Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Jan Rüther

Entwicklung eines Systems zur Kartografie von Innenräumen mittels 2D-Bilddaten eines Quadrocopters

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Jan Rüther

Entwicklung eines Systems zur Kartografie von Innenräumen mittels 2D-Bilddaten eines Quadrocopters

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Marc Hensel
Zweitgutachter : Prof. Dr. -Ing. Dipl. -Kfm. Jörg Dahlkemper

Abgegeben am 2. Januar 2020

Jan Rüther

Thema der Masterthesis

Entwicklung eines Systems zur Kartografie von Innenräumen mittels 2D-Bilddaten eines Quadrocopters

Stichworte

Quadrocopter, Kartierung, probabilistische Filter, Erweiterter Kalman Filter SLAM, ArUco-Marker

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines Systems zur Kartografie von Innenräumen, welches als Messdaten lediglich die 2D-Bilddaten eines Quadrocopters verwendet. Zur Lösung des gleichzeitigen Lokalisierens und Kartierens wird eine EKF-SLAM Algorithmus im Verbindung mit künstlichen planaren Markern verwendet.

Jan Rüther

Title of the paper

Development of a system for interior cartography using 2D image data of a quadrocopter

Keywords

Quadrocopter, cartography, probabilistic filters, extended Kalman filter SLAM, ArUco-Marker

Abstract

This thesis deals with the development of a system for the cartography of interiors, which uses as measurement data only the 2D image data of a quadrocopter. An EKF-SLAM algorithm in combination with artificial planar markers is used to solve the problem of simultaneous localization and mapping.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation und Ziel

Die Navigation mit dem Handy im Freien gehört bereits seit Jahren zum alltäglichen Leben dazu. Mittels GPS ermöglichen Apps wie Google Maps oder Waze ein kostengünstiges und vor allem effizientes Navigieren in ländlichen sowie urbanen Umgebungen. Doch sobald ein Gebäude betreten wird, ist das GPS Signal meist nur noch schwach oder gar nicht mehr vorhanden. Dies gilt insbesondere für große Gebäude wie Krankenhäuser oder Ämter, in denen eine Navigationshilfe meist von enormem Vorteil wäre.
Eine mögliche Lösung für dieses Problem ist eine App, welche basierend auf den Kameradaten eines Handys Merkmale aus der Umgebung extrahiert und sich anhand dieser Merkmale in einer a-priori bekannten Karte des Gebäudes lokalisieren kann. Das Erstellen einer solchen Karte des Gebäudes unterliegt jedoch den selben Restriktionen wie das Navigieren an sich: Die Nutzung eines externen Systems zur Lokalisierung wie GPS während der Kartierung ist nicht möglich. Das System zur Kartierung muss sich demnach beim Kartieren der Umgebung gleichzeitig in der erstellten Karte lokalisieren. Dieses Problem wird in der Literatur als SLAM (Simultaneous Localization and Mapping) bezeichnet.
Für das Erstellen einer solchen Karte eignen sich unbemannte Luftfahrzeuge wie Quadrocopter besonders gut: Sie können in drei Dimensionen durch das Gebäude navigieren und verfügen meist schon über geeignete Sensorik wie Kameras oder Ultraschallsensoren, um ihr Umfeld wahrzunehmen.

Das Ziel dieser Arbeit ist es, die erste Teilaufgabe zur Umsetzung eines solchen Kartierungssystems für Gebäude zu entwickeln. Diese Teilaufgabe umfasst die Entwicklung eines grundlegenden Systems zur Kartierung von Innenräumen unter Verwendung eines SLAM-Algorithmus' mit Hilfe eines Quadrocopters. Besonders hierbei ist, dass hierfür ein kostengünstiger Quadrocopter verwendet wird, welcher als Sensorik zur Wahrnehmung der Umgebung nur über eine Kamera verfügt. Es ist demnach ein System zu entwickeln, welches basierend auf den Steuersignalen und den 2D-Bilddaten der Drohne eine Karte eines Innenraumes erstellt. Die Drohne wird hierbei von dem Benutzer per Hand gesteuert und durch den Raum navigiert, während mit Hilfe eines SLAM-Algorithmus' eine Karte des Raumes erstellt wird.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist in sieben Teile unterteilt. In Kapitel 1 wird die Motivation und das Ziel dargestellt. Im folgenden 2. Kapitel werden ausgewählte Grundlagen erläutert, welche zum Verständnis dieser Arbeit beitragen sollen. Kapitel 3 befasst sich mit dem aktuellen Stand der Technik. Hierzu gehören unter anderem verwandte Arbeiten zum Thema sowie aktuelle Lösungsansätze für in dieser Arbeit auftretende Probleme. Eine Analyse der Anforderungen wird in Kapitel 4 durchgeführt. Basierend auf diesen Anforderungen wird das Konzept und Design des zu entwickelnden Systems in Kapitel 5 entworfen. Die Genaue Umsetzung ist in Kapitel 6 festgehalten. In Kapitel 7 wird auf die Messergebnisse sowie die Erfüllung der Anforderungen eingegangen. Das abschließende 8. Kapitel reflektiert die Arbeit und gibt einen Ausblick auf weiteren Handlungsbedarf.

2 Grundlagen

Eine Auswahl an theoretischen Grundlagen der in dieser Arbeit behandelten Themengebiete wird in diesem Kapitel erläutert. Hierzu gehören unter anderem die Abbildungsmathematik einer Lochkamera sowie eine Einführung in probabilistische Filter. Abschließend werden zwei der gängigsten Algorithmen zum Kartieren und Lokalisieren eingeführt.

2.1 Kameramodell

Bei der Bildaufnahme wird der dreidimensionale Raum auf die zweidimensionale Bildebene abgebildet. Es kann nach [Jäh13] vereinfacht als Projektion aus einem drei- in den zweidimensionalen Raum betrachtet werden. Um die grundlegenden Aspekte dieser Projektion zu beschreiben, wird in [BLF16] als stark vereinfachtes Modell der Kamera das Lochkameramodell angenommen. Dieses Modell verfügt über eine Blende mit einem infinitesimal kleinen Loch. Parallel hierzu ist die Bildebene angeordnet. Die Lochblende lässt nur genau einen Lichtstrahl pro Lichtbündel durch, welches von einem Objektpunkt in der Umgebung ausgeht. In homogenen Koordinaten[1] kann dies mathematisch beschrieben werden durch

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \begin{pmatrix} -b & 0 & 0 & 0 \\ 0 & -b & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix}. \tag{2.1}$$

Die Variablen x'_c, y'_c und z'_c sind die Koordinaten des Objektpunktes im Kamerakoordinatensystem, welche auf die Koordinaten x' und y' im Bildkoordinatensystem abgebildet werden. b entspricht in dieser Notation der Bildweite, also dem Abstand von Lochblende zur Bildebene. Die Größe w_c ist hierbei der Skalierungsfaktor der homogenen Koordinaten und kann zu einem beliebigen Wert ungleich 0 gewählt werden. Häufig wird dieser einfach als 1 angenommen.

[1] Für die Darstellung in homogenen Koordinaten werden die kartesischen Koordinaten um eine Dimension erweitert. Vorteil dieser Darstellung ist es, dass affine Transformationen hierbei durch Matrixmultiplikationen dargestellt werden können.[BLF16]

Wird das Lochkameramodell um das Modell eines Bildsensors erweitert, so kann die Gleichung zum Ermitteln der Bildkoordinaten zu

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{b}{s_x} & 0 & c_x & 0 \\ 0 & -\frac{b}{s_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\boldsymbol{P}_i} \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix} \tag{2.2}$$

erweitert werden. Hierbei stehen c_x und c_y für den Versatz von optischer Achse zum Bildkoordinatensystem. Dieses Koordinatensystem hat meistens seinen Ursprung in der linken oberen Ecke des Bildes. Bei einem perfekt vor der Lochblende zentriert ausgerichteten Bildsensor würden diese Werte genau der Hälfte der jeweiligen Auflösung in x- und y-Richtung entsprechen. Die Werte s_x und s_y geben die Pixelgröße in der jeweiligen Richtung an. Die in der Matrix $\boldsymbol{P}_i$ enthaltenen Werte beschreiben die Eigenschaften der abbildenden Optik sowie des Bildsensors und werden als intrinsische Kameraparameter bezeichnet. Sollte keine Veränderung an dem optischen System vorgenommen werden, sind diese Parameter statisch.
Die Lage und Orientierung der Kamera im Raum wird durch die extrinsischen Kameraparameter in der Matrix $\boldsymbol{P}_e$ beschrieben. Diese Matrix besteht aus einer Translation beschrieben durch den Vektor $\boldsymbol{t}$ sowie einer Rotation beschrieben durch die Matrix $\boldsymbol{R}$. Die extrinsischen Kameraparameter beschreiben die Transformation eines Punktes aus dem Weltkoordinatensystem $\begin{pmatrix} x_w & y_w & z_w \end{pmatrix}^T$ in das Kamerakoordinatensystem $\begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$. Der Ursprung des neu eingeführten Weltkoordinatensystems kann beliebig im Raum gewählt werden, ist aber fest. Diese Transformation kann in homogenen Koordinaten durch

$$\begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix} = \underbrace{\begin{pmatrix} & \boldsymbol{R} & & \boldsymbol{t} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{P}_e} \begin{pmatrix} x'_w \\ y'_w \\ z'_w \\ w_w \end{pmatrix} \tag{2.3}$$

berechnet werden.
Aus (2.2) und (2.3) folgt die Gleichung zum Transformieren von Welt- in Bildkoordinaten:

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{b}{s_x} & 0 & c_x & 0 \\ 0 & -\frac{b}{s_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} & \boldsymbol{R} & & \boldsymbol{t} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{P}} \begin{pmatrix} x'_w \\ y'_w \\ z'_w \\ w_w \end{pmatrix} \tag{2.4}$$

2.1.1 Kamerakalibrierung

In der Praxis sind die intrinsischen Kameraparameter häufig nicht bekannt oder weichen durch Ungenauigkeiten in der Fertigung der Kamera von den errechneten Werten ab. Durch die Kamerakalibrierung können die intrinsischen Kameraparameter ermittelt werden. Für die Kalibrierung der Kamera wird nach [BLF16] wie folgt vorgegangen:
Zunächst werden von einem Kalibrierkörper, bei dem die Koordinaten von definierten Punkten zueinander bekannt sein müssen, eine oder mehrere Aufnahmen gemacht. Die definierten Punkte müssen im Bild identifiziert und lokalisiert werden. Es bietet sich demnach an, auffällige Strukturen zu verwenden, wie beispielsweise schachbrettartige Muster. Die Verwendung von Korrespondenzpunkten aus nur einer Ebene des Raums führt zu einem unterbestimmten Gleichungssystem der Kameraparameter. Deshalb wird entweder ein dreidimensionaler Kalibrierkörper verwendet oder es werden mehrere Aufnahmen eines planaren Kalibrierkörpers an unterschiedlichen Positionen im Raum gemacht. Das Ergebnis des ersten Schrittes sind demnach m korrespondierende Punkte im Welt- und Bildkoordinatensystem, für die gilt:

$$\boldsymbol{a}_j = \boldsymbol{P}_j \boldsymbol{a}_{wj}, \quad j = 1, \ldots, m \tag{2.5}$$

Die Projektionsmatrix $\boldsymbol{P}_j$ besteht hierbei aus den intrinsischen und den extrinsischen Kameraparametern. Wird keine Veränderung am optischen System der Kamera vorgenommen, so sind die intrinsischen Parameter konstant und die Gleichung (2.5) kann zu

$$\boldsymbol{a}_j = \boldsymbol{P}_i \boldsymbol{P}_{e,j} \boldsymbol{a}_{wj}, \quad j = 1, \ldots, m \tag{2.6}$$

erweitert werden. Über geeignete mathematische Verfahren können nun über die Minimierung einer Abbildungsfehlerfunktion die intrinsischen Kameraparameter ermittelt werden. [Han11; BLF16; SR14]

2.2 Probabilistische Filter

Viele der bekanntesten und am weitesten verbreiteten Algorithmen in der Robotik basieren auf probabilistischen Annahmen. Hierzu gehören unter anderem auch einige in dieser Arbeit verwendete Filteralgorithmen, mit deren Erläuterung sich dieses Kapitel befasst. Es soll ein grundlegendes Verständnis der angewandten Filter vermittelt werden. Für eine ausführliche Beschreibung und Herleitung aller folgenden Algorithmen sei auf [TBF05], [HLN12], [MD17] und [Kal60] verwiesen.

Darstellung von Systemen im Zustandsraum

Lineare zeitinvariante dynamische Systeme können durch lineare gewöhnliche Differentialgleichungen n-ter Ordnung beschrieben werden, wobei n die Systemordnung angibt und anschaulich für die Anzahl der Energiespeicher im System steht. In der Zustandsraumdarstellung wird die Differentialgleichung n-ter Ordnung in n Differentialgleichungen 1. Ordnung zerlegt. Im Folgenden wird davon ausgegangen, dass das System in der Zustandsraumdarstellung

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{2.7}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t). \tag{2.8}$$

vorliegt. Eine Beschreibung der Variablen ist der folgenden Aufzählung zu entnehmen:

$\boldsymbol{x}(t)$ n -dimensionaler Zustandsvektor. Enthält die Zustände zu Zeitpunkt t.

$\dot{\boldsymbol{x}}(t)$ n -dimensionaler Vektor. Ableitung des Zustandsvektors. Gibt die Änderung des Systems an.

$\boldsymbol{u}(t)$ m -dimensionaler Eingangsvektor. Enthält die eingehenden Eingangssignale zum Zeitpunkt t.

$\boldsymbol{y}(t)$ k -dimensionaler Ausgangsvektor. Enthält die Ausgangsgrößen des Systems zum Zeitpunkt t. Häufig sind diese identisch mit den messbaren Größen des Systems.

$\boldsymbol{A}$ $n \times n$ -dimensionale Systemmatrix. Gibt die internen Systemdynamiken an.

$\boldsymbol{B}$ $n \times m$ -dimensionale Eingangsmatrix. Gibt den Einfluss der Steuersignale auf die Zustände an.

$\boldsymbol{C}$ $k \times n$ -dimensionale Ausgangsmatrix. Gibt den Einfluss der Zustände auf die Ausgangsgrößen an.

$\boldsymbol{D}$ $k \times m$ -dimensionale Durchgriffsmatrix. Gibt den direkten Einfluss der Eingangssignale auf die Ausgangsgrößen an.

Da das in dieser Arbeit betrachtete System (Quadrocopter) über keinen Durchgriff verfügt, wird die Durchgriffsmatrix im Folgenden vernachlässigt.

Durch die Darstellung dynamischer Systeme im Zustandsraum ist die Analyse von Systemen nicht nur auf das Ein- und Ausgangsverhalten beschränkt, sondern kann auf die inneren Zustände des Systems erweitert werden.

2.2.1 Kalman Filter

Der Kalman Filter aus [Kal60] wird im Allgemeinen zur Fusion von zwei unterschiedlichen, stochastisch unabhängigen Informationen genutzt. In der Robotik sind dies häufig Aktionen $\boldsymbol{u}$ (basierend auf Odometriedaten[2]) und die Daten aus einer Messung $\boldsymbol{z}$.
Das Ergebnis des Filtervorgangs ist der geschätzte Systemzustand

$$\boldsymbol{x}_k = \begin{pmatrix} x_{1,k} & x_{2,k} & \dots & x_{n,k} \end{pmatrix}^T, \tag{2.9}$$

wobei n der Anzahl der Systemzustände entspricht und k den Abtastschritt angibt. Des Weiteren liefert die Filterung durch den Kalman Filter eine Kovarianzmatrix $\boldsymbol{\Sigma}_k$:

$$\boldsymbol{\Sigma}_k = \begin{pmatrix} \sigma_{x_1 x_1} & \dots & \sigma_{x_1 x_n} \\ \vdots & \ddots & \vdots \\ \sigma_{x_n x_1} & \dots & \sigma_{x_n x_n} \end{pmatrix}. \tag{2.10}$$

Diese Matrix gibt die Kovarianzen σ der Zustände untereinander an und ist ein Maß für die Unsicherheiten der geschätzten Systemzustände.

Der Kalman-Filter ist eine diskrete, mehrdimensionale Schätzung im Zustandsraum. Es wird demnach ein diskretes Zustandsraummodell benötigt. Eine exakte Diskretisierung des Systems aus (2.7) und (2.8) kann nach [Lun10] durch

$$\boldsymbol{A}_d = \boldsymbol{e}^{\boldsymbol{A}\Delta t} = \mathcal{L}^{-1}\{(s\boldsymbol{I} - \boldsymbol{A})^{-1}\} \mid_{t=\Delta t} \tag{2.11}$$

$$\boldsymbol{B}_d = \boldsymbol{A}^-1 \left(\boldsymbol{A}_d - \boldsymbol{I}\right) \boldsymbol{B} \tag{2.12}$$

$$\boldsymbol{C}_d = \boldsymbol{C} \tag{2.13}$$

[2] Schätzung der Pose des Systems basierend auf den Daten des Vortriebs. Ein Beispiel für ein solches System ist ein Inkrementalgeber. [HLN12]

erfolgen, wobei Δt die Abtastzeit des Systems, $\boldsymbol{e}$ die Matrixexponentialfunktion[3] und $\boldsymbol{I}$ die Einheitsmatrix ist. In diskreter Schreibweise ergeben sich (2.7) und (2.8) zu

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_d \boldsymbol{x}_k + \boldsymbol{B}_d \boldsymbol{u}_k \tag{2.14}$$

$$\boldsymbol{y}_{k+1} = \boldsymbol{C}_d \boldsymbol{x}_{k+1}. \tag{2.15}$$

Zusätzlich zu den Aktionen und Messungen werden die zugehörigen Fehlermodelle in Form von Kovarianzmatrizen benötigt. Für die Aktionen ist dies die Kovarianzmatrix des Prozessrauschens $\boldsymbol{\Sigma}_u$. Diese Matrix beschreibt das Rauschen der Systemzustände. Das Rauschen der Messungen wird in der Kovarianzmatrix $\boldsymbol{\Sigma}_z$ beschrieben.
Das Ziel der Filterung ist es, zu jedem Abtastschritt eine Schätzung des Systemzustands zu liefern. Für den geschätzten Systemzustand $\overline{\boldsymbol{x}}_{k+1}$ wird das zugehörige Fehlermodell als Kovarianzmatrix $\overline{\boldsymbol{\Sigma}}_{k+1}$ angegeben. Es kann somit zu jedem Zeitpunkt der geschätzte Zustand und die Unsicherheit dieses geschätzten Zustandes angegeben werden.
Der Filter arbeitet hierbei in zwei Phasen:

Prädiktion In jedem Abtastschritt wird der a-priori Systemzustand $\overline{\boldsymbol{x}}_{k+1}$ anhand des letzten Systemzustands $\boldsymbol{x}_k$ und der durchgeführten Aktionen $\boldsymbol{u}_k$ prädiziert. Zudem wird das a-priori Fehlermodell des geschätzten Zustandes $\overline{\boldsymbol{\Sigma}}_{k+1}$ mit Rücksichtnahme auf die Kovarianzmatrix des Prozessrauschens $\boldsymbol{\Sigma}_u$ aktualisiert.

Filterung Sollte eine Messung $\boldsymbol{z}_k$ vorliegen, so wird diese in den prädizierten Zustand integriert, sodass das Fehlermodell des aktuellen Zustands $\boldsymbol{\Sigma}_{k+1}$ basierend auf dem a-priori Fehlermodell $\overline{\boldsymbol{\Sigma}}_{k+1}$ minimiert wird. Dies führt dazu, dass anhand des a-priori Systemzustands $\overline{\boldsymbol{x}}_{k+1}$ der Zustand $\boldsymbol{x}_{k+1}$ bestimmt wird, der optimal zu den Aktionen und der Messung passt.

Unter der Annahme, dass alle Fehlermodelle Gaußverteilungen entsprechen, entspricht auch der geschätzte Zustand einer solchen Verteilung. Demnach kann der Zustand $\overline{\boldsymbol{x}}_k$ als Normalverteilung mit der Unsicherheit $\overline{\boldsymbol{\Sigma}}_k$ angesehen werden.
Der genaue Algorithmus geht wie folgt vor:

Schritt 1 Basierend auf dem Systemmodell und dem Steuersignal $\boldsymbol{u}_k$ wird a-priori der Folgezustand $\overline{\boldsymbol{x}}_{k+1}$ und seine Unsicherheit $\overline{\boldsymbol{\Sigma}}_{k+1}$ vorhergesagt:

$$\overline{\boldsymbol{x}}_{k+1} = \boldsymbol{A}_d \boldsymbol{x}_k + \boldsymbol{B}_d \boldsymbol{u}_k \tag{2.16}$$

$$\overline{\boldsymbol{\Sigma}}_{k+1} = \boldsymbol{A}_d \boldsymbol{\Sigma}_k \boldsymbol{A}_d^T + \boldsymbol{\Sigma}_u \tag{2.17}$$

Schritt 2 Sollte eine Messung vorliegen, wird die a-priori Schätzung $\overline{\boldsymbol{x}}_{k+1}$, $\overline{\boldsymbol{\Sigma}}_{k+1}$ aus Schritt 1 korrigiert. Es wird die Matrix $\boldsymbol{K}_k$ (Kalman-Gewinn) bestimmt, die die Kovarianz-

[3] Die Matrixexponentialfunktion ist analog zur skalaren Exponentialfunktion definiert durch $e^{\boldsymbol{X}} = \sum_{k=0}^{\infty} \frac{\boldsymbol{X}^k}{k!}$

matrix $\overline{\boldsymbol{\Sigma}}_{k+1}$ des geschätzten Zustands minimiert. Die Lösung dieses Optimierungsproblems ergibt sich zu:

$$\boldsymbol{K}_k = \overline{\boldsymbol{\Sigma}}_{k+1}\boldsymbol{C}_d^T \left(\boldsymbol{C}_d\overline{\boldsymbol{\Sigma}}_{k+1}\boldsymbol{C}_d^T + \boldsymbol{\Sigma}_z\right)^{-1}. \tag{2.18}$$

Die mit der Messung optimierte Zustandsschätzung sowie deren Unsicherheit können durch

$$\boldsymbol{x}_{k+1} = \overline{\boldsymbol{x}}_{k+1} + \boldsymbol{K}_k\left(\boldsymbol{z}_k - \boldsymbol{C}_d\overline{\boldsymbol{x}}_{k+1}\right) \tag{2.19}$$

$$\boldsymbol{\Sigma}_{k+1} = \overline{\boldsymbol{\Sigma}}_{k+1} - \boldsymbol{K}_k\boldsymbol{C}_d\overline{\boldsymbol{\Sigma}}_{k+1} \tag{2.20}$$

errechnet werden.

