{x}_i. \quad (3.11)$$

In (3.10) eingesetzt erhält man:

$$\dot{X}_{iz}x_i + X_{iz}\dot{x}_i = \hat{\omega}X_{iz}x_i + v. \quad (3.12)$$

Um \dot{X}_{iz} zu eliminieren, multipliziert man (3.12) mit der Schiefsymmetrischen Matrix \hat{x}_i , folglich gilt $\hat{x}_i x_i = x_i \times x_i = 0$:

$$X_{iz}\hat{x}_i\dot{x}_i = \hat{x}_i(\hat{\omega}X_{iz}x_i + v). \quad (3.13)$$

Der Annahme nach befinden sich alle Punkte in einer Ebene. Dementsprechend gilt:

$$1 = \frac{n^T X_{iz} x_i}{d}, \quad (3.14)$$

wobei n der Normalvektor der Ebene ist und d die dazugehörige Distanz.
Zusammenfassend erhält man:

$$\begin{aligned} X_{i_z} \hat{x}_i \dot{x}_i &= X_{i_z} \hat{x}_i (\hat{\omega} + \frac{v}{d} n^T) x_i \\ \hat{x}_i \dot{x}_i &= \hat{x}_i (\hat{\omega} + \frac{v}{d} n^T) x_i. \end{aligned} \quad (3.15)$$

Man nennt die Matrix $\hat{\omega} + \frac{v}{d} n^T =: H$ auch Homographie. In ihr ist die gesamte Bewegungsinformation der Kamera enthalten.

In [12] haben die Autoren eine direkte Messung von ω verwendet, um \dot{x}_i von der Rotation zu bereinigen. Dabei ist ihnen nicht nur ein Fehler unterlaufen (siehe Anhang 8.1 für die korrekte Lösung), es ist aufgrund der Linearität von Matrizen auch mehr als umständlich. Werden alle messbaren Größen nämlich auf die rechte Seite verschoben, so gilt schlicht:

$$\hat{x}_i v = \hat{x}_i \frac{d}{n^T x_i} (\dot{x}_i - \hat{\omega} x_i). \quad (3.16)$$

Da $\det(\hat{x}_i) = 0$ gilt, ist \hat{x}_i nicht invertierbar und das LGS unterbestimmt. Um dieses Problem zu lösen, wird ein LGS aus allen Punkten i aufgebaut, sodass gilt:

$$Av = B, \quad (3.17)$$

mit

$$A = \begin{bmatrix} n^T x_1 \hat{x}_1 \\ n^T x_2 \hat{x}_2 \\ \dots \\ n^T x_l \hat{x}_l \end{bmatrix} \quad B = d \begin{bmatrix} \hat{x}_1 \dot{x}_1 - \hat{\omega} x_1 \\ \hat{x}_2 \dot{x}_2 - \hat{\omega} x_2 \\ \dots \\ \hat{x}_l \dot{x}_l - \hat{\omega} x_l \end{bmatrix}. \quad (3.18)$$

Dabei wurde $n^T x$ wieder auf die andere Seite geschoben um etwaige Polstellen zu vermeiden, welche die *Least-Squares* Lösung stark beeinträchtigen würden. Außerdem gilt, im Gegensatz zu [12], noch zu beachten, dass die so berechnete Geschwindigkeit im koinzidenten Kameraframe ausgedrückt ist. Für die

Geschwindigkeit im Flightcontroller koinzidenten System, muss einerseits die Orientierung der SUAV und andererseits die transversale Geschwindigkeitskomponente berücksichtigt werden. Diese ist eine Folge der Translation T_1 zwischen Controller und Kamera. Für eine nichtrotierte Kamera gilt:

$$\tilde{v} = v - \hat{\omega}T = A^+B - \hat{\omega}T_1 \quad (3.19)$$

3.3.1 Fehlerberechnung

Es folgt eine ausführliche Fehleranalyse des Ansatzes, da diese in [12] nicht vorgenommen wurde und wertvolle Informationen über die Güte liefern kann. Bei der Quantifizierung des Fehlers sollte man sich zunächst klar machen, welche Arten von Fehlern in einem solchen Verfahren auftreten können.

Unter der Annahme, dass der Eingang normalverteilt um den richtigen Wert rauscht, ist die intuitive Annahme, dass das Ausgangrauschen ebenfalls um den wahren Ausgangswert rauscht. Oder anders gesagt, dass der Mittelwert des Verrauschten Signals, dem Wert bei unverrauschem Eingang entspricht. Dies ist jedoch nur bei linearen Systemen der Fall. Zur Illustrierung stelle man sich eine normalverteilte Zufallsvariable P vor. Während $E(P) = 0$ gilt, ist $E(P^2) \neq 0$ da P^2 nur positive Werte annehmen kann.

Die Fehler des hier vorgestellten Verfahrens verhalten sich linear, mit Ausnahme des Fehlers in x_i . Hier kann es zu systematischen Fehlern kommen, welche vorher bestimmt werden sollten, da sie so möglicher Weise herausgerechnet werden können. Eine traditionelle Störungsanalyse eines LGS (Lineares Gleichungssystem), wie sie in der Numerik häufig angewendet wird, ist hier ungeeignet, da sie nicht zwischen systematischen Fehlern und Rauschen unterscheiden kann. Aufgrund der interessanten Einsichten die diese dennoch liefert, wird sie in Kapitel 3.3.2 durchgeführt.

Hier betrachtet man stattdessen ein lediglich in x_i gestörtes System und die einzelnen Messpunkte als Messungen eines stochastischen Prozesses, wobei jede Messung eine Lösung v berechnet. Es stellt sich die Frage, weshalb dies eine sinnvolle Annahme darstellt. Modelliert man x sowie Δx als Zufallsvariable, so lässt sich folgende Gleichung aufstellen:

$$v = \hat{x}^+ \frac{d}{n^T x} \hat{x}(\dot{x} - \hat{\omega}x). \quad (3.20)$$

Bei \hat{x}^+ handelt es sich um das sogenannte Pseudo- oder Moore-Penrose-Inverse. Es gilt $A^+ = (A^T A)^{-1} A^T \forall A \in \mathbb{R}^{n \times n}$. Mit Hilfe von (8.4) lässt sich \dot{x} ebenfalls durch x ausdrücken. Die gemessenen Punkte und ihren Fluss kann man nun als Stichproben x_i betrachten. Um aus Stichproben Aussagen über statistische Größen zu erlangen, bedient man sich einer sogenannten stochastischen Schätzfunktion. Ein Beispiel eines solchen Schätzers, ist die sogenannte *Least-Squares* Methode:

$$v = \min_v \sum_{i=1}^k \|v - \hat{x}^+ \frac{d}{n^T x} \hat{x}(\dot{x} - \hat{\omega}x)\|_2^2. \quad (3.21)$$

Da die Lösung eines überbestimmten LGS über die Pseudoinverse den quadratischen Fehler minimiert, ist der Erwartungswert des Schätzers äquivalent zur Lösung des LGS aus (3.17). Das bedeutet, dass der systematische Fehler sich über den stochastischen Prozess bestimmen lässt. Betrachtet man nun das gestörte System mit gestörtem Ausgang $v + \Delta v$, so findet man:

$$n^T(x + \Delta x)(\hat{x} + \Delta \hat{x})(v + \Delta v) = d(\hat{x} + \Delta \hat{x})(\dot{x} - \hat{\omega}(x + \Delta x)). \quad (3.22)$$

Auflösen nach Δv führt auf:

$$\Delta v = (\hat{x} + \Delta \hat{x})^+ \left(\frac{d}{n^T(x + \Delta x)} \Delta \hat{x}(\dot{x} - \hat{\omega}(x + \Delta x)) - \hat{x} \hat{\omega} \Delta x - \Delta \hat{x} v \right). \quad (3.23)$$

Unter Zuhilfenahme von (8.4) lässt sich \dot{x} eliminieren und man erhält:

$$\Delta v = (\hat{x} + \Delta \hat{x})^+ \left(\frac{d}{n^T(x+\Delta x)} \Delta \hat{x} ((v - v_z x) \frac{n^T x}{d} + \hat{\omega} x - (\hat{\omega} x)_z x - \hat{\omega}(x+\Delta x)) - \hat{x} \hat{\omega} \Delta x - \Delta \hat{x} v \right). \quad (3.24)$$

Das Pseudoinverse einer mit dem Kreuzprodukt assoziierten Matrix \hat{A} ist $\hat{A}^+ = -\frac{\hat{A}}{\|\hat{A}\|_2^2}$. Damit gilt für die Fehlerrechnung:

$$\Delta v = -\frac{\hat{x} + \Delta \hat{x}}{\|\hat{x} + \Delta \hat{x}\|_2^2} \left(\frac{d}{n^T(x+\Delta x)} \Delta \hat{x} ((v - v_z x) \frac{n^T x}{d} + \hat{\omega} x - (\hat{\omega} x)_z x - \hat{\omega}(x+\Delta x)) - \hat{x} \hat{\omega} \Delta x - \Delta \hat{x} v \right). \quad (3.25)$$

Diese unhandliche Formel lässt sich in zwei Teile spalten: einen von ω abhängigen und einen von v abhängigen. Für ersteren gilt:

$$\Delta v_\omega = \frac{d}{n^T x \|x + \Delta x\|_2^2} (\hat{x} + \Delta \hat{x})(\Delta \hat{x} x (\hat{\omega} x)_z + \hat{\omega} \Delta x + \hat{x} \hat{\omega} \Delta x), \quad (3.26)$$

ausmultiplizieren liefert:

$$\begin{aligned} \Delta v_\omega = & \frac{d}{n^T x \|x + \Delta x\|_2^2} (\hat{x} \Delta \hat{x} x (\hat{\omega} x)_z + \Delta \hat{x} \Delta \hat{x} x (\hat{\omega} x)_z + \hat{x} \hat{\omega} \Delta x + \\ & \Delta \hat{x} \hat{\omega} \Delta x + \hat{x} \hat{x} \hat{\omega} \Delta x + \Delta \hat{x} \hat{x} \hat{\omega} \Delta x). \end{aligned} \quad (3.27)$$

Unter Zuhilfenahme der Grassmann Identität und Wegstreichen von orthogonalen Skalarprodukten lässt sich der Term mit den in Kapitel 8.3 zusammengefassten Regeln weiter vereinfachen und man erhält für den gesamten

Erwartungswert:

$$\begin{aligned}
E(\Delta v_\omega) = & d \left[E\left(\frac{(\hat{\omega}x)_z \Delta x x^T x}{n^T x \|x + \Delta x\|_2^2}\right) - E\left(\frac{(\hat{\omega}x)_z x \Delta x^T x}{n^T x \|x + \Delta x\|_2^2}\right) \right. \\
& + E\left(\frac{(\hat{\omega}x)_z \Delta x \Delta x^T x}{n^T x \|x + \Delta x\|_2^2}\right) - E\left(\frac{(\hat{\omega}x)_z x \Delta x^T \Delta x}{n^T x \|x + \Delta x\|_2^2}\right) \\
& + E\left(\frac{\hat{\omega}x^T \Delta x}{n^T x \|x + \Delta x\|_2^2}\right) - E\left(\frac{\Delta x x^T \omega}{n^T x \|x + \Delta x\|_2^2}\right) \\
& + E\left(\frac{\hat{\omega} \Delta x^T \Delta x}{n^T x \|x + \Delta x\|_2^2}\right) - E\left(\frac{\Delta x \Delta x^T \omega}{n^T x \|x + \Delta x\|_2^2}\right) \\
& + E\left(\frac{x x^T \hat{\omega} \Delta x}{n^T x \|x + \Delta x\|_2^2}\right) - E\left(\frac{\hat{\omega} \Delta x x^T x}{n^T x \|x + \Delta x\|_2^2}\right) \\
& \left. - E\left(\frac{\hat{\omega} \Delta x \Delta x^T x}{n^T x \|x + \Delta x\|_2^2}\right) \right]. \tag{3.28}
\end{aligned}$$

Die einzelnen Erwartungswerte hängen nun von den spezifischen Verteilungen ab. Generell führt der Bruch jedoch zu einer Pol- und damit Unstetigkeitsstelle, wodurch nach dem Fundamentalsatz der Analysis keine geschlossene Form des Erwartungswertintegrals gefunden werden kann.

In einfachen Fällen ist es allerdings zumindest möglich, die Dynamik des Systems abzuschätzen. Vereinfachend wird davon ausgegangen, dass der Messfehler Δx normalverteilt ist mit der Standardabweichung σ und dem Erwartungswert 0. Des Weiteren sei x achsensymmetrisch bezüglich der x- und y-Achse verteilt. Zusätzlich sei der Normalvektor parallel zur Z-Achse. Es werden zuerst Summanden, die nur ein x im Zähler enthalten, betrachtet (hier beispielhaft $\frac{x_y \Delta x_x}{n^T x \|x + \Delta x\|_2^2} \omega_x$).

$$E\left(\frac{x_y \Delta x_x}{n^T x \|x + \Delta x\|_2^2}\right) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{x_y \Delta x_x}{\|x + \Delta x\|_2^2} \frac{\exp\left(-\frac{x^2}{2\sigma^2}\right)}{\sqrt{2\pi\sigma}} d\Delta x dx \tag{3.29}$$

Mithilfe der Dreiecksungleichung lässt sich eine Minorante und Majorante

für das Integral finden:

$$\begin{aligned}
& \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{x_y \Delta x_x}{\|x\|_2^2 + \|\Delta x\|_2^2} \frac{\exp\left(-\frac{x^2}{2\sigma}\right)}{\sqrt{2\pi\sigma}} d\Delta x_x dx_y \\
& \leq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{x_y \Delta x_x}{\|x + \Delta x\|_2^2} \frac{\exp\left(-\frac{x^2}{2\sigma}\right)}{\sqrt{2\pi\sigma}} d\Delta x_y dx_x \\
& \leq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{x_y \Delta x_x}{\|x\|_2^2 - \|\Delta x\|_2^2} \frac{\exp\left(-\frac{x^2}{2\sigma}\right)}{\sqrt{2\pi\sigma}} d\Delta x_y dx_x.
\end{aligned} \tag{3.30}$$

Sowohl die Majorante, als auch die Minorante sind nun in Δx_y und x_x um die y-Achse symmetrische Funktionen, d.h. es gilt $f(-x) = -f(x)$. Das Integral über eine derartige symmetrische Funktion ist gleich null. Da beide Integrale also null sind, ist die das Integral über die Funktion selbst null. Analog lässt sich ableiten, dass jeder Erwartungswert mit einem einzigen $x_x, x_y, x_z, \Delta x_x, \Delta x_y, \Delta x_z$ unter den getroffenen Annahmen gleich null ist. Wird zusätzlich angenommen, dass sich das Problem symmetrisch in x_x und x_y verhält, erhält man vereinfachend:

$$E(\Delta v_\omega) = d \left[E\left(\frac{x_x^2 \Delta x_x^2}{\|x + \Delta x\|_2^2}\right) - E\left(\frac{\Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \right] \begin{pmatrix} \omega_y \\ -\omega_x \\ 0 \end{pmatrix}. \tag{3.31}$$

Analog lässt sich auch der von v abhängige Teil bestimmen:

$$\Delta v_v = \frac{-(\hat{x} + \Delta \hat{x})}{\|x + \Delta x\|_2^2} \left(\left[\frac{n^T x}{n^T (x + \Delta x)} - 1 \right] \Delta \hat{x} v - \frac{n^T x}{n^T (x + \Delta x)} v_z x \right). \tag{3.32}$$

Daraus kann wieder der Erwartungswert berechnet werden und nach Ausmultiplizieren und Trennen der Summanden folgt:

$$\begin{aligned}
E(\Delta v_v) &= E\left(\frac{n^T}{n^T(x + \Delta x)} \frac{\Delta x x^T v}{\|x + \Delta x\|_2^2}\right) - E\left(\frac{\Delta x x^T v}{\|x + \Delta x\|_2^2}\right) \\
&\quad - E\left(\frac{n^T x}{n^T(x + \Delta x)} \frac{v x^T \Delta x}{\|x + \Delta x\|_2^2}\right) + E\left(\frac{v x^T \Delta x}{\|x + \Delta x\|_2^2}\right) \\
&\quad + E\left(\frac{n^T x}{n^T(x + \Delta x)} \frac{\Delta x \Delta x^T v}{\|x + \Delta x\|_2^2}\right) - E\left(\frac{\Delta x \Delta x^T v}{\|x + \Delta x\|_2^2}\right) \\
&\quad - E\left(\frac{n^T x}{n^T(x + \Delta x)} \frac{v \Delta x^T \Delta x}{\|x + \Delta x\|_2^2}\right) + E\left(\frac{v \Delta x^T \Delta x}{\|x + \Delta x\|_2^2}\right) \\
&\quad - E\left(\frac{n^T x}{n^T(x + \Delta x)} v_z \hat{x} x\right) - E\left(\frac{n^T x}{n^T(x + \Delta x)} v_z \Delta \hat{x} x\right).
\end{aligned} \tag{3.33}$$

Unter den oben genannten Annahmen über die Verteilungen gilt hier vereinfachend:

$$E(\Delta v_v) = E\left(\frac{\Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \begin{pmatrix} 0 \\ 0 \\ -2v_z \end{pmatrix} \tag{3.34}$$

und damit insgesamt:

$$E(\Delta v) = E\left(\frac{\Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \begin{pmatrix} -d\omega_y \\ d\omega_x \\ -2v_z \end{pmatrix} + E\left(\frac{x_x^2 \Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \begin{pmatrix} d\omega_y \\ -d\omega_x \\ 0 \end{pmatrix}. \tag{3.35}$$

Dies ist natürlich ein besonders einfacher Spezialfall. Vor allem die Annahme der Achsensymmetrie in x und y, ist in der Regel nicht erfüllt, da je nach Bild keine Rücksicht auf eine optimale Positionierung der Punkte genommen werden kann.

Dennoch ist es selbst hier nicht möglich, die Erwartungswerte analytisch zu bestimmen, da die Funktionen aufgrund der Polstellen keine Stammfunktionen haben. Es folgt, dass der systematische Fehler nicht in Realzeit korrigiert werden kann, obwohl seine Dynamik beschreibbar ist. Um den Fehler

zumindest nach oben abschätzen zu können, folgt hier noch eine numerische Störungsanalyse.

3.3.2 Klassische Numerische Störungsanalyse

Beim Quantifizieren des Fehlers dieses Verfahrens, muss beachtet werden, dass hier nicht nur der Messfehler eine Rolle spielt, sondern auch die Lösung des LGS einen Fehler hinzufügt, welcher selbst bei perfekten Messungen auftreten würde. Man betrachte hierzu das gestörte LGS:

$$(A + \Delta A)(v + \Delta v) = B + \Delta B \quad (3.36)$$

Wobei der jeweilige Fehler mit Δ bezeichnet wird. Es gilt $\dim(\Delta A) = \dim(A)$, $\dim(\Delta v) = \dim(v)$ und $\dim(\Delta B) = \dim(B)$. Auflösen nach Δv führt auf:

$$\Delta v = (A + \Delta A)^+(B + \Delta B - (A + \Delta A)v). \quad (3.37)$$

Der zweite Beitrag lässt sich dabei als negatives Residuum $R = B + \Delta B - (A + \Delta A)v$ des LGS unter Einsetzen der korrekten Lösung verstehen. Entsprechend folgt die triviale Aussage, dass der Fehler der Geschwindigkeit verschwindet, wenn das Residuum gegenüber der korrekten Lösung verschwindet.

Es lässt sich aber noch ein wenig mehr herauslesen, denn $Av - B$ ist das Residuum des ungestörten LGS. Es folgt, dass dieses Residuum nicht null sein sollte, sondern in der Größenordnung des Fehlers ist. Insgesamt fällt es aber schwer, die Form von R zu quantifizieren, da nur die Beträge von ΔA und ΔB abgeschätzt werden können.

Dementsprechend wird hier die Norm des Fehlers betrachtet. Unter Verwendung der mit der Euklidischen Norm verträglichen Spektralnorm (siehe

Anhang 8.2), gilt:

$$\|\Delta v\|_2 = \|(A + \Delta A)^+ \|_2 \|B + \Delta B - (A + \Delta A)v\|_2. \quad (3.38)$$

$\|(A + \Delta A)^+\|_2$ entspricht dabei gerade $[\sigma_{\min}(A + \Delta A)]^{-1}$, wobei σ_{\min} der kleinste Singulärwert der Matrix ist (für eine Herleitung siehe Anhang 8.2). Entsprechend erhält man:

$$\|\Delta v\|_2 \leq \frac{\|B + \Delta B - (A + \Delta A)v\|_2}{\sigma_{\min}(A + \Delta A)}. \quad (3.39)$$

Dabei sind $B + \Delta B$ sowie $A + \Delta A$ direkt messbar. Diese Abschätzung erleichtert zwar die Betrachtung des Fehlers massiv, man sollte jedoch vorsichtig sein, denn durch die obige Abschätzung vergrößert sich der numerische Fehler mit erhöhter Anzahl an betrachteten Features, während er sich im Falle einer normalen *Least-Squares* Lösung verkleinern sollte. Wie problematisch dieses Phänomen ist, wird in Kapitel 5.1 simulativ betrachtet.

