SENSOR_FUSION_REV_A Dokumentation

Release single

Dipl.-Ing. (FH) Andreas Gschossmann

01.09.2015

The Kalman Filter in its various forms is clearly established as a fundamental tool for analyzing and solving a broad class of estimation problems.

Leonhard McGee and Stanley Schmidt,
Ames Research Center, NASA

Abkürzungsverzeichnis

UAV	Unmanned Aerial Vehicle
PID	Proportional Integral Derivative
GPS	Global Positioning System
IMU	Inertial Measurement Unit
EKF	Extended Kalman-Filter
GNSS	Globales Navigationssatellitensystem
UKF	Unscented Kalman-Filter
DKF	Distributed Kalman-Filter
MARG	Magnetic, Angular Rate, Gravity
AHRS	Attitude Heading Reference System
RTK	Realtime-Kinematics
DGPS	differential GPS
QZSS	Quasi Zenit Satelliten System
GLONASS	Globalnaja Nawigazionnaja Sputnikowaja Sistema
MEMS	Microelectromechanical System
GPL	Gnu General Public Lizenz
SAS	Stabilized Augmented Systems

KAPITEL 1

Einleitung

1.1 Ziel der Arbeit

Für **Fluglageregelung**, **Positionsregelung** sowie **Höhenregelung** des Flugroboters werden stabile und genaue Werte unterschiedlicher Größen wie **Fluglage**, **Flughöhe**, **Geschwindigkeit**, oder **Position** des Flugroboters benötigt.

Diese Größen können nicht immer direkt durch Sensormessungen in ausreichender Genauigkeit erhalten werden. Ziel der Arbeit ist es durch Sensordatenfusion aus fehlerbehafteten Messwerten verschiedener Sensoren präzise Werte jener benötigten Größen zu schätzen. Dabei werden Werte des globalen Navigationssatellitensystems (GNSS) sowie inertiale Größen wie Beschleunigung, Drehraten und weitere Größen wie barometrischer Höhendruck oder Magnetfeld herangezogen.

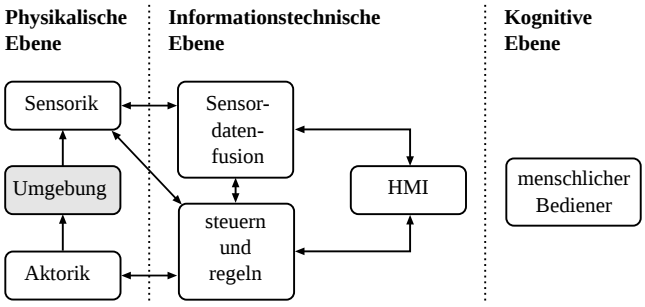

Abbildung 1.1: Rolle der Sensordatenfusion im Gesamtsystem [33]

Unter Sensordatenfusion versteht man die Kombination von Sensordaten mit dem Ziel die Qualität der erhaltenen Information im Vergleich zu Einzelmessungen zu verbessern. Dabei müssen die verwendeten Sensoren nicht zwingend die selben Größen liefern und auch keiner gemeinsamen Zeitbasis unterliegen. Es reicht, wenn Größen gemessen werden, welche in systematischem Zusammenhang stehen. Durch dieses Konzept können aus einer Fülle von mannigfaltigen Messungen eines Systems, Größen abgeleitet werden, die nicht direkt gemessen werden. Auch können, durch die Synergie von mehreren Sensorwerten und Systemverständnis, Messfehler und Ausreißer reduziert, sowie das Wissen über die Plausiblität der Größen verbessert werden. Die Rolle der Sensordatenfusion im Gesamtsystem wird in Abbildung 1.1 dargestellt. Zur genaueren Lektüre von Multi-Sensordatenfusion und Sensorvernetzung können [2] und [33] herangezogen werden.

Die Sensordatenfusion bedient sich vieler Werkzeuge unterschiedlicher Disziplinen, wie beispielsweise neuronaler Netze, Fuzzy-Logic, bayesscher Filter oder lernender Systeme. In diesem Rahmen kommen vor allem Konzepte wie Komplementär- oder Kalman-Filterung zur Anwendung.

1.2 Stand der Technik

1.2.1 Kalman-Filter

Das Kalman-Filter [23] ist bereits seit den 1960er Jahren bekannt und daher in zahlreicher Literatur gut dokumentiert. Ein Klassiker, ebenfalls aus den sechziger Jahren, ist die Arbeit [37], dessen Titel eine humoristische Anlehnung an Stanley Cubrick's satirischen Film über den Kalten Krieg und die nukleare Abschreckung - *Dr. Strangelove* - ist. Andere intuitive und anwendungsbezogene Einführungen zum Thema Kalman-Filter finden sich in [44] und [9]. Eine Heranführung an die Grundidee der Herleitung des Kalman-Filters wird in [10] geliefert. Für die Lektüre einer vollständigen Herleitung des Filters eignet sich [46] besonders gut. Auch [25] geht auf die Herleitung des Kalman-Filters ein und liefert eine umfassende Diskussion der Filteralgorithmen für Ingenieure. Für ein Grundverständnis der Anwendung des Kalman-Filters kann jedoch zunächst die Herleitung übersprungen werden und es genügt eine pragmatische Herangehensweise an das Thema wie sie in [44] und [9] vorliegt. Vertiefungen bezüglich numerischer Optimierung und Anwendung der Filteralgorithmen finden sich in [13], [45] und [10]. Hier werden anwendungsbezogene Themen, wie numerische Integrationsmethoden oder numerische Vereinfachungen der Kalman-Filteralgorithmen mit Berücksichtigung von Rechenaufwand, Speicherauslastung und numerischer Stabilität untersucht. Ein Studium dieser Werke lohnt sich daher sehr, vor allem wenn das Filter auf eingebetteten Systemen mit beschränkten Ressourcen Anwendung findet.

Bereits in den sechziger Jahren wurde mit dem extended Kalman-Filter (EKF) eine Lösung auch für nichtlineare Systeme gefunden, welches ebenfalls sehr anschaulich in [46] beschrieben wird. Die Grundidee ist die kontinuierliche Linearisierung um den aktuellen Punkt mithilfe einer Taylorreihe. Mitte der neunziger Jahre wurde mit dem Unscented Kalman-Filter (UKF) ein weiteres abgewandeltes Kalman-Filter für nichtlineare Systeme vorgestellt. Auch dieses findet sich in [46]

Auch für Sensornetzwerke gibt es Ansätze für Kalman-Filter. Wenn jeder Knoten in einem Sensornetzwerk einen lokalen Kalman-Filter ausführt und die jeweiligen Schätzungen von Knoten zu Knoten zur Weiterverarbeitung weitergereicht werden, spricht man vom sogenannten Decentralized Kalman-Filter [2]. Da diese Art von Kalman-Filter eine Kommunikation von allen Knoten mit allen anderen Knoten erfordert, ist sie nicht förderlich, wenn die Knoten batteriebetrieben sind, da eine drahtlose Kommunikation Leistung kostet [2]. Für diesen Zweck ist der sogenannte Distributed Kalman-Filter (DKF) besser geeignet, da dieser nur eine Kommunikation von Knoten zu Knoten benötigt. [2]

1.2.2 Fluglageschätzung

Eine inertiale Messeinheit (*Inertial Measurement Unit*, acsimu) besteht aus Drehraten- und Beschleunigungssensoren. Mit ihnen kann Rotation und Translation eines Körpers gemessen werden. Ein MARG (*Magnetic, Angular Rate, Gravity*) ist eine IMU, die zusätzlich einen Magnetfeldsensor beinhaltet. Damit kann die Orientierung zum Magnetfeld der Erde, sowie zum Erdschwerevektor bestimmt werden. Ein solches System zur Schätzung der Orientierung bezeichnet man als AHRS (*Attitude Heading Reference System*). Die wohl bekanntesten Ansätze sind jene von Madgwick [27], [26] und Mahony [28]. Beide Filter verwenden

die Quaternionendarstellung und leiden daher nicht unter Singularitäten, wie sie unter bestimmten Winkeln bei der Euler-Winkeldarstellung auftreteten. Diese Algorithmen zeichnen sich vor allem durch ihre Performanz, trotz verhältnismäßig wenigen Floating-Point-Berechnungen, aus. Sie können wahlweise mit einer IMU oder einem MARG-System betrieben werden und wurden in Bezug auf Floating-Point-Berechnungen zulasten des Speicherplatzes in C optimiert. Unter [51] werden die genannte C-Codes, neben weiteren Formulierungen in C# und Matlab, zum Download angeboten. Da sie auch bei niedrigen Samplingraten noch gute Schätzungen liefern, sind sie auch ohne weiteres auf schwachen 8-Bit Controllern noch lauffähig, wenn man nicht an die Grenzen des Speicherplatzes stößt.

Sehr häufig werden Ansätze zur Schätzung der Orientierung mit Kalman-Filtern formuliert. In [29] wird ein auf Quaternionen basierender Ansatz vorgestellt. Der Zustandsvektor setzt sich aus den Komponenten der Winkelgeschwindigkeit und dem Quaternion zur Darstellung der Orientierung zusammen. Es werden zwei alternative Messvektoren vorgestellt. Zunächst besteht der Messvektor aus den neun Messungen des MARG-Systems. Da diese Darstellung zu vielen Matrixberechnungen führt wird ein alternativer Messvektor vorgestellt. Dieser besteht aus den Komponenten der Winkelgeschwindigkeit und dem Quaternion welches optimal die Messungen von Magnetfeld und Beschleunigung in den Folgezustand überführt. Dieses Quaternion wird mithilfe des Gauss-Newton-Verfahrens ermittelt. Dies führt dazu, dass sich der Messvektor von neun Elementen auf sieben reduziert.

In [31] wird ein sogenannter Error Space State Kalman-Filter zur Fluglageschätzung formuliert. Hier werden im Zustandsvektor nicht die absoluten Werte, sondern die Fehler geschätzt. Im Speziellen werden die Fluglagewinkel und der Biasdrift des Drehratensensors geschätzt. In [42] werden kinematische Formeln für das Systemmodell verwendet. Auch hier wird der Drift des Drehratensensors mitgeschätzt.

Einfachere und damit weniger rechenintensive Algorithmen sind die in [41] und [36] vorgestellten Komplementärfilter. Diese Algorithmen sind jedoch nicht so robust gegen Fehler durch Beschleunigungen, die den Erdvektor verfälschen.

In [1] werden die Algorithmen von Magwick [27], [26] und Mahony [28] mit einem Ansatz mit extended Kalman-Filter experimentell verglichen. Es wird auch auf Genauigkeit und Laufzeit der Algorithmen eingegangen. In [3] wird der nichtlineare Komplementärfilter von Mahony [28] und der auf einer Optimierungsrechnung mithilfe des gradient descent Algorithmus basierende Filter von Madgwick [27], [26] verglichen. Für diesen Vergleich werden eine Simulation, sowie Messungen von realen Sensorwerten herangezogen. Es werden die Fehler der jeweiligen Algorithmen miteinander verglichen. Außerdem wird die Performanz der jeweiligen Filter bei unterschiedlichen Filterparametern verglichen.

1.2.3 Positionsschätzung

Am Fraunhofer-Institut in Stuttgart wurden 2004 in [19] Kalman-Filteralgorithmen zur rein inertialen Positionsschätzung für Indoor-Navigation, basierend auf Drehraten-, Beschleunigungs- und Magnetfeldsensoren, vorgestellt. Auch an der ETH-Zürich [8] sowie an der Universität von Pennsylvania [32] wurden Indoor-Navigationslösungen basierend auf einem kommerziellem Motion Capture System der Firma Vicon vorgestellt. Der Zweck dieser Lösung ist vor allem die Erforschung von Kontrollalgorithmen für Quadrocopter. In [52] wird ein Lokalisierungssystem, basierend auf Ultrabreitband-Funktechnik, untersucht, welches für Umgebungen ohne GPS-Empfang, wie beispielsweise in Gebäuden, eingesetzt werden soll.

Die Genauigkeit von Globalen Navigationssystemen (GNSS) allein reicht nicht aus um die Position des Quadrocotpers im Freien zu regeln. Deshalb wurden zahlreiche Lösungen vorgeschlagen, die Genauigkeit durch die Fusion von GNSS-Systemen und inertialer Messtechnik zu verbessern. Der Zustandsvektor des

Error Space State Kalman-Filter aus [31] wird bei Verfügbarkeit von GPS zur Positionsschätzung erweitert. Dieser Filter stammt ursprünglich aus [49]. Dies ist die Fortsetzung der Forschungen an einem direkten Kalman-Filteransatz aus [48], von Jan Wendel, dem Autor von [49]. In [34] wird ein direkter Kalman-Filter zur Positionsschätzung, basierend auf Quaternionen und der Lösung des Strapdownalgorithmus, vorgestellt. In dieser Arbeit wird ein sehr ähnlicher Ansatz verwendet.

Eine weitere Möglichkeit die Genauigkeit von GNSS-Systemen zu verbessern ist differential GPS (**DGPS**) in Verbindung mit Realtime-Kinematics (RTK). Hier werden durch ein ortsfestes Global Positioning System (GPS)-Modul, das seine Position kennt, sogenannte Pseudorange-Daten berechnet. Pseudorange-Daten beinhalten die Differenz der gemessenen Werte dieses Moduls zur bekannten Position, hervorgerufen durch den Ionosphären- und Stratosphärenfehler des GPS-Signals. Diese Daten können zur Korrektur des GPS-Signals verwendet werden. Außerdem wird das Rohsignal der Trägerphase des GPS-Signals ausgewertet, um die Genauigkeit zu verbessern (**RTK**).

Seit etwa einem Jahr ist ein kommerzielles GPS-Modul der Firma *Swift Navigation* lieferbar, welches **RTK**-Technologie bietet [54]. Im Gegensatz zu den industriellen Lösungen ist es leicht und preiswert. Die Module wurden in einem Kickstarter-Projekt entwickelt. Deren Software ist Open Source und deren Preis beträgt um die tausend Euro. Dies ist zwar immer noch teuer, aber wesentlich billiger als industrielle Lösungen.

Navigation durch RTK in Verbindung mit DGPS wurde im Rahmen der Applied Reseach Conference in [16] getestet. Hier hat sich gezeigt, dass mit dieser Methode eine ausreichende Genauigkeit für die Positionsregelung eines UAVs erreicht werden kann, jedoch haben sich auch einige Nachteile herausgestellt. Es kann unter Umständen mehrere Sekunden dauern, bis ein positionsgenaues Signal erhalten wird. Diese Wartezeit kann sich auch bei Verlust des GPS-Signales wiederholen. Außerdem wurden die Tests bei weitgehend wolkenlosem Wetter durchgeführt. Es sind weitere Tests bei schlechtem Wetter notwendig.

Aus diesem Grund wird in diesem Rahmen auf einen Kalman-Filter, als unabhängige Lösung, zurückgegriffen und differential GPS als Ergänzung in Situationen, wo eine absolut genaue Ortung notwendig ist.

1.3 Rahmenbedingungen

1.3.1 Sensoren

GNSS-Systeme: Mit einem GNSS-Empfänger lassen sich Signale von Satellitensystemen wie GPS, QZSS, GLONASS, sowie Galileo [1] empfangen. Die Genauigkeit liegt im Bereich von einigen Metern. GPS und Galileo sind im Bezug auf die Übertragungsfrequenz und Modulierungsverfahren weitgehend kompatibel, herkömmliche Empfänger können den sogenannten C/A-Code auslesen, welches auf der L1-Frequenz (1575,42 MHz) gesendet wird und das typische Zeitsignal enthält. Es gibt auch Empfänger, die zusätzlich das L2-Signal (1227,6 MHz) auswerten können. Das L2-Signal ist zwar nur für militärische Zwecke vorgesehen und daher mit dem sogenanntem Y-Code verschlüsselt. Durch spezielle Auswertealgorithmen, welche sich die Rohwerte der Trägerphasen der L1- und L2-Frequenz zunutze machen, können jedoch trotzdem die Ionosphärenfehler kompensiert und somit das Signal verbessert werden. Diese sogenannten Zweifrequenzempfänger benötigen jedoch Elektronik eines zusätzlichen Empfängers und sind damit schwerer, teurer und haben eine höhere Leistungsaufnahme [49]. In dieser Arbeit werden daher Algorithmen beschrieben, welche die Positionsinformation von herkömmlichen GPS-Empfängern verbessern.