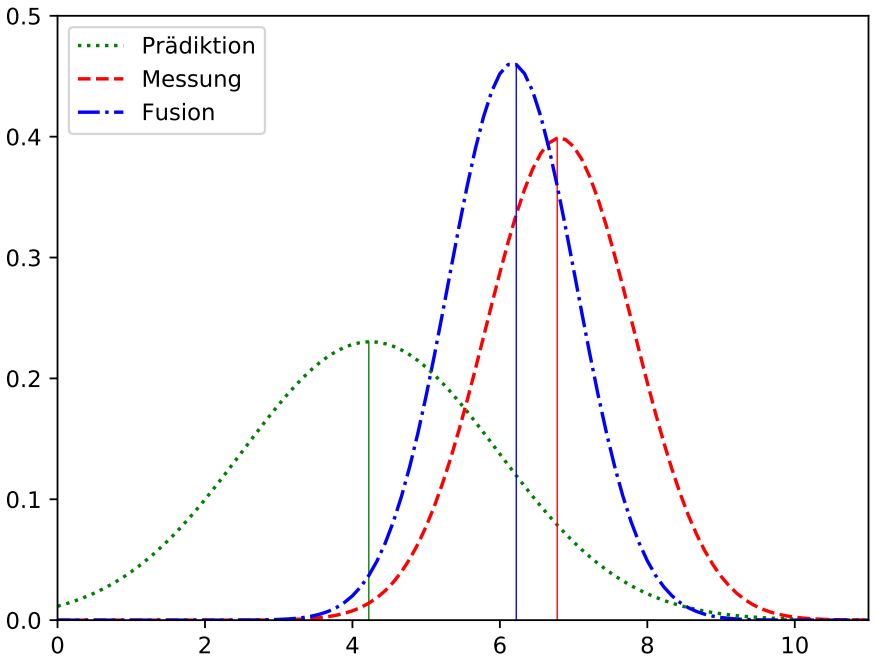


Abbildung 2.1: Verhalten der Gaußverteilungen bei Kalman Filterung

Ein exemplarischer Filtervorgang ist in Abbildung 2.1 anhand eines eindimensionalen Beispiels dargestellt. Der Graph links in der Abbildung beschreibt den prädizierten Zustand und dessen Unsicherheit nach Schritt 1 des Algorithmus. Der rechte Graph bildet die Messung samt Unsicherheit ab. Die optimale Filterung der Messung und Prädiktion ist im blauen Graphen zu sehen. Der resultierende Erwartungswert liegt näher an der Messung als an der

Prädiktion, da die Messung eine geringere Varianz aufweist. Die Unsicherheit des resultierenden Graphen ist geringer als die von Prädiktion und Messung.

Wie bereits erwähnt, handelt es sich bei dem Kalman Filter um einen optimalen Filter. Die vorhandenen Informationen werden optimal ausgenutzt. Kein anderes Verfahren schätzt den Systemzustand mit einer geringeren Abweichung zum realen System unter Voraussetzung folgender Bedingungen:

1. Das Systemrauschen ist mittelwertfrei und normalverteilt.
2. Die Fehlerquellen sind stochastisch unabhängig.
3. Das System ist linear. Es liegt in der Form von (2.7) und (2.8) vor.

Der Kalman Filter ist auch anwendbar, sollten diese Bedingungen nicht erfüllt sein. Die Berechnungen ergeben weiterhin Sinn. Die Filterung der Daten ist jedoch nicht mehr beweisbar optimal.

Ein großer Nachteil dieses Verfahrens ist es, dass nur Schätzungen für lineare Systeme vorgenommen werden können. Im folgenden Kapitel wird eine Erweiterung des Kalman Filters vorgestellt, mit der es möglich ist, die Zustände nichtlinearer Systeme zu schätzen.

2.2.2 Erweiterter Kalman Filter

Der zuvor eingeführte Kalman Filter liefert unter gewissen Bedingungen eine nachweisbar optimale Zustandsschätzung für lineare Systeme, welche durch die Gleichungen (2.7) und (2.8) beschrieben werden können. Um dieses Konzept auf nichtlineare Systeme zu erweitern, können diese Nichtlinearitäten durch eine Taylor-Entwicklung erster Ordnung (Jacobi-Matrizen) approximiert werden. Dieses Verfahren wird als Erweiterter Kalman Filter (EKF) bezeichnet. Das lineare System, beschrieben durch die Matrizen $\boldsymbol{A}_d$, $\boldsymbol{B}_d$ und $\boldsymbol{C}_d$ (vgl. (2.14), (2.15)), wird durch ein nichtlineares System ersetzt, welches durch die nichtlinearen Funktionen $\boldsymbol{g}$ und $\boldsymbol{h}$ beschrieben werden kann. Das System ergibt sich zu:

$$\boldsymbol{x}_{k+1} = \boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) \tag{2.21}$$

$$\boldsymbol{y}_{k+1} = \boldsymbol{h}\left(\boldsymbol{x}_{k+1}\right). \tag{2.22}$$

Der Algorithmus sowie (2.17) bis (2.20) bleiben gültig, es wird lediglich die Formel für die Prädiktion des Zustandes (2.16) durch ihr nichtlineares Äquivalent (2.21) sowie die Matri-

zen $\boldsymbol{A}$ und $\boldsymbol{C}$ durch die Jacobi-Matrizen

$$\boldsymbol{G}_k = \frac{\delta \boldsymbol{g}}{\delta \boldsymbol{x}} (\boldsymbol{x}_k, \boldsymbol{u}_k) = \begin{pmatrix} \frac{\delta g_1}{\delta x_1} & \cdots & \frac{\delta g_1}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta g_n}{\delta x_1} & \cdots & \frac{\delta g_n}{\delta x_n} \end{pmatrix}_{(\boldsymbol{x}_k, \boldsymbol{u}_k)} \quad \text{und} \tag{2.23}$$

$$\boldsymbol{H}_k = \frac{\delta \boldsymbol{h}}{\delta \boldsymbol{x}} (\overline{\boldsymbol{x}}_{k+1}) \quad = \begin{pmatrix} \frac{\delta h_1}{\delta x_1} & \cdots & \frac{\delta h_1}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta h_m}{\delta x_1} & \cdots & \frac{\delta h_m}{\delta x_n} \end{pmatrix}_{(\overline{\boldsymbol{x}}_{k+1})} \tag{2.24}$$

ersetzt. Der erweiterte Algorithmus besteht weiterhin aus zwei Schritten:

Schritt 1 Prädiktion des Systemzustands anhand der Aktion $\boldsymbol{u}_k$:

$$\overline{\boldsymbol{x}}_{k+1} = \boldsymbol{g} (\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.25}$$

$$\overline{\boldsymbol{\Sigma}}_{k+1} = \boldsymbol{G}_k \boldsymbol{\Sigma}_k \boldsymbol{G}_k^T + \boldsymbol{\Sigma}_u \tag{2.26}$$

Schritt 2 Berechnung des Kalman-Gewinns und Korrektur anhand der Messung $\boldsymbol{z}_k$:

$$\boldsymbol{K}_k = \overline{\boldsymbol{\Sigma}}_{k+1} \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \overline{\boldsymbol{\Sigma}}_{k+1} \boldsymbol{H}_k + \boldsymbol{\Sigma}_z \right)^{-1} \tag{2.27}$$

$$\boldsymbol{x}_{k+1} = \overline{\boldsymbol{x}}_{k+1} + \boldsymbol{K}_k \left(\boldsymbol{z}_k - \boldsymbol{h} \left(\overline{\boldsymbol{x}}_{k+1} \right) \right) \tag{2.28}$$

$$\boldsymbol{\Sigma}_{k+1} = \overline{\boldsymbol{\Sigma}}_{k+1} - \boldsymbol{K}_k \boldsymbol{H}_k \overline{\boldsymbol{\Sigma}}_{k+1} \tag{2.29}$$

Wie bereits im Abschnitt 2.2.1 erwähnt, berechnet der Kalman Filter unter gewissen Bedingungen beweisbar die optimale Fusion von prädizierten Werten und den Messwerten. Da hier ein nichtlineares System approximativ durch Jacobi-Matrizen linearisiert wird, sind die Ergebnisse nicht beweisbar optimal, ergeben aber weiterhin Sinn. [TBF05; HLN12]

2.2.3 Partikelfilter

Der Partikelfilter, auch Monte-Carlo Filter genannt, wird nach [HLN12] in der Informatik insbesondere dort eingesetzt, wo eine exakte Lösung des Problems nur sehr aufwendig zu berechnen ist, es jedoch akzeptabel ist, dies durch Berechnen von speziellen Lösungen stichprobenartig zu approximieren. In dieser Arbeit wird der Partikelfilter dahingehend betrachtet, dass er, ebenso wie der zuvor behandelte Erweiterte Kalman Filter, eine Zustandsschätzung für ein nichtlineares System liefert.
Der Partikelfilter generiert hierfür eine Menge an Zustandsschätzungen, so genannte Partikel. Jedes Partikel für sich steht für eine separate Schätzung des Zustands. Zudem enthält jedes Partikel ein Gewicht, welches angibt, wie gut der geschätzte Zustand zum aktuellen Zustand passt. Das Gewicht des Partikels wird über eine Messung bestimmt, welche den

geschätzten Systemzustand des Partikels mit dem aktuellen Systemzustand vergleicht. Je besser die Messung zum geschätzten Zustand passt, desto größer ist das Gewicht des Partikels.

Algorithmus 1 Partikelfilter($\boldsymbol{\chi}_k, \boldsymbol{u}_k, \boldsymbol{z}_k$)

1: $\overline{\boldsymbol{\chi}}_{k+1} = \boldsymbol{\chi}_{k+1} = \emptyset$
2: $M =$ **length**$(\boldsymbol{\chi}_k)$
3: **for all** $\boldsymbol{x}_k^{[m]}$ **in** $\boldsymbol{\chi}_k$ **do**
4: sample $\boldsymbol{x}_{k+1}^{[m]} \sim P\left(\boldsymbol{x}_{k+1} \mid \boldsymbol{u}_k, \boldsymbol{x}_k^{[m]}\right)$
5: $w_{k+1}^{[m]} = P(\boldsymbol{z}_k \mid \boldsymbol{x}_{k+1}^{[m]})$
6: $\overline{\boldsymbol{\chi}}_{k+1} = \overline{\boldsymbol{\chi}}_{k+1} \cup \left(\boldsymbol{x}_{k+1}^{[m]}, \quad w_{k+1}^{[m]}\right)$
7: **end for**
8: **for** $m = 1$ **to** M **do**
9: draw i with propability $\propto w_{k+1}^i$
10: $\boldsymbol{\chi}_{k+1} = \boldsymbol{\chi}_{k+1} \cup \boldsymbol{x}_{k+1}^i$
11: **end for**
12: **return** $\boldsymbol{\chi}_{k+1}$

Der leicht adaptierte Algorithmus 1 aus [TBF05] beschreibt das Vorgehen bei der Anwendung des Partikelfilters.
Die Schleife von Zeile 3 bis Zeile 7 sorgt für das „Streuen“ neuer Partikel. Für jedes vorhandene Partikel in der Menge $\boldsymbol{\chi}_k$ wird ein neuer Zustand $\boldsymbol{x}_{k+1}^{[m]}$ und eine neue Gewichtung $w_{k+1}^{[m]}$ ermittelt. Das m steht hierbei für das m-te Partikel aus der Eingangsmenge. Die Berechnung des Zustands $\boldsymbol{x}_{k+1}^{[m]}$ in Zeile 4 erfolgt hierbei ähnlich wie beim EKF über die Systemdynamik, beschrieben durch (2.21). Anders hierbei ist jedoch, dass keine reine Prädiktion ausgeführt wird, sondern die prädizierten Zustände der Partikel mit einem Rauschen beaufschlagt werden, welches einer Zufallsverteilung unterliegt.
In der Zeile 5 wird das Gewicht des Partikels ermittelt. Hierbei wird die Messung $\boldsymbol{z}_k$ in die Partikelmenge integriert. Die Wahrscheinlichkeit $P\left(\boldsymbol{z}_k \mid \boldsymbol{x}_{k+1}^{[m]}\right)$ ist ein Maß dafür, wie gut der Zustand des Partikels zur aktuellen Messung passt.
Aus der so ermittelten Partikelmenge $\overline{\boldsymbol{\chi}}_{k+1}$ werden in der Schleife von Zeile 8 bis 11 M Partikel für die Ergebnismenge $\boldsymbol{\chi}_{k+1}$ ausgewählt. Die Auswahl der Partikel basiert auf einer Wahrscheinlichkeit, die proportional zum Gewicht des Partikels ist. Die Ergebnismenge enthält demnach tendenziell eine größere Anzahl an Partikeln mit höherem Gewicht.

2.3 Mobile Robotik

Um die in späteren Kapiteln erläuterten Algorithmen und deren Notationen besser zu verstehen, behandelt dieses Kapitel ausgewählte Grundlagen der mobilen Robotik. Alle Grundlagen aus diesem Kapitel basieren auf [HLN12] und [TBF05].

2.3.1 Karten

Bevor im nachfolgenden Abschnitt verschiedene Typen von Landmarken genauer erläutert werden, gibt dieser Abschnitt einen Überblick über die verschiedenen Möglichkeiten, Umgebungen in Karten zu repräsentieren.
Eine Karte in der Robotik ist eine explizite Repräsentation eines Referenzsystems, welche darauf ausgelegt ist, dass der Roboter sie effizient für die jeweilige Aufgabe nutzen kann. Dies bedeutet, dass sie nicht notwendigerweise für den Menschen lesbar sein muss. Vielmehr sind solche Karten auf die Anwendung und die Sensorik des Roboters zugeschnitten.
Auch wenn es viele verschiedene Arten von Karten gibt, können in der Robotik zwei grundlegende Kartentypen voneinander unterschieden werden:

Topologische Karten: Bei dieser Art von Karten stehen nicht die metrischen Relationen der Referenzsysteme im Vordergrund, sondern deren Beziehungen zueinander. Häufig wird diese Art von Karten durch Graphen realisiert. Die Knoten der Graphen enthalten Informationen zu Landmarken oder Sensordaten. Die Kanten beschreiben die Beziehung der Daten zueinander. Kanten können unter anderem Informationen wie den Abstand von zwei Knoten oder deren Sichtbarkeit enthalten.
Ein Beispiel für eine solche Karte ist der Plan des öffentlichen Verkehrsnetzes in Hamburg (Abb. 2.2). Knoten enthalten hierbei die Informationen von Landmarken: Jeder Knoten steht für einen Bahnhof. Die Kanten geben die Erreichbarkeit zwischen den Bahnhöfen an. Metrische zusammenhänge der Bahnhöfe sind hier nur grob abgebildet. Der Anwendungsfall für solche Karten ist meiste eine Start-Ziel Pfadplanung.

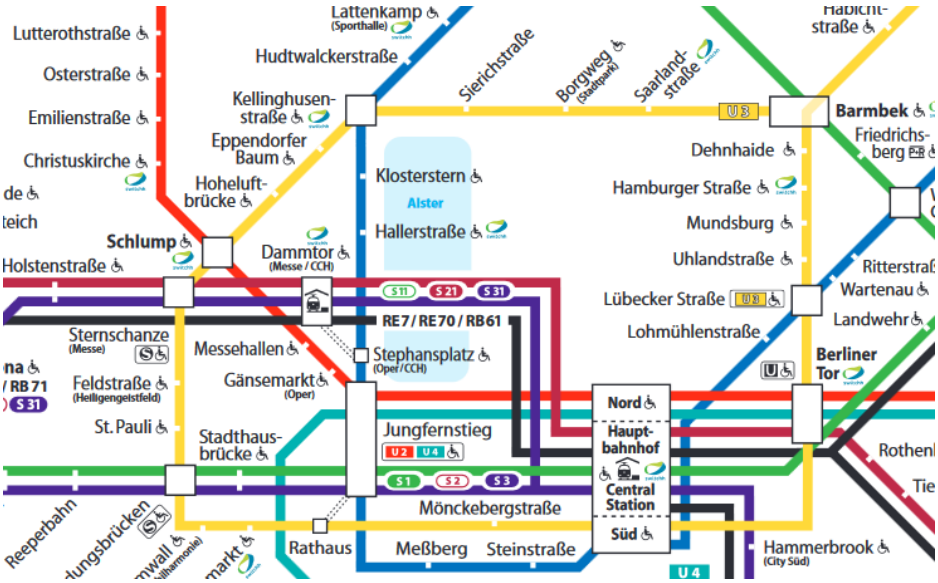
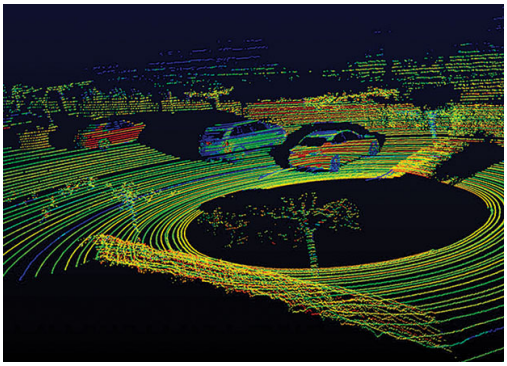
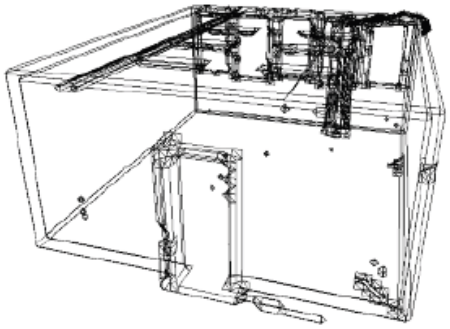


Abbildung 2.2: Auschnitt aus dem öffentlichen Verkehrsnetz Hamburgs [HVV19]

Geometrische Karten: Diese Art von Karten basieren auf den metrischen Relationen des Referenzsystems. Diese können als reine Sensordaten vorliegen oder als aus diesen Sensordaten extrahierte syntaktische Merkmale.
Die in Abbildung 2.3a dargestellte Messung eines Parkplatzes ist ein Beispiel für eine geometrische Karte basierend auf den reinen Sensordaten eines 3D-Lidar Sensors. Alle Messwerte sind in Form einer Punktwolke dargestellt.
Eine Karte basierend auf syntaktischen Merkmalen ist in Abbildung 2.3b dargestellt. Die Sensordaten wurden vorverarbeitet und nur die im Raum vorhandenen Kanten werden in der Karte abgebildet.



(a) Geometrische Karte basierend auf den Daten einer 3D-Lidar Messung [Ack16]



(b) Geometrische Karte basierend auf syntaktischen Merkmalen [TBF05]

Abbildung 2.3: Geometrische Karten

2.3.2 Landmarken

Als Landmarke bezeichnet man in der Umgangssprache ein eindeutig zuzuordnendes Merkmal, welches feststehend und meist gut sichtbar positioniert ist. Bei bekannter Position der Landmarke kann sie zur Lokalisierung verwendet werden. Für einen Roboter trifft dies ebenfalls zu. Eine Lokalisierung des Roboters anhand einer Landmarke ist möglich, sollte eine Karte der Umgebung vorliegen, in der die Landmarke verzeichnet ist. Bedingung hierfür ist, dass der Roboter die Landmarke mit Hilfe seiner Sensorik erkennen und mit der zugehörigen Landmarke in der Karte assoziieren kann.
Bei Landmarken wird zwischen künstlichen und natürlichen Landmarken unterschieden.

Künstliche Landmarken

Künstliche Landmarken sind speziell in der Umgebung errichtet, um zur Orientierung zu dienen. Sie sind meist speziell auf die Sensorik des Roboters abgestimmt um möglichst effizient erkannt zu werden. Ein Vorteil solcher Landmarken ist, dass sie dem Roboter häufig noch zusätzliche Informationen liefern wie beispielsweise eine einzigartige Signatur für jede Landmarke. Je nach verfügbarer Sensorik des Roboters gibt es die unterschiedlichsten Formen von künstlichen Landmarken.
In [PGB18] werden als aktive Landmarken Bluetooth Low Energy (BLE) Sender verwendet. Anhand der Stärke des empfangenen Signals kann die Entfernung von Roboter zum BLE-Sender bestimmt werden. Eine eindeutige Zuordnung des Senders ist über dessen Hardware-ID möglich, welche vom Sender an den Empfänger übertragen wird. Voraussetzung für eine fehlerfreie Messung der Entfernung ist eine konstante Übertragungsleistung des BLE-Senders. Die Spannungsversorgung bei aktiven Landmarken ist meist über Batterien realisiert, was dazu führen kann, dass mit der Zeit durch sinkende Versorgungsspannung die Sendeleistung der BLE-Sender abnimmt. Bei kleineren Singalstärken der Sender steigt die gemessene Entfernung von Roboter zu Sender, was zu einer fehlerhaften Entfernungsmessung führt.
Abhilfe bei schwankenden Signalstärken schaffen passive Landmarken wie die in [Mar15] verwendeten RFID-Transponder. Die Transponder verfügen ebenfalls über eine eindeutige ID, welche vom Empfänger ausgelesen werden kann. Die Entfernung von Roboter zu RFID-Transponder wird über die Signalstärke und die Phasenverschiebung des empfangenen Signals ermittelt.
Beide dieser künstlichen Landmarken haben den Nachteil, dass die Messung nur die Entfernung von Roboter zu Landmarke liefert. Landmarken, die neben der Entfernung noch weitere Informationen wie Rotation und Translation des Roboters zur Landmarke liefern, sind planare Marker wie ArUco-Marker [GJ+14] in Abbildung 2.4a, AprilTags [WO16] in Abbildung 2.4b oder ARToolKit [KB99] in Abbildung 2.4c. Mit diesen speziell für Kamerasysteme entworfenen planaren Landmarken können neben der 3D-Position des Roboters zur Landmarke auch dessen Rotation um alle drei Achsen bestimmt werden.

(a) ArUco (b) AprilTag (c) ARToolKit

Abbildung 2.4: Planare Marker

Natürliche Landmarken

Anders als bei den künstlichen Landmarken ist bei den natürlichen Landmarken kein Eingriff in die Umgebung nötig. Als Landmarken werden signifikante Merkmale aus der Umgebung genutzt.
In dem Konzept aus [IIJ16] werden in einer urbanen Umgebung Häuserecken als Landmarken verwendet. Ein 3D-LIDAR liefert Messdaten, anhand derer zunächst Linien aus der Umgebung extrahiert werden. Wie in Abbildung 2.5 zu sehen ist, befindet sich eine Ecke genau dort, wo sich zwei dieser detektierten Linien treffen. Um sicherzustellen, dass es sich bei der detektierten Ecke auch um eine Häuserecke handelt, wird eine Ecke erst als Häuserecke angesehen, wenn mehrere einzelne Ecken übereinander auftreten.
Eine robuste Methode um aus 2D-Bilddaten Landmarken zu extrahieren, ist der in [Low04] beschriebene Scale Invariant Feature Transformation (SIFT) Algorithmus. Um markante Bildpunkte ausfindig zu machen, wird das Bild in den Skalenraum transformiert. Hierzu wird das Bild Gauß-gefiltert und die Differenzbilder aufeinanderfolgender Stufen gebildet (Difference of Gaussians, DoG), was einer Approximation des Laplace-of-Gaussian (LoG) Filter zur Kantendetektion entspricht, jedoch schneller berechnet werden kann. Dieses Vorgehen wird für kleiner werdende Skalierungen des Eingangsbildes wiederholt. Anschließend werden markante Punkte anhand von Extremwerten ermittelt. Um zu jedem markanten Punkt Ort, Orientierung und Größe zu bestimmen, wird ein Modell anhand der näheren Umgebung gefittet. Ausgerichtet an diesem Modell wird die Beschreibung des markanten Punktes in einem 128-dimensionale Merkmalsraum ermittelt. Die aus dem Algorithmus resultierenden Landmarken sind weitestgehend invariant gegenüber perspektivischen Verzerrungen, Beleuchtungsschwankungen und Koordinatentransformationen wie Rotation oder Translation.