Interessant sind Aussagen über die Dynamik des Fehlers, also sein Verhalten unter beliebigen Geschwindigkeiten. Hierzu ist es sinnvoll, die Darstellung so zu ändern, dass der Einfluss von ω direkt beobachtet werden kann. Dazu wird hier in einer Abschätzung jedes Feature einzelnen betrachtet. Für jedes Feature gilt dabei:

$$\Delta v_i = \|(\hat{x}_i + \Delta \hat{x}_i)[(d + \Delta d)(\dot{x}_i + \Delta \dot{x}_i + (\hat{x}_i + \Delta \hat{x}_i)(\omega + \Delta \omega)) - (n + \Delta n)^T(x_i + \Delta x_i)v]\|_2. \quad (3.40)$$

Es ist außerdem zu erwähnen, dass nur Fehler 1. Ordnung beachtet werden, da höhere Ordnungen bei einem Grundfehler der Größenordnung 10^{-3} tausendmal kleiner sind (siehe Kapitel 4.1). Außerdem kann nun \dot{x}_i mit (8.4)

eliminiert werden, wodurch sich folgender Fehler ergibt:

$$\begin{aligned} & \|(\hat{x}_i + \Delta\hat{x}_i)[\Delta d((v - e_z^T v x_i) \frac{n^T x_i}{d} - \hat{x}_i \omega + e_z^T \hat{x}_i \omega x_i + \hat{x}_i \omega) \\ & + d((v - e_z^T v x_i) \frac{n^T x_i}{d} - \hat{x}_i \omega + e_z^T \hat{x}_i \omega x_i + \Delta \dot{x}_i + \hat{x}_i \omega + \Delta \hat{x}_i \omega \hat{x}_i \Delta \omega) \\ & - n^T x_i v - \Delta n^T x_i v - n^T \Delta x_i v]\|_2. \end{aligned} \quad (3.41)$$

Da $\hat{a}a = a \times a = 0 \forall a \in \mathbb{R}^3$, lassen sich einige Terme streichen. Insgesamt folgt nach dem Kürzen:

$$\|\hat{x}\left[\left(\frac{\Delta d}{d}n^T x_i - \Delta n^T x_i - n^T \Delta x_i\right)v + d(\Delta \dot{x}_i + \Delta \hat{x}_i \omega + \hat{x}_i \Delta \omega + \Delta x_i)\right]\|_2. \quad (3.42)$$

Der Fehler lässt sich nun in drei Teile aufteilen: einen von v abhängigen, einen von ω abhängigen und einen dynamikunabhängigen. Des Weiteren fällt auf, dass der durch die Drehgeschwindigkeit hervorgerufene Fehler unabhängig von der Orientierung ist. Die statistische Betrachtung impliziert aber genau das Gegenteil. Sie impliziert, dass der lineare Geschwindigkeitsanteil unabhängig ist. Es stellt sich die Frage, wie das zusammenpasst. Die Erklärung ist, dass der Fehler durch den minimalen Singulärwert der Matrix $A + \Delta A$ geteilt wird. Dort findet sich ein $n^T x_i$, was den Unterschied genau erklärt.

3.4 Vorfilterung valider Punkte

Das oben vorgestellte Verfahren arbeitet unter der Annahme, dass sich alle Punkte in einer Ebene befinden und ihre Eigenbewegung nur durch die Bewegung der Drohne induziert wird. Für ein vorgegebenes Feature ist aber nicht von vornehmehin klar, ob es diese Annahmen erfüllt. In der folgenden Grafik sind einige der Probleme dargestellt.



Abbildung 3.4: Hochhäuserschlucht mit zum Boden parallele Ebenen

In Dunkelrot sind hier bewegte Strukturen eingezeichnet, deren Bewegung nicht alleine aus der Bewegung der SUAV resultiert. Die Lila eingezzeichneten Strukturen sind Spiegelungen, deren Bewegung nicht direkt mit der Bewegung der Drohne korrespondiert. Außerdem sind im Bild mehrere Ebenen zu sehen, wobei die Höhenmessung nur für eine dieser Ebenen gilt. Die Be trachtung der anderen Ebenen verfälscht das LGS folglich. Wie bereits in der numerischen Fehlerrechnung gesehen wurde, ist die Minimierung des Residuums ein gutes Maß für Fehlerminimierung und wird daher auch von [12] verwendet. Dieses Residuum wird dabei für jedes Feature einzeln berechnet:

$$R_i = \hat{x}_i v - \hat{x}_i (\dot{x}_i - \hat{\omega} x_i) \frac{d}{n^T x}. \quad (3.43)$$

Damit diese Aussortierung jedoch funktioniert, muss es eine dominante Ebene im Bild geben, sodass alle nicht zu dieser Ebene zugehörigen Punkte

identifiziert werden können. Wie in Abbildung 3.4 zu sehen ist, muss dies jedoch nicht unbedingt der Realität entsprechen. Mithilfe der Beschleunigungsmessungen aus einer IMU kann dieses Problem behoben werden:

Unter der Annahme, dass die optische Stützung häufig genug erfolgt, sollte es möglich sein, die driftbehaftete Geschwindigkeit zu nutzen, um zumindest eine grobe Schätzung \bar{v} zu erhalten. Damit lässt sich nun eine andere Art Residuum bestimmen.

$$\bar{R}_i = \hat{x}_i(\bar{v} - \hat{\omega}T_1) - \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\frac{d}{n^T x}. \quad (3.44)$$

Dieser Vektor wird nur dann Null, wenn

$$\hat{x}_i(\bar{v} - \hat{\omega}T_1) = \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\frac{d}{n^T x} \quad (3.45)$$

gilt. Für die Kategorisierung der Punkte wird eine Metrik gesucht, welche eine scharfe Unterscheidung zwischen korrekten und inkorrekten Werten ermöglicht. Die Wahl einer optimalen Metrik ist allerdings ein separates Problem, welches in dieser Arbeit nicht behandelt werden soll. Einige Beispielmetriken sind in [3] aufgeführt.

Eine Möglichkeit, eine Metrik zu finden, ist über die Verwendung einer Norm oder eines Skalarproduktes. In [11] nutzen die Autoren die euklidische Distanzmetrik um $\|R\|_2$ zu berechnen, analog kann hier $\|\bar{R}\|_2$ berechnet werden. Der inhärente Nachteil dieser Metrik ist jedoch, dass sie nicht zwischen Features anderer Ebenen oder bewegten bzw. gespiegelten unterscheiden kann.

Dies ist für eine simple Aussortierung zwar nicht nötig, hat aber praktische Vorteile. Wie einfach einzusehen ist, ist das suchen neuer Features deutlich langsamer, als das verfolgen bekannter (siehe 4.5). Überfliegt die SUAV eine Ebene und stößt auf eine andere, muss sie sich im herkömmlichen Fall sofort neue Features suchen.

Können parallele Ebenen jedoch separat getrackt werden, ist es möglich, einfach mit einer weiteren Untermenge an Punkten weiterzurechnen. Folglich lässt sich hoher Rechenaufwand bei Ebenenwechseln gegen leicht erhöhten Rechenaufwand bei einem Flug über eine Ebene eintauschen.

Daneben ist es sogar möglich, so den Höhenunterschied zwischen mehreren Ebenen zu bestimmen, um sich dieser zur Geschwindigkeitsbestimmung zu bedienen, wenn die momentan überflogene Ebene keine geeigneten Features besitzt.

Zur Aussortierung sich selbst bewegender, sowie gespiegelter Features lässt sich der Winkel zwischen $\hat{x}_i(\bar{v} - \hat{\omega}T_1)$ und $\hat{x}_i(\dot{x}_i - \hat{\omega}x_i) \frac{d}{n^T x}$ betrachten. Beide Vektoren sollten antiparallel sein; eine Bedingung, die für jede beliebige Ebene gilt, da diese sich nur in d unterscheiden. Mathematisch lässt sich dies über ein normiertes Skalarprodukt beschreiben, welches in [3] Kosinusähnlichkeit (*Cosine Similarity*) genannt wird:

$$\frac{\hat{x}_i(\bar{v} - \hat{\omega}T_1) \cdot 2\hat{x}_i(\dot{x}_i - \hat{\omega}x_i)}{\|\hat{x}_i(\bar{v} - \hat{\omega}T_1)\|_2 \|\hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\|_2} \stackrel{!}{=} -1. \quad (3.46)$$

Nach der Aussortierung der bewegten Punkte bleibt jetzt noch die Gruppierung der Features in Ebenen. Dabei gilt es auch zu beachten, dass sich im Bild auch Einzelfeatures befinden können, welche nicht Teil einer Ebene sind und ebenfalls aussortiert werden sollten. Bis jetzt wurde der Winkel betrachtet, nun wird sich dem Längenunterschied $|\|\hat{x}_i(\bar{v} - \hat{\omega}T_1)\|_2 - \|\hat{x}_i(\dot{x}_i - \hat{\omega}x_i) \frac{d}{n^T x}\|_2|$ zugewandt. Punkte der korrekten Ebene minimieren diesen Abstand, wobei für d gilt:

$$d = \frac{\|\hat{x}_i(\bar{v} - \hat{\omega}T_1)\|_2}{\|\hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\|_2} n^T x. \quad (3.47)$$

Um parallele Ebenen zu finden, werden statistische Methoden bemüht. Die Annahme ist, dass die Distanzen d jeder Ebene normalverteilt sind und der

Höhenunterschied zwischen Ebenen größer als ihre Standardabweichung ist. So lässt sich das Problem umformulieren in die Suche von Normalverteilungen in einer verrauschten Statistik.

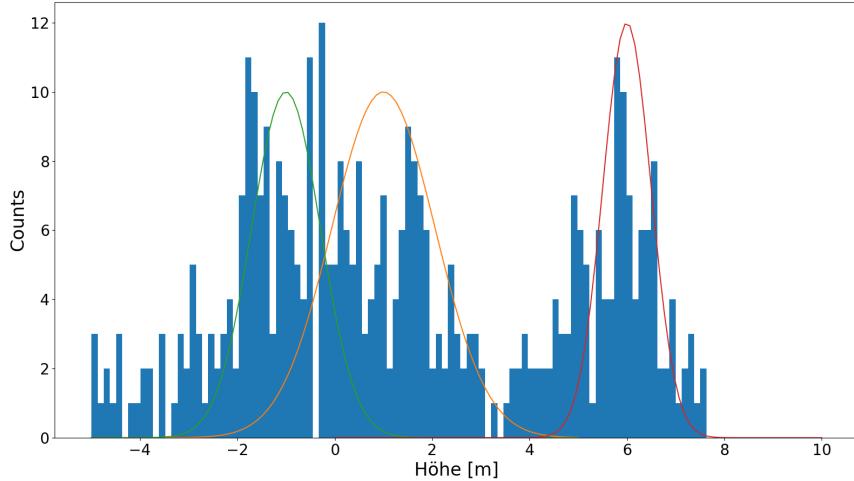


Abbildung 3.5: Schematische Darstellung einer verrauschten Statistik mit 3 Gaußverteilungen

Der erste Schritt ist das Aufteilen der Distanzen in potentielle Intervalle. Hierzu werden die mit (3.47) bestimmten Distanzen sortiert und die Differenz gegenüber dem Nachbar gebildet. Die Teilung des Intervalls erfolgt nun zwischen allen Elementen, deren Abstand größer ist, als die erwartete Standardabweichung einer einzelnen Ebene. Dies ist eine sehr konservative Schranke, die der potentiell kleinen Statistik pro Ebene geschuldet ist.

Für jedes Intervall muss nun festgestellt werden, ob es sich um eine Gaußverteilung handelt, oder nicht. Hierzu werden verschiedene statistische Momente verwendet:

$$\begin{aligned}
\text{Mittelwert} &:= m_1 = \frac{1}{n} \sum_{i=1}^n d_i, \\
\text{Varianz} &:= m_2 = \frac{1}{n} \sum_{i=1}^n (d_i - m_1)^2, \\
\text{Schiefe} &:= m_3 = \frac{1}{n} \sum_{i=1}^n \left(\frac{d_i - m_1}{m_2} \right)^3, \\
\text{Wölbung} &:= m_4 = \frac{1}{n} \sum_{i=1}^n \left(\frac{d_i - m_1}{m_2} \right)^4.
\end{aligned} \tag{3.48}$$

Von einer Gaußverteilung wird dabei erwartet, dass die Varianz der in der Fehlerrechnung vorhergesagten entspricht und dass sich der Mittelwert in der Mitte des Intervalls befindet, da die Verteilung symmetrisch ist. Die Schiefe ist entsprechend 0. Die Wölbung wiederum ist 3 [8] (und damit ist die sogenannte kurtosis, welche als die Wölbung minus Drei definiert ist, gleich Null). Zusammenfassend wird folgendes erwartet:

$$\begin{aligned}
m_1 &= \frac{d_{\max} - d_{\min}}{2}, \\
m_2 &= \sigma, \\
m_3 &= 0, \\
m_4 &= 3.
\end{aligned} \tag{3.49}$$

Die Nützlichkeit dieser Kriterien hängt einerseits vom Messfehler der Höhe, anderseits von der Stärke der Statistik ab. Zunächst sollte man sich dabei klar machen, welche Arten von Intervallen auftreten:

1. Das Intervall enthält keine Gaußfunktion
2. Das Intervall enthält eine Gaußfunktion (mit Rauschen)

3. Das Intervall enthält mehrere Gaußfunktionen (mit Rauschen).

Der erste Fall, unterscheidet sich in mindestens einem der vier Maße, sonst würde es sich um eine Gaußverteilung handeln.

Der zweite Fall besitzt zwar dieselbe Schiefe, durch den angehängten Fortsatz befindet sich der Mittelwert jedoch nicht in der Mitte des Intervalls. Da auch die Wölbung näherungsweise übereinstimmen sollte, ist es so möglich, den zweiten Fall zu identifizieren.

Der dritte Fall ist insofern besonders, als das er je nach Anzahl der Gaußfunktionen in mehreren Formen auftritt. Allgemein ist seine Varianz allerdings zu groß.

Es wird hier nicht weiter versucht ein Intervall mit mehreren Gaußverteilungen in einzelne aufzuteilen. Wäre eine rechnerische einfache Methode bekannt, so würde diese hier bereits zur Identifizierung der Intervalle verwendet werden.

In den Abbildungen 3.6, 3.7 und 3.8 ist im Folgenden das Verhalten der verschiedenen Maße simuliert. Dabei werden drei Verteilungen betrachtet, eine Gaußverteilung mit $\sigma = 1$ und $\mu = 0$, eine doppelte Gaußverteilung wobei sich der zweite Peak bei $\mu = 2$ befindet, sowie gleichverteiltes Rauschen auf dem Intervall [-2,2]. Diese werden jeweils bei verschiedener Samplegröße verglichen, die Simulation jeder Größe wird dabei hundert Mal durchgeführt, um einen Mittelwert und eine Standardabweichung bestimmen zu können.

Es bestätigen sich hier die oberen Aussagen über die Einzelverteilungen und gleichzeitig lässt sich eine Mindestanzahl an Punkten pro Ebene ablesen, welche für eine Unterscheidung jenseits einer Standardabweichung nötig sind. In (c) kann man dabei ablesen, dass ab 40 Features zwischen Rauschen und

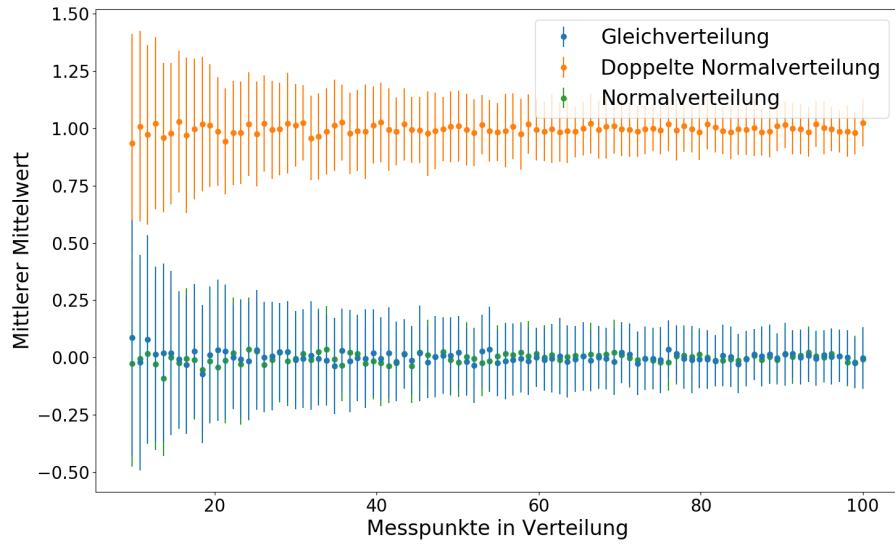


Abbildung 3.6: Erwartungswert m_1

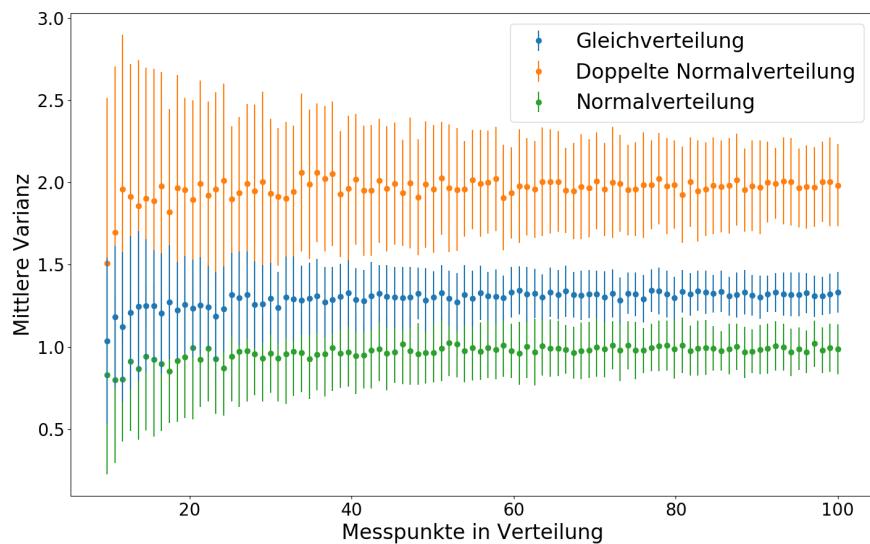


Abbildung 3.7: Varianz m_2

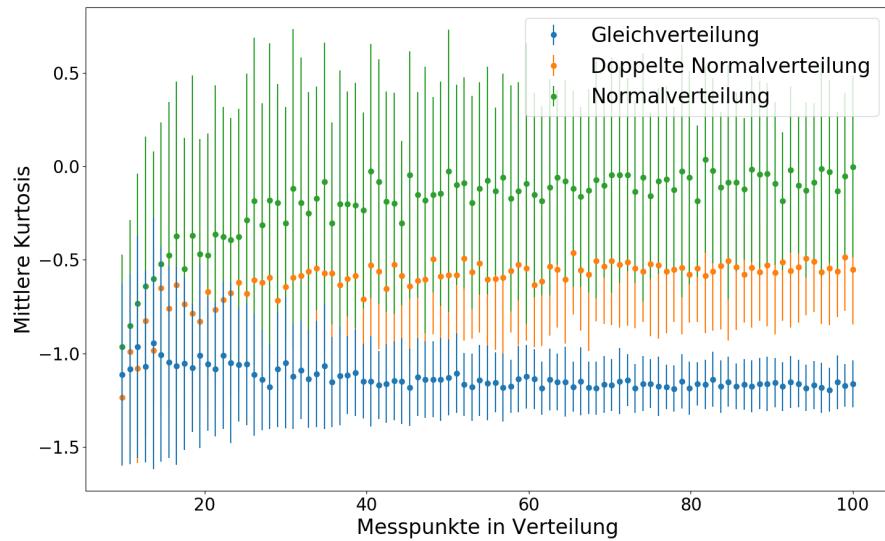


Abbildung 3.8: Kurtosis m_4

gaußartiger Verteilung unterschieden werden kann. In (a) wiederum erkennt man, dass bereits ab 20 Features zwischen Fall zwei und drei unterschieden werden kann.

Diese Größen sind für den endgültigen Algorithmus natürlich nicht aussagekräftig, da die Unterscheidungsfähigkeit von der Standardabweichung der Gaußkurven sowie der Breite des Rauschens abhängt, illustrieren aber, dass eine Unterscheidung anhand dieser Größen prinzipiell möglich ist.

Kapitel 4

Implementierung



Abbildung 4.1: SUAV

In diesem Kapitel wird die Implementierung des eben beschriebenen Algorithmus beschrieben. Hierbei wird auf das Zeitverhalten des Verfahrens, die betrachtete Hardware und das *tuning* eingegangen.

4.1 Hardware

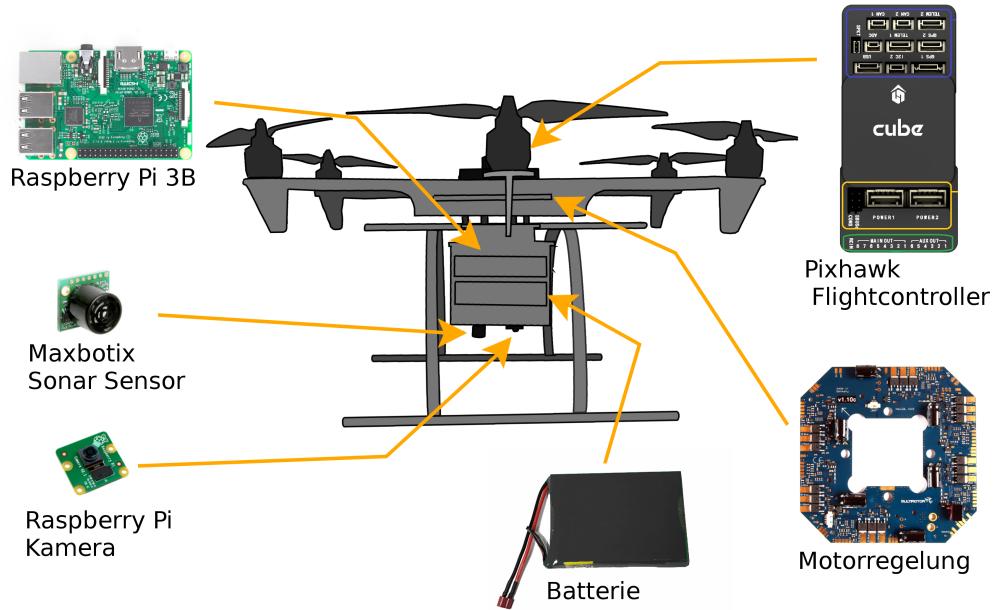


Abbildung 4.2: SUAV schematischer Aufbau

Ein weiterer Bestandteil dieser Arbeit ist der Aufbau einer geeigneten Hardwareplattform. Hierfür wird ein eigener Hexacopter aufgebaut, wobei bei der Spezifikation nicht nur auf die Anforderungen dieser Arbeit, sondern auch zukünftiger Lehrstuhltätigkeiten geachtet wird. Der genaue Aufbau der SUAV, sowie die zugehörige, eigens geschriebene Bedienungsanleitung ist im Kapitel 8.4 zu finden.