[1] Bis 2016 sollen die ersten Galileo-Dienste zur Verfügung stehen, das gesamte Netz von 30 Satelliten soll bis 2020 fertiggestellt werden. Die Signale von Galileo sind kompatibel zu GPS. Es werden jedoch im Vergleich zu GPS noch weitere Dienste, wie etwa Echtzeit-Ortung von Notrufen. [7] angeboten.

Weiter Information zum Thema GNSS wurden im Rahmen einer Literaturrecherche in [17] zusammengetragen.

Beschleunigung: Der Beschleunigungssensor dient zur Berechnung von Geschwindigkeit und Position mithilfe des Strapdownalgorithmus. Die Güte aktueller MEMS-Beschleunigungssensoren hat sich in den letzten Jahren sehr verbessert. Die Bias-Fehler, wie sie in [49] beschrieben werden, sind nicht mehr besonders ausgeprägt und können vernachlässigt werden. Damit ist die einzige Fehlerquelle des Beschleunigungssensors sein Rauschen.

Drehraten: Mit dem Drehratensensor kann durch Integration der Winkelgeschwindigkeit auf die aktuellen Fluglagewinkel geschlossen werden. Sind diese bekannt, so kann mithilfe des Strapdownalgorithmus durch Integration der Beschleunigung auf Geschwindigkeit und Position zurückgeschlossen werden. Für gewöhnlich ist der Messwert der Drehraten jedoch mit einem konstanten Bias-Fehler behaftet. Das führt zu Integrationsfehlern in der Winkelberechnung und muss bei den folgenden Fusionsalgorithmen berücksichtigt werden.

Magnetfeld: Die Methode das Erdmagnetfeld für die Navigation zu nutzen ist bereits fast tausend Jahre alt und dessen Entdeckung hatte dramatische Auswirkung auf die Geschichte der modernen Zivilisation [5]. Die Feldlinien des Erdmagnetfeldes treten an der Südhalbkugel aus der Erde aus und am Nordpol wieder in die Erde ein. Dies wird in Abbildung 1.2 dargestellt. Die Inklination [2] beträgt hierzulande in etwa zwischen $62°$ und $70°$ [53]. Der Betrag der magnetischen Flussdichte variiert auf der Erde, je nach Ort, zwischen 22 μT und 67 μT [31].

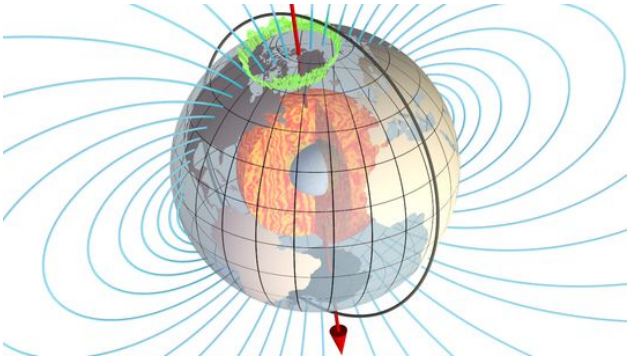

Abbildung 1.2: Magnetfeld der Erde [53]

Durch die Messung des Richtungsvektors des Magnetfeldes kann eine Orientierung des Flugroboters bezüglich der Erde bestimmt werden.

Barometrischer Höhendruck: Für die Schätzung der Höhe des Flugroboters steht ein barometrischer Höhensensor zur Verfügung. Über inertiale Sensorik allein lässt sich die Höhe nicht bestimmen. Die Positionsmessung des GNSS-Systems beinhaltet zwar ebenfalls die Höheninformation, diese ist jedoch dessen ungenauster Wert. Außerdem ist es sinnvoll auch bei Ausfällen der Satellitennavigation, in der Lage zu sein die Höheninformation zuverlässig zu stützen. Dies kann durch die Fusion der Drucksensorwerte mit den Werten von Beschleunigung und Drehraten gewährleistet werden.

[2] Inklination bezeichnet den Neigungswinkel zwischen Horizontalebene der Erde und dem Richtungsvektor des Erdmagnetfeldes [31]. Unter [53] finden sich Deklinationskarten.

1.3.2 Anforderungen

Ein Quadrocopter ist ein instabiles System, das nur durch eine permanente und zeitlich performante Regelung der Aktoren in Balance gehalten werden kann. Für diesen Zweck gibt es folgende Regelschleifen:

- **Fluglageregelung:** Aufgrund der naturgemäßen Instabilität des Systems werden in einer inneren, schnellen Regelschleife die Fluglagewinkel geregelt und so der Quadrocopter in Balance gehalten. Für die Regelgröße dieses Reglers werden die Fluglagewinkel benötigt.
- **Positions- sowie Höhenregelung und Navigation:** Flughöhen- und Positionsregelung werden in langsameren, die Fluglageregelung umschließenden Reglern realisiert. Für diese Regler werden Geschwindigkeit und Position benötigt.

Fluglageschätzung: Für die Fluglageregelung ist es notwendig qualitativ hochwertige Fluglagewerte in 6 Freiheitsgraden zur Verfügung zu haben. Die direkte Ermittlung der Fluglage aus den einzelnen Sensoren ist aufgrund deren Fehlercharakteristik nicht möglich. Beispielsweise würde eine Integration des Drehratensensors durch dessen Fehler-Bias schnell zu großen Driftfehlern führen. Auch Rückschlüsse auf den Lagewinkel des Flugroboters durch die Messung des Gravitationsvektors, führen ohne Mittelung nur im ruhenden Zustand zu wahren Ergebnissen. Um ausreichend genaue Werte zu erhalten, ist es deshalb notwendig verschieden Sensoren zu fusionieren. Algorithmen zur Fluglagebestimmung mithilfe der inertialen Messeinheit, bestehend aus Beschleunigungs- und Drehratensensor, werden in Kapitel 3.1 untersucht.

Positionsschätzung: Die Genauigkeiten von GNSS-Systemen wie GLONASS, Galileo oder GPS genügen, durch deren hohe Standardabweichung von mehreren Metern, nicht den Anforderungen einer Positionsbestimmung des Flugroboters. Auch die Lösung des Strapdownalgorithmus würde innerhalb kurzer Zeit zu großen Integrationsfehlern führen. Da die **GNSS-Systeme** langzeitgenaue Positionswerte und **Trägheitsnavigation** [3] kurzzeitgenaue Positionswerte liefern, müssen sie in geeigneter Weise klug fusioniert werden, dass sie über alle Zeitbereiche gute Werte liefern. Diese Kombination des GNSS-Systems mit Trägheitsnavigation bezeichnet man als Loosely Coupled. Zusätzlich wird diese Lageschätzung durch den Magnetfeld- und den barometrischen Höhendrucksensor gestützt. Hierfür wird in Kapitel 3.2 ein extended Kalman-Filter entworfen.

Als Tightly Coupled bezeichnet man die Stützung der GNSS-Werte durch Pseudorange-Daten. Da ein solches System jedoch erheblich mehr Integrationsaufwand mit sich bringt und rechenintensiver ist wird meistens ein Loosely-Coupled-System bevorzugt. [49], [31]

Eine weitere Möglichkeit genauere Positionswerte zu erhalten, ist es mithilfe einer stationären Referenzstation Pseudorange- und Trägerphasenmessungen vorzunehmen. Da die Position der Referenzstation bekannt ist, können systematische Fehler, wie Ionosphären- oder Stratosphärenfehler herausgerechnet werden. Dieses Verfahren wird als differential GPS (DGPS) bezeichnet. Als Referenzstationen können geostationäre Satelliten oder ein fest montiertes GPS-Modul, mit über längere Zeit gemitteltem GPS-Signal, dienen. Mit zweiterer Methode können Genauigkeiten im Dezimeter-Bereich erreicht werden. In [16] wurde getestet ob sich ein solches System zur Lokalisierung eines Flugroboters eignet. Es hat sich gezeigt, dass es, nach Verlust der Verbindung zu GPS-Satelliten, sehr lange dauern kann bis wieder ein stabiles Korrektursignal berechnet wird. Aus diesem Grund ist diese Technik für eine Positionsregelung nur bedingt geeignet. Vielmehr ist es ein Zusatzfeature, wenn eine positionsgenaue Landvermessung vorgenommen werden soll, oder wenn Punkte sehr exakt angeflogen werden sollen. Aus diesem Grund wird in diesem Rahmen auf ein Loosely Coupled System in Form eines extended Kalman-Filters gesetzt.

[3] Trägheitsnavigation (Inerital Navigation) ist die Schätzung von Orientierung und Position einzig durch inertiale Sensorik (Beschleunigung, Drehraten). Der aktuelle Wert wird basierend auf zurückliegender Werte berechnet (Dead-Reckoning).

KAPITEL 2

Theorie

2.1 Komplementärfilter

2.1.1 Hintergrund

Oft hat man Werte von unterschiedlichen Sensoren, eines zuverlässig in niedrigen und ein anderes in höheren Frequenzbereichen, welche für die Bestimmung einer einzelnen Größe geeignet sind. Um jene Größe zu schätzen bietet sich dann ein Komplementärfilter an.

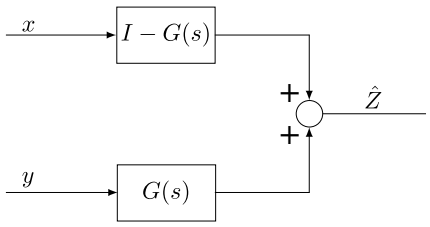

Abbildung 2.1: Grundprinzip Komplementärfilter

Beim Komplementärfilter werden die Sensorsignale mit einem Hochpass und einem Tiefpass entsprechend obiger Abbildung vereint. Die Sensorsignale, die in niedrigen Frequenzbereichen verlässlich sind werden durch einen Tiefpass und jene, welche in hohen Frequenzbereichen verlässlich sind durch einen Hochpass gefiltert.

2.1.2 Ansatz nach Colton

In [41] wird ohne Herleitung ein Filter zur Schätzung eines Winkels $\hat{\theta}$ aus der Messung der Winkelgeschwindigkeit ω_{gyro} und dem aus den Messungen eines Beschleunigungssensors berechneten Winkels θ_{acc} nach der Grundgleichung (2.1) vorgestellt.

$$\hat{\theta} = a \cdot (\hat{\theta} + \omega_{gyro} dt) + (1 - a) \cdot \theta_{acc} \tag{2.1}$$

Dabei lässt sich die Zeitkonstante τ für die Trennfrequenz des Hoch- und Tiefpassfilters aus der Konstanten a durch $\tau = \frac{a \cdot dt}{1-a}$ berechnen, wobei dt die Periodendauer der Frequenz, mit welche der Filter druchgeführt wird, darstellt.

2.1.3 Ansatz nach Oliviera/Pascoal/Kaminer

Im Folgenden wird ein Komplementärfilter anhand eines Beispiels beschrieben, wie es in [36] vorgestellt wird.

Sei ψ der Steuerkurs eines Fahrzeuges. Er soll durch die Sensorwerte eines Drehratensensors r_ω und eines Magnetfeldsensors ψ_ω gemessen werden. Daraus ergibt sich der Zusammenhang $r = \dot{\psi}$. Beide Messungen sind naturgemäß störungsbehaftet. Nimmt man die Laplace-Transformierten $\psi(s)$ und $r(s)$ von ψ und r kann man für $k > 0$ folgende Gleichung aufstellen [36]:

$$\psi(s) = \frac{s+k}{s+k}\psi(s) = T_1(s)\psi(s) + T_2(s)\psi(s) \tag{2.2}$$

Der Bruch aus Gleichung (2.2) lässt sich zerlegen zu $T_1(s) = k/(s+k)$ und $T_2(s) = s/(s+k)$. Außerdem muss die Gleichung $T_1(s) + T_2(s) = I$ erfüllt werden. Durch den Zusammenhang $r(s) = s\psi(s)$ kann man die Gleichung (2.2) umschreiben zu $\psi(s) = F_\psi\psi(s) + F_r(s)r(s)$, mit $F_\psi(s) = T_1(s) = k/(s+k)$ und $F_r(s) = 1/(s+k)$. Multipliziert man diese Gleichung aus kann sie auf den Term $s\psi(s)$ aufgelöst werden. Damit lässt sich der Filterterm nach Gleichung (2.3) im Zustandsraum darstellen [36]:

$$\dot{\hat{\psi}} = -k\hat{\psi} + k\psi_m + r_m = r_m + k(\psi_m - \hat{\psi}) \tag{2.3}$$

Diese Filterberechnung wird in folgender Abbildung grafisch dargestellt [36].

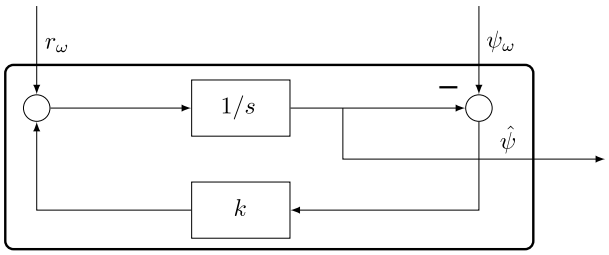

Abbildung 2.2: Komplementärfilter nach [36]

Zur Erläuterung sollen noch folgende Zusammenhänge genannt werden:

- $T_1(s)$ ist ein Tiefpassfilter, angewendet auf die Messungen des Magnetfeldsensors. Hochfrequenten Messungen wird hier nicht vertraut.
- $T_2(s)$ ist ein Hochpassfilter, angewendet auf die Werte des Drehratensensors. Er bildet den komplementären Bereich, in welchem der Magnetfeldsensor keine Werte liefert, ab.
- Der Bereich der Grenzfrequenz wird durch die Wahl des Parameters k gewählt

2.2 Kalman-Filter

2.2.1 Hintergrund

1960 stellte R.E. Kalman [1] seine berühmte Arbeit [23] über das Kalman-Filter, eine rekursive Lösung des diskreten linearen Filter Problemes, vor. Es gewann seither durch die Verbesserung der Rechenleistung immer mehr Anwendungen. Vor allem in der autonomen Navigation wurden viele Lösungen basierend auf dem Kalman-Filter vorgestellt. Die bekannteste Anwendung ist wohl das Apollo Programm der NASA in den sechziger Jahren, die Neil Armstrong als ersten Mann auf den Mond brachte. [14]

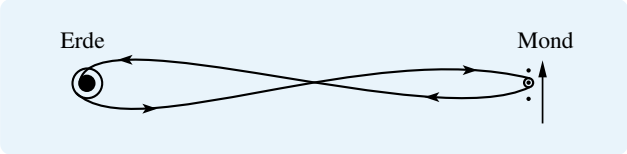

Abbildung 2.3: Nominale Appollo Trajektorien, skizziert ohne die Berücksichtigung der orbitalen Geschwindigkeit des Mondes um die Erde (1 km/h). Jene orbitale Geschwindigkeit des Mondes macht die Berechnung der Trajektorien noch komplizierter. [14]

Das Kalman-Filter ist eine Reihe von Gleichungen, welche auf effektive Weise rekursiv den Zustand eines Prozesses schätzt und dabei den mittleren quadratischen Fehler minimiert. Für eine erste Schätzung wird ein gewisses Systemverständnis verlangt. Diese erste Schätzung wird dann durch eine Messung korrigiert. Sowohl das Systemmodell, als auch das Messmodell werden als gestörte statistische Prozesse gesehen, welche mit System- und Messstörungen in Form von weißem gaußverteiltem Rauschen betroffen sind. *Das Kalman-Filter ist kein Filter im klassischem Sinne, welches ein Signal im Frequenzbereich beschneidet. Es schätzt für ein lineares, dynamisches Systemmodel mit stochastischer Erregung den zeitlich veränderlichen Zustand aus den gestörten Beobachtungen des Sytems.* [25]

Abbildung 2.4: R.E. Kalman bei der Überreichung des Charles Stark Draper Preises [40]

Der ursprüngliche Ansatz setzte ein lineares System- und Messmodell voraus. Im Zuge der Apollo Missionen wurde jedoch eine Version für nichtlineare Systeme, das extended Kalman-Filter (EKF), vorgestellt [30]. Es wurde durch den Vergleich mit Monte Carlo Simulationen gezeigt, dass der Schätzfehler einer Linearisierung des Problems mithilfe einer Taylorreihe klein genug ist um immer noch ausgezeichnete Ergebnisse zu liefern.