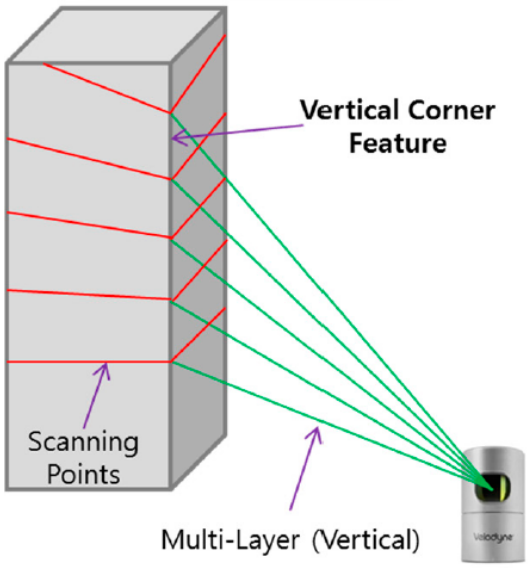


Abbildung 2.5: Schematische LIDAR-Messung eines Hauses [IIJ16]

2.3.3 SLAM

In Abschnitt 2.3.1 wurden die geometrischen Karten vorgestellt. Häufige Einsatzgebiete für solche Karten sind Pfadplanung oder Lokalisierung von Robotern. Bevor ein Roboter jedoch eine solche Karte verwenden kann, muss diese zunächst für das vorgesehene Einsatzgebiet erstellt werden. Eine händische Erstellung einer solchen Karte ist meist mit hohem Aufwand verbunden. Bei Innenräumen hingegen bietet sich die Möglichkeit an, die Karte basierend auf Bauplänen zu erstellen.
Einige Anwendungsfälle hingegen bieten nicht den Vorteil einer a-priori bekannten Karte. In diesen Fällen muss der Roboter selber eine Karte der Umgebung erstellen, die er fortlaufend aktualisiert und in der er sich gleichzeitig lokalisiert. In der Literatur [TBF05; HLN12] wird häufig von einem Henne-Ei Problem gesprochen: Um eine exakte Karte der Umgebung zu erstellen, muss der Roboter sich präzise lokalisieren können; um sich präzise zu lokalisieren, benötigt der Roboter eine exakte Karte der Umgebung.
Das Problem der gleichzeitigen Lokalisierung und Kartierung wird SLAM (engl. simultaneous localization and mapping) genannt. Beide Teilprobleme einzeln gesehen lassen sich relativ einfach lösen. Die Roboterlokalisierung bei gegebener Karte und die Kartierung bei bekannter Roboterpose sind nicht Teil dieser Arbeit. Grundlagen hierzu können aber in [HLN12] und [TBF05] nachgelesen werden. Das gleichzeitige Lösen beider Probleme wird in diesem Kapitel behandelt.

Überblick

Als Eingaben für die SLAM-Algorithmen stehen die Steuereingriffe $\boldsymbol{u}_k$ sowie die Messungen $\boldsymbol{z}_k$ des Roboters zur Verfügung. Gesucht ist der Pfad des Roboters inklusive aktueller Pose $\boldsymbol{x}_{k:1}$ sowie die Schätzung der Karte $\boldsymbol{m}$. Bei der Verarbeitung der Sensordaten des Roboters können die Algorithmen in zwei Arten unterschieden werden:

Direktes SLAM Die Messung des Roboters besteht aus den rohen Sensordaten. Die Algorithmen arbeiten bei Kamerabildern direkt mit den Pixelintensitäten des Bildes.

Indirektes SLAM Die Messungen des Roboters bestehen aus Landmarken, welche aus den Sensordaten extrahiert werden. Bei Laserscans sind dies detektierte Linien oder Ecken. In Kamerabildern können *SIFT* Merkmale verwendet werden (vgl. Kapitel 2.3.2).

Für jede dieser Methoden existieren bei der Messung im Idealfall Tiefen- und Richtungsinformationen. Da nicht jeder Roboter mit einem Sensor ausgestattet ist, der beide Informationen liefert, kann bei der Art der Messung ebenfalls eine Unterscheidung vorgenommen werden:

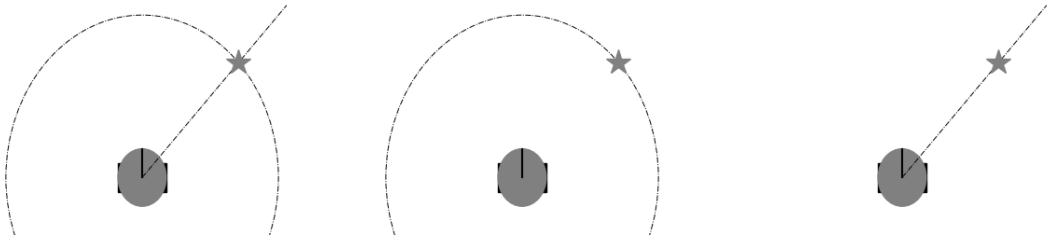
Tiefen- und Richtungsinformationen Die Messung enthält Tiefen- und Richtungsinformationen. Die Position des gemessenen Objekts kann relativ zum Roboter genau bestimmt werden (Abb. 2.6a). Beispiele für solche Sensoren sind RGB-D Kameras, Ultraschall- oder LIDAR-Sensoren.

Nur Tiefeninformationen Die Messung liefert nur die Tiefeninformationen zum gemessenen Objekt. Die Position des Objektes kann auf einen Radius um den Roboter eingegrenzt werden (Abb. 2.6b). Ein Beispiel hierfür ist eine WLAN-Laufzeitmessung.

Nur Richtungsinformationen Die Messung liefert nur die Richtungsinformationen. Die Position des gemessenen Objektes liegt auf einer Linie ausgehend von dem Roboter (Abb. 2.6c). Ein Beispiel für solch einen Sensor ist eine Kamera.

Ein weiteres Problem ist die Datenassoziation: Sind zwei verschiedene Landmarken nicht eindeutig voneinander zu unterscheiden, hat dies schwerwiegende Auswirkungen auf den Algorithmus. Wird eine Landmarke fälschlicherweise mit einer anderen assoziiert, so ist die Schätzung der Pose und somit auch die der Karte nicht nur fehlerhaft, sondern gänzlich inkorrekt.

In den nachfolgenden Erläuterungen wird davon ausgegangen, dass es sich um ein indirektes SLAM mit Tiefen- und Richtungsinformationen handelt. Zudem wird davon ausgegangen, dass jede Landmarke über eine individuelle Signatur verfügt, über die sie zugeordnet werden kann.
Für diese Form des SLAMs liegt für jede sichtbare Landmarke die Richtung und Entfernung zu dieser Landmarke durch die Messung vor. Basierend auf dieser Messung und der



(a) Tiefen- und Richtungsmessung (b) Nur Tiefenmessung (c) Nur Richtungsmessung

Abbildung 2.6: Verschiedene Sensortypen für SLAM-Algorithmen

aktuellen Positionsschätzung des Roboters wird die globale Lage der Landmarke errechnet und in der Karte eingetragen. Hierbei wirken sich die Ungenauigkeiten in der Positionsschätzung und der Messung auf die Genauigkeit der kartierten Landmarke aus. Da diese Ungenauigkeiten sich mit fortschreitendem Vermessen des Areals akkumulieren, kann der Fehler prinzipiell unbegrenzt wachsen. Sobald der Roboter jedoch eine bereits kartierte Landmarke wahrnimmt, kann er seine aktuelle Pose korrigieren. Es ist hierbei sogar je nach Algorithmus möglich, dass zurückliegende Landmarken und Posen des Roboters nachträglich korrigiert werden.

EKF-SLAM

Die historisch gesehen erste Lösung des SLAM-Problems basiert auf dem Erweiterten Kalman Filter wie er in Kapitel 2.2.2 beschrieben ist. Der im Folgenden aufgeführte Algorithmus ist aus [TBF05] entnommen und wurde leicht adaptiert.
Der Zustandsvektor des Kalman Filters setzt sich aus der Position des Roboters $\boldsymbol{x}_{r,k}$ und den Positionen der Landmarken $\boldsymbol{x}_{m,k}$ zusammen. Für einen Roboter mit dem Freiheitsgrad 3 (x, y, φ) und den Landmarkenpositionen bestehend aus x- und y-Koordinaten gilt demnach:

$$\boldsymbol{x}_k = \begin{pmatrix} \boldsymbol{x}_{r,k} \\ \boldsymbol{x}_{m,k} \end{pmatrix} \tag{2.30}$$

$$= \begin{pmatrix} x_k & y_k & \varphi_k & m_{1x,k} & m_{1y,k} & m_{2x,k} & m_{2y,k} & \dots & m_{Nx,k} & m_{Ny,k} \end{pmatrix}^T . \tag{2.31}$$

x_k, y_k und φ_k stehen hierbei für die Pose des Roboters zum Zeitpunkt k; $m_{i,x}$ und $m_{i,y}$ stehen für die Position der i-ten Landmarke. Die Dimension des Zustandsvektors ist $2N+3$, wobei N die Anzahl an Landmarken in der Karte angibt. Die Kovarianzmatrix $\boldsymbol{\Sigma}_k$ ergibt sich somit

zu:

$$\boldsymbol{\Sigma}_k = \left(\begin{array}{ccc|ccccc} \sigma_{xx} & \sigma_{xy} & \sigma_{x\varphi} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{N,x}} & \sigma_{xm_{N,y}} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\varphi} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{ym_{N,x}} & \sigma_{ym_{N,y}} \\ \sigma_{\varphi x} & \sigma_{\varphi y} & \sigma_{\varphi\varphi} & \sigma_{\varphi m_{1,x}} & \sigma_{\varphi m_{1,y}} & \cdots & \sigma_{\varphi m_{N,x}} & \sigma_{\varphi m_{N,y}} \\ \hline \sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{m_{1,x}\varphi} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{N,x}} & \sigma_{m_{1,x}m_{N,y}} \\ \sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{m_{1,y}\varphi} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{N,x}} & \sigma_{m_{1,y}m_{N,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{N,x}x} & \sigma_{m_{N,x}y} & \sigma_{m_{N,x}\varphi} & \sigma_{m_{N,x}m_{1,x}} & \sigma_{m_{N,x}m_{1,y}} & \cdots & \sigma_{m_{N,x}m_{N,x}} & \sigma_{m_{N,x}m_{N,y}} \\ \sigma_{m_{N,y}x} & \sigma_{m_{N,y}y} & \sigma_{m_{N,y}\varphi} & \sigma_{m_{N,y}m_{1,x}} & \sigma_{m_{N,y}m_{1,y}} & \cdots & \sigma_{m_{N,y}m_{N,x}} & \sigma_{m_{N,y}m_{N,y}} \end{array}\right). \tag{2.32}$$

Der Index k für den aktuellen Abtastschritt wurde innerhalb der Kovarianzmatrix aus Gründen der Übersicht weggelassen. Die entstehende Kovarianzmatrix kann in drei verschiedene Typen von Kovarianzen aufgeteilt werden. Der obere linke 3×3 Abschnitt der Matrix repräsentiert die Unsicherheit in der aktuellen Roboterpose. Die $2N \times 2N$ Teilmatrix unten rechts gibt die Unsicherheiten der Karte an, da hier nur die Kovarianzen der Marken untereinander angegeben sind. Der obere rechte sowie der untere linke $3 \times 2N$ bzw. $2N \times 3$ Teil der Matrix korreliert die Landmarken mit den Roboterposen. Diese beiden Teile der Matrix sind identisch und lediglich transponiert zueinander.

Der grobe Ablauf des EKF-SLAM ist sehr ähnlich zu dem des EKFs und wird im Folgenden kurz erklärt.
Wie beim EKF wird der neue Systemzustand anhand der Gleichungen der Systemdynamik prädiziert. Hierbei ist jedoch nur eine Prädiktion für die Zustände der Drohne nötig. Die Zustände in $\boldsymbol{x}_k$, welche die Landmarkenpositionen angeben, unterliegen keiner Dynamik und müssen deshalb nicht prädiziert werden.
Anschließend wird für jede gemessene Landmarke separat der Kalman-Gewinn ermittelt und verrechnet. Hierfür wird zunächst anhand des prädizierten Systemzustands die zu erwartende Messung der Landmarke bestimmt. Der Kalman-Gewinn wird dann analog zum EKF mit den selben Gleichungen ermittelt und mit den aktuellen Systemzuständen sowie deren Kovarianz verrechnet.
Sollte eine Landmarke zuvor noch nicht gesehen worden sein, so werden Zustandsvektor und Kovarianzmatrix erweitert und die neue Landmarke initialisiert.

Fast-SLAM

Der EKF-SLAM verwendet zur Darstellung der Unsicherheiten von Pose und Karte eine Normalverteilung, repräsentiert durch Mittelwert und Kovarianzen. Beim Fast-SLAM Algorithmus wird ein auf Partikelfilter basierter Ansatz verfolgt. Eine direkte Implementie-

rung des Partikelfilter-SLAMs wäre nach [TBF05] jedoch zum Scheitern verurteilt, da die Dimension des Schätzproblems des Partikelfilters exponentiell mit der Anzahl der Variablen steigt. Abhilfe schafft hier der Rao-Blackwellized Partikelfilter.
Jedes der Partikel enthält die gesamte Trajektorie des Roboters $\boldsymbol{x}_{1:k}$ sowie eine komplette Karte der Umgebung $\boldsymbol{m}$. Jede Landmarke $\boldsymbol{m}_j$ in dieser Karte wird wiederum durch einen eigenen EKF repräsentiert, wie in Tabelle 2.1 dargestellt.

Tabelle 2.1: Darstellung der Partikel beim Fast-SLAM

	Roboter Pfad	LM 1	LM 2	...	LM N
Partikel $p = 1$	$\boldsymbol{x}_{1:k}^{[1]} = \{\begin{pmatrix} x & y & \varphi \end{pmatrix}^T\}_{1:k}^{[1]}$	$\boldsymbol{m}_1^{[1]}, \boldsymbol{\Sigma}_1^{[1]}$	$\boldsymbol{m}_2^{[1]}, \boldsymbol{\Sigma}_2^{[1]}$	...	$\boldsymbol{m}_N^{[1]}, \boldsymbol{\Sigma}_N^{[1]}$
Partikel $p = 2$	$\boldsymbol{x}_{1:k}^{[2]} = \{\begin{pmatrix} x & y & \varphi \end{pmatrix}^T\}_{1:k}^{[2]}$	$\boldsymbol{m}_1^{[2]}, \boldsymbol{\Sigma}_1^{[2]}$	$\boldsymbol{m}_2^{[2]}, \boldsymbol{\Sigma}_2^{[2]}$	...	$\boldsymbol{m}_N^{[2]}, \boldsymbol{\Sigma}_N^{[2]}$
		$\vdots$			
Partikel $p = M$	$\boldsymbol{x}_{1:k}^{[M]} = \{\begin{pmatrix} x & y & \varphi \end{pmatrix}^T\}_{1:k}^{[M]}$	$\boldsymbol{m}_1^{[M]}, \boldsymbol{\Sigma}_1^{[M]}$	$\boldsymbol{m}_2^{[M]}, \boldsymbol{\Sigma}_2^{[M]}$	...	$\boldsymbol{m}_N^{[M]}, \boldsymbol{\Sigma}_N^{[M]}$

Das grundlegende Vorgehen des Fast-SLAM Algorithmus' kann in drei Schritte unterteilt werden:

Prädiktion der neuen Partikel Für jedes Partikel in der Eingangsmenge $\boldsymbol{\chi}_k$ wird eine separate Prädiktion ausgeführt. Diese Prädiktion betrifft nur den Zustand und wird anhand der Wahrscheinlichkeit $P\left(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k^{[p]}, \boldsymbol{u}_k\right)$ bestimmt. Die entstehenden Partikel werden zunächst in einer temporären Partikelmenge $\overline{\boldsymbol{\chi}}_{k+1}$ hinzugefügt.

Landmarken Update Anhand der Messung $\boldsymbol{z}_k$ werden die Landmarken für jedes Partikel der temporären Menge aktualisiert. Hierbei wird zudem mit einer Metrik das Gewicht des Pixels bestimmt. Dieses Gewicht ist ein Maß dafür, wie gut die Messungen zur aktuellen Karte und dem prädizierten Zustand passen.

Ergebnismenge bestimmen Die Ergebnismenge wird wie beim normalen Partikelfilter in Kapitel 2.2.3 bestimmt. Aus der temporären Menge werden zufällige Partikel ausgewählt. Die Wahrscheinlichkeit der Auswahl ist proportional zu dem Gewicht des Partikels. Es kann demnach vorkommen, dass das gleiche Partikel mehrmals in der Ergebnismenge auftritt.

Eine genaue Erläuterung und Diskussion der Vor- und Nachteile beider Algorithmen ist in Kapitel 5.1.2 auf Seite 47 zu finden.

3 Stand der Technik

Dieser Abschnitt befasst sich mit dem aktuellen Stand der Technik. Hierzu gehören Verfahren zur Tiefenbsetimmung anhand von 2D-Bilddaten sowie ein Überblick über aktuelle Quadrocoptermodelle. Abschließend werden dieser Arbeit ähnliche Publikationen vorgestellt.

3.1 Tiefeninformationen aus Bilddaten

Wie bereits in Abschnitt 2.3.3 erwähnt, können die SLAM-Algorithmen je nach verwendeter Sensorik in drei Kategorien eingeteilt werden: SLAM mit Tiefen- und Richtungsinformationen, SLAM mit reinen Richtungsinformationen und SLAM mit reinen Tiefeninformationen (Vgl. Abb. 2.6). Wird als Sensorik für diese Aufgabe nur eine Kamera verwendet, wie es in dieser Arbeit der Fall ist, so liefert diese ohne weitere Verarbeitung der Bilddaten lediglich die Richtungsinformationen. Im Folgenden werden Verfahren erläutert, wie anhand von künstlichen neuronalen Netzen (KNN) oder a-priori Wissen über Gegenstände im Bild die Tiefeninformation aus den Bilddaten gewonnen werden kann. Abschließend wird noch auf Verfahren für kamerabasierte SLAM-Algorithmen eingegangen.

3.1.1 Künstliche Neuronale Netze

Neuronale Netze sind mittlerweile effiziente Lösungen für gängige Bildverarbeitungsprobleme wie Objekterkennung, Klassifizierung und Gestenerkennung bei Menschen. Auch zu der Schätzung von Tiefeninformationen anhand von 2D Bilddaten einer einzelnen Kamera existieren zahlreiche Lösungsansätze.
In [EPF14] wird ein Ansatz vorgestellt, welcher mit zwei KNNs arbeitet. Ein Netz bestimmt zunächst auf dem gesamten Bild die groben Tiefeninformation. Das zweite Netz nutzt diese Informationen und errechnet die genaueren Tiefeninformationen auf Teilbereichen des Bildes. Zum Trainieren beider Netze werden Bilddaten mit den zugehörigen Tiefeninformationen verwendet. Die zum Trainieren der Netze verwendete Kostenfunktion ist invariant gegenüber Skalierung, was dazu führt, dass die resultierenden Tiefeninformationen nur im Verhältnis stimmen.
Einen Ansatz, der ohne direkte Tiefeninformationen beim Anlernen des Netzes auskommt,

wird in [Gar+16] vorgestellt. Für das Training des vorgestellten Netzes werden lediglich Bilderpaare der gleichen Szene mit bekannter Kamerabewegung zwischen den Bildern verwendet. Ähnlich wie bei der 3D-Rekonstruktion durch einen Stereo-Kameraaufbau. Hierzu muss jedoch die Rotation und Translation der Kamera für die Trainingsdaten bekannt sein. Ähnlich zu dem Ansatz aus [Gar+16] wird in [Zho+17] ebenfalls mit mehreren aufeinanderfolgenden Bildern einer Szene gearbeitet. Die Rotation und Translation der Kameraposen der einzelnen Bilder müssen jedoch nicht bekannt sein, sondern werden über ein separates neuronales Netz ermittelt. Während des Trainings sind beide Netze über die Kostenfunktion miteinander gekoppelt, können aber vollständig losgelöst voneinander betrieben werden.

3.1.2 Tiefenbestimmung anhand von bekannten Punkten

In Kapitel 2.1 wird bereits darauf eingegangen, dass mit Hilfe der Projektionsmatrix $\boldsymbol{P}$ ein Punkt in Weltkoordinaten $\begin{pmatrix} x'_w & y'_w & z'_w & w_w \end{pmatrix}^T$ in einen Punkt in Bildkoordinaten $\begin{pmatrix} x' & y' & w \end{pmatrix}$ transformiert werden kann. Die Projektionsmatrix $\boldsymbol{P}$ besteht hierbei aus der internen Kameraparametermatrix $\boldsymbol{P}_i$ und der externen Kameraparametermatrix $\boldsymbol{P}_e$. Bei gegebenen intrinsischen Kameraparametern und gegebener Zuordnung von n Punkten im Bild- zu n Punkten im Weltkoordinatensystem kann demnach durch das Bestimmen der extrinsischen Kameraparametermatrix die Rotation und Translation der Kamera relativ zu den Weltkoordinaten ermittelt werden. Dieser Vorgang wird in der Literatur als PnP (Perspective-n-Point) Problem bezeichnet.
Die minimale Anzahl an benötigten korrespondierenden 3D- zu 2D- Punktepaaren beträgt drei (P3P). Nach [FS37] kann für das P3P-Problem bewiesen werden, dass maximal vier Lösungen existieren. Aus diesen vier Lösungen kann anhand eines vierten Punktepaares die richtige Lösung ermittelt werden. Ein Überblick über aktuelle Algorithmen zum Lösen von PnP und P3P Problemen ist in [Lu18] zu finden.
Speziell für die in 2.3.2 erwähnten planaren Marker bedeutet dies, dass bei einer korrekten Detektion des Markers dessen vier Eckpunkte verwendet werden können, um die Rotation und Translation der Kamera relativ zu dem Marker bestimmen zu können.

3.1.3 SLAM Methoden zur Bestimmung von Tiefeninformationen

Die zuvor genannten Verfahren zur Bestimmung der Tiefeninformationen gehen davon aus, dass für die Bestimmung nur ein einziges Bild aus einer festen Kameraperspektive zur Verfügung steht. Da sich beim SLAM jedoch die Kamera durch den Raum bewegt, kann die Bewegungsinformation zwischen zwei Bildern dazu verwendet werden, die Entfernungsinformationen ähnlich wie bei einem Stereo-Kameraaufbau zu ermitteln. Das in [Jen+06] vorgestellte Verfahren beruht genau auf diesem Prinzip. Der hier verwendete EKF-SLAM wird um N Bilder verspätet betrieben. Diese N Bilder Verspätung werden zum einen dazu

verwendet, robuste natürliche Landmarken zu identifizieren und zum anderen, um die 3D-Position dieser Landmarken zu bestimmen. Um diese 3D-Positionen der Landmarken aus den N Bildern zu bestimmen, werden die Kamerapositionen anhand von Odometriedaten ermittelt.
Ein Ansatz, bei dem der SLAM-Algorithmus nicht verzögert arbeitet, wird in [CDM08] vorgestellt. Bei der hier vorgestellten *inversen Tiefen-Parametrisierung* wird eine Landmarke im Raum nicht durch ihre 3D-Koordinaten (x, y, z) beschrieben, sondern durch einen Strahl, ausgehend von der Position der Kamera, an dem die Landmarke zum ersten Mal beobachtet wurde, bis zur eigentlichen Landmarke. Eine Landmarke besteht demnach aus einem 6D-Zustandsvektor, welcher aus der Position der Kamera (x_m, y_m, z_m) bei der ersten Sichtung der Landmarke, den beiden Winkeln ψ_m und θ_m in Richtung der Landmarke und der inversen Entfernung ρ von Landmarke zur Position x_m, y_m und z_m besteht. Die inverse Entfernung ρ ist hierbei lediglich der Kehrwert der eigentlichen Distanz zwischen den beiden Punkten. Durch die Darstellung in dieser Form werden die Nichtlinearitäten in den Update-Gleichungen des Systems deutlich verringert. Zudem können hierdurch bereits kleine Parallaxen zur Verbesserung der Position und Karte verwendet werden. Nachteilig hierbei ist der höhere Verbrauch an Speicherkapazität sowie der erhöhte Rechenaufwand, welcher den doppelt so großen Landmarken geschuldet ist.

3.2 Quadrocopter

Der in dieser Arbeit zu verwendende Quadrocopter ist zwar vorgegeben, jedoch soll in diesem Abschnitt ein kurzer Überblick über weitere Drohnen im Hobbybereich gegeben werden.

3.2.1 Parrot ANAFI

Die Firma Parrot bietet mit der ANAFI einen Quadrocopter mit hochauflösender 4K HDR-Kamera. Durch einen elektronischen Gimbal kann die Kamera um bis 180° geneigt werden. Durch eine Vorrichtung zum Einklappen der Ausleger kann die Drohne kompakt gelagert und transportiert werden.
Mit dem *GroundSdk* [Anaa] wird von Parrot ein Software Developement Kit (SDK) für Android und IOS bereitgestellt. Dieses SDK bietet Zugriff auf die Steuerung, den Videostream sowie die Einstellungen der Drohne. Es wird zudem ein Python Interface namens *Olympe* [Anab] bereitgestellt, welches ebenfalls den vollen Zugriff auf die Funktionalitäten der Drohne bereitstellt.