Relevant und Gegenstand der Arbeit ist vor allem die Sensorspezifikation, sowie die Recheneigenschaften des flugbegleitenden Raspberry Pi. In dieser Arbeit wird angenommen, dass Orientierung, Beschleunigung und Winkelgeschwindigkeit bekannt sind. Diese Größen werden über das *RobotOperating-System* (ROS) aus dem Flightcontroller ausgelesen. Bei diesem handelt es sich, wie im Anhang 8.4 beschrieben, um einen Pixhawk 2.1 Cube, welcher aufgrund seiner ausgezeichneten Sensorik ausgewählt wird [21]. Eine Rolle spielte auch, dass die zugrundeliegende Software Open Source ist und sich deshalb eine große *community* gebildet hat. Dies macht es möglich, softwareseitige Probleme schnell und effizient zu lösen.

Für die vom Pixhawk gelieferten Werte ist zu beachten, dass sie mit dem internen *Extendet Kalman Filter* (EKF) geschätzt werden [1]. Es wäre hier nicht korrekt, die Messgenauigkeit des Sensors als Fehler des Systems anzugeben, sondern die vom Kalmanfilter gelieferte Ungenauigkeit der Schätzung (auch *believe* genannt).

Die ausgegebenen Fehler scheinen allerdings an einigen Stellen nicht glaubwürdig und sind entweder viel zu groß oder zu klein, weshalb sie im folgenden experimentell überprüft werden.

4.1.1 Orientierungs- und Winkelgeschwindigkeitsfehler

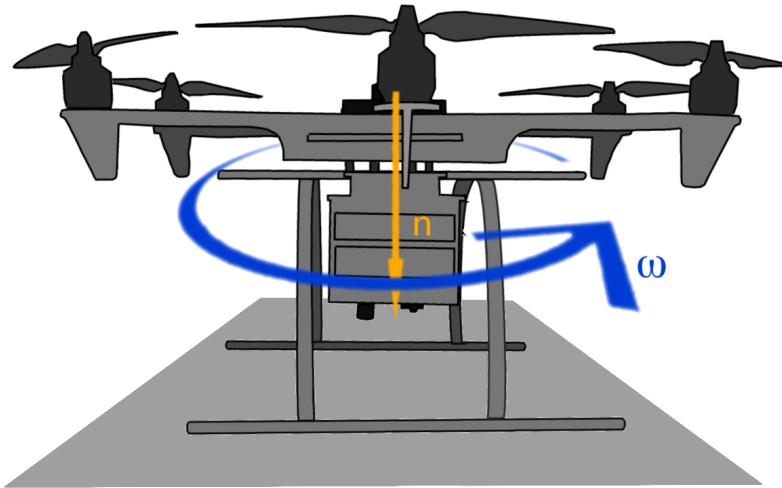


Abbildung 4.3: Aufbau: Messung des Normalvektor- und Winkelgeschwindigkeitsfehlers

Zu diesem Zweck wird die Drohne zunächst in eine aufrechte Lage gebracht und es werden Werte im Ruhezustand aufgenommen. Die Winkelgeschwindigkeit ω sollte dann null und der Normalvektor n parallel zur z-Achse sein.

Messgröße	Experimenteller Fehler	Theoretischer Fehler	Einheit
$\ n\ $	0,00065	1	–
ω	0,00071	1,218E-7	$\frac{\text{rad}}{\text{s}}$

Tabelle 4.1: Fehlerbestimmung von Normalvektor und ω

Es sollte erwähnt werden, dass der Fehler der Drehgeschwindigkeit im dynamischen Flug vermutlich noch größer ist. Die Berechnung des Normalvektors erfolgt über die vom Pixhawk ausgegebenen Quaternionen $[q_w, q_x, q_y, q_z]$

[14]. Aus diesen wird die Rotationsmatrix R berechnet

$$R := \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_w q_y + q_x q_z) \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(yz - wx) \\ 2(q_x q_z - q_w q_y) & 2(q_w q_x + q_y q_z) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}, \quad (4.1)$$

welche dann mit dem Normalvektor e_z multipliziert wird, um n zu erhalten:

$$n = Re_z \quad (4.2)$$

Die ausgegebenen Quaternionen sind allerdings nicht normiert. Folglich ist die daraus berechnete Drehmatrix R nicht unitär. In der weiteren Berechnung muss der Normalvektor deshalb normiert werden, was in den oben beschriebenen Fehler bereits einfließt.

Als Nächstes sollte auf die optischen Daten eingegangen werden. Zur Aufnahme wird das *Raspberry Pi Camera Module 2* eingesetzt, da dieses besonders kompatibel zum Pi ist und keinen Autofokus besitzt, welcher die Modellbeschreibung verkomplizieren würde. Die Kamera wird dabei möglichst genau parallel zu der IMU angebracht, um die in Kapitel 3.3 getroffene Annahme einer reinen Translation zu erfüllen. Besagte Translation wird zunächst mithilfe eines Messschiebers auf $((2, 0 \pm 0, 5)\text{E-}2; (0 \pm 5)\text{E-}3; (2, 05 \pm 0, 05)\text{E-}1)^T m$ bestimmt. In Kapitel 4.2 ist zusätzlich eine Methode detailliert dargestellt, welche den Abstand präziser bestimmen sollte.

Die hier verwendete Implementierung des Lucas Kanade Algorithmus aus Kapitel 4.4 bestimmt ebenfalls einen eigenen Fehler. Dieser berücksichtigt die Ungenauigkeiten der Kamera, sowie etwaige Linseneffekte, allerdings nicht.

4.1.2 Optischer Fehler

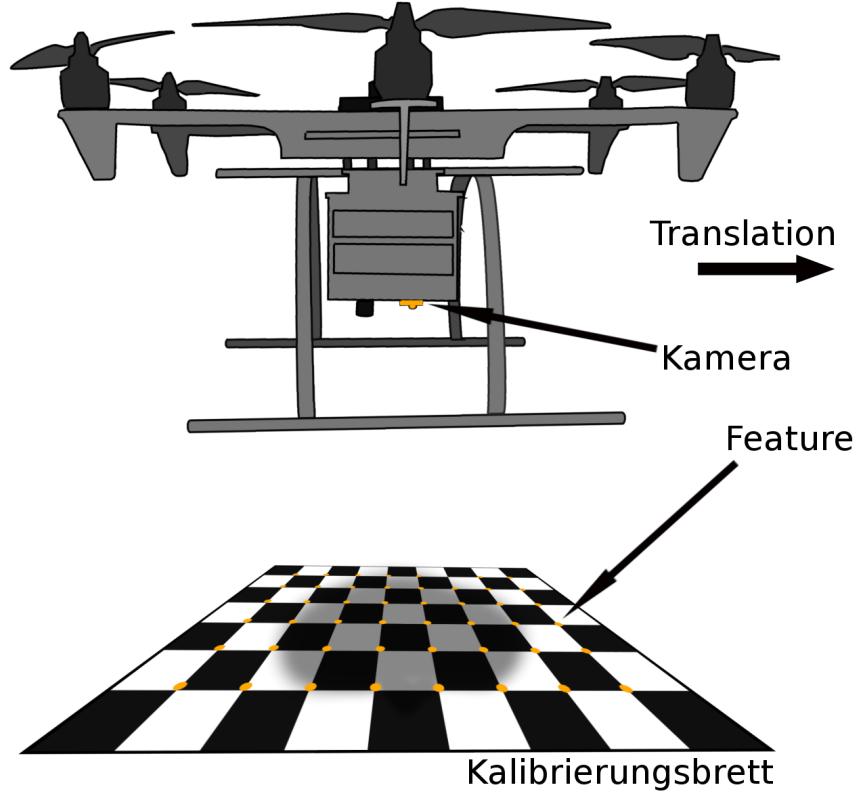


Abbildung 4.4: Aufbau: Messung des Optischen Fehlers

Um die Ungenauigkeit der Kamera, sowie des Algorithmus selbst zu überprüfen, wird ein Schachbrett verwendet, dessen Features (also Ecken) nun verfolgt werden. Wird dieses nun parallel zur Bildebene verschoben, sollten alle Verschiebungen der einzelnen Features gleich groß sein. Aus der Abweichung der einzelnen Verschiebungen lässt sich so ein mittlerer Fehler generieren. Die experimentellen Ergebnisse sind in der folgenden Tabelle zusammengefasst:

Messgröße	Experimenteller Fehler	Theoretischer Fehler	Einheit
Feature Position	0,056	1,3	px

Tabelle 4.2: Fehler in der Feature Position

Dies ist allerdings eine Vereinfachung, da Linsenfehler beispielsweise am Rand des Bildes stärker sind, als in der Mitte.

4.1.3 Umrechnung von Pixeln auf Meter

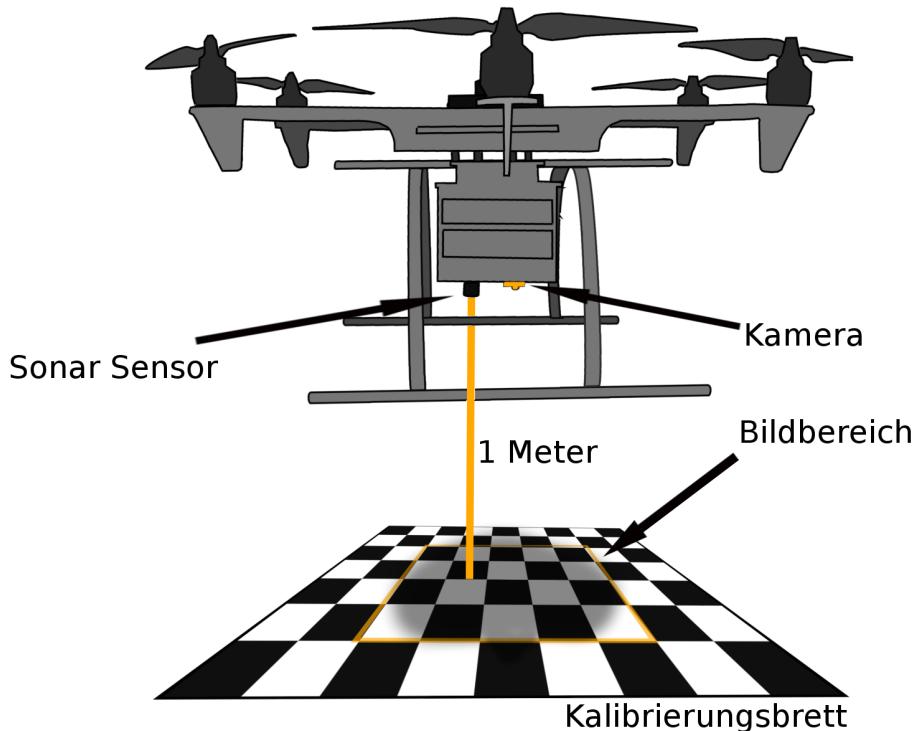


Abbildung 4.5: Aufbau: Messung zur Umrechnung von Pixeln auf Meter

Bis jetzt erfolgte die Distanzmessung in Pixeln. In der Theorie erfolgt die Umrechnung aber mit einem in Meter gemessenen Abstand d . Damit diese

Umrechnung korrekt ist, muss die Bildebene so skaliert sein, dass jede Koordinate in der Bildebene der eines Weltpunkts bei einem Meter Entfernung entspricht. Dabei fließt hier sowohl die Größe der einzelnen Pixel, als auch die Fokuslänge der Kamera ein. Ziel ist es also, die richtige Skalierung zu finden.

Hierzu wird die Kamera einen Meter vor einer Ebene angebracht und die Ecken des Bildbereiches vermessen. Es ergibt sich hier eine Breite von $(1,27 \pm 0,01)\text{cm}$ und eine Höhe von $(0,93 \pm 0,01)\text{cm}$. Dadurch lässt sich der Fehler für jedes Pixel nun mit gaußscher Fehlerfortpflanzung berechnen. Unter Verwendung des konservativeren theoretischen Fehlers gilt:

$$\Delta\text{feature} = 3,558\text{E-}6 \text{ m.}$$

4.1.4 Fehler der Höhenmessung

Als Höhenmesser wird ein Maxbotix Sonar Sensor verwendet, welcher speziell für den Indoor Einsatz gedacht ist und eine effektive Reichweite von 0,2-7,65 Meter besitzt. Dies ist eine sinnvolle Reichweite, da der Algorithmus bei geringeren Höhen präziser ist. Der Sonarsensor wird vor allem aus praktikablen Gründen gewählt, da am Lehrstuhl bereits mehrere Sonarsensoren zur Kollisionsvermeidung verwendet werden. Hier gilt zu beachten, dass die erhaltene Höhe nicht der benötigten Höhe d über dem Boden entspricht.

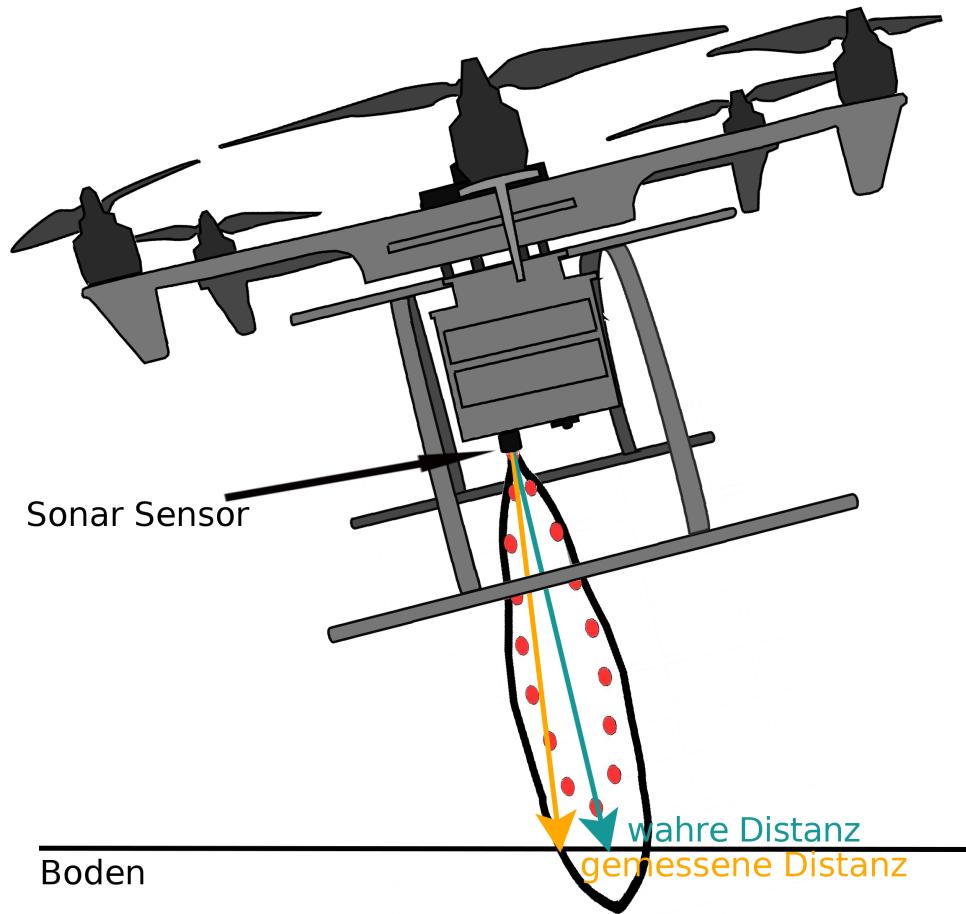


Abbildung 4.6: Höhenmessung mit Ultraschallkeule

Wie in Abbildung 4.6 dargestellt ist, misst der Sonarsensor die kürzeste Distanz zum Boden, die mathematische Annahme einer zu e_z parallelen Messung ist folglich verletzt. Um diese Effekte zu minimieren, wurde der Maxbotix Sensor mit der schmalsten Keule verwendet (Spezifikationen im Anhang 8.4). Im folgenden Modell wird ein idealisierter Distanzsensor angenommen, der die real auftreten Sonareffekte ignoriert (um diese zu verarbeiten, müssten Filter verschiedener Art verwendet werden [16]). Zuerst muss die tatsächliche Höhe d_{dist} im Höhensensorkoordinatensystem aus der gemessenen Höhe d_{meas}

berechnet werden. Hierbei gilt es, die Schräglage des Sensors auszugleichen. Dabei folgt:

$$d_{dist} = \cos(\alpha)d_{meas} = d_{meas}e_3^T n. \quad (4.3)$$

n ist dabei der Normalvektor der betrachteten Ebene. Die Translation T_2 zwischen Höhensensor und Kamera führt ebenfalls zu einem Höhenunterschied, welcher für die Umrechnung in das Koordinatensystem der Kamera ausgeglichen werden muss:

$$d = d_{dist} - \Delta d = d_{dist} - T_2^T n = d_{meas}(e_3 - T_2)^T n. \quad (4.4)$$

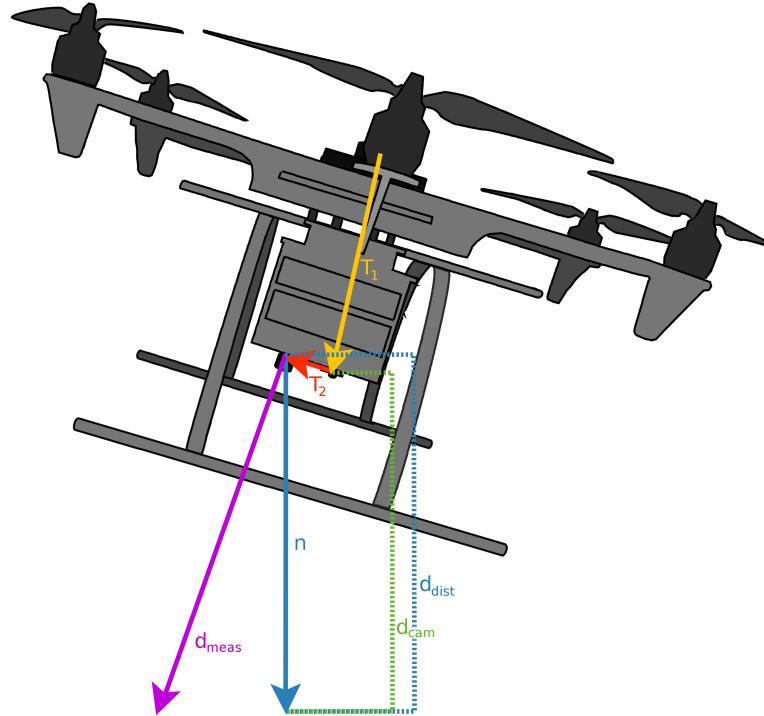


Abbildung 4.7: Höhenbestimmung

Der Fehler der Höhe setzt sich aus drei Komponenten zusammen: dem

Messgröße	Δd_{meas}	Δd	Einheit
Höhe	0,01	0,2	m

Tabelle 4.3: Fehler in der Distanzmessung

Fehler der Orientierung, dem Fehler der Translation T_2 und dem Fehler der Distanzmessung selbst. Nach gaußscher Fehlerfortpflanzung gilt:

$$\Delta d = \sqrt{(\|T_2\|^2 + d_{\text{meas}}^2)\Delta n^2 + \Delta T_2^2 + (\Delta d_{\text{meas}}n_z)^2}. \quad (4.5)$$

Dem Datenblatt des Sensors kann eine Auflösung von 1 cm entnommen werden. Hierbei muss beachtet werden, dass diese nicht der Genauigkeit des Sensors entspricht. Der dominierende Fehler des Sensors resultiert, wie in Abbildung 4.6 zu sehen ist, aus der Form der Ultraschallkeule. Er wird hier konservativ angesetzt, weshalb die maximale Keulenöffnung bei 7,65 Metern verwendet wird, um die Abweichung mithilfe des Satzes von Pythagoras zu bestimmen. Es sollte erwähnt werden, dass durch einen schrägeren Schnitt des Ultraschallkegels mit dem Boden ein größerer Fehler entstehen kann, da n_z in diesem Fall entsprechend kleiner ist, wurde dieser Einfluss allerdings ignoriert. Der resultierende maximale Fehler ist in der Tabelle 4.3 abzulesen.

4.2 Kalibrierung der Kamera

Für den hier vorgestellten Ansatz ist eine genaue Kenntnis der Translation T_1 zwischen Kamera und IMU notwendig. Diese kann nur näherungsweise gemessen werden, da hier der Abstand der virtuellen Nullpunkte beider Systeme gemeint ist. Im Folgenden ist ein möglicher Kalibrierungsalgorithmus vorgestellt, welcher jedoch nicht mehr implementiert und getestet wird. Wie in Abbildung 4.8 zu sehen ist führt die Translation der Kamera der IMU gegenüber zu einer zusätzlichen von $\hat{\omega}$ abhängigen Geschwindigkeitskompo-

nente.