[1] R.E. Kalman gilt nicht nur als einer der einflussreichsten Forscher der Systemtheorie, sondern auch als einer der Wegbereiter dieses Forschungsgebietes. [4] Seine Leistungen wurden mehrmals durch Auszeichnungen honoriert, zuletzt 2008 mit dem Charles Stark Draper Preis, überreicht durch den amtierenden Präsidenten der USA, Barack Obama. Dieser Preis ist für die Ingenieurswissenschaften ähnlich bedeutend wie der Nobelpreis für die Naturwissenschaften. [40]

2.2.2 Gleichungen Kalman-Filter

Das zeitdiskrete Kalman-Filter setzt voraus, dass der Zustand $\mathbf{x_t}$ durch ein lineares zeitdiskretes Systemmodell nach Gleichung (2.4) zu einem Zeitpunkt t, für den aus dem vorhergehenden Zustand zum Zeitpunkt $t-1$ berechnet werden kann:

$$\mathbf{x_t} = \mathbf{F_t}\mathbf{x_{t-1}} + \mathbf{B_t}\mathbf{u_t} + \mathbf{w_t} \tag{2.4}$$

Wobei $\mathbf{x_t}$ den Zustandsvektor, $\mathbf{u_t}$ den Vektor für die Eingangswerte des Systems, $\mathbf{F_t}$ die Systemmatrix, $\mathbf{B_t}$ die Kontrollmatrix und $\mathbf{w_t}$ den Vektor, der das Systemrauschen enthält darstellen.

Außerdem muss durch eine Messung, dessen Werte im Messvektor $\mathbf{z_t}$ enthalten sind, mithilfe des Zustandsvektor $\mathbf{x_t}$ ein lineares System entsprechend Gleichung (2.5) herleitbar sein:

$$\mathbf{z_t} = \mathbf{H_t}\mathbf{x_t} + \mathbf{v_t} \tag{2.5}$$

Wobei $\mathbf{H_t}$ die Messmatrix und $\mathbf{v_t}$ das Messrauschen jeweils zum Zeitpunkt t darstellen.

Das Kalman-Filter arbeitet rekursiv in zwei Stufen. Zunächst wird mithilfe des Systemmodells der Zustandsvektor geschätzt. Anschließend wird die Schätzung durch die Erfassung der Messwerte korrigiert:

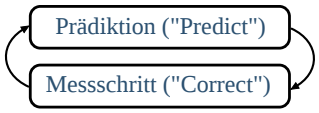

Abbildung 2.5: Ablauf Kalman-Filter [47]

Liegen keine Messungen vor, kann auch mehrmals der Prädiktionsschritt ausgeführt werden, solange bis wieder Messungen verfügbar sind. Folgende Abbildung zeigt jeweils die Wahrscheinlichkeitsdichtefunktionen des Systemmodells (a), des Messmodells (b) und schließlich der Schätzung des Kalman-Filters für einen Durchlauf von Prädiktion und Messschritt.

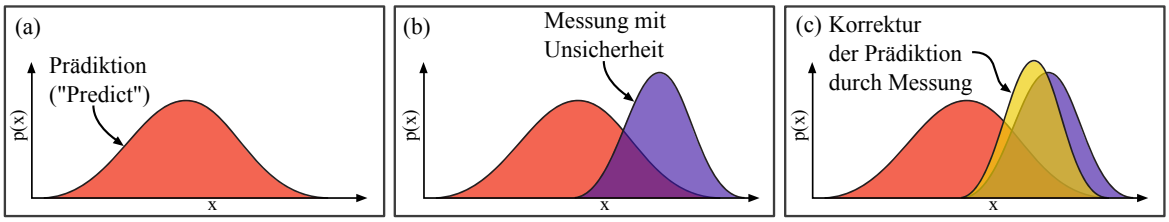

Abbildung 2.6: Ablauf des Kalman-Filters: (a) Schätzung (a priori) anhand des Systemmodells, (b) Erhebung der Messung, (c) Korrektur der Schätzung (a priori) durch die Messung (a posteriori) [46] [9]

Die Breite (Kovarianz) der Wahrscheinlichkeitsdichtefunktion kann als Maß der Vertrauenswürdigkeit des Wertes aufgefasst werden. In obiger Abbildung (c) sieht man, dass die Gaußsche Verteilung der Schätzung des Kalman-Filters sowohl schmaler ist als jene des propagierten und des gemessenen Wertes als auch, dass sie zwischen den beiden liegt.

Folgende Abbildung zeigt die Gleichungen des Kalman-Filteralgorithmus nach [9] im Detail [2]. Die Grundidee ist, dass der geschätzte Zustand durch eine Wahrscheinlichkeitsdichtefunktion dargestellt wird. Hierfür werden ein Vektor für den geschätzten Zustand $\hat{\mathbf{x}}_\mathbf{t}$ sowie eine Kovarianzmatrix $\mathbf{P_t}$ für den Schätzfehler benötigt.

Prädiktion ("Predict")

(1) $\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} + B_t u_t$

(2) $P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t$

Messschritt ("Correct")

(3) $K_t = P_{t|t-1} H_t^T (H_t P_{t|t-1} H_t^T + R_t)^{-1}$

(4) $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(z_t - H_t \hat{x}_{t|t-1})$

(5) $P_{t|t} = P_{t|t-1} - K_t H_t P_{t|t-1}$

Abbildung 2.7: Kalman-Filtergleichungen [9]

Prädiktion: Zunächst wird mithilfe des Systemmodells aus Gleichung (2.4) der aktuelle Zustand $\hat{\mathbf{x}}_\mathbf{t}$ des Filters über die Zeit propagiert **(1)** [3] [31]. Außerdem wird die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ rekursiv aus dessen vorhergehenden Werten, sowie der Kovarianzmatrix der Unsicherheit des Systemmodells $\mathbf{Q_t}$ bestimmt **(2)**.

Messschritt: Im Messschritt werden die Schätzung $\hat{\mathbf{x}}_{\mathbf{t}|\mathbf{t-1}}$, sowie die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ korrigiert. Hierfür wird die Messung $\mathbf{z_t}$ herangezogen. Zunächst wird zu diesem Zweck die sogenannte Kalman-Gain-Matrix $\mathbf{K_t}$ berechnet **(3)**. Sie bestimmt die Gewichtung inwieweit die Messung Einfluss auf den Systemzustand hat. Anschließend wird mithilfe jener Kalman-Gain-Matrix der Systemzustand durch die Messung $\mathbf{z_t}$ korrigiert **(4)**. Schließlich wird auch die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ korrigiert **(5)**. [31], [46]

Die Kovarianzmatrix der Unsicherheit des Systems $\mathbf{Q_t}$ beinhaltet das Vertrauen des Systems [9]. Sie berechnet sich aus dem Rauschvektor des Systems $\mathbf{w_t}$ nach Gleichung (2.6):

$$\mathbf{Q_t} = \mathbf{E}(\mathbf{w_t} \mathbf{w_t^T}) \tag{2.6}$$

Die Kovarianzmatrix der Messunsicherheit $\mathbf{R_t}$ beinhaltet das Vertrauen in die Messung [9]. Sie berechnet sich aus dem Rauschvektor der Messung $\mathbf{v_t}$ nach Gleichung (2.7):

$$\mathbf{R_t} = \mathbf{E}(\mathbf{v_t} \mathbf{v_t^T}) \tag{2.7}$$

Das Vertrauen in das Systemmodell $\mathbf{Q_t}$ sowie das Vertrauen in die Messung $\mathbf{R_t}$ können bei Bedarf zu jeder Zeit angepasst werden. Sind beispielsweise bei einer GPS-Messung wenig Satelliten vorhanden, kann das Vertrauen in die Messung so angepasst werden, dass den GPS-Messungen weniger Gewichtung zukommt, indem die Werte für die GPS-Messung im Rauschvektor $\mathbf{v_t}$ entsprechend geändert werden. Dadurch wird in diesem Fall der Einfluss des Systemmodells verstärkt. Sind wieder mehr Satelliten vorhanden, kann diese Änderung wieder rückgängig gemacht werden.

[2] In $\hat{\mathbf{x}}_{\mathbf{t-1|t-1}}$ steht das Dach exemplarisch für einen Schätzwert, der erste Index steht für den Schätzschritt und der zweite Index für den Updateschritt (a priori/a posteriori).

[3] Die fett gedruckten Nummern in Klammern beziehen sich auf die Nummerierung in Abbildung 2.7.

2.2.3 Gleichungen extended Kalman-Filter

Für die korrekte Funktion des Kalman-Filters ist es zwingend notwendig, dass das Systemmodell linear ist, denn nur dann resultiert die Beobachtung einer Gaußschen Zufallsvariable wieder in einer Gaußschen Zufallsvariable [46]. Das extended Kalman-Filter (EKF) ist die nichtlineare Version des Kalman-Filters. Beim EKF wird eine Linearisierung durch eine Taylorreihe, die nach dem erstem Glied abgeschnitten wird, vorgenommen.

Prädiktion ("Predict")

(1) $\hat{x}_{t|t-1} = g(u_t, x_{t-1|t-1})$

(2) $P_{t|t-1} = \Psi_t P_{t-1|t-1} \Psi_t^T + Q_t$

Messschritt ("Correct")

(3) $K_t = P_{t|t-1} \Gamma_t^T (\Gamma_t P_{t|t-1} \Gamma_t^T + R_t)^{-1}$

(4) $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (z_t - h(\hat{x}_{t|t-1}))$

(5) $P_{t|t} = P_{t|t-1} - K_t \Gamma_t P_{t|t-1}$

Abbildung 2.8: Extended Kalman-Filtergleichungen

Der Algorithmus des EKFs ähnelt sehr jenem des Kalman-Filters. Folgende Tabelle zeigt die Hauptunterschiede [46]:

	Kalman-Filter	EKF
Systemmodell Zeile **(1)**	$F_t \hat{x}_{t-1\|t-1} + B_t u_t$	$g(u_t, x_{t-1})$
Messmodell Zeile **(4)**	$H_t \hat{x}_{t\|t-1}$	$h(x_t)$

Die linearen Schätzungen des Kalman-Filters werden durch die nichtlinearen Funktionen des EKFs ersetzt. Außerdem werden im EKF statt der linearen Systemmatrizen $\mathbf{F_t}$, $\mathbf{B_t}$ und $\mathbf{H_t}$ die Jacobi-Matrizen $\mathbf{\Phi_t}$ und $\mathbf{\Gamma_t}$ verwendet. Dabei korrespondiert $\mathbf{\Phi_t}$ mit $\mathbf{F_t}$ und $\mathbf{B_t}$ und $\mathbf{\Gamma_t}$ korrespondiert mit $\mathbf{H_t}$. [46]

2.2.4 Rechenaufwand und Präzision

Rechenaufwand

Der größte Rechenaufwand des Filters liegt in der Inversion der Matrix S aus Gleichung **(3)** des Kalman-Filteralgorithmus aus Abbildung 2.7:

$$P = PH^T (\overbrace{HPH^T + R}^{S})^{-1}$$

Hier muss eine $r \times r$-Matrix invertiert werden, wobei r der Anzahl der Elemente des Messvektors z entspricht. Da man zeigen kann, dass die Matrix S immer symmetrisch positiv definit ist lässt sich deren Inverse durch das Lösen folgender Gleichung mithilfe des Cholewsky-Verfahrens finden:

$$SX = I \text{ mit der Lösung } X = S^{-1}$$

Diese Methode ist numerisch stabil und ihr Rechenaufwand beträgt $O(^1/_3 r^3)$ [38]. [4] Neben dem Cholewsky-Verfahren gibt es schnellere Methoden die Matrixinversion durchzuführen. Laut [46] basieren die derzeit schnellsten Inversionsalgorithmen auf den schnellen Matrixmultiplikationen von Coppersmith und Winograd [6] und weisen damit einem Rechenaufwand von $O(r^{2,4})$ auf. In diesem Rahmen wurde jedoch nicht geprüft, ob diese Verfahren praxistauglich sind.

Eine Möglichkeit den Rechenaufwand zu reduzieren ist es die Inversion jener Matrix massiv zu vereinfachen, indem man Messungen nacheinander und einzeln verarbeitet (Sequential Filtering) [45]. Damit reduziert sich jene Inversion einer $r \times r$-Matrix zur Inversion einer 1×1-Matrix und damit zu einer schlichten skalaren Division. Der Rechenaufwand ist dann nur noch proportional zu r.

Wenn die Matrizen R und Q, sowie das zugrunde liegende Systemmodell zeitinvariant sind, kann es vorkommen, dass die Kovarianzmatrix des Schätzfehlers P und damit auch die Kalman-Gain-Matrix K nach einigen Filterdurchläufen zu konstanten Werten konvergieren. Dann besteht die Möglichkeit diese vorher zu berechnen. Am einfachsten geht dies indem man den Kalman-Filteralgorithmus offline für eine bestimmte Zahl von Durchgängen durchlaufen lässt. Dadurch kann man jene Rechenschritte im Filteralgorithmus, die zu deren Berechnung dienen, einsparen. Man spricht von Steady-State Filtering [45].

Präzision

Auf eingebetteten Systemen mit geringer Wortbreite kann es unter bestimmten Umständen dazu kommen, dass die Kovarianzmatrix aus der Riccati-Gleichung [5] eine schlechte Konditionszahl erhält und damit die Präzision nicht mehr ausreichend ist. Dieses Problem trat erstmalig auf dem Apollo Guidance Computer, konstruiert zur Berechnung der Trajektorie von der Erde zum Mond, auf.

Abbildung 2.9: Apollo Guidance Computer (15-bit Fixpoint-Arithmetik) und Display/Keyboard-Einheit [50]

Da sich Änderungen häufig nur in den letzten Nachkommastellen niederschlagen, wird für das Kalman-Filter prinzipiell Auflösung in der Arithmetik benötigt [6]. In [45] werden zwei Alternativen zu den Riccati-Gleichungen vorgestellt, die auf Kosten des Rechenaufwandes die Genauigkeit erhöhen:

- Square-Root Filtering [7]

[4] Um numerische Matrix-Operationen in C/C++ durchzuführen lohnt es sich einen Blick auf die GNU Scientific Library (GSL) [12] zu werfen. Beispielsweise wird eine Funktion für das Cholewski-Verfahren angeboten.

[5] Die Gleichungen im Kalman-Filter, die zur Berechnung der Kovarianzmatrix führen werden zusammen Riccati-Gleichung genannt [45]

[6] Aussage von Wolfgang Högele

[7] Diese Methode fand auf dem 15-Bit Appollo Guidance Computer Anwendung. Der Appollo Guidance Computer wurde konstruiert um die Trajektorie von der Erde zum Mond zu berechnen. Eine schlechte Kondition der Kovarianzmatrix führte auf der 15-Bit Maschine jedoch zu Problemen. Die Lösung fand der MIT Student J.E. Potter, der als Student Teilzeit am Instrumentation Laboratory der NASA in Mountain View arbeitete. Der Trick war, mithilfe des Cholewski-Verfahrens die Kovarianzmatrix zu

- U-D Filtering

Diese beiden Methoden erhöhen die Präzision sowie die Stabilität des Kalman-Filters, auf Kosten des Rechenaufwandes. Da jene Methoden in diesem Rahmen keine Anwendung finden, wird hier lediglich auf deren Existenz hingewiesen. Der geneigte Leser sei bei näherem Interesse auf die Lektüre von [45] verwiesen.