Tabelle 3.1: Technische Daten der ANAFI

Name	ANAFI
Hersteller	Parrot
Flugzeit	25 min
Videoauflösung	4096 x 2160 Pixel
Bilder pro Sekunde	24
Gewicht	320 g
Abmessungen (Höhe, Länge, Breite)	65 mm, 175 mm, 240 mm
Flugakku	2S LiPo / 2700 mAh
Kosten	699 €

3.2.2 Ryze Tech Tello Edu

Die Tello Edu von Ryze Tech ist ein kostengünstiger Quadrocopter, welcher sich durch seine Größe und sein Gewicht perfekt dazu eignet, in Innenräumen eingesetzt zu werden. Durch ein von Ryze Tech bereitgestelltes Software Development Kit kann die Drohne vom Computer aus über WLAN mittels UDP-Protokoll[1] gesteuert werden. Des Weiteren werden ein Videostream sowie ein Stream mit aktuellen Messungen der Drohne über UDP bereitgestellt. Eine genaue Beschreibung des SDKs ist in [Tel] zu finden.
Ein zusätzlich anzubringender Schutzkäfig bietet erhöhte Flugsicherheit und verringert das Risiko von Personen- und Sachschäden. Die technischen Daten der Tello sind in der Tabelle 3.2 aufgeführt.

Tabelle 3.2: Technische Daten der Tello

Name	Tello EDU
Hersteller	Ryze Tech
Flugzeit	13 min
Videoauflösung	1080 x 720 Pixel
Bilder pro Sekunde	30
Gewicht	80 g
Abmessungen (Höhe, Länge, Breite)	41 mm, 98 mm, 92.5 mm
Flugakku	1S LiPo / 1100 mAh
Kosten	159 €

[1] UDP steht für User Datagram Protocol. Es ist im Gegensatz zu TCP ein Transportprotokoll für die verbindungslose und ungesicherte Kommunikation. [Man18]

3.2.3 DJI Spark

Die Spark der Firma DJI verfügt ähnlich wie die Parrot Anafi über einen mechanischen Gimbal zur Bildstabilisation. Dieser kann jedoch nur in einem Bereich von -85 ° bis 0 ° verstellt werden.
In einer vorangegangenen Arbeit [Sch19] wurde das SDK für Android und IOS von DJI genauer bezüglich der unterstützten Funktionalitäten untersucht. Hierbei kam heraus, dass das Steuern der Drohne über das SDK nur unzuverlässig funktioniert. Es werden keine Messungen der Sensoren zur Verfügung gestellt und das Dekodieren des Videostreams blieb erfolglos.

Tabelle 3.3: Technische Daten der DJI Spark

Name	Spark
Hersteller	DJI
Flugzeit	16 min
Videoauflösung	1920 x 1080 Pixel
Bilder pro Sekunde	30
Gewicht	300 g
Abmessungen (Höhe, Länge, Breite)	55 mm, 143 mm, 143 mm
Flugakku	3S LiPo / 1480 mAh
Kosten	599 €

3.3 Ähnliche Arbeiten

In [MSMC18] wird ein auf ArUco Markern basierendes echtzeitfähiges System beschrieben, welches dynamisch eine Karte der im Raum platzierten ArUco-Markern erstellt und gleichzeitig die Kameraposition schätzt. Bedingung hierfür ist jedoch, dass durchgehend 2 Marker im Bild zu sehen sind. Die Eingabe für den Algorithmus ist ein Video aus einer Kamerafahrt durch den Raum. Im Vergleich zu ORBSLAM2 [MAT17] und LSD-SLAM [ESC14] wies der SPM-SLAM unter Verwendung eines speziell für diese Arbeit entworfenen Testdatensatzes einen deutlich geringeren ATE[2] auf. Dieser Vorteil ist darauf zurückzuführen, dass der SPM-SLAM im Gegensatz zu den anderen Algorithmen die zusätzlichen Informationen (6D-Pose der Kamera relativ zum Marker) aus den ArUco-Markern nutzt. Vergleiche mit in der Forschung gängigen Datensätzen konnten nicht durchgeführt werden, da diese keine künstlichen Landmarken enthalten.
Das auf dem SPM-SLAM aufbauende UcoSLAM [MSMC19] erweitert den Algorithmus

[2] Absolute Trajectory Error. In der Literatur existieren hierfür unterschiedliche Definitionen. Meist ist hiermit jedoch der RMSE (Root Mean Squared Error, Wurzel aus den gemittelten Fehlerquadraten) gemeint.

dahingehend, dass neben den ArUco-Markern auch natürliche Landmarken zum Kartieren der Umgebung und Lokalisieren der Kameraposition herangezogen werden[3]. Bei Tests mit gängigen Datensätzen[4] schnitt der UcoSLAM Algorithmus, gemessen an einer Metrik basierend auf der Genauigkeit der bestimmten Pose, deutlich besser als der LSD-SLAM und geringfügig besser als der ORB-SLAM2 ab.
Ein weiterer auf künstlichen Markern basierender Ansatz ist der in [PD19] vorgestellte TagSLAM. In dieser Arbeit wird ein Frontend für den Faktorgraph-Optimierer GTSAM [Kae+11] entwickelt, welches den direkten Einsatz von AprilTags [WO16] unterstützt. Besonders hierbei ist, dass zwei separate Graphen verwendet werden: Der *vollständige Graph* enthält alle Faktoren und Variablen, die die Messungen bis zum aktuellen Zeitpunkt betreffen, während der *optimierte Graph* nur die Variablen und Faktoren enthält, welche ein gut konditioniertes Optimierungsproblem bilden. Im direkten Vergleich zu ORB-SLAM2 erzielte der TagSLAM bessere Ergebnisse bezüglich der Genauigkeit der geschätzten Kamerapose. Die hier verwendeten Testdaten bestehen aus den Bilddaten eines stereo Kameraaufbaus sowie den Daten einer IMU[5].
In der Fachpublikation [NBB16] wird eine visuell inertiale SLAM Variante vorgestellt, welche mit Hilfe von AprilTags und IMU Daten ein kostengünstiges Echtzeit-Bewegungserfassungssystem bereitstellt. Das verwendete SLAM-Backend ist ein EKF, welcher neben der Karte und der Roboterpose auch noch die extrinsischen Parameter von IMU zu Kamera sowie den Offset der IMU schätzt. Anders als bei den meisten Arbeiten zum Thema „SLAM mit künstlichen Landmarken“ wird hier nicht die Pose des Markers in 6D verwendet, sondern jeweils die 3D Posen der Markerecken. Zur Bewertung der Genauigkeit der Karte wurde die Distanz zweier Marker vermessen und mit der geschätzten Distanz aus dem EKF verglichen, was eine Abweichung von unter 2 cm ergab.
Ein autonomes, nicht auf Markern basierendes System zur Kartografie von Innenräumen wird in [Stu+17] beschrieben. Als Grundlage für die Kartierung dient der LSD-SLAM aus [ESC14]. Mit einer Parrot Beebop Drohne wird der Raum autonom erkundet. Als Eingangsdaten für den SLAM-Algorithmus dienen hierbei die Bilddaten der hochauflösenden onboard Kamera mit Fisheye Objektiv. Zur Kartierung des Raumes wird ein binäres 3D-Belegungsraster verwendet, bei dem Zellen entweder den Zustand frei oder belegt annehmen können. Das Erkunden des Raumes ist in zwei Phasen eingeteilt: Bei der lokalen sternförmigen Erkundung fliegt die Drohne ein kleines Areal sternförmig ab und füllt dabei das Belegungsraster. In der zweiten Phase wird anhand der bereits vorhandenen Daten im Raster ein neuer zu erkundender Ort ausgewählt und angeflogen. Das Resultat aus Testflügen ist, dass im besten Fall 92% aller wissentlich freien Zellen ebenfalls als frei eingestuft wurden. Die Rate von als frei eingestuften belegten Zellen liegt bei 12%. Ein solch gutes Ergebnis hängt jedoch stark von der Fluggeschwindigkeit und den Texturen der Umgebung ab.

[3] Die Veröffentlichung zum UcoSLAM befand sich während des Zeitpunktes der Erstellung dieser Arbeit noch im Review.
[4] Die verwendeten Datensätze sind Kitti, Euroc-MAV, TUM und SPM
[5] Inertial Measurement Unit. Messeinheit bestehend aus mehreren Inertialsensoren zur Bewegungsdetektion

4 Analyse der Anforderungen

Zur Ermittlung der Anforderungen an das zu entwickelnde System werden unterschiedliche Quellen herangezogen. Zum einen wurden die Anforderungen in mehreren Treffen mit Professor Dr. Hensel in einem iterativen Vorgang gemeinsam erarbeitet. Ein weiterer Aspekt bei der Erstellung der Anforderungen ist die Wiederverwendbarkeit und Erweiterbarkeit des Systems für weiterführende Arbeiten. Des Weiteren sind beim Umgang mit Drohnen spezielle Vorschriften und Normen zu beachten, um Personen- und Sachschäden zu vermeiden.

4.1 Anwendungsfälle

Um die Systemanforderungen zunächst grob zu identifizieren, wird ein UML Anwendungsfall Diagramm angefertigt, welches in der Abbildung 4.1 dargestellt ist. Für alle hier aufgeführten Anwendungsfälle werden die Drohne sowie ein Pilot benötigt. Der Pilot ist hierbei entweder ein Mensch oder ein autonomes System, welches beispielsweise die Planung und Regelung des Pfads der Drohne übernimmt. Die Entwicklung dieses autonomen Systems ist nicht Teil dieser Arbeit. Das System soll lediglich für eine Interaktion mit einem solchen externen Modul ausgelegt werden.

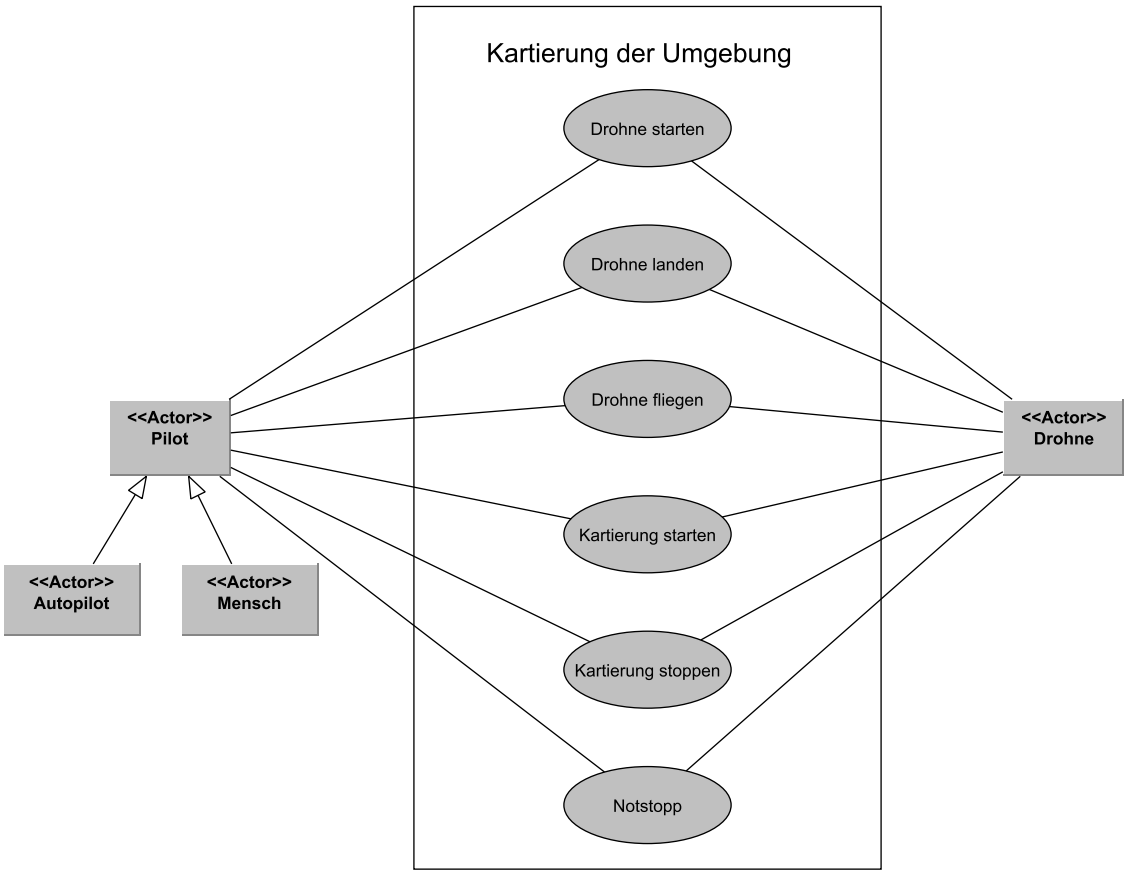


Abbildung 4.1: Diagramm zur Darstellung der Anwendungsfälle

Kartierung starten

Das Starten einer Kartierung kann vom Piloten unabhängig davon angefordert werden, ob die Drohne sich in der Luft oder am Boden befindet. Es muss lediglich die Kommunikation zur Drohne aufgebaut sein. Nach Initialisierung der Kartierung wird der Videostream der Drohne abgerufen. Auf den empfangenen Bildern wird der Bildverarbeitungsalgorithmus zum Ermitteln der Messungen ausgeführt. Abschließend wird die Karte erweitert oder aktualisiert. Dieser Vorgang wird wiederholt, bis die Kartierung gestoppt wird. Es können nicht mehrere Kartierungen parallel laufen.

Tabelle 4.1: Anwendungsfall: Kartierung starten

Name des Anwendungsfalls	Kartierung starten
Kurzbeschreibung	Der Pilot startet die Kartierung. Basierend auf den Kameradaten der Drohne wird eine Karte der Umgebung erstellt.
Vorbedingungen	Die Kommunikation zur Drohne ist aufgebaut. Keine Kartierung ist aktiv.
Ziel	Eine Karte der Umgebung wird automatisch erstellt.
Teilnehmer	Pilot, Drohne
Trigger	Der Pilot startet die Kartierung.
Standardablauf	1. Der Algorithmus zur Kartierung wird initialisiert und gestartet.
	2. Der Videostream der Drohne wird ausgelesen.
	3. Die Messungen werden anhand des Videostreams durchgeführt. 4. Die Karte wird erweitert / aktualisiert.

Kartierung stoppen

Das Stoppen einer Kartierung kann ebenfalls unabhängig vom Flugzustand der Drohne vom Piloten angefordert werden. Nach dieser Anforderung wird der Algorithmus gestoppt und die erstellte Karte wird abgespeichert.

Tabelle 4.2: Anwendungsfall: Kartierung stoppen

Name des Anwendungsfalls	Kartierung stoppen
Kurzbeschreibung	Die Kartierung wird durch den Piloten gestoppt. Die bisher erstellte Karte der Umgebung wird abgespeichert.
Vorbedingungen	Die Kartierung wurde gestartet.
Ziel	Die Karte der Umgebung ist abgespeichert. Der Kartierungsalgorithmus ist gestoppt.
Teilnehmer	Pilot, Drohne
Trigger	Pilot stoppt Kartierung.
Standardablauf	1. Der Algorithmus zum Kartieren wird gestoppt.
	2. Die Karte der Umgebung wird abgespeichert.

Drohne starten

Ist eine Kommunikation mit der Drohne aufgebaut, so kann der Pilot einen Start anfordern. Die Steuerung wird nach dem Start für den Piloten freigegeben.

Tabelle 4.3: Anwendungsfall: Drohne starten

Name des Anwendungsfalls	Drohne starten
Kurzbeschreibung	Die Drohne führt einen kontrollierten Start aus. Nach dem Start befindet sich die Drohne in der Luft und kann gesteuert werden.
Vorbedingungen	Die Kommunikation zur Drohne ist aufgebaut.
Ziel	Die Drohne fliegt und kann gesteuert werden.
Teilnehmer	Pilot, Drohne
Trigger	Der Pilot startet Drohne.
Standardablauf	1. Der Befehl zum Abheben wird gesendet.
	2. Warten bis Drohne gestartet ist.
	3. Die Steuerung wird freigegeben.

Drohne fliegen

Nachdem ein erfolgreicher Start ausgeführt wurde, übernimmt der Pilot die Steuerung der Drohne. Die zu unterstützenden Steuerbefehle der Drohne werden basierend auf denen der Tello festgelegt. Die Steuerung bei der Tello erfolgt über die Geschwindigkeiten v_x, v_y, v_z und v_φ.

Tabelle 4.4: Anwendungsfall: Drohne fliegen

Name des Anwendungsfalls	Drohne fliegen
Kurzbeschreibung	Die Drohne fliegt und wird in der Luft vom Piloten gesteuert.
Vorbedingungen	Die Drohne wurde gestartet.
Ziel	Die Drohne wird vom Piloten gesteuert.
Teilnehmer	Pilot, Drohne
Trigger	Die Drohne wurde gestartet.
Standardablauf	1. Die Drohne wurde gestartet.
	2. Die Steuerbefehle werden gelesen.
	3. Die Steuerbefehle werden an die Drohne gesendet.
	4. Weiter bei Schritt 2.

Drohne Landen

Analog zum Starten der Drohne kann der Pilot eine kontrollierte Landung anfordern. Bedingung hierfür ist, dass die Drohne sich in der Luft befindet. Nach Abschluss einer Landung können die nächsten Befehle gesendet werden.

Tabelle 4.5: Anwendungsfall: Drohne landen

Name des Anwendungsfalls	Drohne landen
Kurzbeschreibung	Die Drohne führt eine kontrollierte Landung aus.
Vorbedingungen	Die Drohne fliegt.
Ziel	Die Drohne ist gelandet.
Teilnehmer	Pilot, Drohne
Trigger	Der Pilot landet Drohne.
Standardablauf	1. Der Befehl zum Landen wird gesendet.
	2. Warten bis Drohne gelandet ist.

Notstopp

Im Fehlerfall soll der Pilot die Möglichkeit haben, einen Notstopp auszuführen. Die Aktorik der Drohne wird abgeschaltet. Die laufende Kartierung wird gestoppt.

Tabelle 4.6: Anwendungsfall: Notstopp

Name des Anwendungsfalls	Notstopp
Kurzbeschreibung	Sicherheitsfunktion zum Stoppen der Aktorik der Drohne.
Vorbedingungen	-
Ziel	Die Aktorik der Drohne wird gestoppt.
Teilnehmer	Pilot, Drohne
Trigger	Der Pilot fordert Notstopp an.
Standardablauf	1. Der Notstopp Befehl wird an Drohne gesendet.
	2. Die laufende Kartierung wird gestoppt.

4.1.1 Anforderungen aus den Anwendungsfällen

Die aus dem Anwendungsfall Diagramm resultierenden funktionalen Anforderungen sind im Folgenden erläutert und in Tabelle 4.7 zusammenfassend aufgeführt.

Das System soll von einem menschlichen oder einem Autopiloten bedient werden können. Es benötigt demnach Schnittstellen für beide Arten der Bedienung (HMI[1], MMI[2]). Das HMI soll hierbei lediglich auf dem MMI aufsetzen und dessen Funktionalitäten verwenden.
Für den menschlichen Bediener sollen Videostream und Karte mit geschätzter Position der Drohne als 2D-Draufsicht auf dem Bildschirm angezeigt werden. Im Videostream sollen dem Piloten zur Kontrolle alle aktuell sichtbaren Landmarken angezeigt werden. Die Steuerbefehle werden über die Tastatur eingegeben.
Aus den Anwendungsfällen bezüglich der Drohnensteuerung geht hervor, dass das System mindestens über die Funktionen Starten, Landen und Notstopp verfügen muss. Des Weiteren soll die Drohne über die Steuerbefehle v_x, v_y, v_z und v_φ angesteuert werden können.
Der Kartierungsalgorithmus soll unabhängig von dem Flugzustand der Drohne gestartet und gestoppt werden. Bei Beendigung des Algorithmus' soll die erstellte Karte abgespeichert werden. Als Informationen soll die Karte für jede Landmarke die Koordinaten in x, y und z enthalten.

Tabelle 4.7: Funktionale Anforderungen aus den Anwendungsfällen

Anforderung	**Beschreibung**
MMI	Das System soll von einem Autopiloten bedient werden können. Es soll ein MMI vorhanden sein.
HMI	Das System soll von einem menschlichen Pilot bedient werden können. Es soll ein HMI vorhanden sein.
Steuerung HMI	Die Steuerbefehle werden über die Tastatur eingegeben.
Video HMI	Die Anzeige des Videostreams erfolgt über den Bildschirm. Es werden alle aktuell sichtbaren Landmarken angezeigt.
Karte HMI	Die aktuelle Karte wird über ein separates Fenster angezeigt.
Start	Die Drohne startet und gibt die Kontrolle nach dem Start an den Piloten.
Landen	Die Drohne landet und ist nach Landung für neue Befehle empfangsbereit.
Notstopp	Die Drohne stellt die Aktorik ab. Die Katierung wird gestoppt.
Steuerbefehle	Allgemeine zu unterstützende Steuerbefehle sind Starten, Landen, Kartierung starten, Kartierung stoppen und Notstopp.
Kartierung	Es soll ein Algorithmus zur Kartierung entwickelt werden. Der Algorithmus kann unabhängig vom Flugzustand der Drohne gestartet und gestoppt werden.
Speichern der Karte	Die Karte soll für jede Landmarke die x, y und z Position im Raum enthalten. Es soll in einem für den Menschen lesbaren Dateityp gespeichert werden.

[1]Human Machine Interface
[2]Machine Machine Interface

4.2 Zusätzliche Anforderungen

Neben den Anforderungen aus den Anwendungsfällen bestehen noch weitere Anforderungen an das System. Diese Anforderungen beziehen sich unter anderem auf die verwendete Hardware, die Algorithmen und die Sicherheitsaspekte.

4.2.1 Anforderungen bezüglich der Drohne

Wie bereits eingangs erwähnt, ist beim Umgang mit Drohnen besondere Vorsicht geboten. Mögliche Gefahren hierbei sind sich drehende Propeller, ein Absturz des Fluggeräts oder ein Brand durch Beschädigung des LiPo-Akkus. Insbesondere in Innenräumen mit beschränkten Ausweichmöglichkeiten kann dies zu gefährlichen Situationen für Mensch und Material führen.
In dieser Arbeit soll die in Abschnitt 3.2.2 vorgestellte Tello EDU von Ryze Tech verwendet werden. Diese Drohne verfügt über einen zusätzlich zu installierenden Schutzkäfig, welcher die Gefahr von Personen- und Materialschäden minimiert. Diese Schutzfunktion wird hierbei als hinreichend angesehen, weshalb keine weiteren Anforderungen an ein Sicherheitskonzept bezüglich der Drohne gestellt werden.

Für nachfolgende Arbeiten soll das System so ausgelegt werden, dass es für unterschiedliche Drohnen verwendet werden kann. Die Schnittstelle für die Drohne soll hierbei gemäß den Funktionalitäten der Tello EDU ausgelegt werden. Als Referenz für die zu unterstützenden Funktionen wird das Tello SDK [Tel] verwendet. Zu diesen Funktionalitäten gehören:

Kommunikationsaufbau Initialisieren der Kommunikation

Start Start der Drohne

Landung Landen der Drohne

Bewegung Kommando zum Bewegen der Drohne in der Form v_x, v_y, v_z und v_φ. Die Steuersignale nehmen hierbei Werte von -100 bis 100 an.

Notstopp Die komplette Aktorik der Drohne wird abgestellt.

Da die Tello EDU nicht dafür vorgesehen ist, zusätzliches Gewicht zu tragen, soll nur die drohneninterne Sensorik verwendet werden.