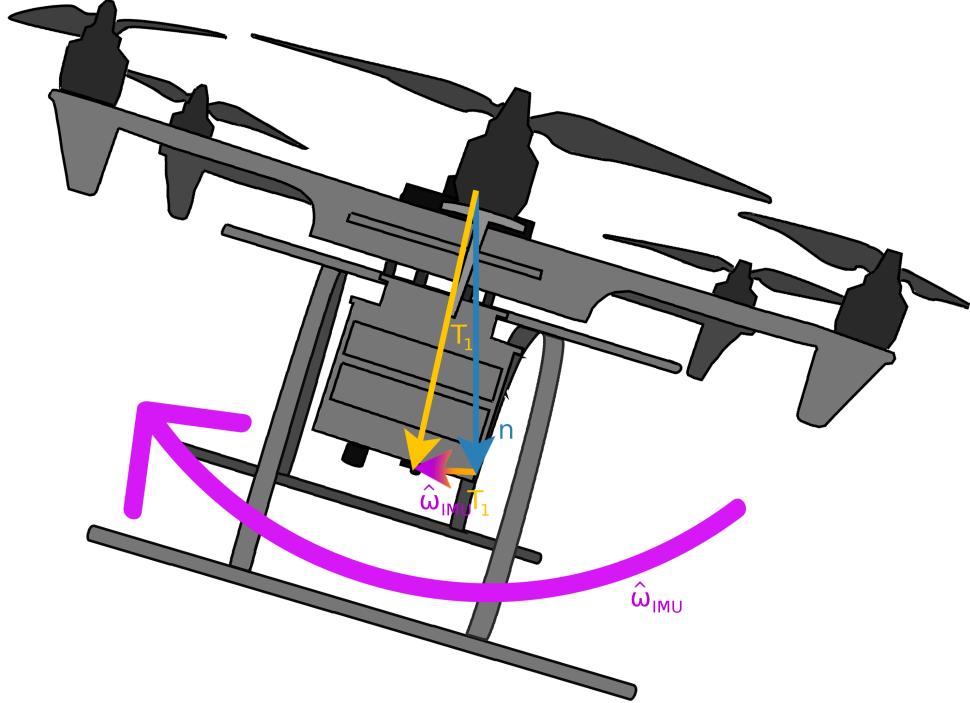


Abbildung 4.8: Translative Geschwindigkeitskomponente

Die Bewegung von Punkten im Raum ergibt sich nach (3.10):

$$\dot{X}_{\text{Cam}} = \hat{\omega}_{\text{Cam}} X_{\text{Cam}} + v_{\text{Cam}}. \quad (4.6)$$

Da hier häufig zwischen Bezugssystemen gewechselt wird, wird das kamerakoinzidente Bezugssystem mit Cam bezeichnet und das flightcontrollerbeziehungsweise IMU-koinzidente mit IMU. Die Winkelgeschwindigkeit ist dabei für die Kamera und IMU gleich. Setzt man für v_{Cam} die Geschwindigkeit der IMU ein, so erhält man:

$$\dot{X}_{\text{Cam}} = \hat{\omega}_{\text{IMU}} X_{\text{Cam}} + v_{\text{IMU}} + \hat{\omega}_{\text{IMU}} T_1. \quad (4.7)$$

Unter Zuhilfenahme des Pseudoinversen lässt sich die Formel nach T_1 umstellen:

$$T_1 = \hat{\omega}_{\text{IMU}}^+ \left(\dot{X}_{\text{Cam}} - \hat{\omega}_{\text{IMU}} X_{\text{Cam}} - v_{\text{IMU}} \right). \quad (4.8)$$

$\hat{\omega}_{\text{IMU}}$ wird dabei direkt von der IMU bestimmt, X_{Cam} und \dot{X}_{Cam} erhält man über Arucomarker [19] [10] aus dem Bild. Ein Arucomarker ist dabei ein schwarzes Quadrat bekannter Abmessung, welches über innere weiße Quadrate eindeutig identifiziert werden kann (Siehe Abbildung 4.9).

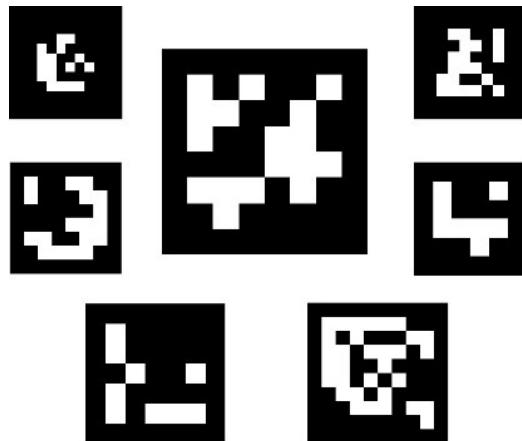


Abbildung 4.9: Beispiele für Arucomarker

Für die Bestimmung von v_{IMU} gibt es nun 2 Möglichkeiten:

Will man sein externes Equipment auf ein Minimum reduzieren, misst man die Beschleunigung und bestimmt daraus die Geschwindigkeit, wobei hier der IMU Drift zu Problemen führen wird und die Genauigkeit verringert. Genauer wäre es, die Geschwindigkeit der IMU von außen zu messen, wozu sich beispielsweise das am Lehrstuhl bereits vorhandene Krypton System anbietet.

Für eine gute Kalibrierung wird wie folgt vorgegangen:

1. Die SUAV wird möglichst ruhig über dem Marker rotiert, sodass dieser

die ganze Zeit im Bild ist.

2. Speichere $X_{\text{Cam}_i}, \dot{X}_{\text{Cam}_i}, \hat{\omega}_{\text{IMU}_i}, v_{\text{IMU}_i}$.
3. Bilde LGS $AT = B$ mit $A = [\hat{\omega}_{\text{IMU}_i}]^T$,
 $B = [\dot{X}_{\text{Cam}_i} - \hat{\omega}_{\text{IMU}_i} X_{\text{Cam}_i} - v_{\text{IMU}_i}]^T$.
4. Löse LGS für *Least-Squares* Lösung von T .

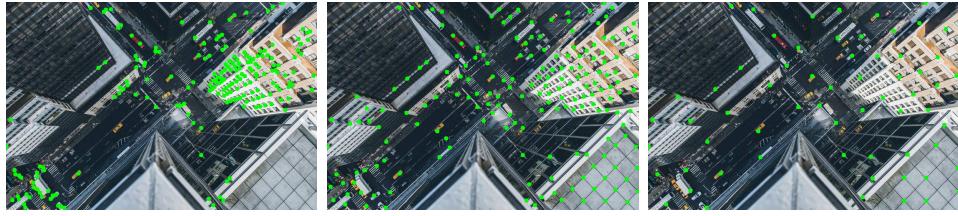
Für die folgenden Kapitel wurde die Translation, wie in Kapitel 4.1.1 erwähnt, mit einem Zollstock vermessen. Hier ergab sich:

$$T_1 = \begin{pmatrix} 0,02 \pm 0,005 \\ 0,0 \pm 0,005 \\ 0,205 \pm 0,005 \end{pmatrix} m.$$

4.3 Feature Detektion

Bevor Features verfolgt werden können, müssen sie zunächst generiert werden. Hierzu wird aus den in Kapitel 2 dargelegten Gründen *Shi-Tomasi-Corner Detection* verwendet. Implementiert wird der Algorithmus mithilfe der OpenCv Bibliothek unter dem Befehl `cv2.goodFeaturesToTrack()`. Der Funktion können mehrere Parameter übergeben werden:

1. Die maximale Anzahl der zu findenden Ecken
2. Die minimale Distanz zwischen den Ecken
3. Das minimal geforderte Shi-Tomasi Auswahlkriterium R (siehe (3.4))
4. Eine Maske, welche das zu untersuchende Gebiet einschränkt
5. Die Blockgröße, für welche die partiellen Ableitungen berechnet werden (entspricht w in (3.1))



(a) $\text{minDistance}=7$ (b) $\text{minDistance}=20$ (c) $\text{minDistance}=50$

Abbildung 4.10: Feature Auswahl in Abhängigkeit der minimalen Distanz

Da, wie im Kapitel 3.4 gezeigt, die Kardinalität der Statistik eine wichtige Rolle spielt, ist es wichtig, eine möglichst große Anzahl an Features zu betrachten. Die maximale Anzahl wird dabei durch die Rechenleistung des Raspberry Pi beschränkt und wird hier auf 30 Features gesetzt.

Die minimale Distanz wird natürlich auch von der Featureanzahl beeinflusst. Ist sie zu klein, kommt es zu starker Haufenbildung und viele Ebenen werden nicht wahrgenommen (siehe Abbildung 4.10 (a)). Ist sie wiederum zu groß, so können nicht genug Features im Bild platziert werden (siehe Abbildung 4.10 (c)).

Am Anfang der Stabilisierung ist es zunächst wichtig, alle Ebenen zu identifizieren. Die minimale Distanz sollte folglich so gewählt werden, dass die 30 Features gleichmäßig verteilt sind. Es gilt:

$$dist_{\max} = \sqrt{\frac{\text{Bildgröße}}{30}} = \sqrt{\frac{600 * 400}{30}} \approx 90px. \quad (4.9)$$

Die ursprüngliche Distanz sollte sich verkleinern sobald geeignete Ebenen identifiziert wurden. Hierzu könnte man als neuen Suchbereich beispielsweise die konvexe Hülle der in der Ebene befindlichen Punkte betrachten. Da der Raspberry Pi mit 30 Punkten allerdings nur zwischen maximal zwei Ebenen unterscheiden kann, wird hier die Minimaldistanz der Einfachheit halber di-

rekt kleiner (auf 10 Pixel) angesetzt.

Um zu verhindern, dass ein einzelnes Feature mehrfach verfolgt wird, wird mit Hilfe des Maskenparameters ein Kreis mit Radius der Minimaldistanz um jedes bereits verfolgte Feature gelegt, sodass diese Regionen ausgeschlossen werden. Um die Blockgröße, sowie R , endgültig zu setzen, werden allerdings experimentelle Daten benötigt. Das *tunen* des gesamten Systems könnte Gegenstand einer Folgearbeit sein (siehe Kapitel 7). Zur Illustration ist hier nochmal die bereits in Abbildung 3.4 dargestellte Hochhäuserschlucht zu sehen. Aufgrund der höheren Auflösung des Bildes im Vergleich zur Raspberry Pi Kamera wird hier eine Minimaldistanz von 20 Pixeln als Optimum angesetzt. Die Einfärbung der einzelnen Punkte entspricht der Färbung in Abbildung 3.4 und soll verdeutlichen, wie viele Punkte pro Ebene gefunden werden. Wie man im Vergleich sieht, konnten für die obere gelbe Ebene keine Punkte gefunden werden, für das rechte Hochhaus wiederum wurde eine Vielzahl von Punkten gefunden. Aufgrund der in urbanen Gebieten häufig auftretenden Fensterstrukturen ist dies häufig nicht vermeidbar, was wiederum die Wichtigkeit eines Sortierverfahrens illustriert.

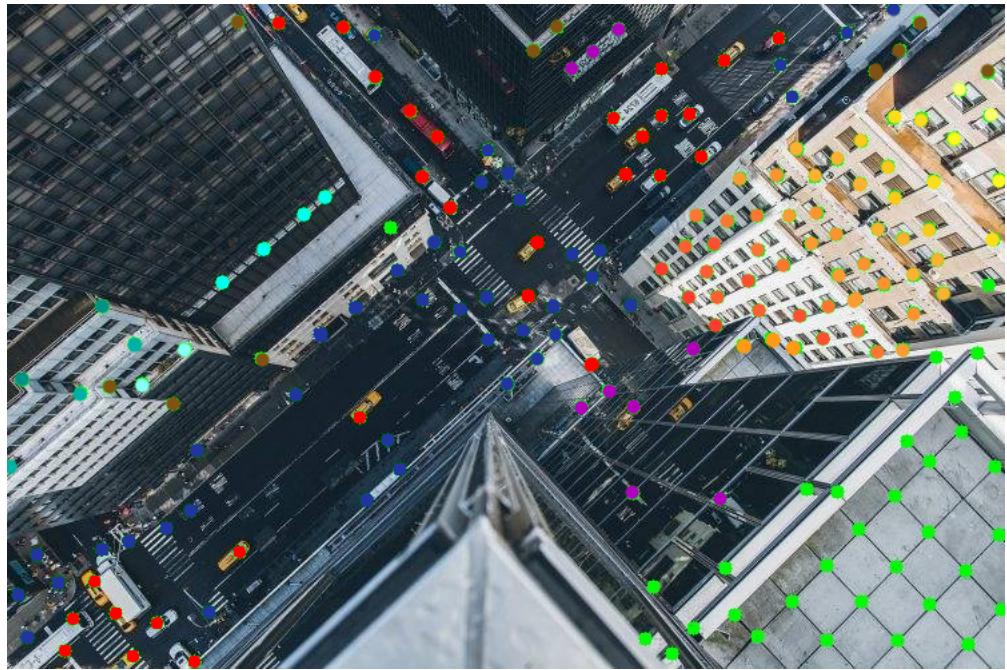


Abbildung 4.11: Hochhäuserschlucht mit *Shi-Tomasi Corner detection*

4.4 Feature Tracking

Für das Verfolgen der Features wird die OpenCv Implementierung des Lucas-Kanade Algorithmus `calcOpticalFlowPyrLK()` verwendet. Dieser hat die folgenden Parameter:

1. Fenstergröße
2. Level

Für die Levelanzahl haben sich drei Level als sinnvoll erwiesen. Für die Fenstergröße sollten weitere Messreihen angestellt werden.

4.5 Raspberry Pi Companion Computer

Die Berechnungen des im Kapitel 2 vorgestellten Algorithmus werden von einem Raspberry Pi 3B vorgenommen. Das Problem, welches es hier zu lösen gilt, ist die Synchronisation der verschiedenen Signale. Die IMU des Pixhawk liefert, verglichen mit der Kamera, Werte mit einer höheren und asynchronen Rate . Das *Robot Operating System* (ROS) stellt nun eine Reihe von Werkzeugen zur Verfügung, um dieses Problem zu lösen. Einzelne Teile eines Programmes sind dabei in sogenannten Nodes verkapselt, welche als eigene Prozesse arbeiten.

Die Geschwindigkeit des Systems wird fortlaufend von der IMU aktualisiert, bis das System ein Bildsignal erhält. Daraufhin wird die momentane Geschwindigkeit zwischengespeichert, sowie die Orientierung und Winkelgeschwindigkeit des Systems. Gleichzeitig wird die Position der optischen Features verfolgt. Sobald alle Werte vorhanden sind, wird die Stützgeschwindigkeit berechnet. Da dieser Prozess selbst Zeit kostet, ergibt sich die neue Geschwindigkeit aus Summe der Stützgeschwindigkeit und der zwischenzeitlich akkumulierten Geschwindigkeitsänderung.

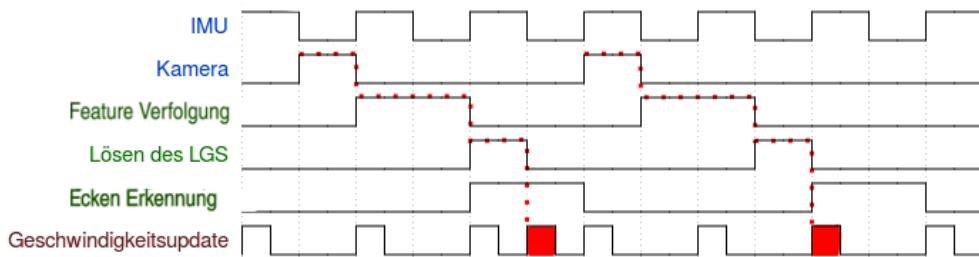


Abbildung 4.12: Schematisches Zeitverhalten der Berechnung

In Abbildung 4.12 ist das Zeitverhalten schematisch dargestellt. Die Frequenz der IMU ist dabei natürlich in Wahrheit deutlich schneller und wird hier nur aus Übersichtlichkeitsgründen langsamer dargestellt. Dies ermöglicht

es zu verdeutlichen, dass die von der äußereren Stützung generierten Geschwindigkeiten (hier rot dargestellt) asynchron zur IMU laufen. Der rot gestrichelte Weg ist dabei die Aufsummierung der einzelnen Zeitkomponenten. Auch ist zu erkennen, dass die Suche neuer Features parallel zur Geschwindigkeitsberechnung stattfinden kann, wodurch sich die Maximalfrequenz ein wenig erhöhen lässt.

Im Code wird das hier beschriebene Zeitverhalten über sogenannte *flags* realisiert. Hierbei handelt es sich um boolsche Werte, die den Zustand des Systems anzeigen. Sie sind nötig, um die asynchronen Werte zur richtigen Zeit auszulesen, denn die Node der Kamera liefert keine Bilder, sondern direkt Bildkoordinaten und Translationen. Diese kommen erst einige Zeit nach der Aufnahme des Bildes an, zu welchem Zeitpunkt bereits Winkelgeschwindigkeit und Orientierung zwischengespeichert werden sollten.

Wird also ein Bild aufgenommen, wird die *flag got_picture_* auf `False` gesetzt. Nun werden die nächsten Winkelgeschwindigkeiten sowie Normalvektoren zwischengespeichert, während die Akkumulierte Beschleunigung zur späteren Addition zwischengespeichert wird. Sind die Positionen und Translationen der Features gesendet worden, wird die *flag* auf `True` gesetzt. Erst dann kann mit der Berechnung der Stützgeschwindigkeit begonnen werden, wodurch die *flag got_vel_* auf `False` gesetzt wird.

Sobald die Berechnung abgeschlossen wurde, wird die *flag* wieder auf `True` gesetzt und die neue Systemgeschwindigkeit auf die Summe der berechneten sowie der akkumulierten Geschwindigkeit gesetzt. Sollte die Berechnung länger dauern, verfolgt die Kamera die Position der Features weiter mit, aber erst sobald *got_vel_* auf `True` ist, kann die *got_picture_ flag* gesetzt werden, wodurch der Prozess von neuem beginnt. Zusammengefasst findet sich dieses Verhalten in Abbildung 4.13

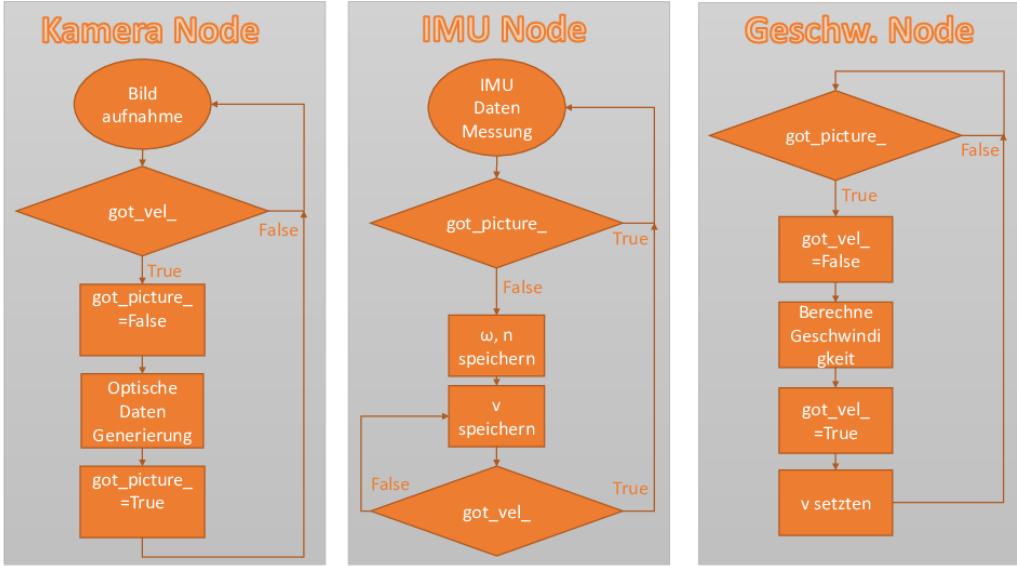


Abbildung 4.13: Flussdiagramm für die Geschwindigkeitsberechnung

Es stellt sich die Frage, wie das zeitliche Verhalten und die maximal mögliche Frequenz des Algorithmus ist. Hier sind ernüchternde Ergebnisse zu verzeichnen. Obwohl die Berechnung des Algorithmus schnell geschieht, dominiert der Zeitaufwand des Verfolgens und Finden der Features. Erstere dauert dabei etwa $(1,3 \pm 0,4)\text{E-}1\text{s}$ und zweiteres sogar ungefähr $(4,4 \pm 0,5)\text{E-}1\text{s}$.

In Frequenzen ausgedrückt bedeutet dies, dass die Stützfrequenz, selbst wenn keine neuen Features gefunden werden müssen, rund $(7 \pm 2)\text{Hz}$ beträgt. Noch schlimmer wird es, wenn die SUAV sich so schnell bewegt, dass die Features jeweils nur zwei Frames im Bild bleiben. In diesem Fall müssen bei einer Gleichverteilung an Features in jedem Frame neue Features gesucht werden. Dadurch fällt die Frequenz auf etwa $(2,3 \pm 0,3)\text{Hz}$.

Aus dieser Grenzfrequenz lässt sich die höchste feststellbare Geschwindigkeit in Abhängigkeit zur Höhe zu bestimmen. Hierbei handelt es sich um die

Geschwindigkeit, bei der ein Teilchen während einer Featureaufnahme das gesamte Bild durchquert. Die maximale Geschwindigkeit ergibt sich also aus der Größe des Bildbereichs in Metern und der maximalen Frequenz. Setzt man die größere Breite des Bildes mit $(1,27 \pm 0,01)d$ an, so gilt:

$$v_{\max} = (2,9 \pm 0,43)\text{Hz} \cdot d \quad (4.10)$$

Nimmt man eine SUAV wie die Phantom 4 Pro als Beispiel, welche $50 \frac{\text{km}}{\text{h}} = 13,8889 \frac{\text{m}}{\text{s}}$ fliegen kann, so wäre diese erst ab einer Flughöhe von $(4,8 \pm 0,7)\text{m}$ im Stande, ihre Geschwindigkeit zu bestimmen.

Dabei wird nicht berücksichtigt, ob der Lucas Kanade Algorithmus im Stande wäre, die Punkte über eine solch große Distanz zu verfolgen. Hier zeigt sich in der Praxis, dass das Verfolgen fehlschlägt, sobald die ersten neuen Features gefunden werden müssen. Der Raspberry Pi erweist sich daher als zu schwach und die Analyse gesammelter Messdaten muss im Nachhinein geschehen. Generell sollte aber beachtet werden, dass eine beliebig hohe Bildfrequenz nicht wünschenswert ist. Es zeigt sich, dass der Verfolgungsfehler für sehr kleine Unterschiede sehr groß wird. Das liegt daran, dass es bei Translationen im Subpixelbereich zu Rundungsfehlern kommt, wie schon in [12] festgestellt wurde.

Kapitel 5

Ergebnisse

Zur korrekten Interpretation realer Messwerte wird in diesem Kapitel eine Simulation durchgeführt, welche das Verfahren unter idealen Bedingungen beschreibt. Damit ist es möglich den Fehlereinfluss nicht beachteter Größen (z.B. die Vibration der SUAV) in einer tatsächlichen Messung zu isolieren.

5.1 Simulation

Die in der Simulation betrachteten Features werden zufällig gleichverteilt in der Bildebene generiert und zur besseren Vergleichbarkeit einzelner Messungen in einer Datei abgespeichert. Der Simulation werden eine lineare Geschwindigkeit, eine Drehgeschwindigkeit, eine Orientierung (in Form eines Normalvektors), eine Höhe sowie eine Translation zwischen Kamera und IMU vorgegeben. Unter Verwendung von

$$\dot{x}_i = (v - e_z \cdot vx_i)n^T \frac{x_i}{d} + \hat{\omega}x_i - e_z \cdot (\hat{\omega}x_i)x_i \quad (5.1)$$

(für Herleitung siehe (8.4)) lässt sich \dot{x}_i bestimmen. Auf diese korrekten Größen werden normalverteilte Fehler addiert, deren Standardabweichung gerade der in Kapitel 4.1 bestimmten Messungenauigkeit entspricht. Aus

diesen fehlerbehafteten Größen wird jeweils eine Lösung des LGS aus (3.16) berechnet. Iterationen dieses Verfahrens ermöglichen es den Mittelwert der bestimmten Werte zu finden, sowie eine statistisch relevante Standardabweichung zu bestimmen.