2.3 Koordinatentransformation

2.3.1 Koordinatensysteme für Erdnahe Navigation

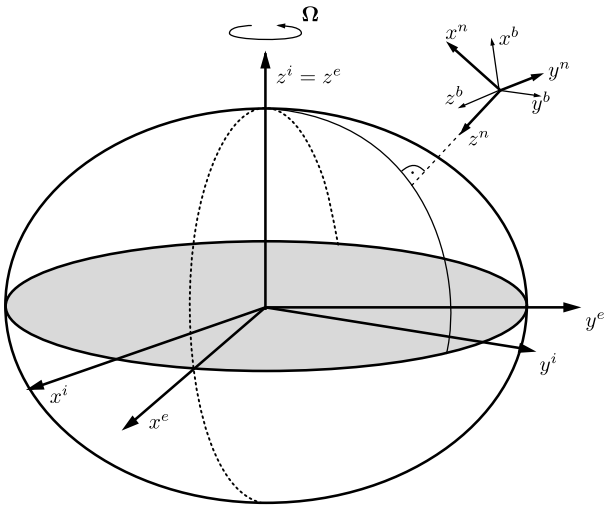

Abbildung 2.10: Verwendete Koordinatensysteme [49]

In Abbildung 2.10 werden die hier verwendeten Koordinatensysteme gezeigt:

- **b-frame:** Das **körperfeste Koordinatensystemem** ist fest mit dem Luftfahrzeug verbunden. Dabei entspricht die Ausrichtung der Achsen jenen aus Abbildung 2.12. Markant ist hier, dass die z-Achse nach unten weist. Der Ursprung befindet sich im Luftfahrzeug. Die Komponenten der Messwerte der IMU werden in diesem Koordinatensystem angegeben. [49]
- **i-frame:** Das **inertiale Koordinatensystem** ist auf Fixsterne ausgerichtet und damit ein ruhendes Koordinatensystem. Dessen z^i-Achse fällt mit der Rotationsachse der Erde zusammen. Die Achsen x^i und y^i befinden sich in der Äquatorebene. Die Messwerte der IMU sind Drehraten und Beschleunigung des körperfesten Koordinatensystems bezüglich des inertialen Koordinatensystems. [49]
- **e-frame:** Der Ursprung und die z^e-Achse des **erdfesten Koordinatensystems** fallen mit jenem des Inertialkoordinatensystems zusammen. Das **erdfeste Koordinatensystem** ist fest mit der Erde verbunden. Es dreht sich, mit der Erdrotationsgeschwindigkeit Ω, mit der Erde mit. Die Achsen x^e und y^e befinden sich in der Äquatorebene. [49]
- **n-frame:** Der Ursprung des **Navigationskoordinatensystems** fällt mit jenem des körperfesten Koordinatensystems zusammen. Die x^n-Achse weist in Nord- und die y^n-Achse weist in Ostrichtung.

zerlegen. Man bezeichnete diese Methode als *Square-Root Filtering*. [14]

Beide liegen tangential zum Erdellipsoid. Die z^n-Achse weist nach unten. Sie enthält den Gravitationsvektor. [49]

Damit die Ausrichtung von Nord, Ost und unten bezüglich des Navigationskoordinatensystems bei einer Drehung des Erdkoordinatensystems erhalten bleibt, muss dieses ständig nachgedreht werden. Die resultierende Drehrate wird als Transportdrehrate bezeichnet.

2.3.2 Nomenklatur

Die Komponenten jedes Vektors haben ihre Gültigkeit in den Koordinatensystemen aus Kapitel *Koordinatensysteme für Erdnahe Navigation*. Auch gelten für jeden Vektor Beziehungen zu den anderen Koordinatensystemen. Die Schreibweise des Vektors ist deshalb so gestaltet, dass dies erkennbar wird. Am Beispiel des Geschwindigkeitsvektors soll exemplarisch die hier verwendete Nomenklatur beschrieben werden.

$$\vec{v}_{eb}^{\,n}$$

Der obere Index gibt an in welchem Koordinatensystem die Größe gegeben ist. In diesem Fall handelt es sich um die Geschwindigkeit im Navigationskoordinatensystem. Die beiden unteren Indizes geben die Beziehungen der Koordinatensysteme untereinander an. Hier ist beispielsweise die Geschwindigkeit des körperfesten Koordinatensystems bezüglich des erdfesten Koordinatensystems angegeben.

Mit Rotationsmatrizen kann ein Vektor aus einem Koordinatensystem durch Rotation in ein anderes Koordinatensystems übergeführt werden. Dabei wird ein Vektor vom Koordinatensystem des unteren Index zum Koordinatensystem des oberen Index gedreht.

$$C_b^n$$

Im obigen Beispiel handelt es sich um die Rotationsmatrix vom körperfesten Koordinatensystem in das Navigationskoordinatensystem. Diese kann beispielsweise dazu verwendet werden die Beschleunigung $\vec{a}_{ib}^{\,n}$ im Navigationskoordinatensystem aus der Beschleunigung $\vec{a}_{ib}^{\,b}$ aus dem körperfesten Koordinatensystems zu berechnen. Da die Sensoren fest mit dem körperfesten Koordinatensystem verbunden sind, kann man die Komponenten von $\vec{a}_{ib}^{\,b}$ den Werten des Beschleunigungssensors entnehmen. Damit ergibt sich Gleichung (2.8).

$$\vec{a}_{ib}^{\,n} = C_b^n \vec{a}_{ib}^{\,b} \tag{2.8}$$

Die Rotationsmatrizen werden im Detail in Kapitel *Rotation* beschrieben.

2.3.3 Rotation

In den vorgestellten Algorithmen ist es häufig notwendig Vektoren eines Koordinatensystems in ein Anderes zu überführen. Hierfür werden Vektorrotationen verwendet. Folgend werden knapp jene Formeln beschrieben, die zum Verständnis dieser Algorithmen benötigt werden. Eine Einführung in dieses Thema ist ebenfalls in [49], [11] zu finden.

Richtungskosinusmatrix

Die Fluglage kann anschaulich durch die Eulerwinkel Roll θ, Pitch ψ und Yaw ϕ beschrieben werden. Die räumliche Zuordnung der Eulerwinkel in einem Luftfahrzeug wird in Abbildung 2.12 dargestellt. Sie geben

die Winkellage zwischen körperfestem Koordinatensytem des Fluggerätes und Navigationskoordinatensystem an. Eine hintereinander ausgeführte und nicht kommutative Drehung eines Vektors um jene Eulerwinkel kann durch Multiplikation mit der Richtungskosinusmatrix aus Gleichung (2.9) durchgeführt werden. Dabei wird in diesem Fall zunächst um die z-Achse (Yaw, ϕ), anschließend um die y-Achse (Pitch, ψ) und zuletzt um die x-Achse (Roll, θ) gedreht. [31] Eine räumliche Darstellung dieser Drehungen findet sich in Abbildung 2.11.

$$C_b^n = \begin{pmatrix} \cos(\theta)\cos(\psi) & -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\phi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -\sin(\theta) & \sin(\phi)cos(\theta) & \cos(\phi)\cos(\theta) \end{pmatrix} \tag{2.9}$$

Mittels der Richtungskosinusmatrix kann entsprechend Gleichung (2.10) auch die zeitliche Änderung der Eulerwinkel angegeben werden [11].

$$R = \begin{pmatrix} \dot{\theta} \\ \dot{\psi} \\ \dot{\phi} \end{pmatrix}_{nb}^{b} = \begin{pmatrix} (\omega_{nb,y}^b \sin(\theta) + \omega_{nb,x}^b \cos(\theta)) \tan(\psi) + \omega_{nb}^b \\ \omega_{nb}^b \cos(\theta) - \omega_{nb,z}^b \sin(\theta) \\ \omega_{nb,y}^b \sin(\theta) + \omega_{nb,z}^b \cos(\theta) / \cos(\psi) + \omega_{nb,x}^b \end{pmatrix} \tag{2.10}$$

In Gleichung (2.10) sieht man, dass bei einem Pitch-Winkel von $\pm\psi = 90°$ durch eine Division durch Null eine Singularität auftritt. Räumlich gesehen entspricht dies dem Aufeinanderfallen von Drehachsen und damit dem Ausfall von Freiheitsgraden in der Drehbewegung. Man bezeichnet dieses Problem der Richtungskosinusmatrix als Gimbal-Lock.

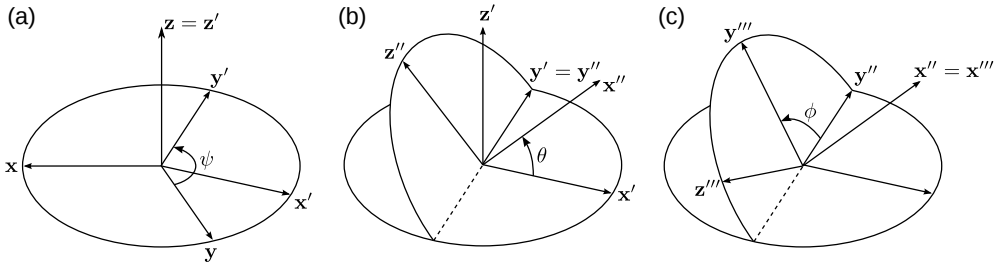

Abbildung 2.11: ZYX-Drehung **(a)** Yaw ϕ um z-Achse **(b)** Pitch ψ um y-Achse **(C)** Roll θ um x-Achse

Links zu Animationen und weiterführenden Informationen, sowie Code zur Veranschaulichung dieser Zusammenhänge finden sich unter [15].

Quaternionen

Die Lage zweier Koordinatensysteme kann mithilfe des Orientierungsvektors $\vec{\sigma}$ beschrieben werden:

$$\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)^T \tag{2.11}$$

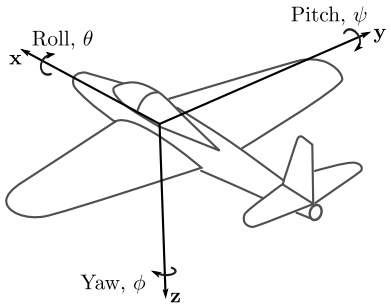

Abbildung 2.12: Roll-, Pitch- und Yaw-Winkel am Luftfahrzeug

Ein Quaternion [8] beschreibt eine Rotation durch eine einzige Drehung. Dies geschieht durch eine Quaternionenmultiplikation, welche näher in Gleichung (2.13) beschrieben wird. Es können auch mehrere Drehungen durch die Aneinanderreihung von Quaternionenmultiplikation durchgeführt werden. Dabei tritt das Problem der Mehrdeutigkeit der Eulerwinkel nicht auf.

Ein Quaternion ist ein vierdimensionaler Vektor [9], dessen Betrag auf eins normiert ist, wie in Gleichung (2.12) dargestellt.

$$q_b^n = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\sigma/2) \\ \sigma_x/\sigma \ \sin(\sigma/2) \\ \sigma_y/\sigma \ \sin(\sigma/2) \\ \sigma_z/\sigma \ \sin(\sigma/2) \end{pmatrix} \tag{2.12}$$

Es kann durch Quaternionenmultiplikation ein Vektor eines Koordinatensystems in ein anderes überführt werden. [10] Hierfür muss der Vektor mit einer führenden Null erweitert werden. Dies ist entsprechend [11] in Gleichung (2.13) dargestellt.

$$\begin{pmatrix} 0 \\ x^n \end{pmatrix} = q_b^n \bullet \begin{pmatrix} 0 \\ x^b \end{pmatrix} \bullet q_n^b$$
$$\text{mit } \ q_b^n \bullet (\ldots) = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \cdot (\ldots) \tag{2.13}$$

Alternativ kann nach [49] aus der Richtungskosinusmatrix die sogenannte Euler-Rodriguez-Darstellung, welche das Navigationskoordinatensystem in das körperfeste Koordinatensystem überführt, [43] berechnet werden. Sie ist in Gleichung (2.14) [11] dargestellt.

$$C_b^n = \begin{pmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{pmatrix} \tag{2.14}$$

[8] In den 1950er Jahren führte die Angst zwischen den Großmächten USA und Russland und weiteren Nationen zu einem Wettrüsten. Dies löste einen Technologieboost im Bereich Luft- und Raumfahrt aus. In diesem Zuge wurden für Anwendungen, wie beispielsweise Fernlenkraketen, die von Hamilton 1844 vorgestellten Quaternionen [20] für die Beschreibung der Orientierung eines Körpers wiederentdeckt. (Es sei an dieser Stelle vom Autor angemerkt, dass zivile Programme, wie die Internationale Raumstation ISS beweisen, dass nicht nur Krieg und Misstrauen, sondern Forscherdrang, Neugier und internationale Zusammenarbeit zu herausragenden technischen Leistungen, ebenfalls im zivilem Bereich, führen können.)

[9] Zur Verbesserung der Lesbarkeit wird bei der Darstellung des Quaternionenvektors $\vec{q}$ auf den Vektorpfeil verzichtet.

[10] Für die Quaternionenmultiplikation wird das Symbol $\bullet$ verwendet.

[11] Die Erfüllbarkeit dieser Gleichung setzt voraus, dass das Quaternion in normalisierter Form $q_{norm} = \frac{q}{|q|}$ vorliegt [29].

In Gleichung (2.15) wird gezeigt wie eine Hintereinanderschaltung von Drehungen vollzogen wird. Diese Operation ist nicht kommutativ [11].

$$q_b^n = q_e^n \bullet q_b^e \tag{2.15}$$

Die Quaternionen-Differentialgleichung (2.16) stellt die zeitliche Änderung eines Quaternions aufgrund von Drehraten dar [11]. Eine Herleitung für diese Gleichung liegt in [22] vor.

$$\dot{q_b^n} = \frac{1}{2} q_b^n \bullet \begin{pmatrix} 0 \\ \omega_{nb}^b \end{pmatrix} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \omega_{nb}^b \end{pmatrix} \tag{2.16}$$

Nach Gleichung (2.17) können Quaternionen in Euler-Winkel umgewandelt werden [11].

$$\begin{pmatrix} \theta \\ \psi \\ \phi \end{pmatrix} = \begin{pmatrix} \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan(\frac{2(q_0 q_3 + q_1 q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}) \\ \arctan(\frac{2(q_0 q_1 + q_2 q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}) \end{pmatrix} \tag{2.17}$$

2.3.4 Strapdown-Algorithmus

Unter Strapdownalgorithmus versteht man die Propagation von Lage, Geschwindigkeit und Position durch die Integration der Werte des Drehratensensors und des Beschleunigungssensors. Da hier sowohl Coriolis-kräfte, welche aufgrund der Erdrotation auftreten, als auch Erd- [12] und Transportdrehrate [13] vernachlässigt werden können, wird hier eine vereinfachte Form des Strapdownalgorithmus vorgestellt, dargestellt in Abbildung 2.13. Eine detailliertere Beschreibung kann in [49] nachgelesen werden.

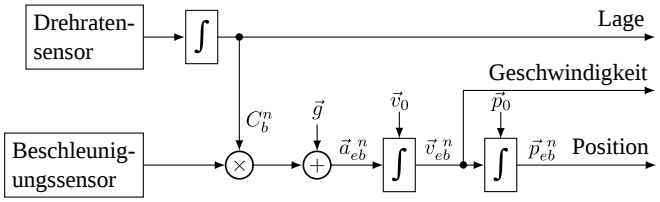

Abbildung 2.13: Vereinfachter Strapdown-Algorithmus [31]

Der Vektor der Drehrate ω_{nb}^b im Navigationskoordinatensystem kann nach Gleichung (2.18) aus dem körperfesten Koordinatensystem berechnet werden. Dabei werden Transportdrehrate und Erddrehrate berücksichtigt [31].

$$\vec{\omega}_{nb}^b = \vec{\omega}_{ib}^b - \underbrace{\vec{\omega}_{ie}^b}_{\substack{\text{Erddreh-} \\ \text{rate}}} - \underbrace{\vec{\omega}_{en}^b}_{\substack{\text{Transport-} \\ \text{drehrate}}} \tag{2.18}$$

[12] Die Erde dreht sich in 24 Stunden einmal um ihre Achse. Dies entspricht einer Winkelgeschwindigkeit von $|\vec{\omega}_{ie}^b| = 7,292115 \cdot 10^{-5}\ rad/s$. Dieser Drehrate ist auch der Drehratensensor ausgesetzt.

[13] Aufgrund der Erdrotation und der Erdkrümmung muss sich bei der Zurücklegung von weiten Strecken das Navigationskoordinatensystem in einer Weise drehen, dass dessen z-Achse immer in Richtung der lokalen Vertikale weist. Dies wird in der Transportdrehrate berücksichtigt [49].