Tabelle 4.8: Anforderungen bezüglich der Drohne

Anforderung	Beschreibung
Tello EDU	In dieser Arbeit soll die Tello EDU von Ryze Tech verwendet werden.
Schnittstelle Drohne	Bei der Schnittstelle zur Drohne wird das SDK der Tello EDU als Referenz verwendet. Es sollen die Funktionalitäten Kommunikationsaufbau, Start, Landung, Bewegung und Notstopp unterstützt werden.
Verwendete Sensorik	Es soll nur die drohneninterne Sensorik verwendet werden.

4.2.2 Allgemeine Anforderungen an das System

Wie bereits in Kapitel 2.3.3 erwähnt, ist das Kartieren eines Raumes nur möglich, wenn die Position der Drohne bekannt ist. Um die Position genau bestimmen zu können, wird wiederum eine Karte benötigt. Das simultane Lösen beider Probleme soll in dieser Arbeit durch einen SLAM-Algorithmus umgesetzt werden. Als Sensorik für diesen Algorithmus sollen nur die drohneninternen Messungen verwendet werden.
Es soll zudem gewährleistet werden, dass das System in Bezug auf die Dynamik der Drohne eine ausreichend hohe Abtastfrequenz aufweist. Um die komplette Dynamik der Drohne mit dem System erfassen zu können, muss dieses nach dem Nyquist-Kriterium eine Abtastfrequenz aufweisen, welche mindestens mehr als doppelt so hoch ist wie die maximale Frequenz, die die Drohne aufweisen kann. Für diese Frequenz wird als Annäherung die Grenzfrequenz der Drohne angenommen, welche approximativ durch die langsamste Drohnenzeitkonstante $T_{D,min}$ beschrieben wird.
Für die minimale Ausführungsfrequenz f_S des Systems gilt demnach:

$$f_S > \frac{2}{T_{D,min}}. \tag{4.1}$$

Die langsamste Streckenzeitkonstante wird in Kapitel 6.1 ermittelt und beträgt $1{,}2\,\mathrm{s}$. Für die Ausführungsfrequenz des Algorithmus' gilt demnach

$$f_S > \frac{2}{1{,}2\,\mathrm{s}} = 1{,}6\,\mathrm{Hz} \tag{4.2}$$

Das System soll in der Lage sein, die aktuelle Pose der Drohne sowie die Karte mindestens mit dieser mittleren Ausführungsfrequenz zur Verfügung zu stellen. Sollte diese Ausführungsfrequenz so niedrig sein, dass sich Bild- oder Sensordaten ansammeln, so sollen nur die neuesten Daten verwendet werden.
Die erstellte Karte soll die Landmarke in den drei Dimensionen x, y und z repräsentieren.

Aus dem Kapitel 2.3.3 auf Seite 24 geht hervor, dass eine fehlerhafte Datenassoziation bei einem SLAM-Algorithmus direkt zu einer falschen Schätzung der Pose sowie der Karte führen kann. Um diesem Problem vorzubeugen, soll das System so ausgelegt werden, dass eine fehlerhafte Datenassoziation weitestgehend ausgeschlossen wird.

Tabelle 4.9: Allgemeine Anforderungen

Anforderung	**Beschreibung**
SLAM-Algorithmus	Zum Kartieren des Innenraumes soll ein SLAM-Algorithmus verwendet werden.
Ausführungsfrequenz	Für die Ausführungsfrequenz des Systems soll gelten: $f_S > 1{,}6\,\mathrm{Hz}$.
Datenauswertung	Es sollen nur die neuesten Sensordaten verarbeitet werden.
3D-SLAM	Der SLAM Algorithmus soll eine dreidimensionale Karte (x, y, z) erstellen.
Datenassoziation	Das System soll so ausgelegt werden, dass eine fehlerhafte Datenassoziation weitestgehend ausgeschlossen werden kann.

4.2.3 Anforderungen an den SLAM-Algorithmus

Um die Genauigkeit von SLAM-Algorithmen bestimmen zu können, existieren in der Literatur mehrere Verfahren [Nar+15; Han+14; Stu+12]. Hierbei wird meist nur auf die Genauigkeit der Trajektorie eingegangen, da eine genaue Referenzkarte der Umgebung schwer zu erstellen ist. Die genaue Trajektorie der Drohne hingegen kann meist über eine geeignete Messvorrichtung (z.B GPS, externes Kamerasystem) ermittelt werden.
Da für diese Arbeit keine solche Messeinrichtung zur Bestimmung der echten Position der Drohne im Raum vorhanden ist und ein manuelles Bestimmen der Drohnenposition nur ungenau und mit großem Aufwand erreicht werden kann, wird auf die Messung der Genauigkeit der Trajektorie verzichtet.
Um die Genauigkeit der Karte zu beurteilen, soll der in [MS+18] verwendete Absolute Corner Error (ACE) berechnet werden. Der ACE wird hierbei als RMSE angegeben. Für den ACE wird der Fehler von realer Landmarke $\boldsymbol{m}_i$ zu geschätzter Landmarke $\hat{\boldsymbol{m}}_i$ durch die Euler-Distanz berechnet:

$$error_i = \sqrt{(m_{ix} - \hat{m}_{ix})^2 + (m_{iy} - \hat{m}_{iy})^2 + (m_{iz} - \hat{m}_{iz})^2}. \tag{4.3}$$

Der ACE kann für alle Landmarken somit zu

$$ACE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} error_i^2} \tag{4.4}$$

berechnet werden, wobei n der Anzahl der gemessenen Landmarken entspricht.

Nach ausgiebiger Literaturrecherche konnte keine direkt vergleichbare Arbeit für Referenzwerte bezüglich des ACE gefunden werden, weshalb als Referenz für die Genauigkeit des Algorithmus' Werte aus der bereits in 3.3 vorgestellten Arbeit [MSMC18] angenommen werden. Der in der Arbeit erreichte Wert von $ACE = 0{,}023\,\mathrm{m}$ wird hierbei nicht als Anforderung, sondern lediglich als Referenz angenommen. Um die Anforderung bezüglich des ACEs überprüfen zu können, muss zudem gewährleistet sein, dass die Landmarken in der Karte eindeutig den zugehörigen Landmarken in der Umgebung zugeordnet werden können. Wie zuvor bereits erwähnt, ist ein genaues Erstellen einer Referenzkarte der Umgebung meist mit großem Aufwand verbunden. Um den Vergleich von Referenzkarte und erstellter Karte möglichst effizient zu gestalten, soll als Kriterium für die Konzeption und das Design des Systems der Messaufwand beim Einmessen der Referenzkarte mit einfließen.
Als letzter Punkt für die Anforderungen an den SLAM Algorithmus sei noch der Implementierungsaufwand genannt. Da diese Arbeit zeitlich begrenzt ist und der Autor vorab keine Erfahrungen bezüglich SLAM Algorithmen hat, soll eine Einschätzung des Implementierungsaufwandes und des daraus folgenden zeitlichen Aufwandes ebenfalls als weitere Anforderung mit einfließen.

Tabelle 4.10: Anforderungen an den SLAM Algorithmus

Anforderung	**Beschreibung**
Genauigkeit	Als Referenz für die Genauigkeit des Algorithmus' soll der $ACE = 0{,}023\,\mathrm{m}$ angenommen werden.
Messaufwand Referenzkarte	Das Einmessen der Referenzkarte soll mit möglichst wenig zusätzlichem Aufwand verbunden sein.
Implementierungsaufwand	Der Implementierungsaufwand des Algorithmus soll in den zeitlichen Rahmen dieser Arbeit passen.

5 Konzeption und Design

Basierend auf den in Kapitel 4 erarbeiteten Anforderungen wird im Folgenden ein Lösungskonzept für das System entworfen. Anschließend wird das Design der Software erarbeitet und erläutert.

5.1 Konzeption

Dieser Abschnitt befasst sich mit der Konzeption des Lösungsansatzes. Zunächst werden Vor- und Nachteile von SLAM-Algorithmen und Landmarkentypen diskutiert. Nach der Diskussion folgt eine Bewertung der Kriterien und abschließend die Auswahl des Konzeptes.

Bei der Bewertung der Kriterien wird folgende Metrik verwendet:

- **-** negativ
- ○ neutral
- **+** positiv

5.1.1 Landmarken

Wie bereits in Kapitel 2.3.2 beschrieben, gibt es eine Vielzahl an unterschiedlichen Landmarken, welche für einen SLAM-Algorithmus verwendet werden können.
Die Auswahl der in diesem Abschnitt betrachteten Landmarkentypen wird vorab dahingehend eingeschränkt, dass nur Landmarken betrachtet werden, welche von einem Kamerasystem erkannt werden können. Dies folgt aus der Anforderung, dass nur die drohneninterne Sensorik verwendet werden soll. Bei den natürlichen Landmarken wird sich hierbei zudem nur auf solche Verfahren bezogen, welche weitestgehend immun gegen perspektivische Verschiebungen wie Rotation und Translation sind (SIFT, SURF, usw.).

Bei der Verwendung von künstlichen Landmarken wird die Umgebung aktiv mit zusätzlicher Hardware präpariert. Speziell für Bilddaten können dies planare Marker oder Lichtzeichen sein, welche in der Umgebung verteilt sind. Die Anzahl der verwendeten Landmarken gibt hierbei auch die Anzahl der diskreten Messpunkte vor, anhand derer der Raum kartiert wird. Natürliche Landmarken hingegen benötigen keine zusätzliche Hardware. Hier ist die Verteilung der Landmarken stark abhängig von der Struktur der Umgebung. Die Anzahl der Landmarken schwankt hierbei stark mit dem Detailreichtum der Umgebung. Zudem können eintönige Räume zu Assoziationsfehlern führen, da einzelne Landmarken sehr ähnlich sein können. Künstliche Landmarken hingegen verfügen meist über eine individuelle Signatur, welche Assoziationsfehler ausschließt.
Der Rechenaufwand zum Erkennen und Zuordnen der unterschiedlichen Landmarken ist bei beiden hier betrachteten Typen recht hoch. Die künstlichen Landmarken sind zwar so entworfen, dass sie sehr schnell von Bildverarbeitungsalgorithmen erkannt werden können, jedoch ist das Extrahieren von zusätzlichen Informationen wie die Entfernung und die Richtung der Landmarke durch Algorithmen meist rechenintensiv. Bei den natürlichen Landmarken nimmt das Extrahieren von robusten Landmarken und das Assoziieren dieser Landmarken in meist hochdimensionalen Merkmalsräumen in Kombination viel Zeit in Anspruch.
Einer der größten Unterschiede der beiden Landmarkentypen sind die für den SLAM-Algorithmus bereitgestellten Informationen. Natürliche Landmarken liefern bei reinen 2D-Bilddaten lediglich Richtungsinformationen. Die Tiefeninformation muss über andere geeignete Verfahren (vgl. Kapitel 3.1.3) ermittelt werden. Bei künstlichen Landmarken hingegen kann durch a-priori Wissen über Form und Größe der Landmarken neben der Richtungs- auch die Tiefeninformation ermittelt werden.
Ein weiterer Aspekt, der hier betrachtet werden muss, ist Zuordnung der Landmarken zu einem Punkt im Raum. Bei den künstlichen Landmarken stellt dies kein Problem dar. Die Landmarken sind offensichtlich im Raum verteilt und mit Signaturen versehen. Eine Zuordnung einer natürlichen Landmarke zu einem Punkt im Raum ist ohne Weiteres nicht möglich. Der hochdimensionale Merkmalsraum ist für einen menschlichen Anwender nur schwer zu interpretieren. Eine Zuordnung müsste demnach händisch über die eingezeichnete Landmarke in den Bilddaten geschehen, was einen hohen manuellen Aufwand mit sich bringt.

Die zuvor aufgeführten Vor- und Nachteile der beiden Landmarkentypen werden im Folgenden den zutreffenden Anforderungen aus Kapitel 4 zugeordnet und bewertet. Das Resultat dieser Bewertung ist in Tabelle 5.1 aufgeführt.
Der Rechenaufwand ist bei beiden Typen relativ hoch, jedoch wird bei den künstlichen Landmarken nur für die Bestimmung der Pose eine rechenintensive Operation ausgeführt, bei den natürlichen Landmarken hingegen bei der Detektion und der Datenassoziation. Da dieser Rechenaufwand direkten Einfluss auf die Ausführungszeiten des Systems hat, werden bei dem Kriterium der Ausführungszeit die natürlichen Landmarken als negativ und die künstlichen Landmarken als neutral bewertet.

Die zuvor erwähnte Datenassoziation ist ebenfalls ein Kriterium aus den Anforderungen. Es wird gefordert, dass eine fehlerhafte Datenassoziation weitestgehend ausgeschlossen werden soll. Eine solche Datenassoziation ist bei natürlichen Landmarken mit einer Klassifizierung in einem Merkmalsraum verbunden. Mehrere sich ähnelnde Landmarken können bei dieser Klassifizierung vertauscht werden, weshalb die natürlichen Landmarken hier nur als neutral bewertet werden. Die künstlichen Landmarken hingegen werden als positiv eingestuft, da durch die einzigartige Signatur eine Fehlassoziation fast gänzlich ausgeschlossen ist.
Als letzter Punkt in diesem Vergleich ist die Zuordnung einer Landmarke zu einem reellen Punkt aufgeführt. Diese kann für künstliche Landmarken mit deutlich weniger Aufwand als für natürliche hergestellt werden, was dazu führt, dass der Messaufwand zum Erstellen einer Referenzkarte deutlich geringer ausfällt. Hinzu kommt noch, dass theoretisch für verschiedene Kartierungsdurchläufe unterschiedliche Punkte im Raum als natürliche Landmarken erkannt werden. Diese müssten für jeden Durchlauf neu eingemessen werden. Der Messaufwand für eine Referenzkarte aus natürlichen Landmarken wird demnach als negativ eingestuft, der für eine Referenzkarte aus künstlichen Landmarken als positiv.

Wie in der Tabelle 5.1 zu sehen ist, werden die künstlichen Landmarken durchgehend besser bewertet als die Natürlichen. Für die Entwicklung des Systems werden künstliche Landmarken verwendet.

Tabelle 5.1: Vergleich von künstlichen und natürlichen Landmarken

	natürliche LM	künstliche LM
Ausführungsfrequenz	-	○
Datenassoziation	○	+
Messaufwand Referenzkarte	-	+

5.1.2 SLAM-Algorithmus

Die beiden bekanntesten Ansätze zum Lösen des SLAM Problems wurden in Kapitel 2.3.3 vorgestellt. Im Folgenden sollen die Vor- und Nachteile der beiden Ansätze erläutert und miteinander verglichen werden.

Ein Vergleich dieser beiden Algorithmen wurde schon in zahlreichen Arbeiten angestellt [KYY12; MSN11; SMN08]. Eines der ersten aufgeführten Kriterien hierbei ist meist der Rechenaufwand der beiden Algorithmen. Die benötigte Zeit für den Updateschritt des EKF steigt quadratisch mit der Anzahl der erkannten Landmarken. Beim Fast-SLAM hingegen steigt die Rechenzeit linear mit der Anzahl der Landmarken. Hinzu kommt hierbei noch, dass die Rechenzeit ebenfalls linear mit der Anzahl der verwendeten Partikel steigt. Die

Anzahl der verwendeten Partikel gibt zudem an, wie viele separate Hypothesen der Karte aufgestellt werden. Durch diesen Multi-Hypothesen-Ansatz führen fehlerhaft assoziierte Landmarken nicht zwangsläufig zu einem falschen Ergebnis, da für jedes Partikel eine separate Landmarkenassoziation durchgeführt wird. Das Gewicht von Partikeln mit falsch assoziierten Landmarken ist zwangsläufig geringer als das der anderen Partikel. Die Wahrscheinlichkeit, dass diese Partikel zu der Ausgangsmenge hinzugefügt werden, ist demnach ebenfalls gering (Vgl. Kap 2.3.3). Beim EKF hingegen wird nur eine einzelne Hypothese für Pose und Karte aufgestellt. Fehlerhaft assoziierte Landmarken können unmittelbar zu einem falschen Ergebnis führen.
Als vorteilhaft beim EKF erweist sich jedoch die Tatsache, dass Landmarkenpositionen über die Kovarianzmatrix des Filters miteinander korreliert werden. Eine korrekte Messung einer Landmarke hat demnach ebenfalls Einfluss auf die Landmarken, welche mit der gemessenen Landmarke korrelieren. Beim Fast-SLAM hingegen wird für jede Landmarke ein separater EKF eingesetzt. Eine direkte Korrelation der Landmarken untereinander ist nicht vorhanden.
Eine weitere wichtige Eigenschaft ist die Genauigkeit der erstellten Karte sowie der geschätzten Position der Drohne. Direkte Vergleiche der Genauigkeit von EKF-SLAM und Fast-SLAM sind unter anderem in den Arbeiten [KYY12], [MSN11] und [SMN08] zu finden. In jedem der Vergleiche schnitt der EKF-SLAM bezüglich der Genauigkeit der Trajektorie deutlich schlechter ab als der Fast-SLAM. In [KYY12] und [SMN08] ist zusätzlich ein Vergleich der Genauigkeit der erstellten Karten aufgeführt. In beiden Publikationen wies der Fast-SLAM eine geringfügig genauere Karte auf als der EKF-SLAM.

Wie bereits im Abschnitt zu vor werden die Vor- und Nachteile beider Alternativen den zutreffenden Anforderungen zugeordnet und bewertet und in tabellarischer Form (Tabelle 5.2) festgehalten. Bei der Bewertung der Kriterien wird hierbei zudem die Wahl des Landmarkentyps berücksichtigt.
Das erste aufgeführte Kriterium ist die Ausführungszeit. Hierbei weist der Fast-SLAM mit einem linear zu der Anzahl der Landmarken steigenden Rechenaufwand einen deutlichen Vorteil gegenüber des EKF-SLAMs mit quadratisch steigendem Rechenaufwand. Der Fast-SLAM wird hierbei als positiv, der EKF-SLAM als negativ bewertet.
Ein weiterer Vorteil des Fast-SLAMs begründet sich in seinem Multi-Hypothesen-Ansatz, welcher robust gegenüber dem fehlerhaften Assoziieren von Landmarken ist. Dies führt zu einer positiven Bewertung des Fast-SLAM bei dem Kriterium der Datenassoziation. Beim EKF-SLAM hingegen kann bereits eine falsch assoziierte Landmarke zu einem fehlerhaften Ergebnis führen. Da jedoch ein fehlerhaftes Assoziieren von künstlichen Landmarken unwahrscheinlich ist, wird der EKF-SLAM bei diesem Kriterium als neutral bewertet.
Bei der Genauigkeit der beiden Algorithmen geht aus [KYY12; SMN08] hervor, dass die erstellte Karte des Fast-SLAM geringfügig genauer ist, als die des EKF-SLAM. Der Fast-SLAM wird als positiv, der EKF-SLAM als neutral bewertet.
Das letzte Kriterium zur Auswahl des Algorithmus' ist der Implementierungsaufwand, welcher für den EKF-SLAM als positiv und den Fast-SLAM als negativ eingestuft wird. Diese

Bewertung beruht darauf, dass der EKF-SLAM lediglich aus einem abgeänderten EKF besteht, der FAST-SLAM hingegen eine Kombination aus einem Partikelfilter und einem EKF darstellt.

Wird die Tabelle 5.2 betrachtet, so weist der Fast-SLAM in fast allen Kriterien eine bessere Bewertung auf als der EKF-SLAM. Die Wahl des zu verwendenden Algorithmus für diese Arbeit fällt jedoch auf den EKF-SLAM, da dem Implementierungsaufwand an dieser Stelle eine höhere Wertigkeit zugewiesen wird. Zudem kann zu den anderen Kriterien gesagt werden, dass der Fast-SLAM bei Datenassoziation und Genauigkeit nur kleinere Vorteile gegenüber dem EKF-SLAM aufweist und die zu erreichende Ausführungsfrequenz von $1{,}6\,\mathrm{Hz}$ selbst bei rechenaufwändigen Aufgaben für moderne Rechensysteme leicht zu bewältigen ist.

Tabelle 5.2: Vergleich von EKF- und Fast-SLAM

	EKF-SLAM	Fast-SLAM
Ausführungsfrequenz	-	+
Datenassoziation	○	+
Genauigkeit	○	+
Implementierungsaufwand	+	-

5.1.3 Systemübersicht

Eine grobe Übersicht über das System und den Datenfluss ist in dem Blockschaltbild in der Grafik 5.1 abgebildet. Das zentrale Modul zur Kommunikation mit der Drohne ist die UDP Sende- und Empfangseinheit. Die Steuerbefehle $\boldsymbol{u}$ werden je nach Anwendung von der HMI oder MMI generiert. Die von der Tello empfangenen Daten sind der Videostream der Kamera sowie der Status der Drohne. Alle im Status enthaltende Informationen sind in der Tabelle 5.3 aufgeführt.
Das ArUco System verarbeitet den Videostream und die resultierenden Positions- und Richtungsinformationen $\boldsymbol{z}$ werden dem SLAM-Algorithmus zur Verfügung gestellt. Des Weiteren werden dem SLAM-Algorithmus die Steuerbefehle $\boldsymbol{u}$ sowie die Statusinformationen der Drohne übermittelt, anhand derer er die Prädiktion und Positionsschätzung verbessern kann.
Das ArUco System stellt dem HMI/MMI zudem die bearbeiteten Bilddaten aus dem Videostream zur Verfügung, in denen alle detektierten ArUco Marker eingezeichnet sind.
Um ein entwicklungsbegleitendes Testen des Systems unter „Laborbedingungen“ zu ermöglichen, soll eine Systemsimulation entwickelt werden. In Abbildung 5.1 zeigt der gestrichelte Kasten die Teilsysteme, die simuliert werden sollen. Genaueres hierzu ist in Abschnitt 6.4 festgehalten.