Die hohe Anzahl an Parametern führt nun zu einem Problem, denn es ist nicht klar, welche Kombination von Parametern zu interessanten Ergebnissen führen. Nur der Normalvektor ist nicht vollkommen willkürlich, da eine zu große Abweichung von der optischen Achse dazu führt, dass die Ebenen von der Kamera nicht mehr wahrgenommen werden. Außerdem kann man vereinfachend annehmen, dass sich das System sowohl in Linear- als auch in Dreh-Geschwindigkeit symmetrisch bezüglich der x- und y-Achse verhält. Wird also beispielsweise die Geschwindigkeit in x-Richtung erhöht, so kann angenommen werden, dass die Dynamik nur in x und y vertauscht wäre, wenn stattdessen die Geschwindigkeit in y-Richtung erhöht werden würde.

5.1.1 Bestimmung der Geschwindigkeit

In den folgenden Simulationen wird von einer Bewegung von $v = [1, 1, 1]^T$, $\omega = [1, 1, 1]^T$ ausgegangen, wobei der Normalvektor parallel zur z-Achse steht. So können alle relevanten Effekte diskutiert werden, während man sich immer noch auf die simulativen Daten des vorherigen Teils beziehen kann.

Es stellt sich die Frage, wie viele Features notwendig sind. Obwohl im Folgenden immer mit denselben 200 prägenerierten Features gearbeitet wird, ist man in der Praxis aufgrund der Hardware beschränkt. In den Abbildungen 5.1 wird die berechnete Geschwindigkeit, sowie der zugehörige Fehler, gegen die Anzahl der verwendeten Punkte aufgetragen.

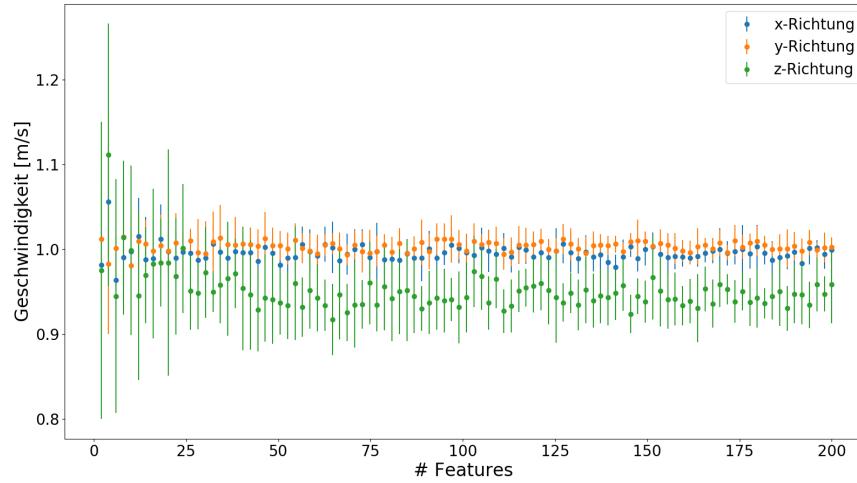


Abbildung 5.1: Lösung in Abhängigkeit von der Featurezahl

Wie man sieht, konvergiert der Fehler des Verfahrens schnell gegen seinen endgültigen Wert und schon 20 Features erzeugen ein zufriedenstellendes Ergebnis. Die Abweichung des Fehlers in z-Richtung lässt sich durch die statistische Fehleranalyse erklären, wobei darauf später genauer eingegangen wird.

Wenn im weiteren davon ausgegangen werden soll, dass die Orientierung immer $[0, 0, 1]^T$ ist, muss vorher betrachtet werden, wie sich das System verhält, wenn die Annahme nicht erfüllt ist. Dies ist besonders wichtig, da die maximale Geschwindigkeit der Drohne im normalen Flugbetrieb auch davon abhängt, wie schräg sie gerade liegt. In Abbildung 5.2 wird die Geschwindigkeit unter einem Winkel α zur Senkrechten, von $0^\circ - 180^\circ$ simuliert, wobei die Drehung in der x-z-Ebene stattfindet.

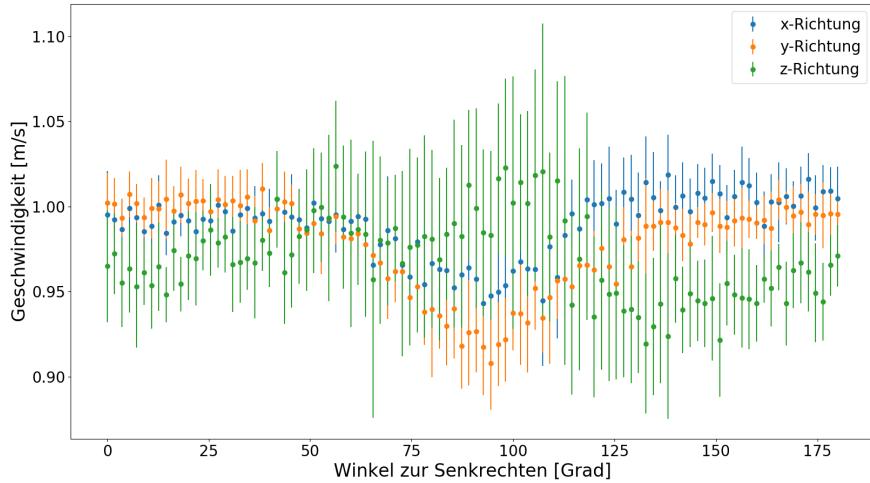


Abbildung 5.2: Simulation der Winkelabhängigkeit

Obwohl die Abweichung um 90° wie erwartet am größten ist, überrascht die Abweichung der x- und y-Fehler voneinander. Dies hat mehrere Gründe. Der unterschiedliche Ausschlag um 90° ist eine Folge der unterschiedlichen Bildgrößen, sowie der Tatsache, dass in der x-z-Ebene rotiert wird. Die leicht unterschiedliche Form kann durch die Translation T_1 zwischen Kamera und IMU erklärt werden, welche nur in x- und z-Richtung ungleich Null ist. Quantitativ lässt sich aber sagen, dass eine Schräglage der Drohne von mehr als 50° zu großen systematischen Fehlern führt. Da die Schräglage einer Drohne im Betrieb durch ihre Geschwindigkeit vorgegeben ist, wäre eine genauere Untersuchung des Geschwindigkeitswinkelverhaltens vital zur Bestimmung tatsächlicher Betriebsfehler.

Als nächstes wird die Höhe betrachtet, da diese die fundamentalen Grenzen unseres Systems aufzeigt. Da der hier verwendete Sensor von 0,2 bis 7,65 Meter ausgelegt ist, fokussieren sich die Untersuchungen auf diesen Bereich. Es wird von derselben Bewegung, wie in der vorherigen Simulation ausge-

gangen. Das Ergebnis der Simulation ist in Abbildung 5.3 zu sehen.

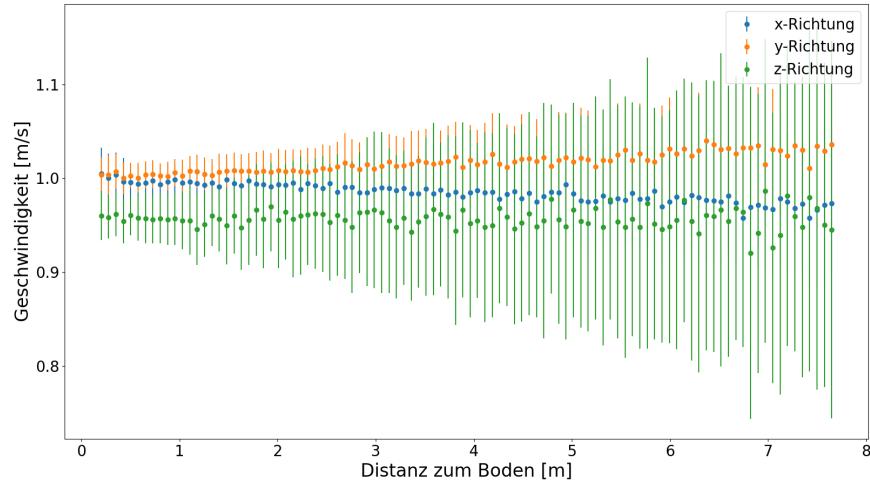


Abbildung 5.3: Simulation der Abhangigkeit der Geschwindigkeit von der Hohe

Sofort fallt hier die systematische Abweichung der Mittelwerte auf. Da es sich hier um nichtlineare Fehler handelt, war dies jedoch zu erwarten. Es findet sich hier exakt der in der Theorie betrachtete Spezialfall, weshalb sich hier nochmal (3.35) heranziehen lsst. Nach besagter Formel wird fur den systematischen Fehler folgende Dynamik erwartet:

$$E(\Delta v) = \left[E\left(\frac{\Delta x_x^2}{\|x + \Delta x\|_2^2}\right) - E\left(\frac{x_x^2 \Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \right] \begin{pmatrix} -d\omega_y \\ d\omega_x \\ 0 \end{pmatrix} - E\left(\frac{\Delta x_x^2}{\|x + \Delta x\|_2^2}\right) \begin{pmatrix} 0 \\ 0 \\ 2v_z \end{pmatrix} \quad (5.2)$$

Dies entspricht einem konstanten systematischen Fehler in z-Richtung, sowie, je nach der Groe der Bildebene, einer positiven oder negativen Steigung der x-Achse, genau spiegelverkehrt zur y-Achse. Das Vorzeichen hangt dabei da-

von ab, ob $E\left(\frac{\Delta x_x^2}{\|x+\Delta x\|_2^2}\right)$ größer als $E\left(\frac{x_x^2 \Delta x_x^2}{\|x+\Delta x\|_2^2}\right)$ ist.

Weitere Untersuchungen zeigen, dass der systematische Fehler in x- und y-Richtung verschwindet, wenn ω_y, ω_x null gesetzt wird. Die systematische Abweichung in z mit v_z wiederum bleibt in diesem Fall bestehen (Siehe Abbildung 5.5). Da eine Vergrößerung der Bildebene (und damit $E(x_x^2)$) zu einer Abflachung und schließlich einem Vorzeichenwechsel des Effekts in x- und y-Richtung führt (siehe Abbildung 5.4), scheint es kurzum so, dass die statistische Herangehensweise die Dynamik des systematischen Fehlers korrekt vorhersagt.

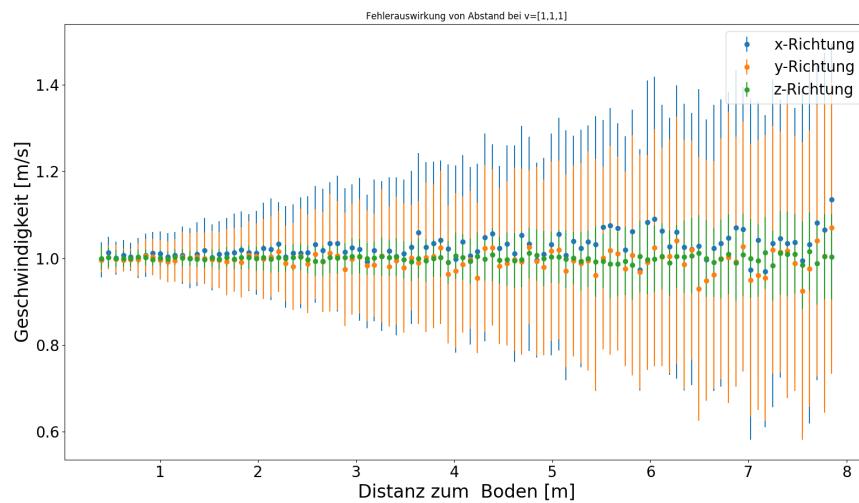


Abbildung 5.4: Simulation der Abhängigkeit der Geschwindigkeit von der Höhe bei zehnfachem Bildbereich

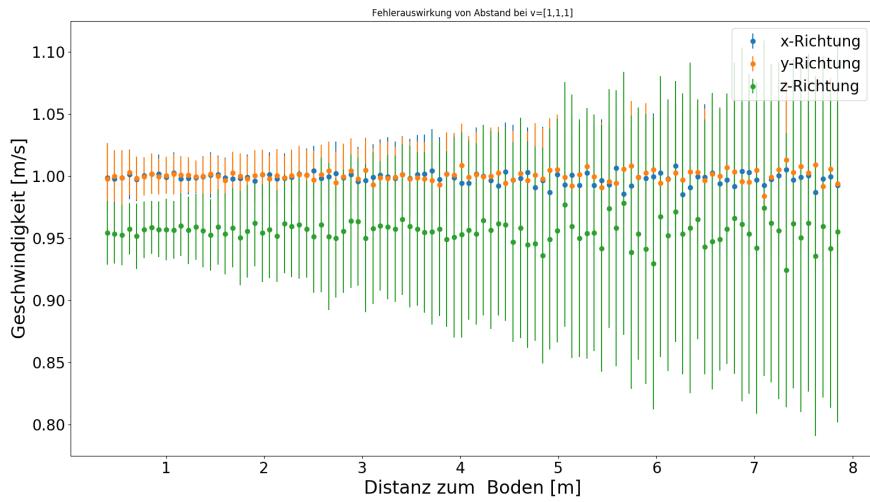


Abbildung 5.5: Simulation der Abhangigkeit der Geschwindigkeit von der Hohe ($\omega_x = \omega_y = 0$)

Unter diesem Gesichtspunkt lassen sich auch weitere Phanomene untersuchen. Wie bereits in der Theorie erwahnt wurde, fuhrt eine ungleiche Verteilung der Bildpunkte zu weiteren systematischen Fehlern. Abbildung 5.6 zeigt eine Simulation, bei der alle Bildpunkte langsam nach oben rechts geschoben werden. Die mittlere Bildposition ist dementsprechend als $\text{mean}(x)$ und $\text{mean}(y)$ zu verstehen.

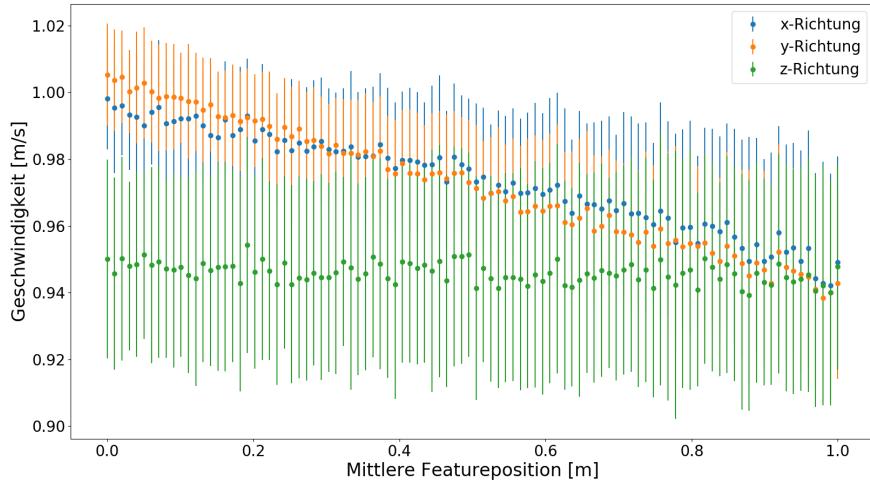


Abbildung 5.6: Simulation der Auswirkung der mittleren Bildverteilung

Hier zeigt sich eine starke, negative Verschiebung des systematischen Fehlers in x- und y-Richtung, während die z-Richtung konstant bleibt. Da der Erwartungswert von Δx nach wie vor null ist, kann diese Verschiebung nur durch einen Term erklärt werden, welcher in Δx quadratisch, aber nur linear in x ist. Diese Terme sind in der Fehlerrechnung zu finden.

Wichtiger ist die Erkenntnis, dass eine ungleiche Verteilung zu einem großen systematischen Fehler führt. Wie in der statistischen Fehlerbetrachtung bereits erwähnt wurde, ist dieser nicht herausrechenbar. Des Weiteren ist leicht einzusehen, dass die Punkteverteilungen bei einem freien Flug außerhalb einer Laborumgebung nicht symmetrisch um das Zentrum sein werden.

Es folgt eine simulative Sensitivitätsanalyse, in welcher der Einfluss der einzelnen Fehler auf das Gesamtergebnis untersucht wird. Dazu wurde jeder Fehler einzeln von 0 bis 0,1 Simuliert, während die gerade nicht betrachteten Fehler, den im Kapitel 4.1 bestimmten entsprechen.

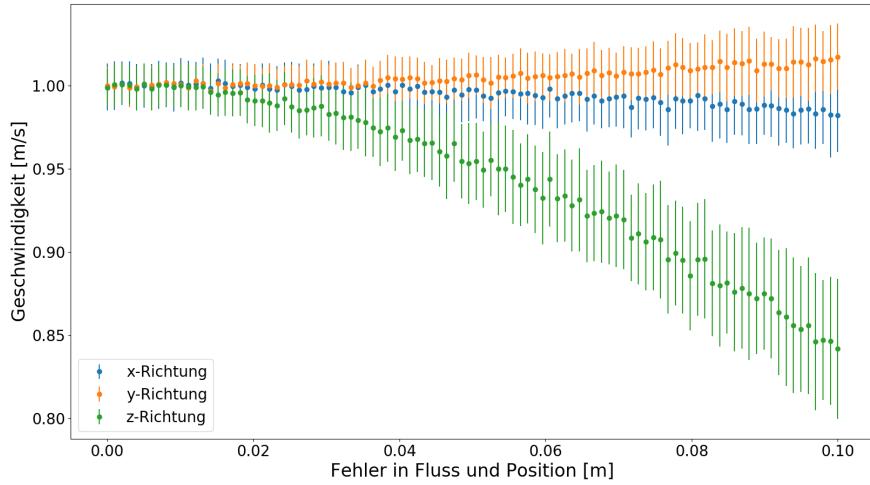


Abbildung 5.7: Auswirkung des optischen Fehlers

In Abbildung 5.7 bestätigt sich zunächst die Erwartung, dass die Nichtlinearität des Positionsfehlers zu systematischen Abweichungen führt. Diese dominieren im Falle der z-Komponente sogar den Gesamtfehler. Der Fehler ist allerdings auch in der x- und y-Komponente sehr groß, mit einer Maximalabweichung von ungefähr $0,03 \frac{m}{s}$.

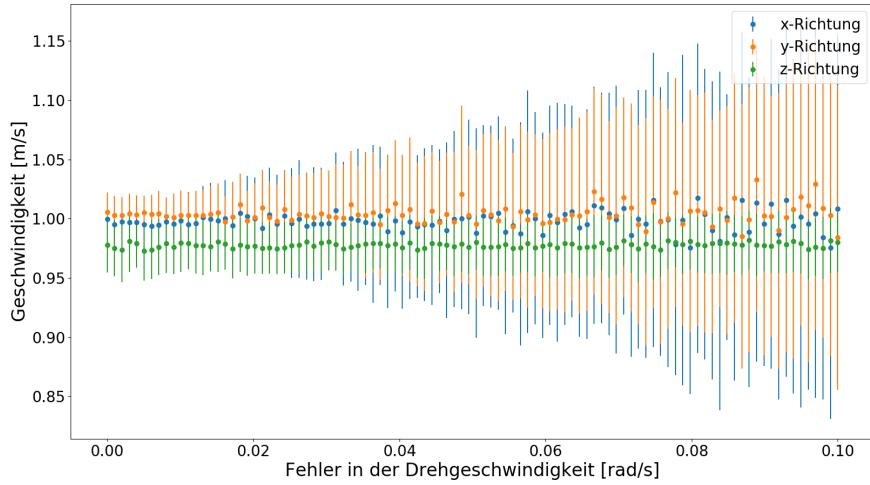


Abbildung 5.8: Auswirkung des Drehgeschwindigkeitsfehlers.

Als besonders fatal für die x- und y-Richtung erweist sich der in Abbildung 5.8 dargestellte Fehler der Drehgeschwindigkeit. Dies war allerdings zu erwarten, da ein Fehler in der Drehgeschwindigkeit jedem Feature eine falsche Komponente in x- und y-Richtung gibt. Da es sich bei diesen Komponenten aber in der Regel um Kreisbewegungen handelt, bleibt die Bewegung in der z-Achse weitgehend unbeeinflusst, denn eine Bewegung in z-Richtung äußert sich in der Bildebene durch eine Bewegung von oder zu einem zentralen Punkt.

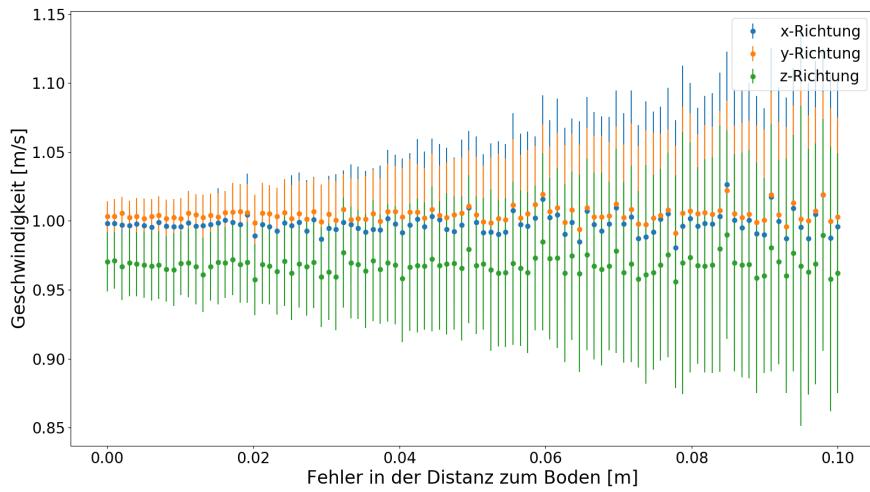


Abbildung 5.9: Auswirkung des Höhenfehlers

Im Gegensatz dazu wirkt sich der Höhenfehler gleich auf alle Komponenten aus, wie in Abbildung 5.9 zu sehen ist. Der Grund dafür ist, dass es sich hier um eine Skalierung handelt. Dementsprechend ist der Einfluss im Verhältnis zur Höhe zu sehen. Je höher die Drohne, desto geringer der Einfluss des Höhenfehlers. Der Fehler bei 1m Höhe kann hier als obere Grenze des so induzierten Fehlers gesehen werden. Es kommt selten vor, dass eine SUAV weit unter einem Meter Höhe betrieben wird und das Verhältnis zwischen Höhenfehler und Höhe ist folglich am größten (von Start und Landung abgesehen).