Da die Geschwindigkeit in der hier gestellten Anforderung gering ist und nur geringe Wege zurückgelegt werden, fällt die Transportdrehrate nicht ins Gewicht. Auch die Erddrehrate kann im Vergleich zum Fehler des Drehratensensors vernachlässigt werden. Unter Berücksichtigung dieser Vereinfachungen ergeben sich die Differentialgleichungen für Lage, Geschwindigkeit und Position aus den gemessenen Werten von Drehrate und Beschleunigung der IMU nach Gleichung (2.19).

$$\begin{aligned} \dot{\vec{\sigma}} &= \vec{\omega}_{ib}^{\,b} \\ \dot{\vec{v}}_{eb}^{\,n} &= C_n^b \vec{a}_{ib}^{\,b} + \vec{g}_l^n \\ \dot{\vec{p}}_{eb}^{\,n} &= \vec{v}_{eb}^{\,n} \end{aligned} \tag{2.19}$$

mit $\vec{g}_l^n = (0, 0, g_0)^T$. Dabei kann der Betrag von g_0 vereinfacht zu $g_0 = 9,80665 m/s^2$ angenommen werden. Aufgrund der Fehlerbeiträge des Beschleunigungssensors und der geringen zurückgelegten Wegstrecken kann hier auf ein genaueres Gravitationsmodell, wie beispielsweise *WGS84*, verzichtet werden. [31]

KAPITEL 3

Sensorfusionsalgorithmen

3.1 Fluglageschätzung

3.1.1 Komplementärfilter

Ansatz nach Madgwick/Mahony

Madgwick hat im Rahmen seiner Dissertation einen AHRS (Attitude Heading Reference System) Algorithmus zur Lagebestimmung entwickelt. [26] [27] Auch Mahony hat einen Fusionsalgorithmus vorgestellt. [28] Die Algorithmen fusionieren jeweils 3-Achsen Beschleunigungs- sowie Drehraten- und wahlweise auch Magnetfeldsensoren. Nimmt man den Magnetfeldsensor mit hinzu werden die Algorithmen rechenintensiver, aber auch stabiler. Außerdem ist dann die Stützung des Yaw-Winkels zuverlässig.

Sowohl Madgwick als auch Mahony haben Code für ihre Algorithmen zur Verfügung gestellt. Dieser Code wird auf [51] unter der *Gnu General License* (GPL) für Matlab, C# und C zur Verfügung gestellt. Auf [21] ist auch eine Labview Version vorhanden.

Eine einfache Abwandlung des von Mahony vorgestellten Codes [51], welche auf Rechenleistung getrimmt wurde befindet sich im Anhang in Kapitel *Code Ansatz Mahony*.

Ansatz nach Oliviera/Pascoal/Kaminer

Für die Schätzung der Fluglage kann das in Kapitel *Ansatz nach Oliviera/Pascoal/Kaminer* beschriebene Filter herangezogen werden. So kann die Winkelgeschwindigkeit ω des Lagewinkels θ nach Gleichung (3.1) geschätzt werden [24]. Dabei sind ω_{gyro} die Messung der Winkelgeschwindigkeit durch den Drehratensensor und ω_{acc} die Schätzung der Winkelgeschwindigkeit durch den Beschleunigungssensor.

$$\dot{\hat{\theta}} = \omega_{gyro} + k_p(\theta_{acc} - \hat{\theta}) \tag{3.1}$$

Da jedoch nach Gleichung (3.1) der Bias des Drehratensensors nicht berücksichtigt wird, wird sie so erweitert, dass dieser kompensiert wird. [24]

$$\begin{aligned} \dot{\hat{\theta}} &= \omega_{gyro} + k_p(\theta_{acc} - \hat{\theta}) \\ \dot{\hat{b}} &= -k_I(\theta_{acc} - \hat{\theta}) \end{aligned} \tag{3.2}$$

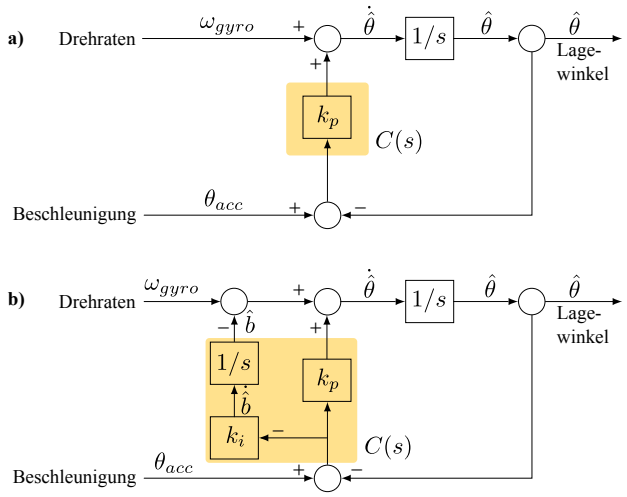

Abbildung 3.1: Umsetzung des Komplementärfilters nach [36] **a)** ohne Driftkompensation und **b)** mit Driftkompensation des Drehratensensors

Folgende Grafik zeigt das Blockdiagramm des Filters sowohl für Gleichung (3.1) als auch (3.2).

Der Code für die Umsetzung dieses Filters, exemplarisch für den Roll-Winkel, befindet sich im Anhang in Kapitel *Ansatz nach Oliviera/Pascoal/Kaminer*.

Ansatz nach Colton

In Kapitel *Ansatz nach Colton* wird ein Filter beschrieben, wie es von Shane Colton in [41] dargelegt wird. Folgender Code zeigt dieses Filter exemplarisch für den Roll-Winkel.

```
#define DELTA_T     0.002 // Cycle Time 2 ms
#define FACTOR_A    0.98

void Complement2Update(float ax, float ay, float az, float gx, float gy, float gz)
{

    float y_gyro_roll = gx;
    float y_acc_roll = atan2(ay,az);

    y_acc_roll = y_acc_roll/PI * 180.0;

    roll = FACTOR_A*(roll+y_gyro_roll*0.005) + (1-FACTOR_A)*y_acc_roll;
}
```

Dabei lässt sich entsprechend den Formeln aus Kapitel *Ansatz nach Colton* die Zeitkonstante τ folgendermaßen berechnen:

$$\tau = \frac{a \cdot dt}{1-a} = \frac{0.98 \cdot 0.002\ sec}{0.02} \approx 0.1\ sec$$

Für Zeitperioden unter $0.1\ sec$ überwiegt der Einfluss der Winkelwerte, berechnet durch die Integration des Drehratensensors, und jene des Beschleunigungssensors werden weggefiltert. Für Zeitperioden über $0.l\ sec$ verhält es sich umgekehrt und den Werten des Beschleunigungssensors kommt mehr Gewichtung zu.

3.1.2 Ansatz mit Kalman-Filter

Für die Fluglageschätzung wurde basierend auf einem Kalman-Filter ein ähnlicher Ansatz wie in [42] gefunden. Im Prädiktionsschritt wird der Lagewinkel θ durch Integration des Drehratensensors propagiert. Da die Integration durch den Bias-Fehler des Drehratensensors mit einem Drift behaftet ist, würde dies dazu führen, dass das Rauschverhalten des Systemmodells nicht rein normalverteilt ist. Abhilfe kann dadurch geschaffen werden, dass der Drift b_θ mitgeschätzt wird und damit dessen Eigenschaften im Modellverhalten enthalten sind. Dadurch wird der Drift des Drehratensensors automatisch korrigiert. Der Fluglagewinkel θ lässt sich durch Gleichung (3.3) beschreiben.

$$\theta_t = \theta_{t-1} - (\omega_{gyro} - b_\theta)dt \tag{3.3}$$

Aus dieser Gleichung lässt sich mit dem Zustandsvektor $x_t = \begin{bmatrix}\theta & b_\theta\end{bmatrix}^{\mathrm{T}}$ das lineare System (3.4) ableiten.

$$\begin{bmatrix}\theta \\ b_\theta\end{bmatrix}_t = \begin{bmatrix}1 & -dt \\ 0 & 1\end{bmatrix}\begin{bmatrix}\theta \\ b_\theta\end{bmatrix}_{t-1} + \begin{bmatrix}dt \\ 0\end{bmatrix}\omega_{t-1} \tag{3.4}$$

Als Messung z wird der Lagewinkel, berechnet durch die Komponenten des Beschleunigungssensors, herangezogen. Der Messschritt führt in diesem Ansatz dazu, dass der Drift des Drehratensensors korrigiert wird. Damit ergibt sich das Messmodell nach Gleichung (3.5).

$$z_t = \begin{bmatrix}1 & 0\end{bmatrix} x_t \tag{3.5}$$

Im Unterschied zum Ansatz nach [42] wird hier der Messschritt nicht in jedem Filterdurchgang ausgeführt. Während im Prädiktionsschritt alle *2 ms* propagiert wird, werden die Werte des Beschleunigungssensors über *200 ms* gemittelt. Erst nach dieser Zeit wird aus diesen gemittelten Komponenten des Beschleunigungssensors der Lagewinkel mithilfe der atan2-Funktion für die Messung berechnet und der durch das Systemmodell propagierte Schätzwert mit dem Messmodell korrigiert.

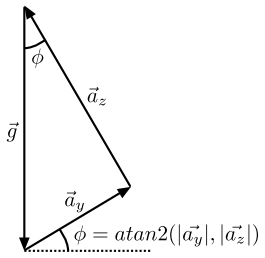

Abbildung 3.2: Trigonometrie des Beschleunigungssensors exemplarisch für den Roll-Winkel.

Da einzig eine Messung herangezogen wird, entfällt die Matrixinversion des Filteralgorithmus und wird zu einer skalaren Division. Dadurch ist die Anzahl der Floating-Point-Berechnungen gering. Noch mehr Floating-Point-Berechnungen können durch die Vorausberechnung der Kalman-Gain gespart werden. Wie in Kapitel *Rechenaufwand und Präzision* beschrieben, kann es dazu kommen, dass diese konvergiert, wenn die Rauschvektoren von System- und Messmodell konstant bleiben [45]. Dies ist hier der Fall. Deshalb kann im hier vorgestellten Ansatz durch die Wahl der Update-Funktion gewählt werden, ob man das Filter herkömmlich berechnet, oder eine vorausberechnete Kalman-Gain verwendet werden soll:

- `KalmanVelUpdate`: Berechnet den Kalman-Filteralgorithmus komplett nach den Formeln wie sie in Kapitel *Gleichungen Kalman-Filter* beschrieben werden.

- `KalmanVelUpdateFast`: Verzichtet im Kalman-Filteralgorithmus auf die Berechnung der Kovarianzmatrix des Schätzfehler P und die Kalman-Gain K. Stattdessen werden vorausberechnete Werte für die Kalman-Gain verwendet. Dabei kann, durch Einfügen des Defines `#define PRECOMP_KP` im Headerfile, zur Compile-Zeit entschieden werden, ob die Kalman-Gain im Initialisierungsschritt berechnet werden soll. Ansonsten können deren Werte in den Membervariablen `pre_comp_k11` und `pre_comp_k12` des Structs `KalmanVelObject` gesetzt werden.

Folgende Grafik zeigt den Ablauf der Funktion `KalmanVelUpdate` als Flussdiagramm. Der komplette Code des hier vorgestellten Ansatz findet sich im Anhang in Kapitel *Code Kalman-Filter zur Lagestützung*.

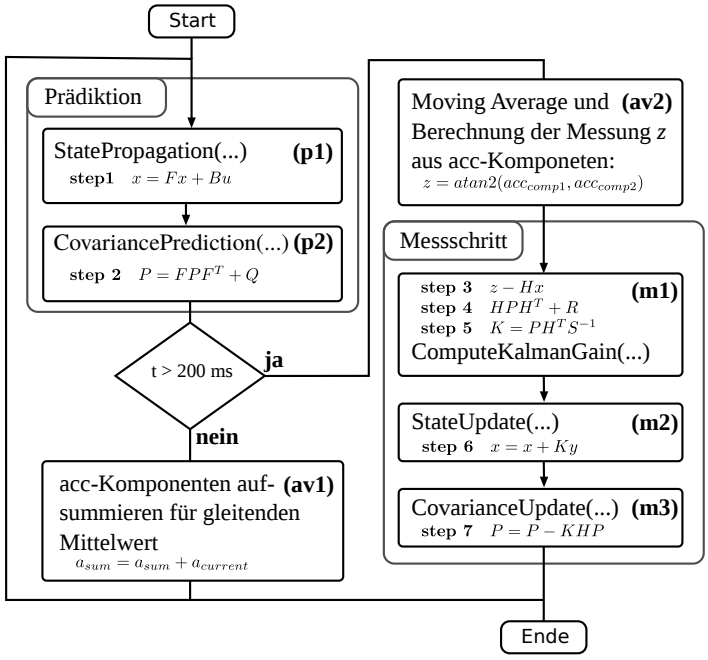

Abbildung 3.3: Flussdiagramm des Kalman-Filteralgorithmus zur Fluglageschätzung in sieben Schritten

Da viele Matrizen dünn besetzt sind fallen viele Berechnungen weg. Außerdem fällt beim Ausmultiplizieren des Algorithmus auf, dass sich manche Berechnungen wiederholen. Dazu kommt, dass die Kovarianzmatrix des Schätzfehlers P symmetrisch ist und es daher genügt, nur eine Seite der Diagonale zu berechnen. Deshalb können die Rechenschritte zur Durchführung des Filters optimiert und Floating-Point-Berechnungen eingespart werden, indem man sie entsprechend zusammenfasst. Die Berechnungen, welche zur Herleitung des Codes geführt haben, finden sich im Anhang in Kapitel *Matrix-Berechnungen*.

3.1.3 Vergleich der Lagefilter und Anwendung im Regler

Vergleich

Sowohl bei den Algorithmen nach Mahony/Madgwick, als auch beim Kalman-Filter muss eine $x \times x$-Matrix invertiert werden. Beide Ansätze sind zwar genau und auch bei Vibrationen noch robust, jedoch sind sie auch rechenintensiv. Ein Vorteil von Mahony/Madgwick ist, dass diese Algorithmen im Gegensatz zum Kalman-Ansatz über einen Winkel von 180 Grad hinaus funktionieren.

Wegen der hohen Rechenintensität wurden die etwas simpleren Komplementärfilteralgorithmen getestet. Diese Filter waren etwas schwerer zu parametrisieren und sind weniger genau als die Ansätze nach Mahony/Madgwick und dem Ansatz mit Kalman-Filter. Dafür ist deren Laufzeit geringer.

Einsatz im Regler

Es hat sich gezeigt, dass die hier vorgestellten Filter zur Fluglageschätzung für manche Anwendungen sehr gut funktionieren (Schlaf-Sensor, Gimbal), für andere Anwendungen (Regelung) jedoch, wegen deren hoher Laufzeit, nur mit Einschränkungen geeignet sind.

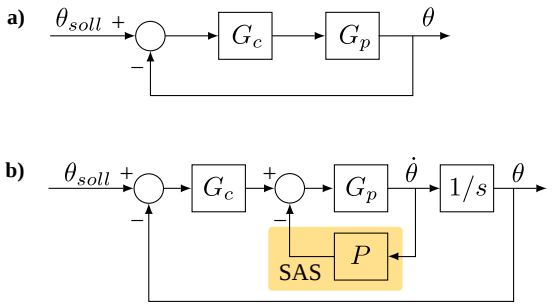

Abbildung 3.4: Konzept Fluglageregler

Desweiteren hat sich gezeigt, dass ein reiner PID-Regler einzig nach Fluglagewerten, wie in Abbildung 3.4 **(a)** dargestellt, sehr instabil ist. Aus diesem Grund wurde der Regler um eine Rückkopplung in Form eines sogenannten Stabilized Augmented Systems (SAS) erweitert wie es in Abblildung 3.4 **(b)** dargestellt ist. Dies ist ein schneller Regelkreis der den Flugroboter über die Drehratensensoren stabilisiert. In der Luftfahrzeugtechnik ist diese Form der Lageregelung über Drehratensensoren gängige Praxis [35]. Deshalb entstand im Rahmen dieser Arbeit die Idee eines zweistufigen Reglers, wie er in Abbildung 3.5 dargestellt ist. Ein innerer Regelkreis sollte in Form einer SAS umgesetzt werden und sehr schnell ablaufen. Da die Drehraten jedoch driften, sollte dieser innere Regelkreis durch einen langsameren äußeren Regelkreis unterstützt werden, der direkt auf die Fluglage regelt. Da dieser Regelkreis langsamer abläuft ist hier genug Zeit um den rechenintensiven Kalman-Filteralgorithmus durchführen zu lassen und mit dessen geschätzem Bias die Drehratensensoren zu korrigieren.