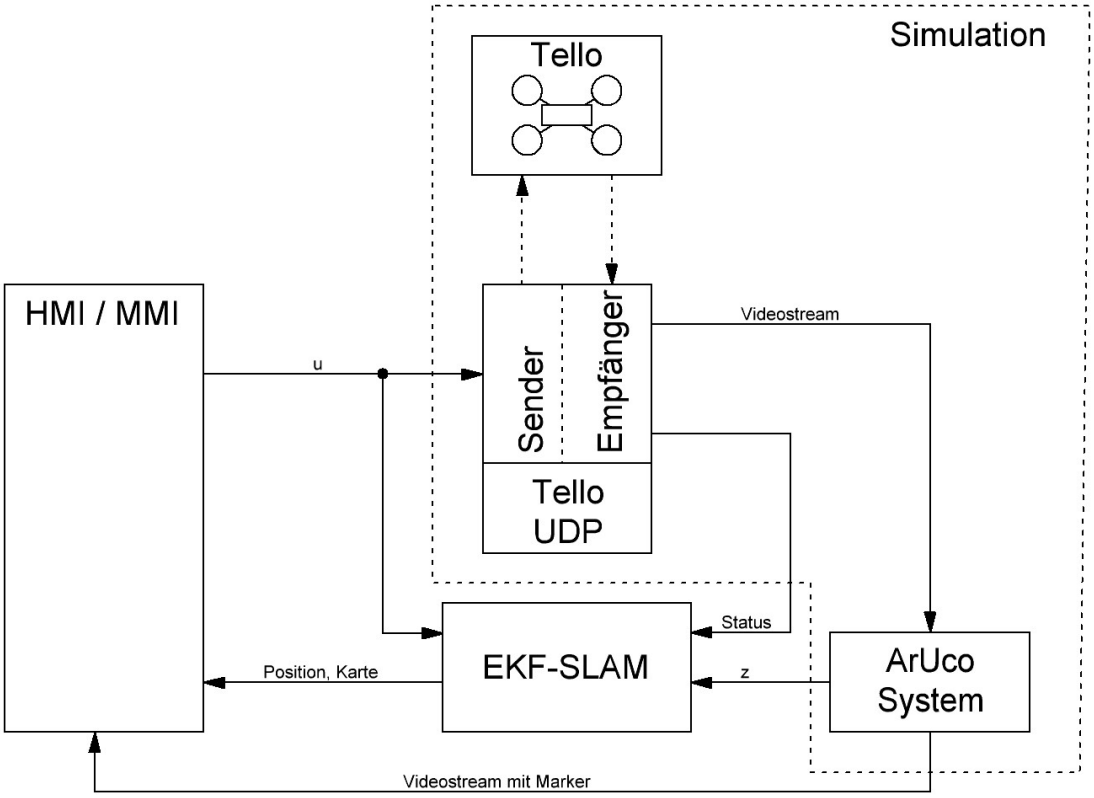


Abbildung 5.1: Schematische Systemübersicht

Tabelle 5.3: Bereitgestellte Statusinformationen der Drohne

Messgröße	Einheit	Messgröße	Einheit
Batteriestand	Prozent	Motorlaufzeit	Sekunden
Geschwindigkeit x	cm/s	Temperatur	°C
Geschwindigkeit y	cm/s	Geflogene Distanz	cm
Geschwindigkeit z	cm/s	Flughöhe	cm
Beschleunigung x	g	Gierwinkel	Grad
Beschleunigung y	g	Rollwinkel	Grad
Beschleunigung z	g	Nickwinkel	Grad

5.1.4 Grober Ablauf einer Kartierung

Das Ablaufdiagramm in Abbildung 5.2 bietet eine Übersicht über das Vorgehen bei der Kartierung. Bevor die Kartierung startet, wird das System initialisiert. Hierzu gehört der Kom-

munikationsaufbau mit der Drohne sowie das Parametrisieren des SLAM-Algorithmus'. Nach der Initialisierung des Systems beginnt der eigentliche Kartierungsvorgang. Zunächst werden die Steuerbefehle eingelesen und an die Drohne übermittelt. Die Bilddaten der Drohne werden ausgelesen. Anschließend werden mögliche ArUco Marker in den Bildern detektiert und die Messdaten für den SLAM-Algorithmus extrahiert. Abschließend wird mit den Messdaten und den Steuerbefehlen ein Schritt des SLAMs ausgeführt. Nach Ausführung dieses Schrittes wird wieder mit dem Einlesen der Steuerbefehle begonnen.

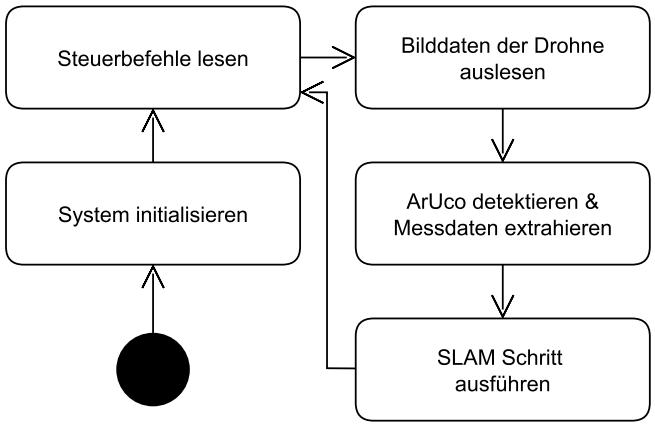


Abbildung 5.2: Ablaufdiagramm der Software

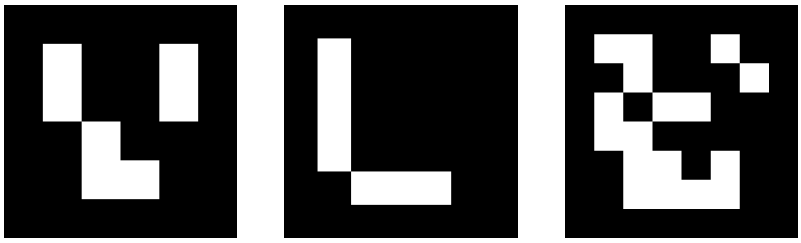
5.2 Design

Dieses Kapitel befasst sich mit dem Design des Systems. Hierzu gehört unter anderem das erarbeiten der Systemarchitektur. Basierend auf dieser Architektur soll eine simple Simulationsumgebung entworfen werden, welche beim Entwickeln der Zielsoftware unterstützt.

5.2.1 ArUco-Marker

Die in dieser Arbeit verwendeten ArUco-Marker sind die quadratischen planaren Marker aus der Veröffentlichung [GJ+14]. Die ArUco-Bibliothek ist Teil von OpenCV, weshalb sie plattformübergreifend auf allen Geräten verwendet werden kann, die C++, Java oder Python unterstützen. Im Gegensatz zu anderen ähnlichen Veröffentlichungen wird bei der ArUco-Bibliothek kein vordefinierter Satz an Markern verwendet, sondern über einen Algorithmus Marker in der gewünschten Größe generiert. Jeder Marker besteht aus einem schwarzen Rand und einer inneren binären Matrix. In dieser inneren binären Matrix, bestehend aus schwarzen und weißen Quadraten, ist die einzigartige Signatur des Markers

rotationsinvariant codiert. Durch ein so genanntes Dictionary werden diese binären Codes mit einer ID verknüpft. In Abbildung 5.3 sind ArUco-Marker aus drei verscheiden Dictionarys abgebildet. Die ArUco Dictionarys sind so aufgebaut, dass der Hamming-Abstand[1] von benachbarten Markern im Dictionary mit steigender Marker-ID sinkt.



(a) 4x4 Dictionary (b) 5x5 Dictionary (c) 6x6 Dictionary

Abbildung 5.3: AruCo-Marker

5.2.2 Entwicklungsumgebung

In diesem Abschnitt soll auf die zu verwendende Entwicklungsumgebung eingegangen werden. Aus den Anforderungen an das System aus Kapitel 4 auf Seite 35 gehen diesbezüglich keine direkten Vorgaben hervor. In dieser Arbeit wird Python verwendet. Im folgenden Abschnitt wird diese Wahl genauer erläutert.
Speziell bei Bildverarbeitungsaufgaben bietet es sich an, die freie Bibliothek OpenCV [Bra00] zu verwenden, welche bereits hochperformante Bildverarbeitungsalgorithmen zur Verfügung stellt. Insbesondere existieren hier Algorithmen zur Detektion und Posenbestimmung von ArUco-Markern. Es existieren von OpenCV Distributionen für Python, Java, C und C++.
Aus den funktionalen Anforderungen aus Tabelle 4.7 auf Seite 40 geht zudem hervor, dass für die manuelle Bedienung des Systems ein HMI vorhanden sein soll. Dieses HMI muss mindestens das Anzeigen von Karte und Videostream sowie die Eingabe der Steuerbefehle über die Tastatur unterstützen. Python bietet hierfür eine Vielzahl an Bibliotheken an. Als Beispiel seien hier *MatPlotLib* zum Plotten von diversen Grafiken sowie *PyGame*, *PyQt* oder *TkInter* zum Programmieren von GUIs genannt.
Zudem bietet Python über die Bibliothek *socket* die Möglichkeit, UDP Pakete zu senden und zu empfangen. Die Kommunikation mit der Tello EDU ist somit ebenfalls realisierbar.
Da Python eine Skriptsprache ist und der Code zur Laufzeit interpretiert wird, sind Laufzeiten von Python Programmen im Vergleich zu äquivalenten C oder C++ Programmen meist

[1] Der Hamming-Abstand gibt die Anzahl der Einzelinformationen an, die zwingend verändert werden müssen, um unterschiedliche Binärketten ineinander zu überführen. [Wei15]

deutlich größer. Da jedoch für alle in dieser Arbeit auftretenden rechenintensiven Operationen wie beispielsweise Matrixmultiplikationen oder Bildverarbeitungsaufgaben bereits Python Bibliotheken wie *NumPy* und *OpenCV* existieren, bei denen diese Operationen durch bereits vorkompilierten C-Code berechnet werden, wird dieser Geschwindigkeitsnachteil relativiert.

5.2.3 Softwaredesign

In diesem Kapitel wird die Softwarearchitektur für das zuvor aufgestellte Konzept ausgearbeitet. Unter Zuhilfenahme von UML Ablauf- und Klassendiagrammen wird die Software geplant. Hierbei wird zunächst auf den groben Ablauf einer Kartierung eingegangen.

Detailliertes Ablaufdiagramm der Software

Der grobe Ablauf des Systems ist in Abbildung 5.4 dargestellt. Zunächst wird das System initialisiert. Alle dazu nötigen Informationen sollen über eine Konfigurationsdatei angegeben werden. Basierend auf dieser Konfiguration wird entweder die Verbindung zur Drohne aufgebaut oder die Simulationsumgebung initialisiert. Auf diese Simulation wird genauer in Abschnitt 6.4 eingegangen. Nach dem Initialisieren der realen oder simulierten Drohne wird die Schleife für den SLAM-Ablauf betreten.
In der Hauptschleife werden zunächst die Steuereingaben eingelesen. Sollte bei diesen Eingaben der Befehl zum Abbruch des SLAMs gegeben werden, so stoppt das Programm. Andernfalls werden die Steuerbefehle an die Drohne weitergegeben und die aktuellen Messwerte ausgelesen. Mit den Messwerten und den Steuerbefehlen wird vom SLAM-Algorithmus die Karte sowie die Drohnenposition geschätzt. Eine genaue Beschreibung des SLAM-Algorithmus ist in Kapitel 6.2 zu finden. Sollte keine Kartierung aktiv sein, so wird die Messung sowie die Aktualisierung des SLAMs übersprungen. Ein Plot der aktuellen Karte und Position der Drohne sowie der Videostream wird in jedem Schleifendurchlauf auf dem Bildschirm angezeigt. Anschließend werden wieder die Steuereingaben eingelesen und die Schleife wird erneut durchlaufen.
Nicht mit in dem Ablaufdiagramm abgebildet ist, dass bei einem Wechsel von Kartierung aktiv zu Kartierung nicht aktiv, die erstellte Karte abgespeichert wird. Die Karte $\hat{\boldsymbol{m}}$ wird in dem Format

$$\hat{\boldsymbol{m}} = \begin{matrix} \hat{m}_{1x} & \hat{m}_{1y} & \hat{m}_{1z} & \hat{m}_{1s} \\ \hat{m}_{2x} & \hat{m}_{2y} & \hat{m}_{2z} & \hat{m}_{2s} \\ & \vdots & & \\ \hat{m}_{Nx} & \hat{m}_{Ny} & \hat{m}_{Nz} & \hat{m}_{Ns} \end{matrix} \tag{5.1}$$

als *csv*-Datei abgespeichert.

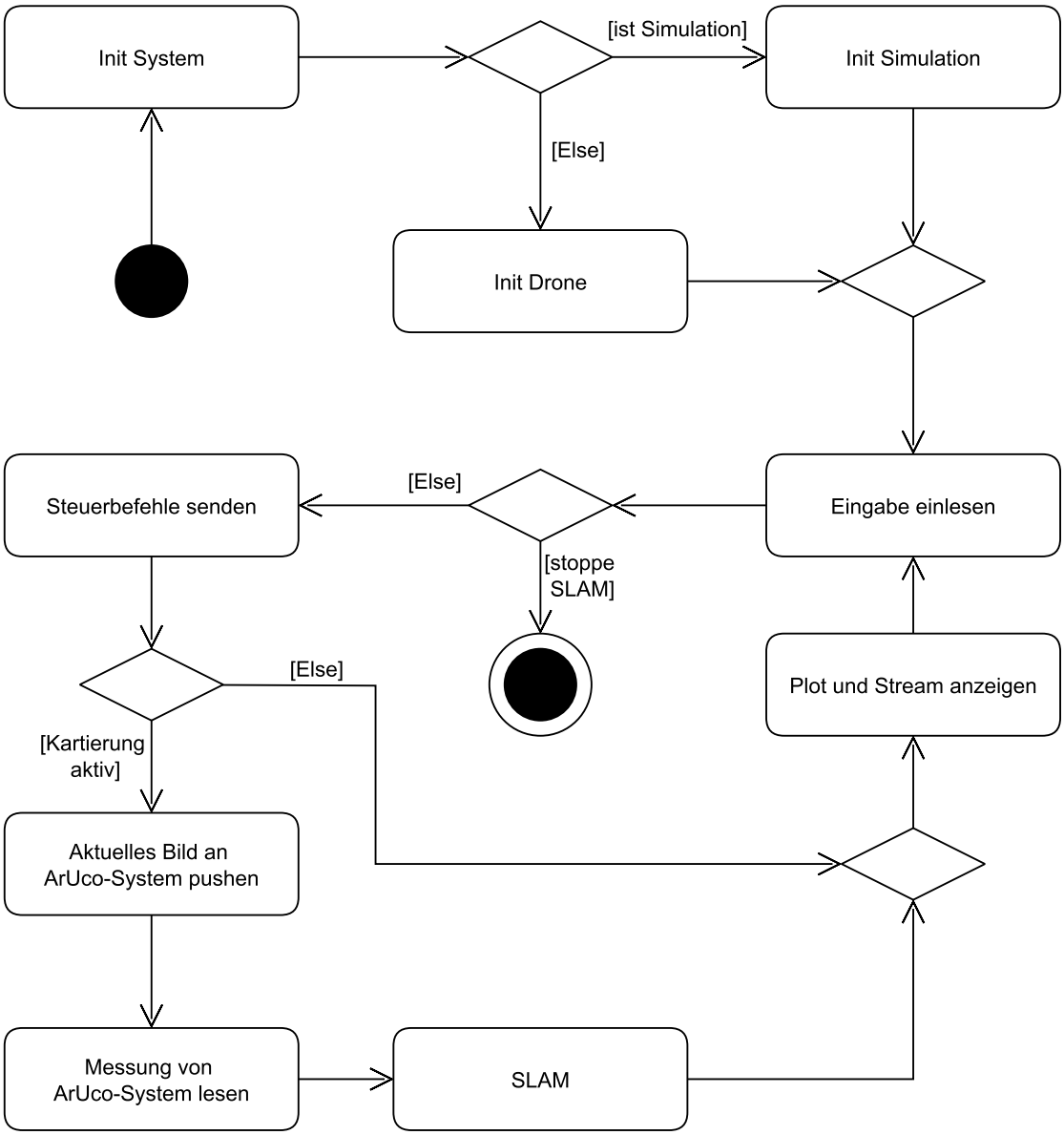


Abbildung 5.4: Ablaufdiagramm der Software

Klassendiagramm

Das Klassendiagramm für die zuvor konzipierte Lösung ist in Abbildung 5.5 dargestellt. Zentrales Element in dem Diagramm ist die Klasse *SLAMSystem*, von der aus alle anderen Klassen bedient werden. Da das System so aufgebaut sein soll, dass der verwendete Quadrocopter durch ein anderes Modell oder eine Systemsimulation ersetzt werden kann,

sollen die Befehle der Drohne über das Interface *DroneInterface* abstrahiert werden. Die zu implementierenden Methoden des *DroneInterface* sind unter anderem *set_velocity()* zum Bewegen der Drohne oder *emergency()* für einen Notstopp.
Eine weitere verwendete Instanz ist *MeasurementInterface*. Diese wird benötigt, da das System entweder mit einer reellen Messung anhand von ArUco-Markern oder einer virtuellen Messung aus der Systemsimulation arbeiten muss. Die Klasse *DroneSimulation* erbt von diesen beiden abstrakten Klassen und simuliert die Drohne samt Umgebung.
Mit den Steuersignalen und den Messungen wird mit der Methode *step()* der Klasse *ExtendedKalmanFilterSLAM* ein Berechnungsschritt des EKF-SLAMs ausgeführt.
Für das MMI wird eine einfache GUI in der Klasse *SimpleFlightGui* realisiert, in der die Karte und der Videostream angezeigt werden. Die Steuerbefehle werden ebenfalls über die GUI eingelesen.

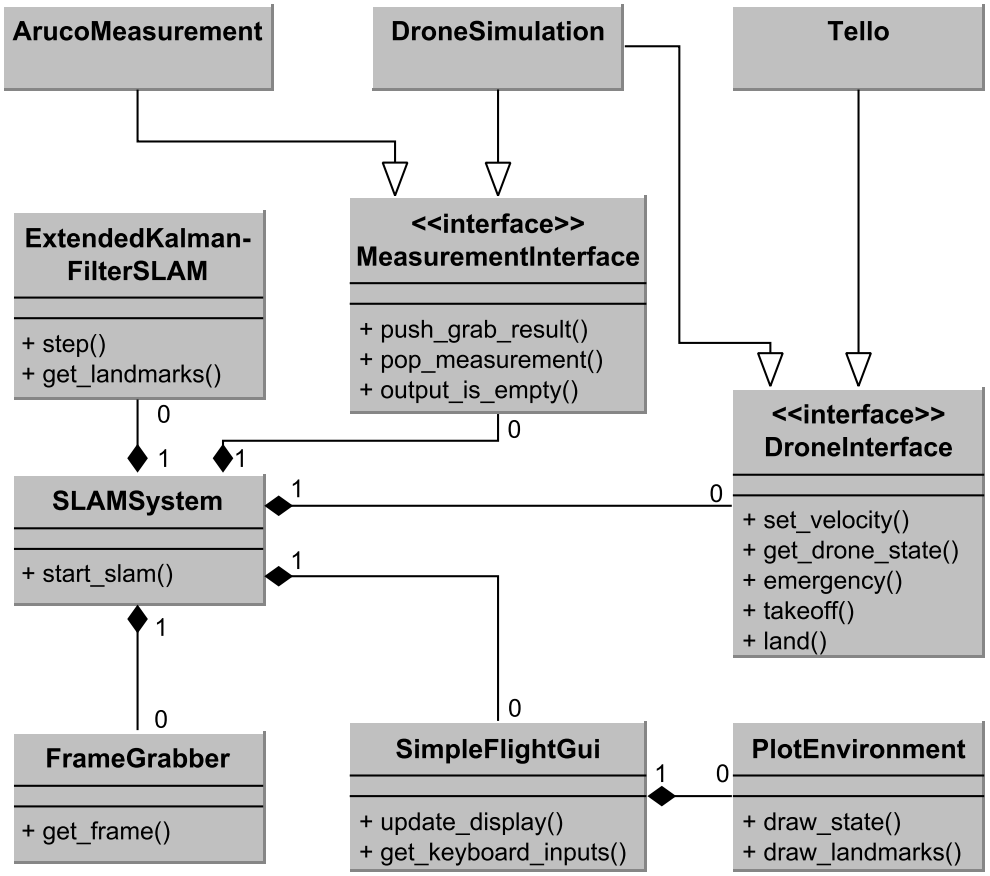


Abbildung 5.5: UML Klassendiagramm

6 Entwicklung und Umsetzung

Die Detaillösung des in Kapitel 5.1 ausgearbeiteten Konzeptes wird im Folgenden erläutert. Hierbei werden insbesondere die Berechnungsvorschriften des EKF-Algorithmus' behandelt. Des Weiteren wird in diesem Kapitel auf die Modellbildung der Drohne und weitere Problemstellungen bei der Umsetzung eingegangen. Für ausgewählte Teile wird die Entwicklung durch Implementierungsbeispiele ergänzt.

6.1 Modellierung der Drohne

Einen großen Einfluss auf die spätere Performance des SLAM-Algorithmus hat die Genauigkeit der Bewegungsprädiktion anhand der Steuersignale. Das hierfür verwendete regelungstechnische Modell der Drohne ist aus [Kra+11] entnommen, da dort eine Drohne verwendet wird, welche dieselben Steuersignale wie die Tello EDU aufweist. Die Zustands- und Eingangsgrößen des Systems sind wie folgt definiert:

$$\boldsymbol{x} = \begin{pmatrix} x & v_x^D & y & v_y^D & z & v_z^D & \varphi & v_\varphi^D \end{pmatrix}^T \tag{6.1}$$

$$\boldsymbol{u} = \begin{pmatrix} u_{vx} & u_{vy} & u_{vz} & u_{v\varphi} \end{pmatrix}^T \tag{6.2}$$

Hierbei steht das hochgestellte D dafür, dass die Zustandsgröße in Bezug zum Drohnen-Referenzsystem definiert ist. Ist kein hochgestellter Buchstabe vorhanden, so ist die Zustandsgröße im globalen Referenzsystem definiert. Abbildung 6.1 veranschaulicht die beiden Koordinatensysteme nochmals. Es wird zunächst davon ausgegangen, dass das Verhalten der Geschwindigkeiten der Drohne im Drohnen-Referenzsystem durch PT_1 Glieder approximiert werden kann. Die zeitdiskreten Systemgleichungen für die Geschwindigkeiten

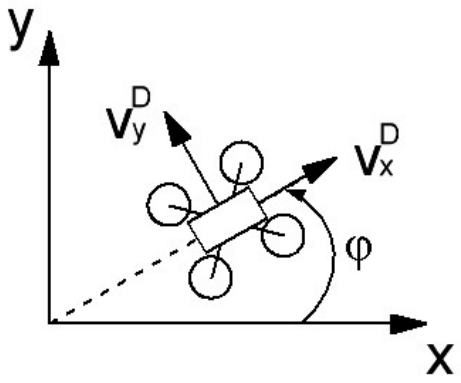


Abbildung 6.1: Globales und Drohnen Koordinatensystem

können demnach im Zustandsraum zu

$$v_{x,k+1}^D = \frac{1}{\frac{T_x}{\Delta_t}+1}\left(K_x u_{vx,k} - v_{x,k}^D\right) + v_{x,k}^D \tag{6.3}$$

$$v_{y,k+1}^D = \frac{1}{\frac{T_y}{\Delta_t}+1}\left(K_y u_{vy,k} - v_{y,k}^D\right) + v_{y,k}^D \tag{6.4}$$

$$v_{z,k+1}^D = \frac{1}{\frac{T_z}{\Delta_t}+1}\left(K_z u_{vz,k} - v_{z,k}^D\right) + v_{z,k}^D \tag{6.5}$$

$$v_{\varphi,k+1}^D = \frac{1}{\frac{T_\varphi}{\Delta_t}+1}\left(K_\varphi u_{v\varphi,k} - v_{\varphi,k}^D\right) + v_{\varphi,k}^D \tag{6.6}$$

angegeben werden. Um von den Geschwindigkeiten v_z und v_φ im Drohnen-Referenzsystem auf die Position z und die Ausrichtung φ der Drohne im globalen Referenzsystem zu schließen, müssen die Geschwindigkeiten integriert werden:

$$z_{k+1} = v_{z,k}^D \Delta_t + z_k \tag{6.7}$$

$$\varphi_{k+1} = v_{\varphi,k}^D \Delta_t + \varphi_k. \tag{6.8}$$

Die globale Position in x und y der Drohne lässt sich nicht durch einfaches Integrieren bestimmen, da die Richtung der lokalen Geschwindigkeiten v_x^D und v_y^D im globalen System von der Ausrichtung der Drohne φ abhängt (vgl. Abb. 6.1). Die Positionen können demnach

durch

$$x_{k+1} = \left(\cos\left(\varphi_k\right) v_{x,k}^D - \sin\left(\varphi_k\right) v_{y,k}\right) \Delta_t + x_k \tag{6.9}$$

$$y_{k+1} = \left(\sin\left(\varphi_k\right) v_{x,k}^D + \cos\left(\varphi_k\right) v_{y,k}\right) \Delta_t + y_k \tag{6.10}$$

bestimmt werden. Aus (6.3) bis (6.10) ergibt sich folgende Systemdynamik:

$$\boldsymbol{x}_{k+1} = \begin{pmatrix} \left(\cos\left(\varphi_k\right) v_{x,k}^D - \sin\left(\varphi_k\right) v_{y,k}\right) \Delta_t + x_k \\ \frac{1}{\frac{T_x}{\Delta_t}+1} \left(K_x u_{vx,k} - v_{x,k}^D\right) + v_{x,k}^D \\ \left(\sin\left(\varphi_k\right) v_{x,k}^D + \cos\left(\varphi_k\right) v_{y,k}\right) \Delta_t + y_k \\ \frac{1}{\frac{T_y}{\Delta_t}+1} \left(K_y u_{vy,k} - v_{y,k}^D\right) + v_{y,k}^D \\ v_{z,k}^D \Delta_t + z_k \\ \frac{1}{\frac{T_z}{\Delta_t}+1} \left(K_z u_{vz,k} - v_{z,k}^D\right) + v_{z,k}^D \\ v_{\varphi,k}^D \Delta_t + \varphi_k \\ \frac{1}{\frac{T_\varphi}{\Delta_t}+1} \left(K_\varphi u_{v\varphi,k} - v_{\varphi,k}^D\right) + v_{\varphi,k}^D \end{pmatrix}. \tag{6.11}$$

Das daraus resultierende Blockschaltbild ist in Abbildung 6.2 dargestellt.