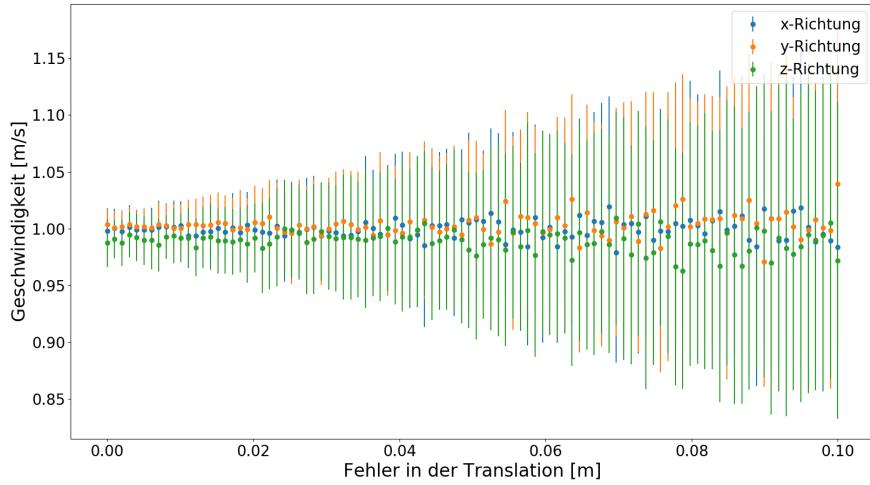


Abbildung 5.10: Auswirkung des Translationsfehlers.

Der Translationsfehler verhält sich ähnlich, wie in Abbildung 5.10 ersichtlich wird. Da er aber nicht nur in z-Achse skaliert, fällt er als Folge der Zusammenwirkung mit der Winkelgeschwindigkeit stärker ins Gewicht. Je höher diese ist, desto höher ist auch der Einfluss des Translationsfehlers. Im Gegensatz zu den bisher betrachteten Fehlern handelt es sich hier aber eigentlich nicht um eine statistische Größe, demnach kann die Translation vorher bestimmt werden und ihr Fehler sollte durch das Kalibrierungsverfahren klein gehalten werden.

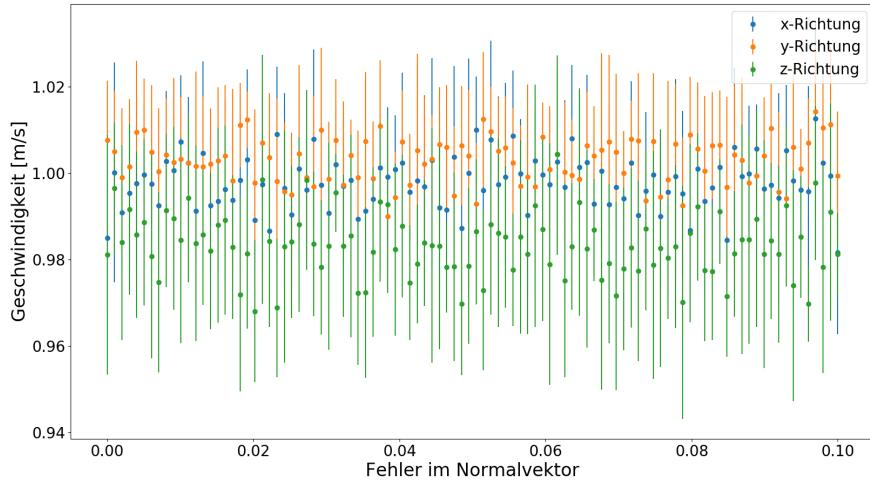


Abbildung 5.11: Auswirkung des Normalvektorenfehlers

Am überraschendsten verhält sich hier die in Abbildung 5.11 dargestellte relative geringe Auswirkung des Normalvektorfehlers. Der Einfluss des Fehlers wird vermutlich durch die Gleichverteilung der Punkte geringgehalten, denn daraus folgt $\text{mean}(\Delta n^T x_i = 0)$ unabhängig von der Korrektheit des Normalvektors. Da der Normalvektor das zu lösende LGS nur in jedem Feature skaliert, könnte das *Least-Squares* Verfahren auch einfach besonders robust gegen zufällige Längenänderungen sein. Insgesamt ist hier aber abzulesen, dass die Verbesserung der optischen Messung, sowie der Drehgeschwindigkeit höchste Priorität haben sollten. Dies ist in der folgenden Tabelle noch einmal zusammengefasst. Std bezeichnet dabei die Standardabweichung und je mehr Pluszeichen ein Punkt hat, desto größer ist der Einfluss des Fehlers.

Fehlerbehaftete Größe	Auswirkung	Einfluss
Translation	steigende Std	++
Optische Messung	starker system. Fehler, steigende Std	++++
Orientierung	keinerlei erkennbare Auswirkung	+
Distanz	schwacher system. Fehler, steigende Std	++
Drehgeschwindigkeit	stark steigender system. Fehler	+++

Tabelle 5.1: Zusammenfassung der einzelnen Fehler mit Priorität

Zuletzt muss überprüft werden, ob die numerische Fehlerformel den Fehler zufriedenstellend bestimmen kann. Hierzu wird ein weiteres Mal die Abhängigkeit der Geschwindigkeit von der Anzahl der verwendeten Features simuliert, mit dem Unterschied, dass der Graph um den numerischen Fehler erweitert wird. Das Ergebnis ist in Abbildung 5.12 dargestellt.

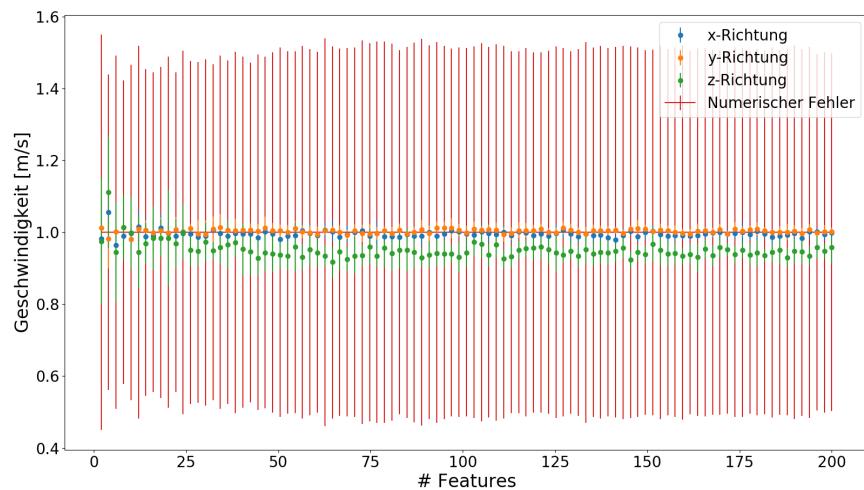


Abbildung 5.12: Numerischer Fehler

Wie man sehen kann, überschätzt der systematische Fehler die simulative Standardabweichung massiv. Die ursprüngliche Vermutung, dass der Fehler mit großer Featureanzahl beliebig groß wird, hat sich wiederum nicht

bestätigt. Ein Grund dafür könnte sein, dass die Norm durch größere Singulärwerte kompensiert wird.

5.1.2 Aussortieren ungeeigneter Features

Die andere zu überprüfende Fähigkeit ist, Features mit korrekter Höhe d zu isolieren und zu bewerten, ob deren Bewegung aus der Eigenbewegung der Drohne resultiert.

Die Performance der verschiedenen Evaluationsverfahren ist in Abbildung 5.13 dargestellt. Es werden dabei zunächst die Klassifikationen von Ebenen betrachtet. Bei diesen Ebenen handelt es sich um drei jeweils 1, 2 und 3 Meter entfernte mit gleich vielen Punkten.

Außerdem wird jede Metrik einmal unter Verwendung der selbst berechneten Geschwindigkeit und einmal unter Verwendung der über die IMU bestimmten Geschwindigkeit betrachtet. Ab jetzt sollen diese Methoden mit Vorwärts (VW) beziehungsweise Rückwärts (RW) deklariert werden, da es die IMU-Geschwindigkeit erlaubt Werte **vor** der Geschwindigkeitssortierung auszusortieren, während die selbstberechnete Geschwindigkeit das erst **rückwärtig** kann. Zur Erinnerung werden hier nochmal alle Metriken zusammengefasst:

Name	Formel
Parallelität	$\frac{\hat{x}_i(\bar{v} - \hat{\omega}T_1) \cdot \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)}{\ \hat{x}_i(\bar{v} - \hat{\omega}T_1)\ _2 \ \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\ _2}$
Distanz	$\frac{\ \hat{x}_i(\bar{v} - \hat{\omega}T_1)\ _2}{\ \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\ _2} n^T x$
$\ \text{Residuum}\ $	$\ \hat{x}_i(\bar{v} - \hat{\omega}T_1) - \hat{x}_i(\dot{x}_i - \hat{\omega}x_i)\ _{n^T x}^d$

Tabelle 5.2: Sortiermetriken (VW)

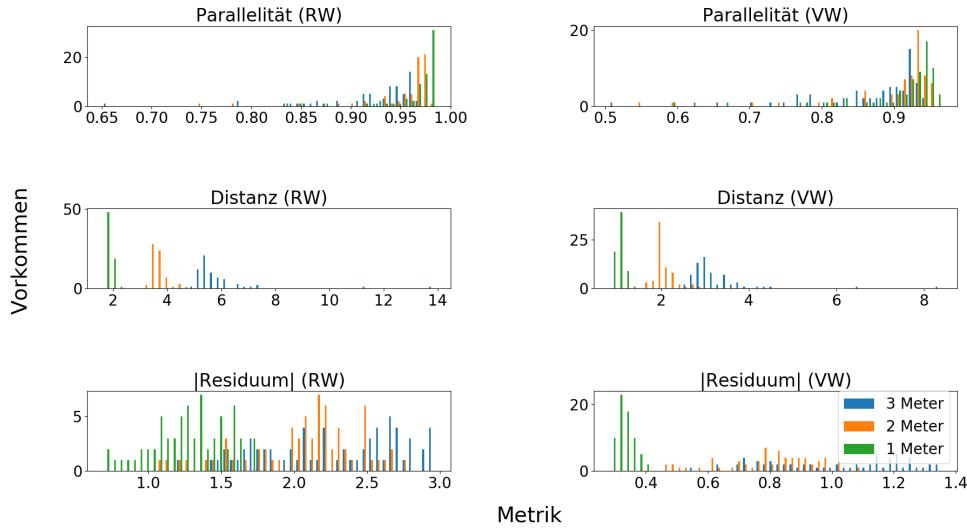


Abbildung 5.13: Simulation der Ebenenklassifikation bei $1 \frac{m}{s}$

Zunächst scheint es hier, dass sich die einzelnen Gaußkurven selbst bei geringen Höhen überlappen. Da keine Lücke größer als die Standardabweichung gegeben ist (wie in Kapitel 3.4 angenommen), kann so zunächst nicht sortiert werden. Eine genauere Untersuchung zeigt jedoch, dass die Standardabweichung der einzelnen Maße mit steigender Geschwindigkeit geringer werden. Fliegt die Drohne mit 10 m/s , so ergibt sich folgendes Bild:

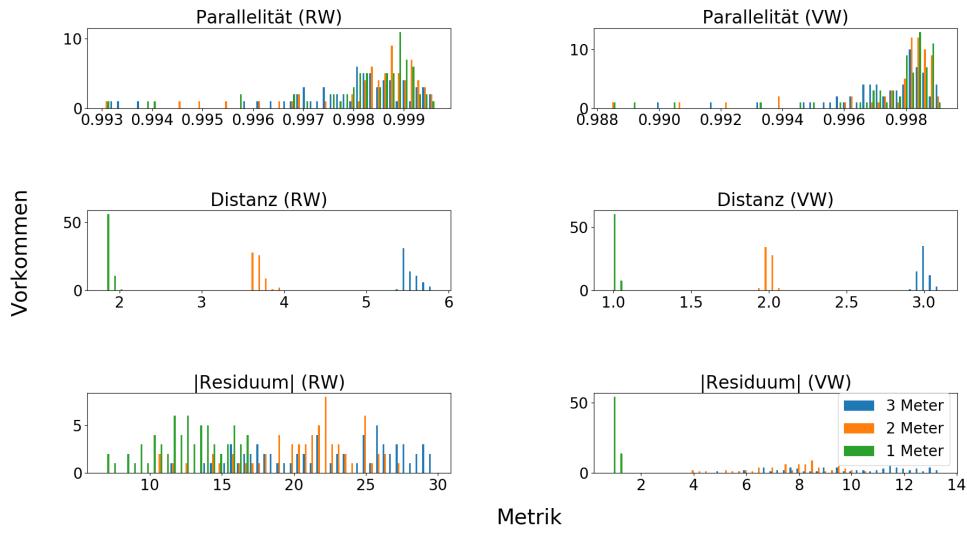


Abbildung 5.14: Simulation der Ebenenklassifikation bei $10 \frac{m}{s}$

Hier ist es möglich die Ebenen über die ihnen zugeordneten Distanzen selbst dann zu trennen, wenn die optisch bestimmte Rückwärtslösung \tilde{v} verwendet wird.

Die Aussortierung bewegter Features, gegenüber unbewegter ist in Abbildung 5.15 dargestellt, dabei unterscheidet sich der Winkel um mindestens 1° .

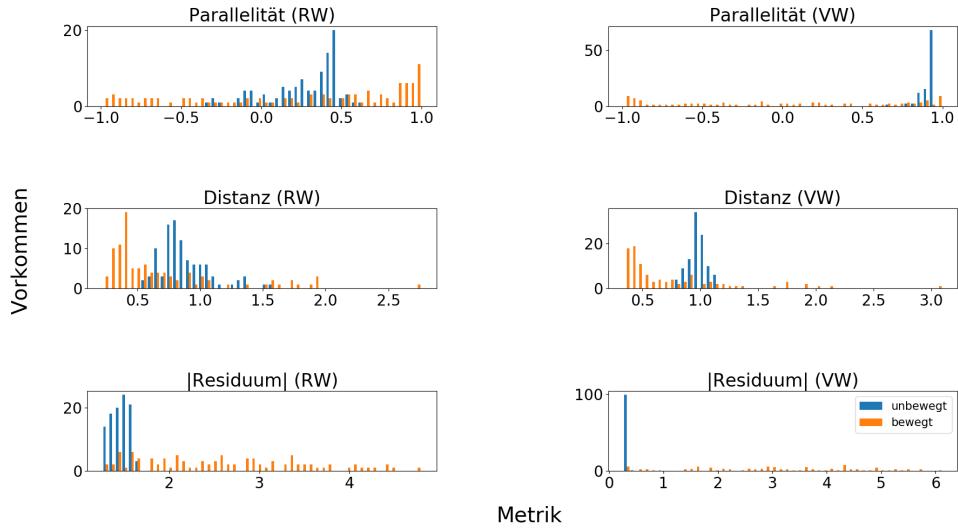


Abbildung 5.15: Simulation der bewegten Objekte Klassifikation bei $1 \frac{m}{s}$

Es ergibt sich ein ähnliches Problem, wie bei der Ebenenortierung. Generell ist außerdem zu bemerken, dass sich das Residuum der Vorwärtslösung gegenüber der Parallelität als bessere Metrik erweist. Dabei fließt allerdings mit ein, dass die bewegten Punkte sich in dieser Simulation nur in ihrer Richtung von korrekten Punkten unterscheiden. Prinzipiell wäre es aber durchaus möglich, dass die ihnen zugewiesene Distanz ebenfalls aufgrund der Eigen Geschwindigkeit falsch ist. Im Folgenden ist eine Simulation bei erhöhter Geschwindigkeit dargestellt.

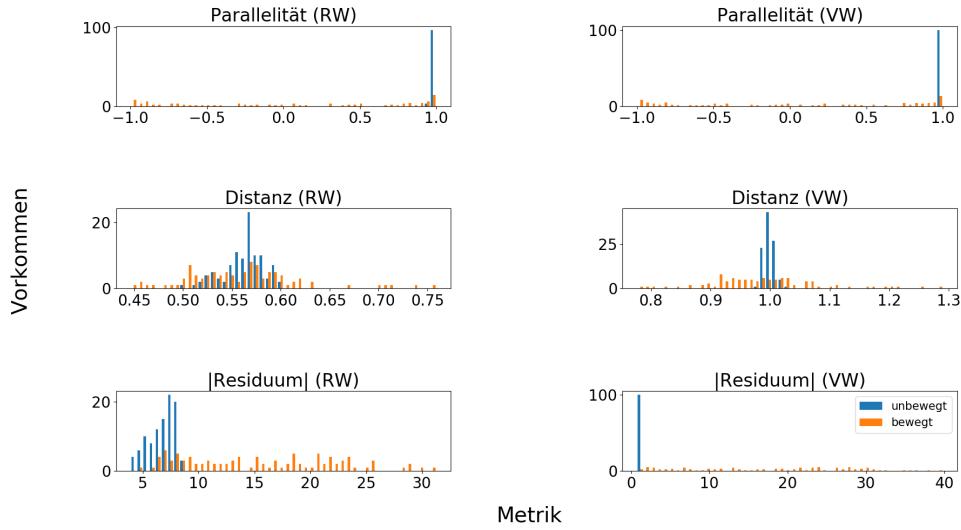


Abbildung 5.16: Simulation der bewegten Objekte Klassifikation bei $10 \frac{m}{s}$

Insgesamt ist das Geschwindigkeitsverhalten nicht weiter verwunderlich, denn je schneller sich die Drohne linear bewegt, desto kleiner wird der relative Driftfehler.

Zusammenfassend kann gesagt werden, dass höhere Geschwindigkeiten zu einer besseren Aufteilung in Ebenen, beziehungsweise bewegte und unbewegte Features führen. Die momentane Implementierung besitzt eine maximale Geschwindigkeit von $v_{\max} = [(2,9 \pm 0,43)d] \frac{m}{s}$. Fliegt sie über zwei, jeweils ein und zwei Meter entfernte Ebenen, auf denen sich bewegte Features befinden, so erhält man Abbildung 5.17.

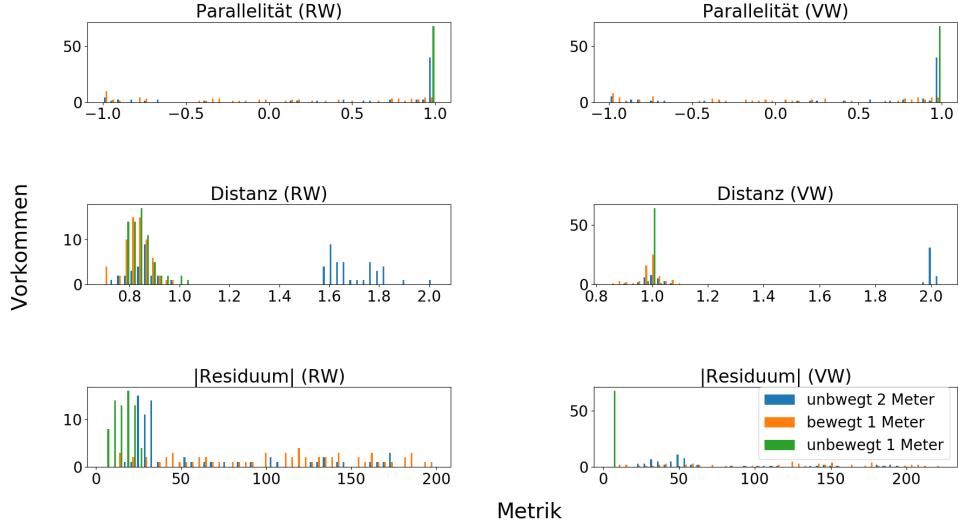


Abbildung 5.17: Simulation mit $v_{\max} = [(2, 9 \pm 0, 43)d] \frac{m}{s}$

Hier lässt sich nun die reale Güte der einzelnen Metriken, dabei sind nur 20% aller Punkte korrekt nutzbar. Bei einer zugelassenen Abweichung von rund 1,2 in der Parallelitätsmetrik, werden für die Vorwärtslösung 82% aller bewegten Features aussortiert und selbst mit der Rückwärtslösung immerhin 75%.

Die Sortierung in Ebenen funktioniert nach der Aussortierung der bewegten Punkte eindeutig über das Distanzmaß sowohl mit (RW) als auch (VW). Hier zeigt sich deutlich der Vorteil der neuen Metrik. Das rückwärtige Residuum, wie es in [12] verwendet wurde, ist nicht in der Lage, die Punkte der zweiten Ebene korrekt zu sortieren. Das vorwärtige Residuum ist im Gegensatz dazu eine geeignete Metrik, wenn nur zwischen verwendbaren und nicht verwendbaren Punkten unterschieden werden soll. Hier verzeichnen sie bessere Ergebnisse, als die bei Distanz- und Parallelitätmetrik. Auf die Vorteile der beiden anderen Metriken wurde jedoch bereits im Kapitel 3.4 eingegangen.

Kapitel 6

Diskussion und Fazit

Ziel dieser Arbeit war die äußere Stützung der Geschwindigkeit einer SUAV. Zu diesem Zweck sollte eine geeignete Hardwareplattform aufgebaut werden und mit einer entsprechenden Software ausgestattet werden.

Der Fokus dieser Arbeit lag dabei in der theoretischen Evaluation, da ausgiebige Messreihen und anschließende Evaluationen aus Zeitgründen nicht mehr möglich waren. Die Diskussion der Ergebnisse beschränkt sich hier folglich auf die simulativen Ergebnisse.

Wie bereits besprochen, verhält sich die Dynamik wie erwartet. Nur der numerische Fehler überschätzt die Standardabweichung stark. Die Metriken zur Aussortierung ungeeigneter Features sind präziser, als bisher verwendete und stellen die größte Verbesserung gegenüber den etablierten Methoden dar.

An dieser Stelle soll noch kurz darauf eingegangen werden, inwiefern die simulativen Ergebnisse aussagekräftig sind, beziehungsweise inwieweit sie reale Fehler sinnvoll abbilden. Der Annahme nach werden auf die korrekten Werte wiederholt gaußverteilte Fehler addiert, um Mittelwert und Standardabweichung der Lösung zu bestimmen.