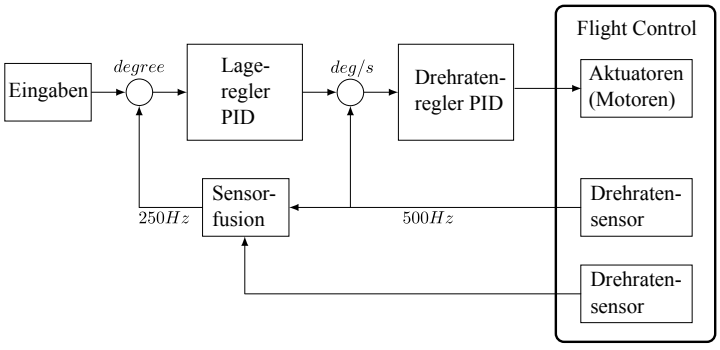

Abbildung 3.5: Alternatives Reglerkonzept

Die Idee nach Abbildung 3.5 wurde jedoch im Rahmen dieser Arbeit nicht mehr getestet, da zuvor ein anderer Ansatz sehr gute Ergebnisse zeigte. Die Idee dieses Ansatzes ist es, ebenfalls mit einem schnellen Regler auf die Drehraten zu regeln. Für diesen Regler wurde eine sehr kurze Zykluszeit von nur *2 ms* gewählt. Über einen sehr langen Zeitraum von fast einer Sekunde werden nebenher die Werte des Beschleunigungssensors durch einen gleitenden Mittelwert erfasst und mithilfe dessen Werten die Bias-Werte des Drehratensensors korrigiert. [1]

3.2 Positionsschätzung

Da der Betrag der Standardabweichung von GNSS-Sensoren im Bereich von einigen Metern liegt, sind dessen Messwerte nicht geeignet für die Verwendung in der Positionsregelung des Flugroboters. Um ein besseres Signal zu erhalten wird ein extended Kalman-Filter verwendet, die Werte des globalen Navigationssatellitensystems Globales Navigationssatellitensystem (GNSS), mit den Werten der Inertialen Messeinheit (IMU), zu fusionieren. Dafür werden die Positionswerte mithilfe des Strapdownalgorithmus propagiert und mit den Werten von Magnetfeld und barometrischem Drucksensor, sowie den Positions- und Geschwindigkeitswerten des GPS-Sensors korrigiert.

Im OpenPilot-Projekt [39] wird ein extended Kalman-Filter zur Positionsschätzung vorgestellt, welches sehr jenem aus [34] ähnelt. Im Unterschied zum Ansatz in [34] wird hier der Drift des Drehratensensors $\vec{b}$ nicht als Random-Walk-Prozess modelliert, sondern im Systemmodell mitgeschätzt.

Zunächst wird das Systemmodell in der Form der nichtlinearen Zustandsgleichung nach Gleichung (3.6) hergeleitet.

$$\dot{\vec{x}} = f(\vec{x}, \vec{u}, \vec{w}) \tag{3.6}$$

mit dem Zustandsvektor $\vec{x} = (\vec{p}, \vec{v}, \vec{q}, \vec{b})^T$, dem Eingangsvektor $\vec{u} = (\vec{a}, \vec{\omega})^T$ sowie dem Rauschvektor $\vec{w} = (w_a, w_\omega)^T$. Die Nomenklatur der Elemente jener Vektoren $\vec{x}$, $\vec{u}$ und $\vec{u}$ wird in nachfolgender Tabelle gezeigt. [2]

Tabelle 3.1: Nomenklatur des Systemmodells des Kalman-Filters zur Positionsschätzung

Variable	Beschreibung
$\vec{p}$	Position im Navigationskoordinatensystems $\vec{p}_{eb}^{\,n}$
$\vec{v}$	Geschwindigkeit im Navigationskoordinatensystem $\vec{v}_{eb}^{\,n}$
$\vec{q}$	Fluglagevektor in Form eines Quaternions
$\vec{b}$	Bias des Drehratensensors $\vec{b}_\omega$
$\vec{a}$	Beschleunigung $\vec{a}_{ib}^{\,n}$ im Koordinatensystem
$\vec{\omega}$	Drehraten im körperfesten Koordinatensystem $\vec{\omega}_{ib}^{\,b}$
$\vec{w_a}$	Messrauschen des Beschleunigungssensors
$\vec{w_\omega}$	Messrauschen des Drehratensensors

Lässt man in Gleichung (2.16) alle Elemente weg, welche mit Null multipliziert werden und damit unerheblich sind und berücksichtigt zudem den Bias des Drehratensensors, wird sie zum dynamischen Modell für

[1] Dieser Regler ist nicht als Teil dieser Arbeit entstanden und stamt von Thorsten Reitmeier.

[2] Es sei hier nocheinmal daran erinnert, dass das erdfeste Koordinatensystem und das Inertialkoordinatensystem näherungsweise aufeinanderfallen.

die Drehraten nach Gleichung (3.7).

$$\dot{q} = \frac{1}{2}\Omega(q)(\vec{\omega}_{ib}^{\,b} - \vec{b}_\omega) \tag{3.7}$$

mit

$$\Omega(q) = \begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix}$$

Damit ergeben sich die Bewegungsgleichungen eines starren Körpers für 6 Freiheitsgrade, mithilfe des Strapdownalgorithmus aus Gleichung (2.19), aus den Werten der IMU. Diese hängen nicht von der Dynamik des bewegten Körpers ab und sind damit ohne Anpassung für jedes Fahrzeug anwendbar.

$$\begin{aligned} \dot{\vec{p}}_{eb}^{\,n} &= \vec{v}_{eb}^{\,n} \\ \dot{\vec{v}}_{ib}^{\,n} &= C_n^b \vec{a}_{ib}^{\,b} + g_l^n \\ \dot{\vec{q}} &= \frac{1}{2}\Omega q(\vec{\omega}_{ib}^{\,n} - \vec{b}_\omega) \end{aligned}$$

Nun lässt sich das **Systemmodell** entsprechend der nichtlinearen Zustandsgleichung $\vec{f}(\vec{x}, \vec{u}, \vec{w})$ nach Gleichung (3.8) aufstellen.

$$\vec{f}(\vec{x}, \vec{u}, \vec{w}) = \begin{pmatrix} \dot{\vec{p}} \\ \dot{\vec{v}} \\ \dot{\vec{q}} \\ \dot{\vec{b}}_\omega \end{pmatrix} = \begin{pmatrix} \vec{v} \\ C_n^b \cdot (\vec{a} + \vec{w}_a) + g_l^n \\ \frac{1}{2}\Omega(\vec{q} \cdot (\vec{\omega} + \vec{w}_\omega - \vec{b}_\omega) \\ \vec{b}_\omega \end{pmatrix} \tag{3.8}$$

Im Gegensatz zum Ansatz nach [39] wird hier der Bias des Drehratensensors $\vec{b}_\omega$ in das Systemmodell aufgenommen, um im Kalman-Filteralgorithmus mitgeschätzt zu werden.

Für die Messung werden Positions- und Geschwindigkeitswerte des GPS-Sensors, sowie Messwerte des Magnetfeldsensors und des barometrischen Drucksensors herangezogen. Damit lässt sich das **Messmodell** als Funktion des Zustandsvektors $\vec{h}(\vec{x})$ entsprechend Gleichung (3.9) formulieren.

$$\vec{z} = \vec{h}(\vec{x}) = \begin{pmatrix} \vec{p} \\ \vec{v} \\ \vec{h}^b \\ p \end{pmatrix} = \begin{pmatrix} \vec{p} \\ \vec{v} \\ C_b^n(\vec{q})\vec{h}^e \\ p \end{pmatrix} \tag{3.9}$$

mit dem Positionsvektor $\vec{p}$, dem Geschwindigkeitsvektor $\vec{v}$, dem Magnetfeldvektor $\vec{h}$ und dem skalarem Druckwert des barometrischem Höhensensors p (nicht zu verwechseln mit dem Positionsvektor $\vec{p}$).

Der extended Kalman-Filteralgorithmus erfordert eine Linearisierung des Systemmodells. Hierfür werden die Jacobi-Matrizen aus Gleichung (3.10) benötigt.

$$\begin{aligned} F &= \frac{\partial \vec{f}}{\partial \vec{x}} \\ G &= \frac{\partial \vec{f}}{\partial \vec{w}} \\ H &= \frac{\partial \vec{h}}{\partial \vec{x}} \end{aligned} \tag{3.10}$$

Die Partielle Ableitung von F ergibt sich nach Gleichung (3.11).

$$F = \left(\begin{array}{c|c|cc} 0_{3\times3} & I_{3\times3} & \multicolumn{2}{c}{0_{3\times3}} \\ \hline & & F_{vq} & 0_{3\times3} \\ 0_{10\times6} & & F_{qq} & F_{qb} \\ & & 0_{3x4} & I_{3\times3} \end{array}\right) \tag{3.11}$$

mit

$$F_{vq} = \begin{pmatrix} F_{vq0} & F_{vq1} & F_{vq2} & F_{vq3} \\ -F_{vq3} & -F_{vq2} & F_{vq1} & F_{vq0} \\ F_{vq2} & -F_{vq3} & F_{vq0} & F_{vq1} \end{pmatrix}$$

$$F_{vq0} = 2(q_0\tilde{a}_x - q_3\tilde{a}_y + q_2\tilde{a}_z)$$
$$F_{vq1} = 2(q_1\tilde{a}_x + q_2\tilde{a}_y + q_3\tilde{a}_z)$$
$$F_{vq2} = 2(-q_2\tilde{a}_x + q_1\tilde{a}_y + q_0\tilde{a}_z)$$
$$F_{vq3} = 2(-q_3\tilde{a}_x - q_0\tilde{a}_y + q_1\tilde{a}_z)$$

$$F_{qq} = \frac{1}{2}\begin{pmatrix} 0 & -(\tilde{\omega}_x - b_{\omega x}) & -(\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_z - b_{\omega z}) \\ (\tilde{\omega}_x - b_{\omega x}) & 0 & (\tilde{\omega}_z - b_{\omega z}) & -(\tilde{\omega}_y - b_{\omega y}) \\ (\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_z - b_{\omega z}) & 0 & (\tilde{\omega}_x - b_{\omega x}) \\ (\tilde{\omega}_z - b_{\omega z}) & (\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_x - b_{\omega x}) & 0 \end{pmatrix}$$

und

$$F_{qb} = -\frac{1}{2}\Omega(\vec{q})$$

Die partielle Ableitung der Matrix G ergibt sich nach Gleichung (3.12).

$$G = \left(\begin{array}{c} 0_{3\times6} \\ \hline \begin{array}{c|c} 0_{3\times3} & C_b^n(\vec{q}) \\ \hline \Omega(\vec{q})/2 & 0_{4\times3} \end{array} \\ \hline 0_{3\times6} \end{array}\right) \tag{3.12}$$

Die partielle Ableitung der Messmatrix H ergibt sich nach Gleichung (3.13).

$$H = \left(\begin{array}{cc|c} \multicolumn{2}{c|}{I_{6\times6}} & 0_{6\times7} \\ \hline \multicolumn{2}{c|}{0_{3\times6}} & \begin{array}{c|c} H_{hq} & 0_{3\times3} \end{array} \\ \hline 0\ 0\ -1 & 0_{1\times3} & 0_{1\times7} \end{array}\right) \tag{3.13}$$

mit

$$H_{hq} = \begin{pmatrix} H_{hq0} & H_{hq1} & H_{hq2} & H_{hq3} \\ H_{hq3} & -H_{hq2} & H_{hq1} & -H_{hq0} \\ -H_{hq2} & -H_{hq3} & H_{hq0} & H_{hq1} \end{pmatrix}$$

$$H_{hq0} = 2(q_0\tilde{h}_x + q_3\tilde{h}_y - q_2\tilde{h}_z)$$
$$H_{hq1} = 2(q_1\tilde{h}_x + q_2\tilde{h}_y + q_3\tilde{h}_z)$$
$$H_{hq2} = 2(-q_2\tilde{h}_x + q_1\tilde{h}_y - q_0\tilde{h}_z)$$
$$H_{hq3} = 2(-q_3\tilde{h}_x + q_0\tilde{h}_y + q_1\tilde{h}_z)$$

Nun lässt sich schließlich das extended Kalman-Filter herleiten. Zunächst muss das **System-** und das **Messmodell** der nichtlinearen Form aus den Gleichungen (3.6) und (3.9) in die linearisierte und diskretisierte Form nach Gleichung (3.14) geführt werden.

$$\begin{aligned} \vec{x}_t &= \Phi \vec{x}_{t-1} + \Gamma \vec{w}_{t-1} \\ \vec{z}_t &= H \vec{x}_t + \vec{v}_t \end{aligned} \tag{3.14}$$

Da das Systemmodell aus Gleichung (3.6) in Form einer kontinuierlichen Differentialgleichung $\vec{f}(\vec{x}, \vec{u}, \vec{w})$ vorliegt muss dieses diskretisiert werden. Hierfür werden die Näherungen aus Gleichung (3.15) verwendet.

$$\begin{aligned} \Phi &\cong I + FT \\ \Gamma &\cong GT \end{aligned} \tag{3.15}$$

Damit steht der komplette extended Kalman-Filteralgorithmus für die Positionsschätzung fest. Er wird in Abbildung 3.6 dargestellt. Die Integration zur Berechnung der Zustandsgleichung im ersten Filterschritt **(p1)**, wird mithilfe des Runge-Kutta-Algorithmus vierter Stufe [3] durchgeführt. Dabei wird Gleichung (3.8) herangezogen. Die partiellen Ableitungen zur Linearisierung des Systemmodells aus Gleichung (3.11) und (3.12) werden in **(p3)** durchgeführt. In **(p4)** wird, wie im Kalman-Filter üblich, der aufgrund des Messrauschens anwachsende Schätzfehler in Form der Kovarianzmatrix P berechnet.

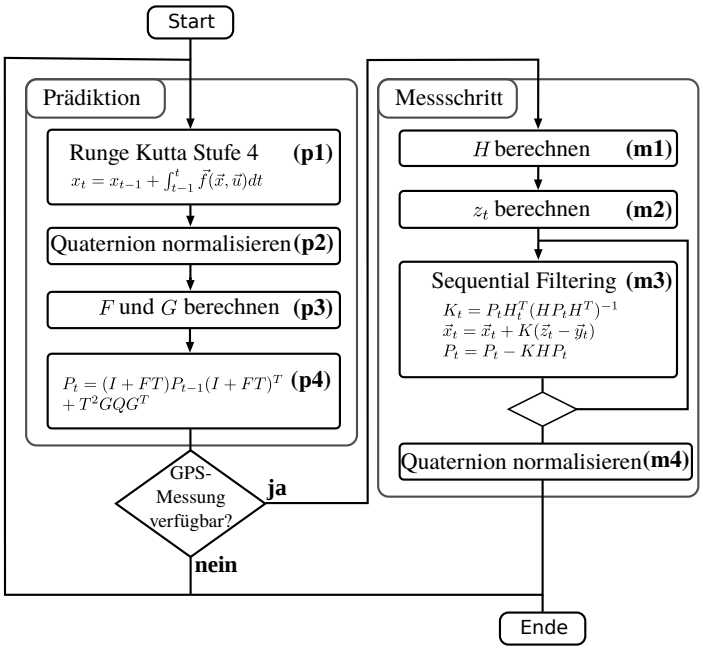

Abbildung 3.6: Extended Kalman-Filteralgorithmus zur Positionsschätzung nach [39]

Für die Berechnung von H in **(m1)** wird Gleichung (3.13) herangezogen. Zur Berechnung des Ausgangswertes $\vec{z}_t$ in Schritt **(m2)** wird Gleichung (3.9) herangezogen. In Schritt **(m3)** wird ein wesentlicher Trick für die Reduktion des Rechenaufwandes angewendet: Um die im Messschritt des Kalman-Filteralgorithmus notwendige Matrixinversion zu verhindern, werden wie in [13] und [10] vorgestellt, die Messungen einzeln verarbeitet. Damit reduziert sich die besagte Matrixinversion zu skalaren Divisionen, einzeln durchgeführt für jede Messung. Dies ist zulässig, da das Messrauschen als korrelationsfrei angenommen wird.

[3] Runge-Kutta-Integration vierter Stufe wird in [45] bündig beschrieben (S. 32ff).