6.1.1 Bestimmung der Modellparameter

Für eine möglichst genaue Prädiktion der Bewegung der Drohne anhand von Steuersignalen müssen die Modellparameter der Gleichung (6.11) bestimmt und das Übertragungsverhalten der Drohne analysiert werden. Die zu bestimmenden Modellparameter sind K_x, K_y, K_z, K_φ, T_x, T_y, T_z und T_φ. Das Übertragungsverhalten der Drohne kann durch zwei Totzeiten approximiert werden. Die Sende-Übertragungstotzeit $T_{t,s}$ gibt an, wie lange die Drohne braucht, um ein vom Rechner gesendetes Signal zu interpretieren. Die zweite Totzeit ist die Empfangs-Übertragungstotzeit $T_{t,r}$, welche die Verzögerung bei der Übertragung von Video- und Statusstream der Drohne zum Rechner beschreibt.
Für die Bestimmung der Empfangs-Übertragungstotzeit wird auf dem Rechnerbildschirm eine Zeit in Millisekunden hochgezählt. Mit der Drohne wird der Rechnerbildschirm abgefilmt. Alle eingehenden Bilder werden auf dem Rechner mit dem aktuellen Zeitstempel in Millisekunden als Name abgespeichert. Die Differenz zwischen Zeitstempel im Bild und dem Zeitstempel im Namen ist die Empfangs-Übertragungstotzeit. Der genaue Wert ergibt sich aus dem Mittelwert von 30 Bildern und beträgt $T_{t,r} = 0{,}43\,\mathrm{s}$. Es wird angenommen, dass die Empfangs-Übertragungstotzeit für Status- und Videostream gleich ist.

Die Modellparameter und die Sende-Übertragungstotzeit der Drohne werden über die

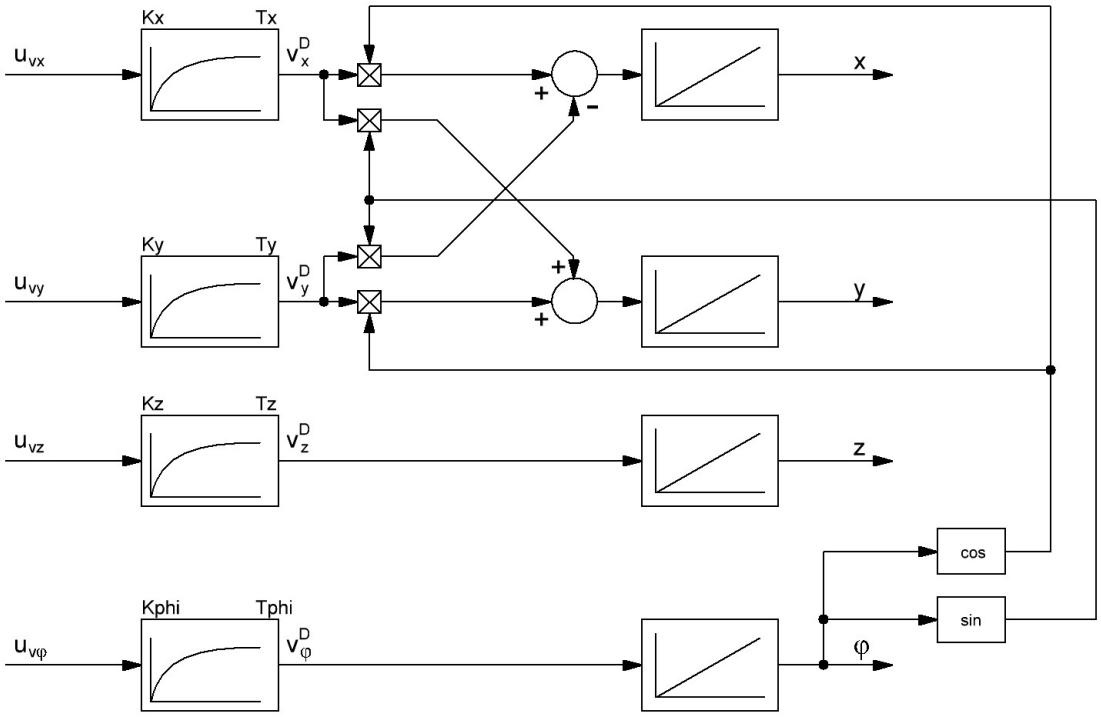


Abbildung 6.2: Blockschaltbild des dynamischen Systems

Sprungantwort des Systems bestimmt. Der Sprung wird hierbei auf die Geschwindigkeit der Drohne in der jeweiligen Flugachse gegeben. Parallel hierzu wird die von der Drohne gemessene Geschwindigkeit aus dem Statusstream der Drohne gelesen. Die so erhaltene Sprungantwort ist in Abbildung 6.3 für die Geschwindigkeit in x-Richtung dargestellt.
Es wird davon ausgegangen, dass sämtliche Totzeiten im System der Übertragung zugeordnet werden können. Für die Sprungantwort bedeutet dies, dass die Gesamttotzeit $T_{t,ges}$ gemessen von Anfang des Sprungs bis zur ersten Reaktion der Strecke die Summe aus den beiden Übertragungstotzeiten $T_{t,s}$ und $T_{t,r}$ ist. Die Sende-Übertragungstotzeit kann demnach durch $T_{t,s} = T_{t,ges} - T_{t,r}$ ermittelt werden. Eine Mittlung über 15 Sprungantworten führt zu einem Ergebnis von $T_{t,s} = 0{,}12\,\mathrm{sec}$.
Die Verstärkung K_x kann anhand des Verhältnisses v_x zu u_x im eingeschwungenen Zustand ermittelt werden. Da angenommen wird, dass die Geschwindigkeit ein reines PT1-Verhalten aufweist, entspricht die Zeitkonstante der Strecke genau der Zeit, an dem die Geschwindigkeit einen Wert von $v_x = K_x \cdot u_x$ aufweist. Hierbei muss jedoch noch die Gesamttotzeit abgezogen werden. Für die Streckenzeitkonstante in x-Richtung gilt demnach $T_x = t|_{v_x = 0.63 u_x K_x} - T_{t,ges}$.
Eine Mittlung aus 6 Sprungantworten ergab hierbei im Mittel die Werte $K_x = 2{,}2\,\mathrm{m\,s^{-1}}$ und

$T_x = 1{,}2\,\mathrm{s}$. Analog hierzu werden die Streckenparameter der verbliebenen Achsen ermittelt und sind in der Tabelle 6.1 aufgeführt.

Tabelle 6.1: Streckenparameter

$K_x = 2{,}2\,\mathrm{m\,s^{-1}}$,	$T_x = 1{,}2\,\mathrm{s}$
$K_y = 2{,}2\,\mathrm{m\,s^{-1}}$,	$T_y = 1{,}2\,\mathrm{s}$
$K_z = 2{,}2\,\mathrm{m\,s^{-1}}$,	$T_z = 1{,}2\,\mathrm{s}$
$K_\varphi = 0{,}2\,\mathrm{m\,s^{-1}}$,	$T_\varphi = 1{,}2\,\mathrm{s}$
$T_{t,s} = 0{,}12\,\mathrm{s}$,	$T_{t,r} = 0{,}43\,\mathrm{s}$

Zudem sei hier noch erwähnt, dass die Drohne nur eine wertedisktrete Abbildung der Geschwindigkeitsmessung zur Verfügung stellt, wie es in Darstellung 6.3 zu sehen ist. Die hier ermittelten Modellparameter sind demnach nur Richtwerte und müssen bei der Inbetriebnahme des Systems angepasst werden.

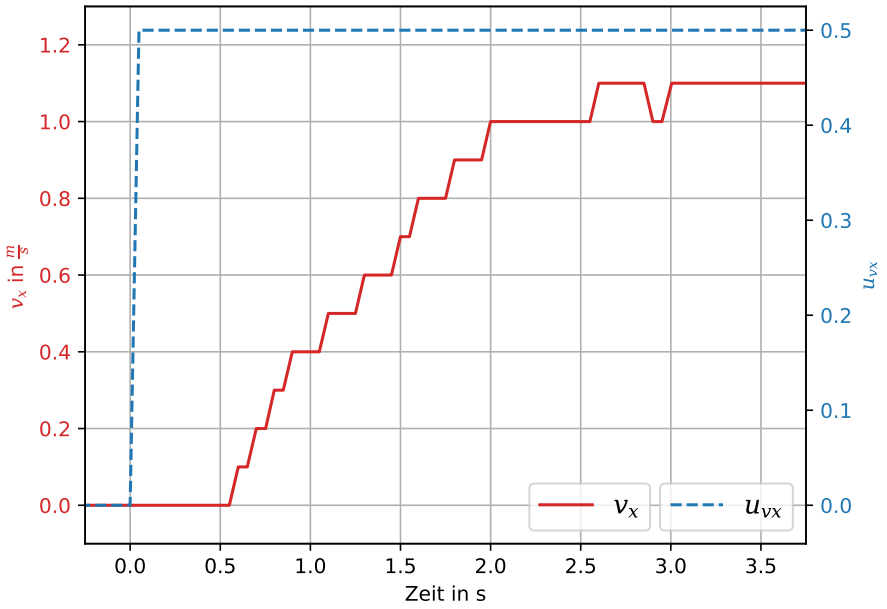


Abbildung 6.3: Sprungantwort der Geschwindigkeit in x-Richtung bei einem Sprung von $u_x = 0.5$.

6.2 EKF-SLAM

Das Kernelement dieser Arbeit ist der EKF-SLAM, welcher bereits grob in Kapitel 2.3.3 vorgestellt wurde. Als Vorlage für den hier zu entwickelnden Algorithmus dient der EKF-SLAM Algorithmus aus [TBF05]. Dieser Algorithmus ist auf einen Roboter mit drei Zuständen (x, y und φ) sowie auf zweidimensionale Landmarken (x und y) ausgelegt. Im Folgenden wird der Algorithmus so angepasst, dass er mit dem Zustandsraummodell der Drohne aus Kapitel 6.1 sowie dreidimensionalen Landmarken arbeiten kann.

6.2.1 Aufbau des Systems

Der Zustandsvektor des EKF-SLAMs besteht analog zu dem aus Gleichung (2.30) aus den Zuständen der Drohne (vgl. (6.1)) $\boldsymbol{x}_{d,k}$ sowie denen der Landmarken $\boldsymbol{x}_{m,k}$. Die Landmarken bestehen, anders als im zweidimensionalen Ansatz aus Kapitel 2.3.3, aus der x-, y- und z-Position der jeweiligen Landmarke. Der Zustandsvektor des EKF-SLAMs ergibt sich somit zu

$$\boldsymbol{x}_k = \begin{pmatrix} \boldsymbol{x}_{d,k} \\ \boldsymbol{x}_{m,k} \end{pmatrix} \tag{6.12}$$

mit

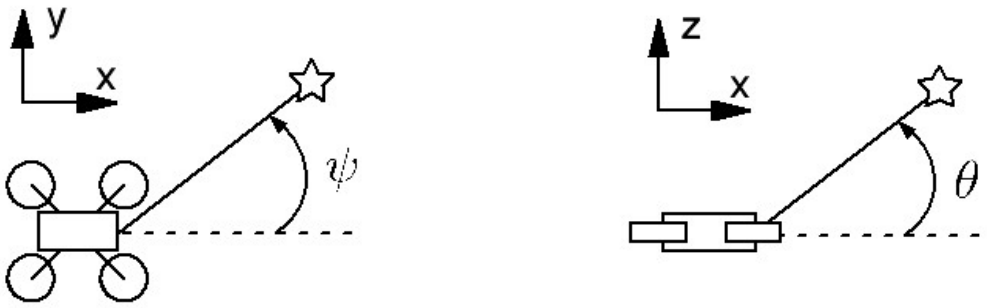
$$\boldsymbol{x}_{d,k} = \begin{pmatrix} x_k & v^D_{x,k} & y_k & v^D_{y,k} & z_k & v^D_{z,k} & \varphi_k & v^D_{\varphi,k} \end{pmatrix}^T \quad \text{und} \tag{6.13}$$

$$\boldsymbol{x}_{m,k} = \begin{pmatrix} m_{1x,k} & m_{1y,k} & m_{1z,k} & \dots & m_{Nx,k} & m_{Ny,k} & m_{Nz,k} \end{pmatrix}^T . \tag{6.14}$$

Die Messung eines ArUco-Markers liefert neben der Entfernung r von Drohne zu Marker auch die beiden Winkel ψ und θ. Wie in Abbildung 6.4 zu sehen ist, beschreibt ψ hierbei den Horizontalwinkel und θ den Höhenwinkel von Drohne zu Landmarke. Die Messung $\boldsymbol{z}$ der i-ten Landmarke kann demnach durch

$$\boldsymbol{z}^i_k = \begin{pmatrix} r^i_k & \psi^i_k & \theta^i_k \end{pmatrix}^T \tag{6.15}$$

beschrieben werden.



(a) Draufsicht der Drohne mit Horizontalwinkel ψ

(b) Seitenansicht der Drohne mit Höhenwinkel θ

Abbildung 6.4: Messbare Winkel von Drohne zu Landmarke

6.2.2 Beschreibung des Algorithmus'

In Algorithmus 2 wird zunächst das Vorgehen beschrieben, ohne dabei auf die Rechenvorschriften der einzelnen Funktionen einzugehen. Im nachfolgenden Abschnitt werden diese hergeleitet und genauer erläutert.
Es wird davon ausgegangen, dass jede Messung eines ArUco-Markers $\boldsymbol{z}^i_{k+1} = \begin{pmatrix} r^i_{k+1} & \psi^i_{k+1} & \theta^i_{k+1} \end{pmatrix}^T$ durch seine Signatur c^i_{k+1} eindeutig einer Landmarke zugeordnet werden kann.
In den Zeilen 1 bis 4 wird die Prädiktion durchgeführt. Eine Besonderheit hierbei ist, dass nicht alle Zustände in $\boldsymbol{x}_k$ von der Prädiktion betroffen sind, da die Positionen der Landmarken keiner Dynamik unterliegen. Es wird nur die Position der Drohne und deren Unsicherheit prädiziert.
Die in Zeile 1 definierte Matrix $\boldsymbol{F}_x$ dient dazu, die Prädiktion, welche nur auf der Drohnenposition ausgeführt wird, auf das komplette System zu projizieren.
In Zeile 2 wird die Prädiktion des Zustandes durchgeführt. Die Schritte für die Prädiktion der Kovarianzmatrix $\overline{\boldsymbol{\Sigma}}_{k+1}$ sind in Zeile 3 und 4 enthalten. $\boldsymbol{G}_{D,k}$ steht hierbei für die Jacobi-Matrix der Drohnendynamik und $\boldsymbol{G}_k$ für die Jacobi-Matrix des gesamten Systems. Anschließend wird die neue Kovarianzmatrix wie beim erweiterten Kalman-Filter berechnet.
In der Schleife von Zeile 5 bis Zeile 16 wird für jede gesichtete Landmarke $\boldsymbol{z}^i_{k+1}$ separat der Kalman-Gewinn $\boldsymbol{K}^i_{k+1}$ bestimmt und mit dem Zustandsvektor und dessen Kovarianzen verrechnet.
In Zeile 10 wird die erwartete Sensormessung $\overline{\boldsymbol{z}}^i_{k+1}$ bestimmt, die bei dem prädizierten Sys-

temzustand $\overline{\boldsymbol{x}}_{k+1}$ für die aktuelle Landmarke j als Messung vorliegen müsste. Die Funktion $\boldsymbol{h}\left(\overline{\boldsymbol{x}}_{k+1}, j\right)$ bildet hierbei den aktuellen Systemzustand auf die erwarteten Messwerte ab.
Da der Kalman-Gewinn $\boldsymbol{K}^i_{k+1}$ separat für jede Messung $\boldsymbol{z}^i_{k+1}$ ermittelt wird, muss anstelle der Jacobi-Matrix aller Zustände wie in (2.24) nur die Ausgangs-Jacobi-Matrix $\tilde{\boldsymbol{H}}^i_{k+1}$ für die Drohnenposition und die aktuelle Landmarke bestimmt werden. Die in Zeile 11 definierte Matrix $\boldsymbol{F}_{x,j}$ dient zur Projektion dieser Ausgangs-Jacobi-Matrix auf die volle Systemgröße.
Die Zeilen 13 bis 15 berechnen wie beim erweiterten Kalman-Filter den Kalman-Gewinn und die neuen prädizierten Zustände sowie deren Unsicherheiten.
Sollte die Landmarke j zuvor nie gesichtet worden sein, so wird sie in Zeile 8 neu initialisiert. Die Funktion $\boldsymbol{f}$ bestimmt hierbei ausgehend von der Messung $\boldsymbol{z}^i_{k+1}$ und dem prädizierten Zustand der Drohe $\overline{\boldsymbol{x}}_{k+1}$ die geschätzte Position der Landmarke in der Karte.

Algorithmus 2 EKF-SLAM$(\boldsymbol{x}_k, \boldsymbol{\Sigma}_k, \boldsymbol{u}_k, \boldsymbol{z}_{k+1}, \boldsymbol{c}_{k+1})$

1: $\boldsymbol{F}_x = \begin{pmatrix} \boldsymbol{I}_8 & \boldsymbol{0}_{8\times 3N} \end{pmatrix}$
2: $\overline{\boldsymbol{x}}_{k+1} = \boldsymbol{x}_k + \boldsymbol{F}_x^T \boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)$
3: $\boldsymbol{G}_k = \begin{pmatrix} \boldsymbol{0}_{8\times 8} & \boldsymbol{0}_{8\times 3N} \\ \boldsymbol{0}_{3N\times 8} & \boldsymbol{I}_{3N} \end{pmatrix} + \boldsymbol{F}_x^T \left(\boldsymbol{I}_8 + \boldsymbol{G}_{D,k}\right) \boldsymbol{F}_x$
4: $\overline{\boldsymbol{\Sigma}}_{k+1} = \boldsymbol{G}_k \boldsymbol{\Sigma}_k \boldsymbol{G}_k^T + \boldsymbol{F}_x^T \boldsymbol{\Sigma}_u \boldsymbol{F}_x$
5: **for all** $\boldsymbol{z}^i_{k+1} = \begin{pmatrix} r^i_{k+1} & \psi^i_{k+1} & \theta^i_{k+1} \end{pmatrix}^T$ **do**
6: $\quad j = c^i_{k+1}$
7: $\quad$ **if** landmark j never seen before **then**
8: $\quad\quad \overline{\boldsymbol{x}}_{mj,k+1} = \boldsymbol{f}\left(\overline{\boldsymbol{x}}_{k+1}, \boldsymbol{z}^i_{k+1}\right)$
9: $\quad$ **end if**
10: $\quad \overline{\boldsymbol{z}}^i_{k+1} = \boldsymbol{h}\left(\overline{\boldsymbol{x}}_{k+1}, j\right)$
11: $\quad \boldsymbol{F}_{x,j} = \begin{pmatrix} \boldsymbol{I}_8 & \boldsymbol{0}_{8\times(3j-8)} & \boldsymbol{0}_{8\times 3} & \boldsymbol{0}_{8\times(3N-3j)} \\ \boldsymbol{0}_{3\times 8} & \boldsymbol{0}_{3\times(3j-8)} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times(3N-3j)} \end{pmatrix}$
12: $\quad \boldsymbol{H}^i_{k+1} = \tilde{\boldsymbol{H}}^i_{k+1} \boldsymbol{F}_{x,j}$
13: $\quad \boldsymbol{K}^i_{k+1} = \overline{\boldsymbol{\Sigma}}_{k+1} \boldsymbol{H}^{iT}_{k+1} \left(\boldsymbol{H}^i_{k+1} \overline{\boldsymbol{\Sigma}}_{k+1} \boldsymbol{H}^{iT}_{k+1} + \boldsymbol{\Sigma}_z\right)^{-1}$
14: $\quad \overline{\boldsymbol{x}}_{k+1} = \overline{\boldsymbol{x}}_{k+1} + \boldsymbol{K}^i_{k+1} \left(\boldsymbol{z}^i_{k+1} - \overline{\boldsymbol{z}}^i_{k+1}\right)$
15: $\quad \overline{\boldsymbol{\Sigma}}_{k+1} = \left(\boldsymbol{I}_{8+3N} - \boldsymbol{K}^i_{k+1} \boldsymbol{H}^i_{k+1}\right) \overline{\boldsymbol{\Sigma}}_{k+1}$
16: **end for**
17: **return** $\overline{\boldsymbol{x}}_{k+1}, \overline{\boldsymbol{\Sigma}}_{k+1}$

Implementierung

Ein Ausschnitt aus der Implementierung des Prädiktionsschrittes ist in Listing 6.1 zu finden, welcher im Algorithmus 2 von Zeile 1 bis Zeile 4 aufgeführt ist. Als Argumente für die Methode werden die vergangen Zeit zur vorherigen Ausführung `dt` sowie die Steuersignale `u_k`

übergeben. Die Methodenaufrufe `self._func_g` in Zeile 15 und `self._jacobian_G_D` in Zeile 20 repräsentieren hierbei jeweils die Funktion $\boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)$ sowie die Jacobi-Matrix $\boldsymbol{G}_{D,k}$. Anders als in Zeile 3 im Algorithmus 2 wird jedoch bereits im Aufruf der Methode `self._jacobian_G_D` die Einheitsmatrix $\boldsymbol{I}_8$ addiert. Die Rückgabewerte der Methode sind der a-priori berechnete Systemzustand `x_k_1_aprio` ($\overline{\boldsymbol{x}}_{k+1}$) sowie dessen Kovarianzmatrix `Sigma_k1_aprio` ($\overline{\boldsymbol{\Sigma}}_{k+1}$).

Listing 6.1: Prädiktionsschritt des EKF-SLAMs

```
1  def _predict(self, dt, u_k):
2      """
3      Prediction step for the EKF-SLAM.
4      :param dt: Passed time since the last call in s
5      :param u_k: Control signal.
6      :return: a-priori covariance matrix, a-priori states
7      """
8      # Get the slicing matrix to inflate and
9      # deflate the system states
10     F_x = np.zeros((self._n, self._x_k.shape[0]))
11     F_x[:self._n, :self._n] = np.eye(self._n)
12
13     # Predict new drone state
14     x_k_1_aprio = self._x_k + F_x.T @ \
15                   self._func_g(u_k, self._x_k, dt,
16                                self._sys_constants)
17
18     # Get jacobian of the system for the predicted state.
19     # The identity matrix gets already added in the method
20     G_k_temp = self._jacobian_G_D(dt, x_k_1_aprio)
21
22     # Prepare G_k
23     G_k = np.eye(self._x_k.shape[0])
24     G_k[:self._n, :self._n] = np.zeros((self._n, self._n))
25     G_k = G_k + F_x.T @ G_k_temp @ F_x
26
27     # Predict covariance matrix
28     Sigma_k1_aprio = G_k @ self._Sigma_k @ G_k.T + \
29                      F_x.T @ self.Sigma_u @ F_x
30
31     return Sigma_k1_aprio, x_k_1_aprio
```

Ein weiterer Ausschnitt aus dem EKF-SLAM Algorithmus ist in Listing 6.2 dargestellt. Der Ausschnitt beschreibt das Initialisieren einer neuen Landmarke. In Algorithmus 2 entspricht

dieser Ausschnitt den Zeilen 7 bis 9. Der hier aufgeführte Code befindet sich somit in der Schleife, welche über alle Messungen $\boldsymbol{z}_{k+1}^{i}$ iteriert. Die jeweilige Messung ist in der Variable `lm` enthalten. Sie besteht aus den Messungen zur Landmarke sowie deren einzigartige Signatur: $\begin{pmatrix} r_{k+1}^{i} & \psi_{k+1}^{i} & \theta_{k+1}^{i} & c_{k+1}^{i} \end{pmatrix}^{T}$. Die Signatur für die erste entdecke Landmarke startet bei eins und wird für jede neue Landmarke inkrementiert. In den Zeilen 1 und 2 kann somit basierend auf dieser ID und der Länge des Zustandsvektors `x_k_1_aprio` bestimmt werden, ob die Landmarke neu initialisiert werden muss.
Ist dies der Fall, so wird die geschätzte Position der Landmarke mit der Methode `self._func_f` berechnet. Anschließend werden Zustandsvektor und Kovarianzmatrix um die neue Landmarke erweitert.