Die dynamischen Eigenschaften der SUAV werden dabei vollkommen ignoriert, vielmehr werden die Werte so gesetzt, dass alle möglichen Fehlereffekte auftreten. Ob die SUAV diese Bewegung überhaupt ausführen kann, war hier nicht von Relevanz.

Dies vernachlässigt assoziierte Fehler, wie die Vibration der SUAV, welche sich vor allem auf die Distanzmessung auswirken würden. Vermutlich müsste hier ein Filter verwendet werden, um die Werte zu glätten. In der Praxis würde dies mit den anderen Fehlern interagieren, was zu einem anderen Endergebnis führen könnte.

Solche Fehler sind beispielsweise die mögliche Bildverwischung bei starken Drehungen, welche den Positionsfehler von ω abhängig macht. Welchen Einfluss all diese Phänomene haben, kann nur durch eine ausführliche Untersuchung tatsächlicher Messungen bestimmt werden, aber aufgrund der hier durchgeführten Simulation, von Verfahrensfehlern unterschieden werden.

Im Folgenden sind einige mögliche Experimente beschrieben, welche entweder mit den simulierten Ergebnissen übereinstimmen müssten oder wahrscheinlich neue Fehlerquellen aufdecken. Ein langsamer Flug mit rund $2\frac{m}{s}$ in einer Höhe von einem Meter über einem gut strukturierten Untergrund, in windstiller Laborumgebung, sollte sich gut mit den simulierten Ergebnissen decken.

Als mögliche Flugbahn wäre ein Quadrat geeignet, wobei die Drohne bei der 1. Ecke seitwärts fliegt und bei der 2. eine Drehung vollführt, um beide Effekte zu beobachten. Anschließend sollte die 3. Kante in einem leichten Aufwärtsflug abgeflogen werden und die 4. entsprechend in einem Abwärtsflug zurück zum Ausgangspunkt.

Mögliche Störbedingung, die neue Fehler aufdecken würden, wären Umgebungen in denen der Sonarsensor leicht falsche Werte produzieren kann. Dazu zählen viele Wände und Objekte im Flugraum. Interessant wäre außerdem ein Flug in wechselhaften Windsituationen.

6.1 Fazit

Aus dieser Arbeit lassen sich einige Schlüsse ziehen, sowohl im Hinblick auf das konkrete Problem, als auch auf die generelle Methodologie.

Zuerst ist hier zu bemerken, dass die statistische Analyse ein geeignetes Werkzeug zur Betrachtung nichtlinearer Fehler ist. Allerdings können durch das jeweilige Problem bedingt Polstellen auftreten, welche eine analytische Lösung unmöglich machen.

Wie in dieser Arbeit gezeigt wurde, bedeutet dies aber nicht, dass aus einer solchen Analyse keine Informationen gewonnen werden können. So ist es durchaus möglich, Aussagen über das dynamische Verhalten des Fehlers zu treffen. Insgesamt ist es jedoch besser, nichtlineare Fehler zu vermeiden, da eine Korrektur, selbst wenn möglich, zusätzlichen Rechenaufwand erfordern würde.

Generell zeigt sich auch der Nachteil äußerer Stützungen, denn der Fehler ist durch die Stützmessung fundamental nach unten beschränkt und es passiert im Gegensatz zum geschlossenen Regelkreis keine Fehlerunterdrückung. Trotz allem bewährt sich das Einbeziehen optischer Daten als Methode der Geschwindigkeitsbestimmung.

Kapitel 7

Verbesserungsvorschläge und Zukunftsansblick

Im Folgenden muss zwischen zwei Arten von Verbesserungsvorschlägen unterschieden werden. Die einen bauen inhaltlich auf der hier präsentierten Arbeit auf, die anderen beziehen sich auf den Ansatz selbst und korrigieren die im Fazit genannten Probleme.

7.1 Inhaltlich aufbauende Verbesserungen

Zuerst sollen die Vorschläge erster Art betrachtet werden. Hierbei ist direkt die Abwesenheit fundierter Messergebnisse und Auswertungen zu bemerken, welche aus Zeitgründen weggelassen wurden. Die in dieser Arbeit präsentierten Grundlagen sind ein Anhaltspunkt für die Güte der Methode, ignorieren aber nicht nur Hardwarefehlerquellen wie Vibration, die Keulenform des Distanzsensors sowie sphärische Aberrationen und andere optische Effekte, sondern auch die Kopplung dynamischer Parameter wie Schräglage und Geschwindigkeit.

Diese sollten ausgiebig in mehreren Messreihen untersucht werden. Zu diesem

Zweck wäre es beispielsweise möglich ein differentielles GPS als Referenzgröße zu installieren und in einem Flug über eine strukturierte Umgebung Daten zu sammeln.

Des Weiteren wäre es möglich, den für diese Arbeit geschriebenen Echtzeitcode auf eine performantere Hardware zu transplantiert, um das Realzeitverhalten des Algorithmus zu verbessern. Hierzu bietet sich zum Beispiel das im Lehrstuhlinventar vorhandenen Jetson Board an. Eine weitere Möglichkeit wäre auch das rechenaufwendige Finden und Verfolgen der Bildpunkte auf einen dedizierten Chip auszulagern wie den PX4FLOW [22]. Eine weitere wichtige Aufgabe wäre die Generierung eines *lookuptable* indem vorher bestimmte Erwartungswertintegrale (vergleiche 3.35) hinterlegt sind. Diese könnten genutzt werden, um Beispielsweise den in 5.6 beschriebenen systematischen Fehler auszugleichen.

Daneben wäre auch eine Implementierung des Kamerakalibriervorgangs interessant, auch wenn der Translationsfehler keinen großen Einfluss auf den Gesamtfehler hat (siehe Abbildung 5.10).

Verbesserungswürdig wäre auch die Ebenendetektion und -sortierung. Für die einzelnen Momente (3.48) sollten erwartungstreure Schätzer verwendet werden, wobei das Ergebnis auch mit den einfachen, den Definitionen genügenden Schätzern zufriedenstellend war. Für die Aufteilung könnte noch nach geeigneteren Maßen gesucht werden, welche das Trennen von noch verrauschteren Ebenen ermöglichen. Ein möglicher Ansatz wäre, die Differenz der Distanzen der Einzelfeatures nach *peaks* zu durchsuchen (siehe Abbildung 7.1 welche mit den Parametern von Abbildung 5.13 simuliert wurde). So könnten Abstände zwischen Ebenen gefunden werden, ohne deren genauen Abstand vorher kennen zu müssen.

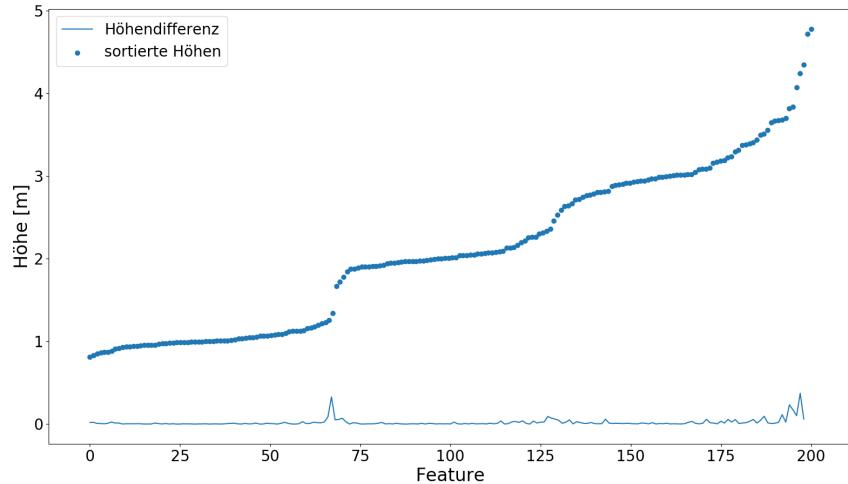


Abbildung 7.1: Differenz der einzeln bestimmten Featuredistanzen

Des Weiteren wäre eine genauere Untersuchung des dynamischen Verhaltens sinnvoll. Hierzu muss ein Maß gefunden werden, welches die Überlappung der Histogramme zufriedenstellend beschreibt, z.B.:

$$ov[a, b]_1 = \begin{cases} m_1(b) + m_2(b) - m_1(a) + m_2(a) & \text{für } m_1(a) \geq m_1(b) \\ m_1(a) + m_2(a) - m_1(b) + m_2(b) & \text{für } m_1(b) \geq m_1(a) \end{cases} \quad (7.1)$$

oder

$$ov[a, b]_2 = \sum_{i \in \text{bins}} \min(\alpha, \beta). \quad (7.2)$$

Dabei beschreiben α, β die Höhen der jeweiligen *Histogrammbins* und m_1 sowie m_2 Mittelwert und Varianz.

Denkbar wäre außerdem die Position der in einer Ebene befindlichen Punkte zu nutzen, um das Suchgebiet für neue Punkte derart einzugrenzen, dass mit hoher Wahrscheinlichkeit Features in derselben Ebene gefunden werden. Dabei könnten die komplexe Hülle der Punkte oder Farbeigenschaften des

Bildes hilfreich sein.

Ein weiterer Ansatzpunkt wäre die Dimensionierung der Funktionsparameter des Verfolgungs- beziehungsweise Detektions-algorithmus. Diese sollten nicht nur an die Dynamik des Systems angepasst werden, sondern mit den oben beschriebenen Eingrenzungen skalieren, sodass innerhalb gleicher Ebenen eine geringere Minimaldistanz herrscht. Wie im Fazit erwähnt, ist der hier verwendete *open-loop* Ansatz prinzipiell nicht ideal, weswegen als Verbesserungsvorschlag zweiter Art eine Möglichkeit der Geschwindigkeitsschätzung detailliert dargestellt wird.

7.2 Verbesserungen des Ansatzes selbst

Im Gegensatz zum bisherigen Ansatz wird nun die erwartete nächste Position für jedes Feature berechnet und mit der später Gemessenen verglichen. Dies erlaubt eine Regelung des internen Geschwindigkeitszustands über einen *Extendet Kalman Filter* [1]. Linearisiert gilt hier für jedes Feature x :

$$x_k = x_{k-1} + \dot{x}_k t, \quad (7.3)$$

wobei t der hier als konstant angenommene Zeitschritt ist. Damit gilt mit (8.4)

$$x_k = x_{k-1} + [(v_k - e_z^T v_k x_{k-1}) n^T \frac{x_{k-1}}{d} + \hat{\omega}_k x_{k-1} - e_z^T (\hat{\omega}_k x_{k-1}) x_{k-1}] t. \quad (7.4)$$

Dies kann nun als Matrix C mit Zustandsvektor $[v, \omega]^T$ ausgedrückt werden. Mit der Messung der Drehgeschwindigkeit ω und der Beschleunigung a als Input wäre so ein System gefunden, welches als Basis für den Kalmanfilter verstanden werden kann. Hier könnte man nach wie vor die oben besprochenen Sortieralgorithmen verwenden.

Literaturverzeichnis

- [1] Florian Bauer. Luenberger-beobachter und extended kalman-filter: Ein vergleich. http://floba.info/Files/Kalman-Filter/Luenberger-Beobachter_und_Extended_Kalman-Filter_Ein_Vergleich.pdf, 2013. Stand 2019.01.17.
- [2] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *OpenCV Document, Intel, Microprocessor Research Labs*, 1, 01 2000.
- [3] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Model. Meth. Appl. Sci.*, 1, 01 2007.
- [4] Haiyang Chao, Yu Gu, and Marcello Napolitano. A survey of optical flow techniques for robotics navigation applications. *Journal of Intelligent & Robotic Systems*, 73(1):361–372, Jan 2014.
- [5] F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics Automation Magazine*, 13(4):82–90, Dec 2006.
- [6] Mike Stephens Chris Harris. A combined corner and edge detector. *Alvey vision conference*, 1988.
- [7] Stefan Schäffler2 David Meinstrup. *Stochastik*. Number 9783540267072. Springer Spektrum, 2005.

- [8] Lawrence T DeCarlo. On the meaning and use of kurtosis. *Psychological methods*, 2(3):292, 1997.
- [9] Gerd Fischer. *Lernbuch Lineare Algebra und Analytische Geometrie*. Number 978-3-8348-2379-3 in SpringerLink : Bücher. Springer Spektrum, Wiesbaden, 3. aufl. 2017 edition, 2017.
- [10] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015.
- [11] V. Grabe, H. H. Bülthoff, and P. R. Giordano. On-board velocity estimation and closed-loop control of a quadrotor uav based on optical flow. In *2012 IEEE International Conference on Robotics and Automation*, pages 491–497, May 2012.
- [12] V. Grabe, H. H. Bülthoff, and P. Robuffo Giordano. Robust optical-flow based self-motion estimation for a quadrotor uav. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2153–2159, Oct 2012.
- [13] Norbert Henze. *Irrfahrten und verwandte Zufälle*. Springer Spektrum, 2013.
- [14] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, Apr 1987.
- [15] M. Irani, B. Rousso, and S. Peleg. Recovery of ego-motion using region alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):268–272, March 1997.
- [16] Alexander Krizski. Collision avoidance for a multi-rotor uav. Master’s thesis, Automation Laboratory, Heidelberg University, 2017.

- [17] H. Romero, R. Benosman, and R. Lozano. Stabilization and location of a four rotor helicopter applying vision. In *2006 American Control Conference*, pages 6 pp.–, June 2006.
- [18] H. Romero, S. Salazar, and R. Lozano. Real-time stabilization of an eight-rotor uav using optical flow. *IEEE Transactions on Robotics*, 25(4):809–817, Aug 2009.
- [19] Francisco Romero Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018.
- [20] J. Shi and C. Tomasi. Good features to track. *9th IEEE Conference on Computer Vision and Pattern Recognition. Springer.* pp. 593–600, 1994.
- [21] AD Team et al. Pixhawk v2 specifications. http://www.hex.aero/wp-content/uploads/2016/07/DRS_Pixhawk-2-17th-march-2016.pdf, März 2016. Stand: 2019.05.21.
- [22] S. Tsai and S. Zhuang. Optical flow sensor integrated navigation system for quadrotor in gps-denied environment. In *2016 International Conference on Robotics and Automation Engineering (ICRAE)*, pages 87–91, Aug 2016.
- [23] Yu Zhang, Tingting Wang, Zhihao Cai, Yingxun Wang, and Zhenxing You. The use of optical flow for uav motion estimation in indoor environment. In *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, pages 785–790, Aug 2016.

Abbildungsverzeichnis

1.1	Drift Messung einer ruhenden SUAV	3
2.1	Variablen Definition für (2.4) aus [18]	8
3.1	Flächen, Ecken und Kanten http://mynextcar.info/pics/allpaper-birds-eye-view	9
3.2	Pyramidische Lucas Kanade Implementierung verändert nach: https://pdfs.semanticscholar.org/2b24/df177471a4d463e4d8af07669909505213ef.pdf	14
3.3	Kameramodell verändert nach: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html	15
3.4	Hochhäuserschlucht mit zum Boden parallele Ebenen http://mynextcar.info/pics/allpaper-birds-eye-view	26
3.5	Schematische Darstellung einer verrauschten Statistik mit 3 Gaußverteilungen	29
3.6	Erwartungswert m_1	32
3.7	Varianz m_2	32
3.8	Kurtosis m_4	33
4.1	SUAV	35
4.2	SUAV schematischer Aufbau	36
4.3	Aufbau: Messung des Normalvektor- und Winkelgeschwindigkeitsfehlers	38

4.4	Aufbau: Messung des Optischen Fehlers	40
4.5	Aufbau: Messung zur Umrechnung von Pixeln auf Meter	41
4.6	Höhenmessung mit Ultraschallkeule	43
4.7	Höhenbestimmung	44
4.8	Translative Geschwindigkeitskomponente	46
4.9	Beispiele für Arucomarker $\text{https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html}$	47
4.10	Feautre Auswahl in Abhängigkeit der minimalen Distanz $\text{http://mynextcar.info/pics/allpaper-birds-eye-view}$	49
4.11	Hochhäuserschlucht mit <i>Shi-Tomasi Corner detection</i> $\text{http://mynextcar.info/pics/allpaper-birds-eye-view}$	51
4.12	Schematisches Zeitverhalten der Berechnung	52
4.13	Flussdiagramm für die Geschwindigkeitsberechnung	54
5.1	Lösung in Abhängigkeit von der Featurezahl	59
5.2	Simulation der Winkelabhängigkeit	60
5.3	Simulation der Abhängigkeit der Geschwindigkeit von der Höhe	61
5.4	Simulation der Abhängigkeit der Geschwindigkeit von der Höhe bei zehnfachem Bildbereich	62
5.5	Simulation der Abhängigkeit der Geschwindigkeit von der Höhe ($\omega_x = \omega_y = 0$)	63
5.6	Simulation der Auswirkung der mittleren Bildverteilung	64
5.7	Auswirkung des optischen Fehlers	65
5.8	Auswirkung des Drehgeschwindigkeitsfehlers.	66
5.9	Auswirkung des Höhenfehlers	67
5.10	Auswirkung des Translationsfehlers.	68
5.11	Auswirkung des Normalvektorenfehlers	69
5.12	Numerischer Fehler	70
5.13	Simulation der Ebenenklassifikation bei $1 \frac{m}{s}$	72
5.14	Simulation der Ebenenklassifikation bei $10 \frac{m}{s}$	73
5.15	Simulation der bewegten Objekte Klassifikation bei $1 \frac{m}{s}$	74

5.16	Simulation der bewegten Objekte Klassifikation bei $10 \frac{m}{s}$	75
5.17	Simulation mit $v_{\max} = [(2, 9 \pm 0, 43)d] \frac{m}{s}$	76
7.1	Differenz der einzeln bestimmten Featuredistanzen	83
8.1	Fernsteuerungskonfiguration	100
8.2	Anschluss der ESC	
	Motoren: http://ardupilot.org/copter/docs/connect-escs-and-motors.html ,	
	ESC: https://www.mikrocontroller.com/index.php?main_page=product_infocPath=69products_id=982zenid=d004212824ed5e4a0d2922897e7ac9c4	104
8.3	Anschluss des Pixhawk	105
8.4	Anschluss von Flightcontroller und Raspberry Pi	106
8.5	Anschluss der ESC	106

Tabellenverzeichnis

4.1	Fehlerbestimmung von Normalvektor und ω	38
4.2	Fehler in der Feature Position	41
4.3	Fehler in der Distanzmessung	45
5.1	Zusammenfassung der einzelnen Fehler mit Priorität	70
5.2	Sortiermetriken (VW)	71

Kapitel 8

Anhang

8.1 Rotatorischer Anteil der Bildbewegung

Es folgt eine Isolierung des rotatorischen Anteils der Bildbewegung, wie sie in [12] durchgeführt wurde. Der Zitation der Veröffentlichung zufolge wurde diese [5] entnommen. Dabei ist den Autoren ein Fehler unterlaufen, während hier die korrekte Isolierung vorgenommen wurde. Um den rotatorischen Anteil der Bildbewegung zu isolieren, wird zunächst (3.12) nach \dot{x}_i aufgelöst.

$$\dot{x}_i = \frac{\dot{X}_i - \dot{X}_{iz} x_i}{X_{iz}}. \quad (8.1)$$

In dieser Beschreibung finden sich allerdings noch die dreidimensionalen Raumpunkte, die nicht direkt gemessen werden können. Umformen und Nutzen von (3.10) führt auf:

$$\dot{x}_i = \frac{\hat{\omega} X_i + v - e_z^T (\hat{\omega} X_i + v) x_i}{X_{iz}}. \quad (8.2)$$

Um X_i ebenfalls zu eliminieren, lässt sich die Definition der Bildpunkte: $x_i := \frac{X_i}{X_{iz}}$ verwenden.

$$\dot{x}_i = \frac{v - e_z^T v x_i}{X_{iz}} + \hat{\omega} x_i - e_z^T (\hat{\omega} x_i) x_i. \quad (8.3)$$

Unter Einführung der in (3.14) genutzten Ebenenforderung ist es nun möglich, auch X_{iz} zu eliminieren:

$$\dot{x}_i = (v - e_z^T v x_i) n^T \frac{x_i}{d} + \hat{\omega} x_i - e_z^T (\hat{\omega} x_i) x_i. \quad (8.4)$$

Damit besteht \dot{x}_i aus Faktoren, die entweder v oder $\hat{\omega}$ enthalten, oder anders gesagt einem translatorischen u und einem rotatorischen Teil Ω .

$$\dot{x}_i = u + \Omega. \quad (8.5)$$

Eine genauere Betrachtung des rotatorischen Anteils Ω liefert:

$$\Omega = \hat{\omega} x_i - e_z^T (\hat{\omega} x_i) x_i. \quad (8.6)$$

Unter der Verwendung einer schiefsymmetrischen Matrix anstelle des Kreuzprodukts und den Vertauschungsregeln des Spatprodukts gilt:

$$\Omega = -\hat{x}_i \omega - (\hat{x}_i e_z)^T \omega x_i. \quad (8.7)$$

Umstellung und Zusammenfassung in einer Matrix liefert:

$$\Omega = L \omega, \quad (8.8)$$

mit

$$L = \begin{bmatrix} -x_{ix} x_{iy} & 1 - x_{ix}^2 & -x_{iy} \\ -1 + x_{iy}^2 & x_{ix} x_{iy} & x_{ix} \\ 0 & 0 & 0 \end{bmatrix}. \quad (8.9)$$

Dies unterscheidet sich von der in [12] verwendeten Matrix

$$\begin{bmatrix} -x_x x_y & 1 + x_x^2 & -x_y \\ -(1 + x_y)^2 & x_x x_y & x_x \\ 0 & 0 & 0 \end{bmatrix}, \quad (8.10)$$

welche nicht einmal symmetrisch in x- und y-Richtung ist. Der Fehler liegt hier allerdings nicht bei den Autoren, welche die Matrix selbst auch nur aus [5] übernommen haben.