KAPITEL 4

Ausblick

Die Fluglagefilter für die Schätzung der Orientierung des UAVs, wie sie in dieser Arbeit vorgestellt wurden, kamen für die Fluglageregelung nicht zum Einsatz. Die Gründe hierfür werden in Kapitel *Einsatz im Regler* dargelegt. Trotzdem wurden aus deren Umsetzung wertvolle Informationen für die numerische Umsetzung des Kalman-Filters zur Positionsschätzung gewonnen.

Auch können die gewonnenen Erfahrungen im Bereich Kalman-Filterung durch dezentrale Konzepte wie Descentralized Kalman-Filter und Distributed Kalman-Filter, beschrieben in [2], in weiteren Anwendungen der Sensorvernetzung zukünftig ausgebaut werden.

Zu den in Kapitel *Positionsschätzung* gefundenen Algorithmen zur Positionsschätzung mit GNSS-Systemen und Trägheitsnavigation finden sich im Anhang noch keine Codes, da diese im zeitlichen Rahmen dieser Arbeit nicht mehr auf der Flughardware getestet werden konnten. Der nächste Schritt wird sein, diese auf der Flughardware umzusetzen.

Sobald gute Positionswerte gefunden werden, kann eine Positionsregelung implementiert werden. Der häufigste Ansatz für eine Regelung in X/Y-Richtung, ist ein zweistufiger Regler von Position und Geschwindigkeit, wie er beispielsweise in [31] oder [18] beschrieben wird. Im inneren Regelkreis wird die Geschwindigkeit und im äußerem Regelkreis die Position des UAVs geregelt. Die Eingangsgröße des Reglers ist die Position und die Ausgangsgröße sind die Stellwerte, die zu den gewünschten Fluglagewerten führen.

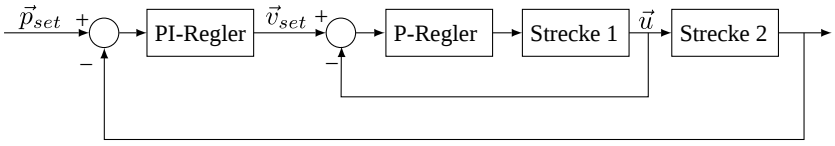

Abbildung 4.1: Konzept Positionsregelung nach [31]

Auch zur Höhenregelung sowie Höhen-Estimation finden sich Algorithmen in [31] und [18]. Die Update-Rate von Höhen- und Positionsregelung sollte weit unter jener der Fluglageregelung liegen.

Komplementär-Filter zur Lageschätzung

A.1 Code Ansatz Mahony

A.1.1 Header-Datei

```
#ifndef _MAHONY_H
#define _MAHONY_H

typedef struct
{
  float twoKp;
  float twoKi;

  float q0;    //< set point
  float q1;    //< error
  float q2;    //< previous error
  float q3;    //< integral

  float integralFBx;
  float integralFBy;
  float integralFBz;  // integral error terms scaled by Ki

} MahonyObject;

void mahonyInit(MahonyObject* mahony);
void mahonyUpdate(MahonyObject* mahony, float gx, float gy, float gz, float ax, float ay, float az, float dt);
#endif
```

A.1.2 C-Datei

```
#include "Mahony_Lage.h"
#include "mainvars.h"

#include <math.h>

#define M_PI 3.14159265358979323846

#define TWO_KP_DEF  (2.0f * 0.4f) // 2 * proportional gain
#define TWO_KI_DEF  (2.0f * 0.001f) // 2 * integral gain

float invSqrt(float x);

float ConvertToDegree(float value)
{
    return (value/PI * 180.0);
}

//---------------------------------------------------------------------------------------------------

const float Epsilon = 0.0009765625f;
const float Threshold = 0.5f - Epsilon;

void CalculateAngles(MahonyObject* mahony)
{
    float XY = mahony->q0 * mahony->q1;
    float ZW = mahony->q2 * mahony->q3;
```

```
27
28      float TEST = XY + ZW;
29
30      if ( (TEST < -Threshold) || (TEST > Threshold) )
31      {
32          float sign;
33          if (TEST < 0.0) sign=-1.0; else sign=1.0;
34          pitch = sign * 2.0 * (float)atan2(mahony->q0, mahony->q3);
35          yaw   = sign * PI/2;
36          roll  = 0;
37      }
38      else
39      {
40
41          float XX = mahony->q0 * mahony->q0;
42          float XZ = mahony->q0 * mahony->q2;
43          float XW = mahony->q0 * mahony->q3;
44
45          float YY = mahony->q1 * mahony->q1;
46          float YW = mahony->q1 * mahony->q3;
47          float YZ = mahony->q1 * mahony->q2;
48
49          float ZZ = mahony->q2 * mahony->q2;
50
51          pitch = atan2(2 * YW - 2 * XZ, 1 - 2 * YY - 2 * ZZ);
52          //yaw   = atan2(2 * XW - 2 * YZ, 1 - 2 * XX - 2 * ZZ);
53          roll  = asin(2 * TEST);
54
55      }//if
56
57      yaw   = ConvertToDegree(yaw);
58      pitch = ConvertToDegree(pitch);
59      roll  = ConvertToDegree(roll);
60
61  }
62
63  void mahonyInit(MahonyObject* mahony)
64  {
65      mahony->twoKp = TWO_KP_DEF;     // 2 * proportional gain (Kp)
66      mahony->twoKi = TWO_KI_DEF;     // 2 * integral gain (Ki)
67
68    mahony->q0 = 0.0001f;
69    mahony->q1 = 0.0001f;
70    mahony->q2 = 0.0001f;
71      mahony->q3 = 0.0001f;
72  }
73
74  void mahonyUpdate(MahonyObject* mahony, float gx, float gy, float gz, float ax, float ay, float az, float dt)
75  {
76    float recipNorm;
77    float halfvx, halfvy, halfvz;
78    float halfex, halfey, halfez;
79    float qa, qb, qc;
80
81    // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer normalisation)
82    if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
83    {
84      // Normalise accelerometer measurement
85      recipNorm = invSqrt(ax * ax + ay * ay + az * az);
86      ax *= recipNorm;
87      ay *= recipNorm;
88      az *= recipNorm;
89
90      // Estimated direction of gravity and vector perpendicular to magnetic flux
91      halfvx = mahony->q1 * mahony->q3 - mahony->q0 * mahony->q2;
92      halfvy = mahony->q0 * mahony->q1 + mahony->q2 * mahony->q3;
93      halfvz = mahony->q0 * mahony->q0 - 0.5f + mahony->q3 * mahony->q3;
94
95      // Error is sum of cross product between estimated and measured direction of gravity
96      halfex = (ay * halfvz - az * halfvy);
97      halfey = (az * halfvx - ax * halfvz);
98      halfez = (ax * halfvy - ay * halfvx);
99
100     // Apply proportional feedback
101     gx += mahony->twoKp * halfex;
102     gy += mahony->twoKp * halfey;
103     gz += mahony->twoKp * halfez;
104   }
105
106   // Integrate rate of change of quaternion
107   gx *= (0.5f * dt);   // pre-multiply common factors
108   gy *= (0.5f * dt);
109   gz *= (0.5f * dt);
110   qa = mahony->q0;
111   qb = mahony->q1;
112   qc = mahony->q2;
113   mahony->q0 += (-qb * gx - qc * gy - mahony->q3 * gz);
114   mahony->q1 += (qa * gx + qc * gz - mahony->q3 * gy);
115   mahony->q2 += (qa * gy - qb * gz + mahony->q3 * gx);
116   mahony->q3 += (qa * gz + qb * gy - qc * gx);
117
```

```
118   // Normalise quaternion
119   recipNorm = invSqrt(mahony->q0 * mahony->q0 + mahony->q1 * mahony->q1 + mahony->q2 * mahony->q2 + mahony->q3 * mahony->q3);
120   mahony->q0 *= recipNorm;
121   mahony->q1 *= recipNorm;
122   mahony->q2 *= recipNorm;
123   mahony->q3 *= recipNorm;
124
125   // Calculate angles from quaternions
126   CalculateAngles(mahony);
127 }
128
129 //-------------------------------------------------------------------------------------------------------
130 // Fast inverse square-root
131 // See: http://en.wikipedia.org/wiki/Fast_inverse_square_root
132 float invSqrt(float x)
133 {
134     float halfx = 0.5f * x;
135     float y = x;
136     long i = *(long*)&y;
137     i = 0x5f3759df - (i>>1);
138     y = *(float*)&i;
139     y = y * (1.5f - (halfx * y * y));
140     return y;
141 }
```

A.2 Code Ansatz Oliviera/Pascoal/Kaminer

```
1  #define K_P         0.02
2  #define K_I         0.001
3
4  #define DELTA_T     0.002 // Cycle Time 2 ms
5
6  void Complement1Update(float ax, float ay, float az, float gx, float gy, float gz)
7  {
8      float dotRoll;
9      float dotBiasRoll;
10
11     float y_gyro_roll = 0.0;
12     float y_acc_roll = 0.0;
13
14     // measurement of gyro
15     y_gyro_roll = gx;
16
17     // measurment of acc
18 #ifdef  SMALL_ANGLE_APPROX
19     y_acc_roll = ay/az;
20 #else // no small angle approximation
21     y_acc_roll = atan2(ay,az);
22 #endif
23
24     // Convert angle calculated
25     // by acc to degrees
26     y_acc_roll = y_acc_roll/PI * 180.0;
27
28     dotRoll = y_gyro_roll - biasRoll + K_P*(y_acc_roll - roll);
29
30     dotBiasRoll = -K_I*(y_acc_roll - roll);
31     biasRoll = biasRoll + dotBiasRoll*DELTA_T;
32
33     roll = roll + dotRoll*DELTA_T;
34 }
```

Kalman-Filter zur Lageschätzung

B.1 Code Kalman-Filter zur Lagestützung

B.1.1 Header-Datei

```
/* ---------------------------------------------------------------------------
 *
 *
 *  ____                  _____     _
 * / ___|  __ _ _ __   _ __|__  /  __| | ___
 * \___ \ / _` | '_ \| '_ \ / /  / _` |/ _ \
 *  ___) | (_| | |_) | |_) / /_ | (_| |  __/
 * |____/ \__,_| .__/| .__/____(_)__,_|\___|
 *             |_|   |_|
 *
 *
 *
 * Author: Andreas Gschossmann
 * Email:  andreas.gschossmann@hs-regensburg.de
 *
 * kalman_vel.h - Kalman Filter for attitude estimation
 * The algorithm estimates the attitude and the bias of
 * the gyro using the gyro for the system model and the
 * accelerometer for the measurement model.
 *
 * ---------------------------------------------------------------------------*/

#ifndef KalmanVel_h
#define KalmanVel_h

#define Q1 0.0f//0.003f
#define Q2 0.1f//0.001f
#define R1 1.0f//0.008f

#define DT 0.002

#define MSR_CNT 20

#define MOVING_AVG_ACC
//#define PRECOMP_KP
//#define SMALL_ANGLE_APPROX

typedef struct
{
    // public member variables
    float x1, x2;

    // private member variables
    float p11, p12, p21, p22;           // Covariance Matrix
    float q1,q2;                                    // Believe in System
    float r1;                                           // Believe Measurement
    float dt;                                           // Delta t

    float y;                                            // Innovation
    float k11, k12;                                 // Kalman Gain

    float acc_comp1_sum;
    float acc_comp2_sum;

    float pre_comp_k11;
    float pre_comp_k12;

```

```
58      int measure_cnt;
59
60  } KalmanVelObject;
61
62  // public functions
63  void KalmanVelInit(KalmanVelObject *data);
64  void KalmanVelUpdate(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega);
65  void KalmanVelUpdateFast(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega);
66
67  // private functions
68  // step 1
69  void StatePropagation(KalmanVelObject *data, float u);
70  // step 2
71  void CovariancePrediction(KalmanVelObject *data);
72  // step 3-5
73  void ComputeKalmanGain(KalmanVelObject *data, float z);
74  // step 6
75  void StateUpdate(KalmanVelObject *data);
76  // step 7
77  void CovarianceUpdate(KalmanVelObject *data);
78
79  void PreCompKP(KalmanVelObject *data);
80
81
82  #endif
```

B.1.2 C-Datei

```
1   /* ---------------------------------------------------------------------------
2    *
3    *
4    *  ____                  _____     _
5    * / ___|  __ _ _ __  _ __|__  /  __| | ___
6    * \___ \ / _` | '_ \| '_ \ / /  / _` |/ _ \
7    *  ___) | (_| | |_) | |_) / /_ | (_| |  __/
8    * |____/ \__,_| .__/| .__/____(_)__,_|\___|
9    *             |_|   |_|
10   *
11   *
12   *
13   * Author: Andreas Gschossmann
14   * Email:  andreas.gschossmann@hs-regensburg.de
15   *
16   * kalman_vel.c - Kalman Filter for attitude estimation
17   * The algorithm estimates the attitude and the bias of
18   * the gyro using the gyro for the system model and the
19   * accelerometer for the measurement model.
20   *
21   * ---------------------------------------------------------------------------*/
22
23  #include "kalman_vel.h"
24  #include "mainvars.h"
25  #include "FlashDtaValues.h"
26  #include <math.h>
27
28  void KalmanVelInit(KalmanVelObject *data)
29  {
30      int i;
31
32      data->x1 = 0.0f;
33      data->x2 = 0.0f;
34
35      data->p11 = 1000.0f;
36      data->p12 = 0.0f;
37      data->p21 = 0.0f;
38      data->p22 = 1000.0f;
39
40      data->q1 = Q1;
41      data->q2 = Q2;
42
43      data->r1 = R1;
44      data->dt = DT;
45
46      data->acc_comp1_sum = 0.0f;
47      data->acc_comp2_sum = 0.0f;
48
49      data->pre_comp_k11 = 0.393326f;
50      data->pre_comp_k12 = 0.0f;
51
52      data->measure_cnt = 0;
53
54  #ifdef PRECOMP_KP
55      for (i=0; i<PRECOMP_KP_CNT; i++) {
56          PreCompKP(data);
57      }
58  #endif
```

```
59  }
60
61  void KalmanVelUpdate(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega)
62  {
63      float z;
64
65      // Step 1
66      // x(k) = Fx(k-1) + Bu + w:
67      StatePropagation(data, omega);
68
69      // Step 2
70      // P = FPF'+Q
71      CovariancePrediction(data);
72
73  ////////////////////////////////////
74  // here  moving average of acc for
75  // 200 ms and then measurement step
76  if (data->measure_cnt < MSR_CNT) {
77      data->measure_cnt ++;
78  #ifdef MOVING_AVG_ACC
79          data->acc_comp1_sum += acc_comp1;
80          data->acc_comp2_sum += acc_comp2;
81  #endif
82  } else {
83      data->measure_cnt = 0;
84  #ifdef MOVING_AVG_ACC
85      acc_comp1 = (data->acc_comp1_sum)/MSR_CNT;
86      acc_comp2 = (data->acc_comp2_sum)/MSR_CNT;
87  #endif MOVING_AVG_ACC
88
89      data->acc_comp1_sum = 0.0f;
90      data->acc_comp2_sum = 0.0f;
91
92      ////////////////////////////////////
93      // do measurement step of kalman Filter
94
95      // measurment of acc
96  #ifdef  SMALL_ANGLE_APPROX
97    z = acc_comp1/acc_comp2;
98  #else // no small angle approximation
99      z = atan2(acc_comp1, acc_comp2);
100 #endif
101     // convert to degrees
102     z = z/PI * 180.0;
103
104     // Step 3
105     // y = z(k) - Hx(k)
106     // Step 4
107     // S = HPT' + R
108     // Step 5
109     // K = PH'*inv(S)
110     ComputeKalmanGain(data, z);
111
112     // Step 6
113     // x = x + Ky
114     StateUpdate(data);
115
116     // Step 7
117     // P = (I-KH)P
118     CovarianceUpdate(data);
119 }
120 }
121
122 void KalmanVelUpdateFast(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega)
123 {
124     float y, z;
125
126     // Step 1
127     // x(k) = Fx(k-1) + Bu + w:
128     StatePropagation(data, omega);
129
130     ////////////////////////////////////
131 // here  moving average of acc for
132 // 200 ms and then measurement step
133 if (data->measure_cnt < MSR_CNT) {
134     data->measure_cnt ++;
135 #ifdef MOVING_AVG_ACC
136         data->acc_comp1_sum += acc_comp1;
137         data->acc_comp2_sum += acc_comp2;
138 #endif
139 } else {
140     data->measure_cnt = 0;
141 #ifdef MOVING_AVG_ACC
142     acc_comp1 = (data->acc_comp1_sum)/MSR_CNT;
143     acc_comp2 = (data->acc_comp2_sum)/MSR_CNT;
144 #endif MOVING_AVG_ACC
145
146     data->acc_comp1_sum = 0.0f;
147     data->acc_comp2_sum = 0.0f;
148
149     ////////////////////////////////////
```

```
150     // do measurement step of kalman Filter
151
152     // measurment of acc
153 #ifdef  SMALL_ANGLE_APPROX
154   z = acc_comp1/acc_comp2;
155 #else // no small angle approximation
156     z = atan2(acc_comp1, acc_comp2);
157 #endif
158     // convert to degrees
159     z = z/PI * 180.0;
160
161     // Step 3
162     // y = z(k) - Hx(k)
163     data->y = z - data->x1;
164
165     // Step 6
166     // x = x + Ky
167     data->x1 = data->x1 + data->pre_comp_k11*data->y;
168     data->x2 = data->x2 + data->pre_comp_k11*data->y;
169 }
170 }
171
172 // private functions
173 // step 1
174 void StatePropagation(KalmanVelObject *data, float u)
175 {
176     // Step 1
177     // x(k) = Fx(k-1) + Bu + w:
178     data->x1 = data->x1 + u*data->dt - data->x2*data->dt;
179     //x2 = x2;
180 }
181
182 // step 2
183 void CovariancePrediction(KalmanVelObject *data)
184 {
185     float a, b;
186
187     // Step 2
188     // P = FPF'+Q
189     a = data->p12*data->dt; // = data->21*data-dt
190     b = data->p22*data->dt;
191     data->p11 = data->p11 - a - a + b*data->dt + data->q1;
192     data->p12 = data->p12 - b;
193     data->p21 = data->p12; // Symmetry of Covariance Matrix
194     data->p22 = data->p22 + data->q2;
195 }
196
197 // step 3-5
198 void ComputeKalmanGain(KalmanVelObject *data, float z)
199 {
200     float s;
201
202     // Step 3
203     // y = z(k) - Hx(k)
204     data->y = z - data->x1;
205
206     // Step 4
207     // S = HPT' + R
208     s = data->p11 + data->r1;
209
210     // Step 5
211     // K = PH'*inv(S)
212   data->k11 = data->p11/s;
213     data->k12 = data->p21/s;
214 }
215
216 // step 6
217 void StateUpdate(KalmanVelObject *data)
218 {
219     // Step 6
220     // x = x + Ky
221     data->x1 = data->x1 + data->k11*data->y;
222     data->x2 = data->x2 + data->k12*data->y;
223 }
224
225 // step 7
226 void CovarianceUpdate(KalmanVelObject *data)
227 {
228     // Step 7
229     // P = (I-KH)P
230     data->p11 = data->p11 - data->k11*data->p11;
231     data->p12 = data->p12 - data->k11*data->p12;
232     data->p21 = data->p12; // symmetry of covariance matrix
233     data->p22 = data->p22 - data->k12*data->p12;
234 }
235
236 void PreCompKP(KalmanVelObject *data)
237 {
238     float a, b;
239     float s;
240     float k11, k12;
```

```
241
242         // Step 2
243         // P = FPF'+Q
244         a = data->p12*data->dt; // = data->21*data-dt
245         b = data->p22*data->dt;
246         data->p11 = data->p11 - a - a + b*data->dt + data->q1;
247         data->p12 = data->p12 - b;
248         data->p21 = data->p12; // Symmetry of Covariance Matrix
249         data->p22 = data->p22 + data->x2*data->q2;
250
251     if (data->measure_cnt < MSR_CNT) {
252         data->measure_cnt ++;
253     } else {
254         data->measure_cnt = 0;
255
256         // Step 4
257         // S = HPT' + R
258         s = data->p11 + data->r1;
259
260         // Step 5
261         // K = PH'*inv(S)
262         data->pre_comp_k11 = data->p11/s;
263         data->pre_comp_k12 = data->p21/s;
264         //printf ("k11: %f k12: %f\n", data->k11, data->k12);
265
266         // Step 7
267         // P = (I-KH)P
268         data->p11 = data->p11 - data->pre_comp_k11*data->p11;
269         data->p12 = data->p12 - data->pre_comp_k11*data->p12;
270         data->p21 = data->p12; // symmetry of covariance matrix
271         data->p22 = data->p22 - data->pre_comp_k12*data->p12;
272     }
273     }
```

B.2 Matrix-Berechnungen

Im Folgenden werden alle Matrixmultiplikationen gezeigt, die für die Herleitung der Codezeilen des Kalman-Filters zur Lageschätzung aus Kapitel *Code Kalman-Filter zur Lagestützung* im Anhang notwendig sind gezeigt.

$$\begin{bmatrix}1 & -dt\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}dt\\ 0\end{bmatrix}u=\begin{bmatrix}x_1-x_2dt+udt\\ x_2\end{bmatrix}$$

Step 1:
$x = Fx + Bu$

$$\begin{bmatrix}1 & -dt\\ 0 & 1\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1 & 0\\ -dt & 1\end{bmatrix}+\begin{bmatrix}q_1 & 0\\ 0 & q_2\end{bmatrix}$$
$$=\begin{bmatrix}p_{11}-p_{21}dt & p_{12}-p_{22}dt\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1 & 0\\ -dt & 1\end{bmatrix}+\dots$$
$$=\begin{bmatrix}p_{11}-p_{21}dt-(\overbrace{p_{12}-\underbrace{p_{22}dt}_{a}}^{b})dt & \overbrace{p_{12}-\underbrace{p_{22}dt}_{a}}^{b}\\ \overbrace{p_{21}-\underbrace{p_{22}dt}_{a}}^{b} & p_{22}\end{bmatrix}+\begin{bmatrix}q_1 & 0\\ 0 & q_2\end{bmatrix}$$
$$=\begin{bmatrix}p_{11}-p_{21}dt-b\cdot dt+q_1 & b\\ b & p_{22}+q_2\end{bmatrix}\quad\text{mit}\quad\begin{matrix}a=p_{22}dt\\ b=p_{12}-p_{22}dt\end{matrix}$$
$$\text{Symmetrie}\begin{cases}=p_{12}-a\\ =p_{21}-a\end{cases}$$

Step 2:
$P = FPF^T + Q$

$$y=z-\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=z-x_1$$

Step 3:
$y = z - Hx$

$$\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}+R$$
$$=\begin{bmatrix}p_{11} & p_{12}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}+R$$
$$=p_{11}+r_1$$

Step 4:
$S = HPH^T + R$

$$\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}s_{11}^{-1}=\begin{bmatrix}p_{11}\\ p_{12}\end{bmatrix}s_{11}^{-1}$$
$$=\begin{bmatrix}p_{11}/s_{11}\\ p_{12}/s_{11}\end{bmatrix}$$

Step 5:
$K = PH^T S^{-1}$

$$\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}k_{11}\\ k_{12}\end{bmatrix}y=\begin{bmatrix}x_1+k_{11}y\\ x_2+k_{12}y\end{bmatrix}$$

Step 6:
$x = x + Ky$

$$\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}-\begin{bmatrix}k_{11}\\ k_{12}\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}=\dots\begin{bmatrix}k_{11} & 0\\ k_{12} & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{12} & p_{22}\end{bmatrix}$$
$$=\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}-\begin{bmatrix}k_{11}p_{11} & k_{11}p_{12}\\ k_{12}p_{11} & k_{12}p_{12}\end{bmatrix}=\begin{bmatrix}p_{11}-k_{11}p_{11} & p_{12}-k_{11}p_{12}\\ p_{21}-k_{12}p_{11} & p_{22}-k_{12}p_{12}\end{bmatrix}$$
$$\text{Symmetrie}\begin{cases}=p_{21}-\frac{p_{21}}{p_{11}+r}\cdot p_{11}\\ =p_{12}-\frac{p_{11}}{p_{11}+r}\cdot p_{12}\end{cases}$$
bei Einsetzen von K wird Symmetrie erkennbar.

Step 7:
$P = P - KHP$

Literaturverzeichnis

[1] Cavallo A., Cirillo A., Cirillo P., De Maria G., Falco P., Natale C., and Pirozzi S. Experimental comparison of sensor fusion algorithms for attitude estimation. *The International Federation of Automatic Control*, 2014.

[2] Ahmed Abdelgawad and Magdy Bayoumi. *Resource-Aware Data Fusion Algorithms for Wireless Sensor Networks*. Springer, 2012.

[3] Fakhri Alam, ZhaiHe Zhou, and JiaJia Hu. A comparative analysis of orientation estimation filters using mems based imu. *2nd International Conference on Research in Science, Engineering and Technology*, 2014.

[4] ac Antoulas. *Mathematical System Theory - The Influence of R. E. Kalman*. Springer, 1991.

[5] David A. Boruchoff. The three greatest inventions of modern times: an idea and its public. *Entangled Knowledge: Scientific Discourses and Cultural Difference*, 2012.

[6] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progression. *J. Symbolic Computation*, 1990.

[7] Offizielle Website der Europäischen Union. Kommentar eu-kommissarin bieńkowska. März 2015. zuletzt aufgerufen März 2015, http://ec.europa.eu/deutschland/press/pr_releases/13213_de.htm.

[8] Guillaume Ducard and Raffaello D'Andrea. Autonomous quadrotor flight using a vision system and accommodating frames misalignment. *IEEE International Symposium Industrial Embedded Systems*, 2009.

[9] Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal Processing Magazine*, 2012.

[10] Jay Farrell and Barth Matthew. *The Global Positioning System and Inertial Navigation*. McGraw-Hill, 1998.

[11] Holger Flühr. *Flugsicherungstechnik*. Springer, Graz, Österreich, 2010.

[12] Free Software Foundation. Gnu scientific license (gsl). zuletzt aufgerufen März 2015, http://www.gnu.org/software/gsl.

[13] Mohinder S. Gewal and Angus P. Andrews. *Kalman Filtering - Theory and Practice Using MATLAB*. WILEY, Hoboken, New Jersey, 2001.

[14] Mohinder S. Grewal and Angus P. Andrews. Application of kalman filtering in aerospace 1960 to the present. *IEEE Control Systems Magazine*, 2010.

[15] Andreas Gschossmann. Download-links. zuletzt aufgerufen April 2015, https://hps.othr.de/gsa39665/downloads.html.

[16] Andreas Gschossmann. Evaluation of an open source realtime-kinematic gps software library. zuletzt aufgerufen April 2015, https://hps.othr.de/gsa39665/files/paper_diffgps.pdf.

[17] Andreas Gschossmann. Satellitennavigation mit dem global positioning system (gps). zuletzt aufgerufen April 2015, http://hps.othr.de/gsa39665/files/Bericht_Breitbandige_Zugangsnetze.pdf.

[18] Daniel Gurdan, Jan Stumpf, Michael Achtelik, Klaus-Michael Doth, Gerd Hirzinger, and Daniela Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 khz. *IEEE International Conference on Robotics and Control*, 2007.

[19] Markus Haid. *Verbesserung der referenzlosen inertialen Objektverfolgung zur Low-cost Indoor-Navigation durch Anwendung der Kalman-Filterung*. Fraunhofer IRB Verlag, 2004.

[20] William Rowan Hamilton. Quaternions, or on a new system of imagineries in algebra. *Philosophical Magazine*, 1844.

[21] Roger Isakson. Labview-quaternion-ahrs. zuletzt aufgerufen März 2014, https://code.google.com/p/labview-quaternion-ahrs/.

[22] Yan-Bin Jia. Quaternions and rotations*. zuletzt aufgerufen April 2015, https://www.cs.iastate.edu/~cs577/handouts/quaternion.pdf.

[23] R.E. KALMAN. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 2012.

[24] Hyon Lim, Jaemann Park, Deawon Lee, and H.J. Kim. Build your own quadrotor: open-source projects on unmanned aerial vehicles. *Robotics and Automation Magazin*, 2012.

[25] O. Loeffield. *Estimationstherie II*. Oldenburg Verlag, 1990.

[26] O.H. Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Internal Report*, 2010.

[27] Sebastian O.H. Madgwick, Andrew J.L Harrison, and Ravi Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. *IEEE International Conference on Rehabilitation Robotics*, 2011.

[28] R. Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on special orthogonal group. *HAL archives-ouverts*, 2010.

[29] Jao Luis Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee, and Michael J. Zyda. An extended kalman filter for quaternion-based orientation estimation using marg sensors. *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[30] Leonard A. McGee and Stanley F. Schmidt. Discovery of the kalman filter as a practical tool for aerospace and industry. *NASA Technical Memorandum 86847*, 1985.

[31] Oliver Meister. *Entwurf und Realisierung einer Aufklährungsplattform auf Basis eines unbemannten Minihelikopters mit autonomen Flugfähigkeiten*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2010.

[32] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro uav testbed. *Robotics and Automation Magazine, IEEE*, 2010.

[33] H.B. Mitchell. *Multi-Sensor Data Fusion: An Introduction*. Springer, 2007.

[34] Micheal R. Molt, Dale E. Schinstock, and Robert M. Caplinger. Gps-aided ins solution with photogrammetry validation. *Aerotech Congress and Exhibition*, 2005.

[35] R. Nelson. *Flight Stability and Automatic Control*. McGraw-Hill, 1989.

[36] Oliveira P. Pascoal A., Kaminer I. Navigation system design using time-varying complementary filters. *International Conference on Control Applications*, 1998.

[37] Roger M. du Piessis. Poor man's explenation of kalman filtering or how i stopped worrying and learned to love matrix inversion. *North American Rockwell Electronics Group*, 1967.

[38] H. William Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vertterling. *Numerical Recipes in C: The Art of Scientific Computing*. Camebridge University Press, 1992.

[39] OpenPilot Project. Open source uav autopilot. zuletzt aufgerufen April 2015, https://www.openpilot.org/.

[40] Peter Rüegg. Medal presented personally by obama. 2009. zuletzt aufgerufen März 2014, http://www.ethlife.ethz.ch/archive_articles/091008_kalman_per/index_EN.

[41] Colton Shane. The balance filter. 2007. zuletzt aufgerufen März 2014, http://robottini.altervista.org/wp-content/uploads/2014/04/filter.pdf.

[42] Wang Shoahua and Ying Yang. Quadrotor aircraft attitude estimation and control based on kalman filter. *Control Conference (CCC)*, 2012.

[43] David J. Siminovitch. *Rotations in NMR: Part I. Euler-Rodrigues Parameters and Quaternions*. WILEY, Lethbridge, Canada, 1997.

[44] Dan Simon. Kalman filtering. *Embedded Systems Programming*.

[45] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. WILEY, Hoboken, New Jersey, 2006.

[46] Fox Dieter Thrun Sebastian, Burgard Wolfram. *Propabilistic Robotics*. MIT Press, 2006.

[47] Greg Welch and Bishop Gary. An introduction to the kalman filter. *Department of Computer Science University of North Carolina*, 2012.

[48] J. Wendel, C. Schlaile, and Trommer F. Direct kalman filtering of gps/ins for aerospace applications. *IEEE Transactions on Aerospace and Electric Systems*, 2002.

[49] Jan Wendel. *Integrierte Navigationssysteme*. Oldenburg Verlag, München, 2011.

[50] Wikimedia. Apollo guidance computer ansicht nasa. zuletzt aufgerufen März 2015, commons.wikimedia.org/wiki/File:Agc_view.jpg.

[51] X-io Technologies. Open source imu and ahrs algorithms. 2012. zuletzt aufgerufen März 2014, www.x-io.co.uk/open-source-imu-and-ahrs-algorithms.

[52] Lukasz Zwirello. *Realization Limits of Impulse-Radio UWB Indoor Localization Systems*. Scientific Publishing, 2013.

[53] Helmholtz-Zentrum Potsdam. Department of Transportation (GFZ). http://www.gfz-potsdam.de.

[54] SWIFT NAVIGATION. Piksi low-cost high-performance GPS receiver with Real-Time Kinematics (RTK). http://www.swiftnav.com/piksi.html.