8.2 Spektralnorm der Pseudoinversen

Sei $\|M\|_2$ die Spektralnorm einer Matrix M

$$\|M\|_2 = \sqrt[2]{\lambda_{\max}(M^T M)}, \quad (8.11)$$

wobei $\lambda_{\max}(L)$ der maximale Eigenwert einer Matrix L ist. Sei $U\Sigma V^*$ die Singulärwertzerlegung der Matrix M . Dann ist $M^+ = V\Sigma^+U^*$. Es gilt außerdem, dass:

$$\|M\| = \|V\Sigma^+U^*\| = \|V\|\|\Sigma^+\|\|U^*\| = \|\Sigma^+\|, \quad (8.12)$$

da U, V^* singuläre Matrizen sind. Weil außerdem Σ eine Diagonalmatrix mit Singulärwerten σ_i oder 0 ist, gilt für die Pseudoinverse Σ^+

$$(\Sigma^+)_{i,j} = \begin{cases} \frac{1}{\sigma_i} & i = j \wedge \sigma_i \neq 0 \\ 0 & sonst \end{cases} \quad (8.13)$$

Aus (8.11), (8.12) und (8.13) folgt:

$$\|M^+\|_2 = \sqrt[2]{\lambda_{\max}(\Sigma^+\Sigma^+)} = \frac{1}{\sigma_{\min}(M)}. \quad (8.14)$$

Hierbei ist $\sigma_{\min}(M)$ der kleinste Singulärwert der Matrix M . Für $(A +$

$\Delta A)^+$ folgt dann:

$$\|(A + \Delta A)^+\|_2 = \frac{1}{\sigma_{\min}(A + \Delta A)}. \quad (8.15)$$

8.3 Erwartungswerte

Es folgt eine kurze Zusammenfassung der Rechenregeln und Eigenschaften stochastischer Erwartungswerte (aus [7] entnommen): Der Erwartungswert ist linear für beliebige Zufallsvariablen X_i :

$$\mathbf{E}(aX_1 + bX_2) = a\mathbf{E}(X_1) + b\mathbf{E}(X_2). \quad (8.16)$$

Sind die Zufallsvariablen außerdem stochastisch unabhängig, so gilt:

$$\mathbf{E}\left(\prod_{i=1}^n X_i\right) = \prod_{i=1}^n \mathbf{E}(X_i). \quad (8.17)$$

Der Erwartungswert einer reellwertigen Zufallsvariablen mit Dichte f ist dabei allgemein gegeben durch:

$$\mathbf{E}[X] = \int_{-\infty}^{\infty} dx x f(x). \quad (8.18)$$

Ist H eine messbare Funktion, so gilt weiterhin:

$$\mathbf{E}[H(X)] = \int_{\mathbb{R}^d} dx H(x) f(x). \quad (8.19)$$

Diese Aussage lässt sich auf Funktionen mehrerer Zufallsvariablen ausweiten. Sei $g(x, y)$ eine reellwertige Funktion, dann ist der Erwartungswert von $g(X, Y)$:

$$\mathbf{E}[g(X, Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) f(x, y) dx dy. \quad (8.20)$$

8.4 In der SUAV verwendete Bauteile

Es folgt eine Liste sämtlicher verwendeter Bauteile:

- Hexa Combi V3.5 Spezial Blue
- F550 Hexa-Rotor 550mm Multirotor Rahmen Satz mit hohem Fahrwerk
- Pixhawk 2.1 Cube
- Mikrokopter Rotoren mit 5mm Mount (Version unbekannt)
- LiPo holder set (GFK)
- Multicopter 10x4.5 Zoll Propeller
- Raspberry Pi
- Graupner GR-16 Radio Empfänger
- Graupner mx-20 Fernbedienung
- Strom-Pi Header
- Raspberry Pi Kamera v2.1
- Maxbotix I2CXL-MaxSonar-EZ4 High Performance Sonar

8.5 Setup und Aufbau der SUAV

8.5.1 Pixhawk Setup

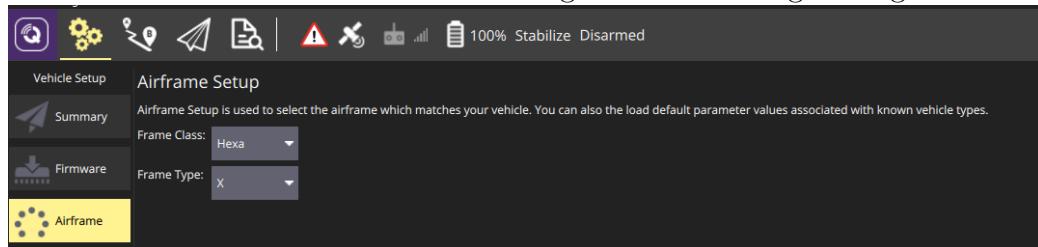
Die SUAV wird mit Hilfe der *QGroundControl* v3.5.0 Software konfiguriert.

Firmware

Unter dem Reiter **Firmware** im **Vehicle Setup Interface** wird die Arducopter Firmware v3.6.8 auf den Pixhawk Flight Controller gespielt.

Airframe Auswahl

Unter dem Reiter **Airframe** werden folgende Einstellungen vorgenommen:



Radio Setup

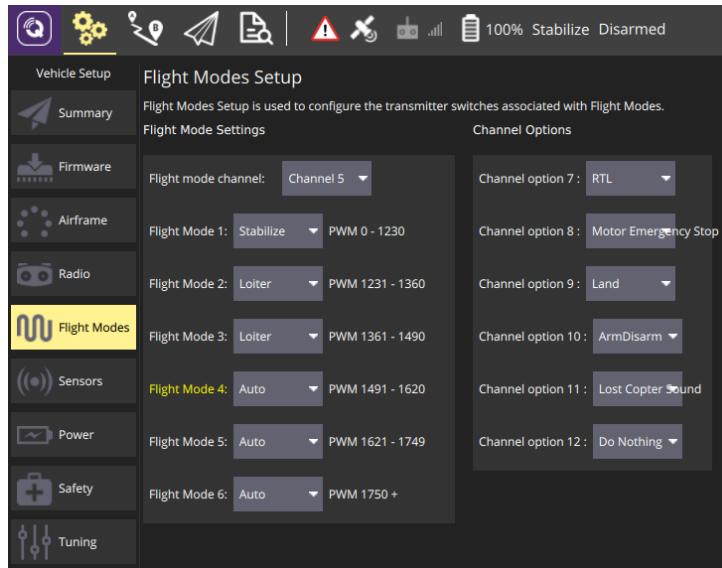
Die Auswahl der Flightmodi sowie die Steuerung und Kontrolle des Hexakopters erfolgt mithilfe einer Graupner mx 20 Fernbedienung, welche über einen GR 16 Radio Empfänger mit dem Pixhawk kommuniziert (Siehe Kapitel 8.5.3). Hierbei folgt man zunächst einfach den Aufforderungen des **Calibrate** Dialogs. Die momentane Konfiguration ist mit Aufklebern auf der Fernbedienung markiert:



Abbildung 8.1: Fernsteuerungskonfiguration

Flight Modes

Die SUAV kann in verschiedenen Flightmodes fliegen. Eine Liste der verfügbaren Flightmodes findet sich auf ardupilot.org. Der in dieser Arbeit verwendete Hexacopter ist wie folgt konfiguriert:

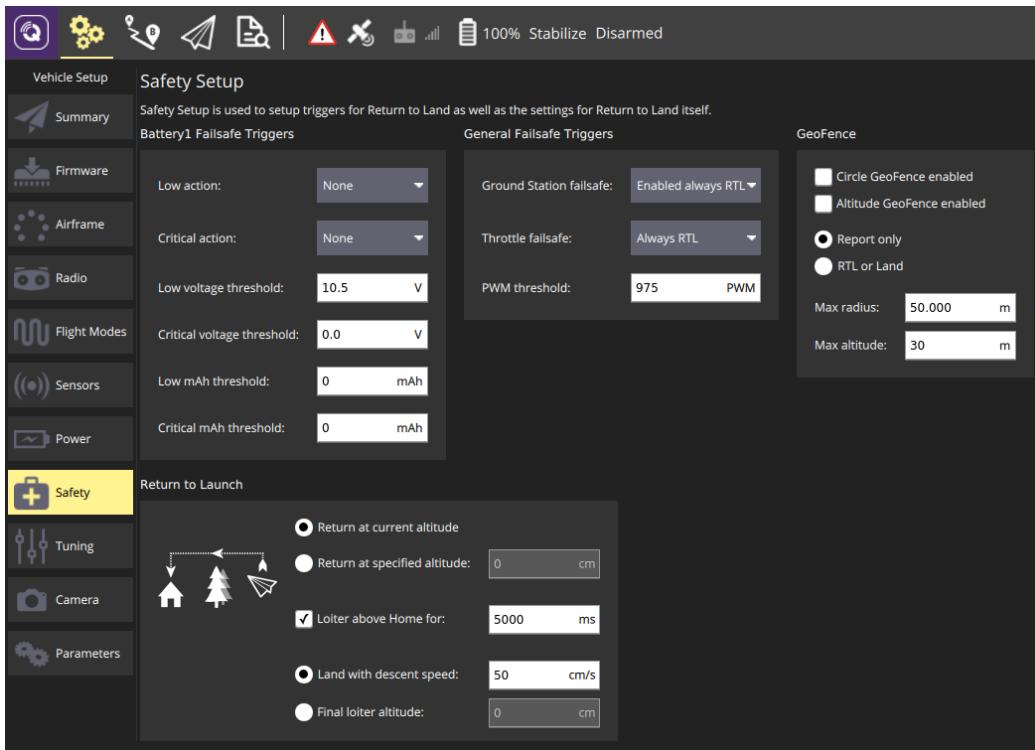


Sensors

Um die Sensoren des Pixhawk zu kalibrieren, folgt man den Instruktionen der Qgroundcontrol software unter dem Reiter **Sensors**. Es ist zu bemerken, dass sich der Pixhawk zur Kalibrierung nicht auf dem Frame befinden muss, was diese erheblich erleichtert.

Safety

Unter dem Reiter **Safety** werden folgende Einstellungen vorgenommen:



Parameters

Damit der Raspberry Pi über Mavlink mit dem Flightcontroller kommunizieren kann, werden im Dialog **Parameters** folgende Einstellungen vorgenommen:

- SERIAL2_PROTOCOL = 1
- SERIAL2_BAUD = 921600
- LOG_BACKEND_TYPE = 3

Vergleiche hierzu die Dokumentation.

Zur Aktivierung des Sonarsensors sind außerdem die folgenden Parameter zu setzen:

- RNGFND_MAX_CM = "700" (i.e. 7m max range)

- RNGFND_TYPE = “2” (MaxbotixI2C sonar)

8.5.2 Raspberry Pi setup

Der Raspberry Pi wird mit dem ubiquity image (2018-11-15-ubiquity-xenial-lxde) geflasht. Das Standartpasswort ist Ubuntu. Auf dem Image ist bereits das komplette ROS Package sowie OpenCv installiert. ROS wird außerdem automatisch gesourced, sodass alle Befehle direkt im Terminal verwendet werden können. Damit der Sonarsensor mit dem Pi kommunizieren kann, muss das *distance_sensor* Add-on von der *blacklist* entfernt werden, wozu die folgenden Schritte notwendig sind:

```
$roscd mavros  
$cd launch/  
$sudo nano px4_pluginlist.yaml  
lösche ”distance_sensor”
```

8.5.3 Hardware Aufbau

Anschluss des ESC an die Motoren

Die ESC (*Electronic Speed Control*) ist in der Mitte des Flamewheels auf Gummistützen geschraubt. Dort werden Löcher in die untere Platte gebohrt. Jeder der Ports wird entsprechend des folgenden Diagramms mit dem entsprechenden Motor verbunden.

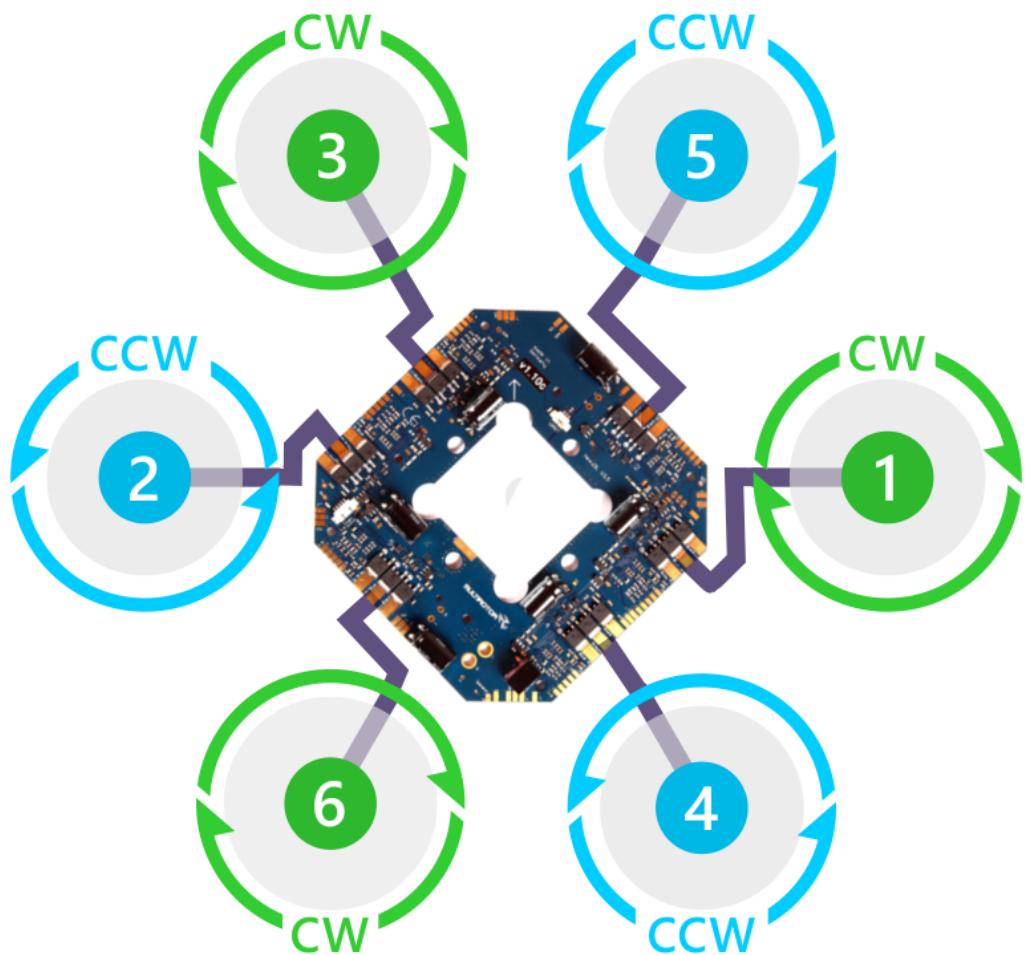


Abbildung 8.2: Anschluss der ESC

Anschluss des Pixhawk

Bevor die Drohne in Betrieb genommen werden kann, muss der Hardware Safety Switch mit dem GPS1 Port des Pixhawk verbunden werden. Außerdem muss der Powerbrick an den POWER1 Port angeschlossen werden. Optional kann auch der Buzzer per USB Port angeschlossen werden. Der Anschluss der restlichen Komponenten ist der folgenden Abbildung zu entnehmen.

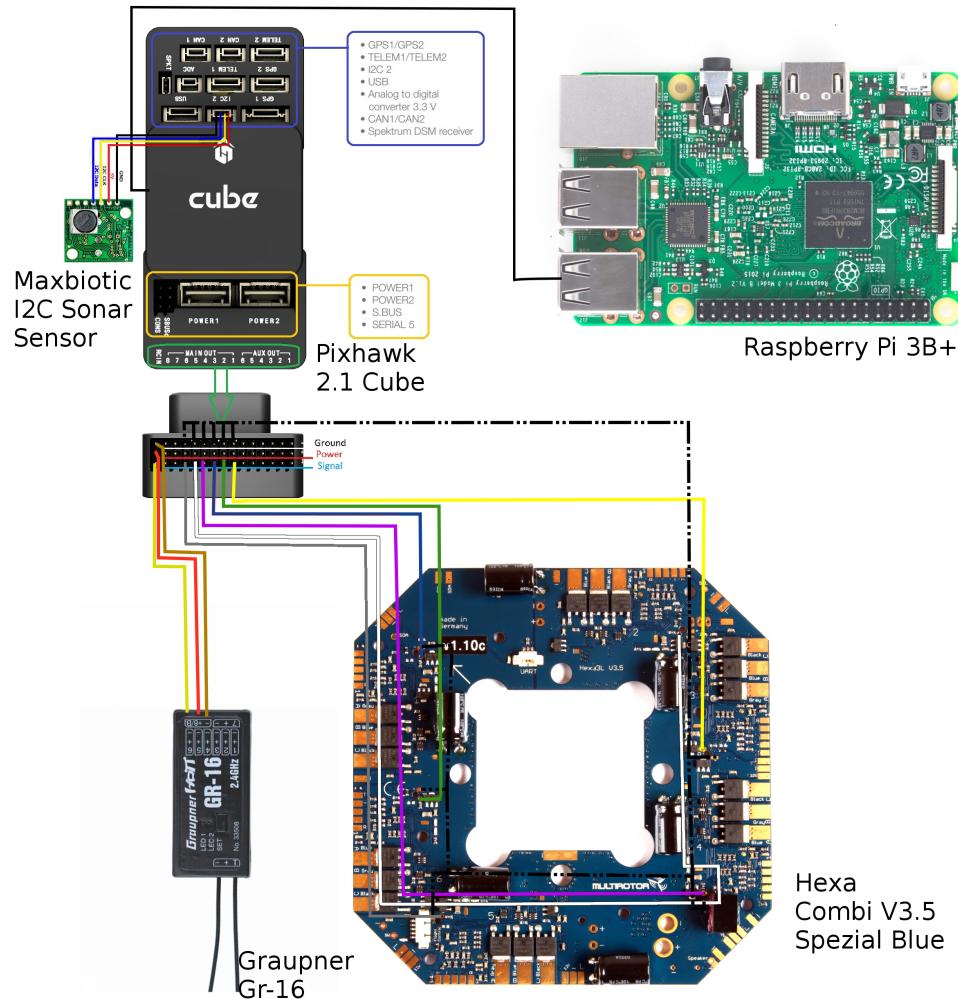


Abbildung 8.3: Anschluss des Pixhawk

Anschluss des Raspberry Pi

Der Raspberry Pi wird von derselben Batterie gespeist, die auch den Pixhawk betreibt. Um dies zu ermöglichen, muss das StromPi 2 Modul auf dem Raspberry Pi installiert werden. Um das Modul in den korrekten Modus zu bringen, folgt man der Anleitung auf joy-it.net.

8.5.4 Starten des Hexacopters

1. Fernbedienung anschalten
2. Flightcontroller-Batterie und Pixhawk Flight Controller und Raspberry Pi anschließen

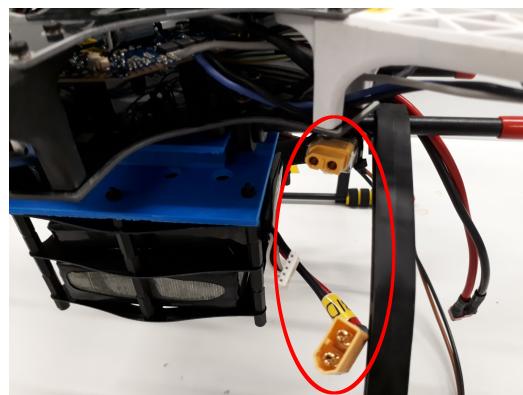


Abbildung 8.4: Anschluss von Flightcontroller und Raspberry Pi

3. Motorbatterie an ESC anschließen, dabei darauf achten, dass die Hände außer Reichweite der Propeller sind. Diese können manchmal nachdrehen.

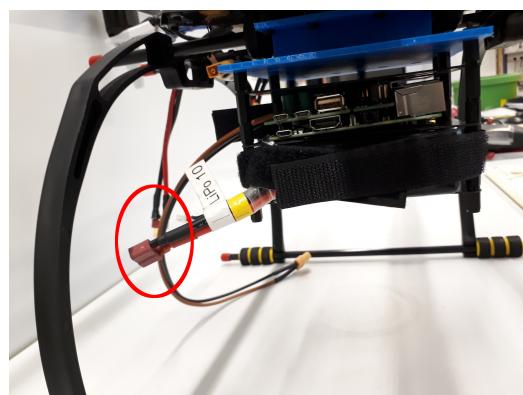


Abbildung 8.5: Anschluss der ESC

4. Den rot blinkenden *hardwaresafetyswitch* drücken bis er konstant leuchtet.
5. Den *armingswitch* an der Fernsteuerung betätigen.
6. In den Flugmodus Stabilizing oder Loitering wechseln.
7. Den linken Stick der Fernsteuerung nach unten rechts drücken bis die Motoren angehen. Bei einer zu langen Verzögerung zwischen den Schritten 5 und 6 muss Schritt 5 wiederholt werden. Ist ein Buzzer angeschlossen, teilt dieser das *disarming* über ein piepen mit.
8. Fliegen.

8.5.5 Landen des Hexacopters

1. SUAV landen.
2. Motorbatterie von ESC trennen.
3. Flightcontroller Batterie von Pixhawk und Raspberry Pi trennen.

Danksagung

Ich möchte diese Stelle nutzen, um den vielen Personen zu danken, die direkt oder indirekt dazu beigetragen haben, dass diese Arbeit möglich wurde. Zunächst gilt mein Dank dem gesamten Lehrstuhl für Automation, welcher mir in seiner Gesamtheit immer mit Rat und Tat zur Seite stand.

Besonderer Dank gilt auch meinem Betreuer Holger Dieterich, der sich immer für mich eingesetzt hat, auch wenn er eigentlich viel zu viel zu tun hatte. Außerdem noch Tobias Bak, ohne dessen technisches Verständnis und Expertise im Drohnenbau der Aufbau des Hexacopters in Hardware und Software Wochen länger gedauert hätte.

Vielen Dank auch an meine Korrekturleser Oliver Fischer, Albert Loran und Jenz Wagner. Dank ihrer unermüdlichen Arbeit wurde diese Arbeit nicht nur inhaltlich, sondern auch formal um ein vielfaches verständlicher.

Zum Schluss möchte ich noch meiner Familie danken, die mir zuhause immer eine Rückzugsmöglichkeit gegeben hat, in der ich meine Batterien wieder aufladen konnte und welche mich an jeder Ecke unterstützt haben, damit ich mich ganz auf meine Arbeit konzentrieren konnte.

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den