Listing 6.2: Initialisierung einer neuen Landmarke

```
1  # Initialize the new lm if lm never seen before
2  if x_k_1_aprio.shape[0] - self._n < \
3          int(lm[-1])*self._mark_size:
4    # # Calculate the initial position of the landmark
5    x, y, z = self._func_f(x_k_1_aprio, lm)
6
7    # Extend state vector and covariance matrix
8    x_k_1_aprio = np.append(x_k_1_aprio,
9                            np.array([[x, y, z]]).T, axis=0)
10   Sigma_temp = np.eye(Sigma_k1_aprio.shape[0] +
11                       self._mark_size) * self._init_cov_lm
12   Sigma_temp[:Sigma_k1_aprio.shape[0],
13              :Sigma_k1_aprio.shape[0]] = Sigma_k1_aprio
14   Sigma_k1_aprio = Sigma_temp
```

6.2.3 Erläuterung der Berechnungen

Der Algorithmus 2 hat zunächst einen groben Überblick über den Ablauf des EKF-SLAMs gegeben, ohne jedoch auf die genauen Berechnungen in den systembezogenen Funktionen $\boldsymbol{g}$, $\boldsymbol{f}$ und $\boldsymbol{h}$ sowie deren Jacobi-Matrizen einzugehen. Im Folgenden werden diese hergeleitet und genauer erläutert.

Die Prädiktion des neuen Systemzustandes wird im Algorithmus anhand der Funktion $\boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)$ durchgeführt. Diese Gleichung ist fast äquivalent zu der Systemdynamik der Drohne aus Gleichung (6.11). Lediglich das Addieren des vorangegangenen Systemzustands $\boldsymbol{x}_k$ wird in dem Algorithmus außerhalb der Funktion realisiert. Die Funktion $\boldsymbol{g}$ ergibt

sich somit zu

$$\boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) = \begin{pmatrix} \left(\cos\left(\varphi_k\right) v_{x,k}^D - \sin\left(\varphi_k\right) v_{y,k}\right) \Delta_t \\ \frac{1}{\frac{T_x}{\Delta_t}+1}\left(K_x u_{vx,k} - v_{x,k}^D\right) \\ \left(\sin\left(\varphi_k\right) v_{x,k}^D + \cos\left(\varphi_k\right) v_{y,k}\right) \Delta_t \\ \frac{1}{\frac{T_y}{\Delta_t}+1}\left(K_y u_{vy,k} - v_{y,k}^D\right) \\ v_{z,k}^D \Delta_t \\ \frac{1}{\frac{T_z}{\Delta_t}+1}\left(K_z u_{vz,k} - v_{z,k}^D\right) \\ v_{\varphi,k}^D \Delta_t \\ \frac{1}{\frac{T_\varphi}{\Delta_t}+1}\left(K_\varphi u_{v\varphi,k} - v_{\varphi,k}^D\right) \end{pmatrix}. \tag{6.16}$$

Durch partielles Ableiten der Funktion $\boldsymbol{g}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \right)$ nach den Systemzuständen $\boldsymbol{x}_{D,k}$ aus (6.13) kann die zugehörige Jacobi-Matrix zu

$$\boldsymbol{G}_{D,k} = \begin{pmatrix} 0 & c_\varphi \Delta_t & 0 & s_\varphi \Delta_t & 0 & 0 & \left(-s_\varphi v_{x,k} - c_\varphi v_{y,k}\right) \Delta_t & 0 \\ 0 & v(T_x) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & s_\varphi \Delta_t & 0 & c_\varphi \Delta_t & (c_\varphi v_{x,k} - s_\varphi v_{y,k})\Delta_t & 0 & 0 & 0 \\ 0 & 0 & 0 & v(T_y) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & v(T_z) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & v(T_\varphi) \end{pmatrix} \tag{6.17}$$

bestimmt werden, wobei gilt:

$$s_\varphi = \sin(\varphi_k), \quad c_\varphi = \cos(\varphi_k), \quad \text{und} \quad v(a) = \frac{1}{\frac{a}{\Delta_t}+1}.$$

Für das Initialisieren neuer Landmarken wird eine Funktion $\boldsymbol{f}$ benötigt, welche anhand des prädizierten Zustandes $\overline{\boldsymbol{x}}_{k+1}$ und der aktuellen Messung zur i-ten Landmarke $\boldsymbol{z}_{k+1}^i = \begin{pmatrix} r_{k+1}^i & \psi_{k+1}^i & \theta_{k+1}^i \end{pmatrix}^T$ die Position der Landmarke berechnet. Aus dem Zustandsvektor des Systems wird die Position der Drohne x_{k+1}, y_{k+1} und z_{k+1} sowie deren Rotation φ_{k+1} benötigt.

Die Position der Landmarke kann zu

$$\boldsymbol{f}\left(\overline{\boldsymbol{x}}_{k+1}, \boldsymbol{z}^i_{k+1}\right) = \begin{pmatrix} \overline{x}_{k+1} + \cos(\theta^i_{k+1}) r^i_{k+1} \cos(\overline{\varphi}_{k+1} + \psi^i_{k+1}) \\ \overline{y}_{k+1} + \cos(\theta^i_{k+1}) r^i_{k+1} \sin(\overline{\varphi}_{k+1} + \psi^i_{k+1}) \\ \overline{z}_{k+1} + \sin(\theta^i_{k+1}) r^i_{k+1} \end{pmatrix} \tag{6.18}$$

bestimmt werden. Für die Berechnung der x- und y-Position der Landmarke wird zunächst über den Term $\cos(\theta^i_{k+1}) r^i_{k+1}$ die Gesamtentfernung von Landmarke zu Drohne in die xy-Ebene projiziert. Anschließend wird über die jeweils geeignete Winkelfunktion die Position der Landmarke relativ zur Drohne berechnet. Um diese ins Weltkoordinatensystem zu verschieben, wird abschließend die aktuelle Drohnenposition addiert.

Die letzte Funktion $\boldsymbol{h}$ ist dafür zuständig, anhand der prädizierten Systemzustände $\overline{\boldsymbol{x}}_{k+1}$ die zu erwartende Messung $\overline{\boldsymbol{z}}^i_{k+1}$ zu der Landmarke mit der Signatur j zu ermitteln. Die erwartete Gesamtentfernung $\overline{r}^i_{k+1}$ von Landmarke zu Drohne kann über

$$\overline{r}^i_{k+1} = \sqrt{(\overline{m}_{jx,k+1} - \overline{x}_{k+1})^2 + (\overline{m}_{jy,k+1} - \overline{y}_{k+1})^2 + (\overline{m}_{jz,k+1}\overline{z}_{k+1})^2} \tag{6.19}$$

berechnet werden. Für die beiden Winkel $\overline{\psi}^i_{k+1}$ und $\overline{\theta}^i_{k+1}$ gilt:

$$\overline{\psi}^i_{k+1} = \arctan 2\left(\overline{m}_{jy,k+1} - \overline{y}_{k+1}, \overline{m}_{jx,k+1} - \overline{x}_{k+1}\right) - \overline{\varphi}_{k+1} \tag{6.20}$$

$$\overline{\theta}^i_{k+1} = \arctan 2\left(\overline{m}_{jz,k+1} - \overline{z}_{k+1}, \sqrt{(\overline{m}_{jx,k+1} - \overline{x}_{k+1})^2 + (\overline{m}_{jy,k+1} - \overline{y}_{k+1})^2}\right). \tag{6.21}$$

Mit den Vereinfachungen

$$\boldsymbol{\delta} = \begin{pmatrix} \delta_x \\ \delta_y \\ \delta_z \end{pmatrix} = \begin{pmatrix} \overline{m}_{jx,k+1} - \overline{x}_{k+1} \\ \overline{m}_{jy,k+1} - \overline{y}_{k+1} \\ \overline{m}_{jz,k+1} - \overline{z}_{k+1} \end{pmatrix} \tag{6.22}$$

und

$$q = \boldsymbol{\delta}^T \boldsymbol{\delta} = \delta_x^2 + \delta_y^2 + \delta_z^2 \tag{6.23}$$

kann die Funktion $\boldsymbol{h}$ zu

$$\boldsymbol{h}\left(\overline{\boldsymbol{x}}_{k+1}, j\right) = \begin{pmatrix} \sqrt{q} \\ \arctan 2(\delta_y, \delta_x) - \overline{\varphi}_{k+1} \\ \arctan\left(\frac{\delta_z}{\sqrt{q - \delta_z^2}}\right) \end{pmatrix} \tag{6.24}$$

umgeschrieben werden.

Für die Jacobi-Matrix $\tilde{\boldsymbol{H}}^{i}_{k+1}$ muss die Funktion $\boldsymbol{h}$ partiell nach den Zuständen der Drohne $\overline{\boldsymbol{x}}_{k+1}$ sowie nach den Zuständen $\overline{m}_{jx,k+1}$, $\overline{m}_{jy,k+1}$, $\overline{m}_{jz,k+1}$ der Landmarke mit der Signatur j abgeleitet werden. Mit den zuvor definierten Substitutionen ergibt sich die Jacobi-Matrix zu:

$$\tilde{\boldsymbol{H}}^{i}_{k+1} = \begin{pmatrix} \frac{-\delta_x}{\sqrt{q}} & 0 & \frac{-\delta_y}{\sqrt{q}} & 0 & \frac{-\delta_z}{\sqrt{q}} & 0 & 0 & 0 & \frac{\delta_x}{\sqrt{q}} & \frac{\delta_y}{\sqrt{q}} & \frac{\delta_z}{\sqrt{q}} \\ \frac{\delta_y}{q-\delta_z^2} & 0 & \frac{-\delta_x}{q-\delta_z^2} & 0 & 0 & 0 & -1 & 0 & \frac{-\delta_y}{q-\delta_z^2} & \frac{\delta_x}{q-\delta_z^2} & 0 \\ \frac{\delta_x\delta_z}{q\sqrt{q-\delta_z^2}} & 0 & \frac{\delta_y\delta_z}{q\sqrt{q-\delta_z^2}} & 0 & \frac{-\sqrt{q-\delta_z^2}}{q} & 0 & 0 & 0 & \frac{-\delta_x\delta_z}{q\sqrt{q-\delta_z^2}} & \frac{-\delta_y\delta_z}{q\sqrt{q-\delta_z^2}} & \frac{\sqrt{q-\delta_z^2}}{q} \end{pmatrix}. \tag{6.25}$$

Implementierung

Als Beispiel für eine Implementierung der zuvor aufgeführten Berechnungen ist im Listing 6.3 der Code für die Berechnung der Funktion $\boldsymbol{h}$ aus (6.24) aufgeführt. Wie bereits erwähnt, berechnet diese Funktion anhand des a-priori Systemzustandes `x_k_1_aprio` die geschätzte Messung `z_est`. Der Code konnte fast direkt aus (6.24) abgeleitet werden. Lediglich die Methode `pi_2_pi` ist hinzugekommen. Diese Methode bildet ein beliebiges Bogenmaß von $-\infty$ bis $+\infty$ auf $-\pi$ bis $+\pi$ ab.

Listing 6.3: Berechnung der Funktion $\boldsymbol{h}$

```
1     def _func_h(self, delta, x_k_1_aprio):
2     """
3     Calculates the estimated measurement z_est based on the
4     actual system state
5     :param delta: Vector containing the distances
6     (d_x, d_y, d_z) between the actual lm and the drone
7     :param x_k_1_aprio: a-priori system states
8     :return: estimated measurement z_est
9     """
10    q = (delta.T @ delta)[0, 0]
11
12    # Estimated measurement for landmark. Range,
13    # Azimuth and elevation angle
14    z_est = np.array(
15      [[np.sqrt(q)],
16       [pi_2_pi(np.arctan2(delta[1, 0], delta[0, 0]) -
17                x_k_1_aprio[6, 0])],
18       [pi_2_pi(np.arctan(delta[2, 0] /
19                np.sqrt(q - delta[2, 0] ** 2)))]])
20    return z_est
```

6.3 ArUco Messung

Das Detektieren der ArUco-Marker im Bild und das Bestimmen der Pose der einzelnen Marker kann durch die teils iterativen Algorithmen unterschiedliche Zeit in Anspruch nehmen. Um hierbei ein mögliches Blockieren des SLAM-Algorithmus durch die Auswertung der Bilddaten zu verhindern, soll diese parallel zum SLAM-Algorithmus in einem Thread laufen.
Die Kommunikation von Thread und Hauptprogramm wird über threadsichere FIFO Speicher mit vordefinierter Länge realisiert. Wird bei vollem FIFO Speicher ein weiteres Element hinzugefügt, so wird das älteste Element des Speichers verworfen. Die Klasse verfügt über zwei solcher Speicher, jeweils einen für die eingehenden Bilder und einen für die ausgehenden Messungen.
In Abbildung 6.5 ist der Ablauf der Bildauswertung grafisch dargestellt. Es wird zyklisch überprüft, ob neue Bilddaten vorhanden sind. Ist dies der Fall, werden alle ArUco-Marker im Bild detektiert. Nach der Detektion wird anhand der Pose die Messung $\boldsymbol{z}_k$ bestimmt, in der für alle detektierten Marker im aktuellen Bild die Entfernung r^i wie auch die Winkel ψ^i und θ^i enthalten sind. Diese Daten werden im Ausgangs FIFO Speicher abgelegt. Sollten keine Marker im Bild zu finden sein, so wird wieder zyklisch überprüft, ob neue Bilddaten vorhanden sind.

Kamerakalibrierung

Für eine genaue Messung der Entfernung von ArUco-Marker zur Drohne wird die in Kapitel 2.1 beschriebene interne Kameramatrix benötigt. Die hier verwendete interne Kameramatrix wird aus der parallel zu dieser Arbeit durchgeführten Arbeit [Dit19] entnommen. Die in [Dit19] ermittelte interne Kameramatrix beträgt

$$P_{i,Tello} = \begin{pmatrix} 912\text{ px} & 0 & 480\text{ px} & 0 \\ 0 & 912\text{ px} & 360\text{ px} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{6.26}$$

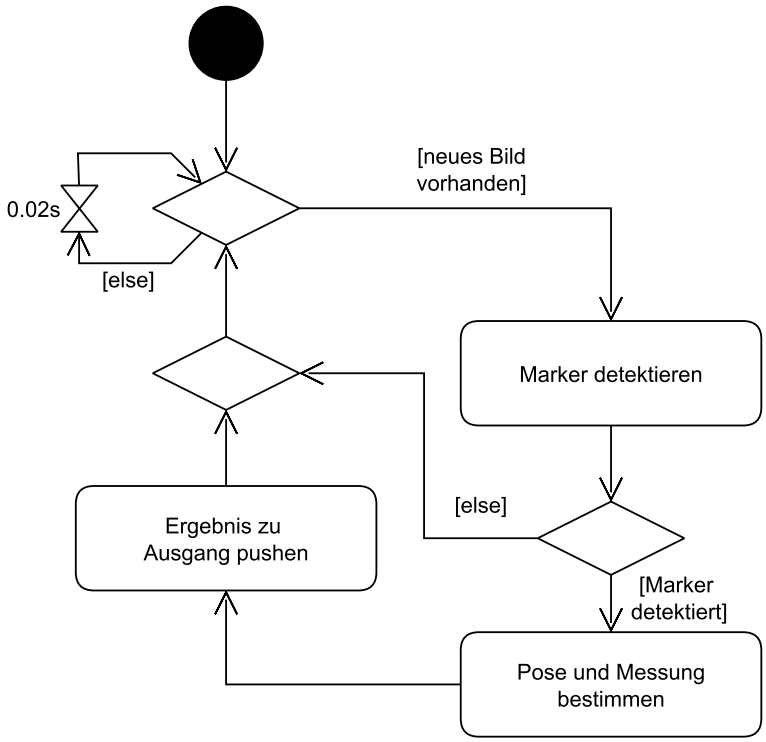


Abbildung 6.5: Ablaufdiagramm der Bildauswertung für die ArUco-Marker

6.4 Simulationsumgebung

Um ein entwicklungsbegleitendes Testen der Software unter Laborbedingungen zu ermöglichen, soll eine Softwaresimulation der Drohne samt Umgebung realisiert werden. Wie dem Klassendiagramm in Abschnitt 5.2.3 zu entnehmen ist, muss die Simulationsumgebung alle Funktionalitäten unterstützen, die zum Steuern der Drohne oder zum Auswerten der Bilddaten verwendet werden.
Zur Simulation der Drohne wird das Modell aus Kapitel 6.1 verwendet. Hierbei wird die Gleichung (6.11) mit den Systemkonstanten aus Tabelle 6.1 verwendet. Die in der Simulation vorhandenen Zustände sind somit identisch mit denen aus (6.13). Logiken wie der Verbindungsaufbau zur Drohne oder Bodenkontakt werden vernachlässigt, da diese für die Entwicklung des Systems unwichtig sind. Ähnlich wie die Tello EDU soll die Simulation eine Abfrage der wichtigsten Systemzustände unterstützen. Hierzu gehören der Winkel φ_k, die Geschwindigkeiten $v_{x,k}$, $v_{y,k}$ und $v_{z,k}$ sowie die aktuelle Flughöhe z_k der Drohne.
Die Landmarken in der Simulationsumgebung werden durch ihre Koordinaten m_x, m_y und m_z dargestellt. Anhand der Ausrichtung φ_k der Drohne, sowie der vorab definierten Sensorspezifikationen wie der Aufnahmewinkel und die Sensorreichweite, wird festgestellt, ob eine Landmarke zum aktuellen Zeitpunkt sichtbar für die Drohne ist. Anschließend kann

die simulierte Messung $\boldsymbol{z}_k^i = \begin{pmatrix} r_k^i & \psi_k^i & \theta_k^i \end{pmatrix}^T$ analog zum EKF-SLAM mit der Funktion $\boldsymbol{h}\left(\overline{\boldsymbol{x}}_k, j\right)$ aus Gleichung (6.24) berechnet werden.
Es besteht ebenfalls die Möglichkeit, das Prozess- und Messrauschen zu simulieren. Hierbei werden die jeweiligen Größen mit mittelwertfreiem gaußschen Rauschen beaufschlagt.

6.5 Umgang mit Übertragungstotzeiten

Aus der Modellbildung in Kapitel 6.1 geht hervor, dass die Kommunikation von Drohne zu Rechner und umgekehrt mit Übertragungstotzeiten behaftet ist. Diese Übertragungstotzeiten betragen $T_{t,s} = 0{,}12\,\mathrm{sec}$ für die Sendeübertragungstotzeit und $T_{t,r} = 0{,}43\,\mathrm{sec}$ für die Empfangstotzeit.
Da die Sendeübertragungstotzeit im Vergleich zur Empfangsübertragungstotzeit und zur Drohnendynamik relativ gering ausfällt, wird diese durch das Erhöhen der Streckenzeitkonstanten T_x, T_y, T_z und T_φ der Drohne approximiert.
Die Empfangstotzeit hingegen ist zu groß und kann nicht allein durch das Anpassen der Modellparameter approximiert werden. Das Problem hierbei ist, dass durch diese Totzeit die Richtungs- und Entfernungsmessung von Drohne zu ArUco-Marker mit einer Verzögerung eintreffen. In der Darstellung 6.6 ist diese Problematik nochmals schematisch dargestellt. Da die Bilder der Kamera mit einer Verzögerung von $T_{t,r}$ eintreffen, stimmt der Zustand im EKF-SLAM nicht mit dem Zustand zum Zeitpunkt der Messung überein.
Um mit diesen verzögerten Messwerten umzugehen, werden alle vergangenen Zustände sowie die Steuerbefehle des EKF-SLAMs über einen Zeitraum von $3 \cdot T_{t,r}$ gespeichert. Sobald eine verzögerte Messung eintrifft, wird der EKF-SLAM auf den letzten Zustand vor dem Messzeitpunkt zurückgesetzt. In der Abbildung 6.6 würde dies dem Zustand $\tilde{\boldsymbol{x}}_{k-3}$ entsprechen. Anschließend wird anhand der Steuersignale der Zustand zum Zeitpunkt der Messung prädiziert. Nachdem die Messung in den SLAM eingeflossen ist, wird das System bis zum aktuellen Systemzustand prädiziert.

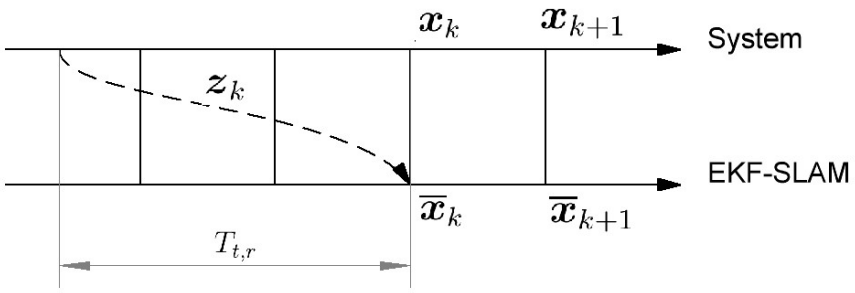


Abbildung 6.6: Schematische Darstellung der Messverzögerung durch die Empfangsübertragungstotzeit

7 Auswertung

Dieses Kapitel befasst sich mit der Auswertung des entwickelten Systems. Hierzu wird zunächst ausführlich auf die erreichte Genauigkeit des Algorithmus' sowie auf mögliche Fehlerquellen eingegangen. Anschließend wird anhand der in Kapitel 4 aufgestellten Anforderungen (vgl. Seite 35 ff.) überprüft, ob das System die geforderten Funktionalitäten erfüllt.

7.1 Genauigkeit des Algorithmus'

Zur Bewertung der Genauigkeit des Algorithmus' wurde in den Anforderungen der ACE genannt. Im Folgenden wird das Vorgehen bei deren Berechnung erläutert und anschließend für verschiedene Kartierungsszenarien ermittelt und miteinander verglichen.

7.1.1 Einmessen der Karte

Zum Einmessen der Referenzkarte $\boldsymbol{m}$ wird der Laser-Enfernungsmesser GLM 50C Professional von Bosch verwendet. Der Entfernungsmesser weist eine Messgenauigkeit von $\pm 1{,}5\,\mathrm{mm}$ auf. Diese Messgenauigkeit könnte als Referenz angenommen werden, wenn die direkte Distanz der Marker zueinander vermessen werden soll. Da jedoch die 3D-Koordinaten der Marker in Bezug zu einem Referenzsystem gemessen werden, kommt zusätzlich zu der Messgenauigkeit des Messgerätes noch eine Toleranz hinzu, die durch das Messverfahren und menschliche Fehler hervorgerufen wird. Bereits kleine Differenzen im Abstrahlwinkel des Messgeräts führen auf größeren Distanzen zu Fehlern. Diese Fehler können nicht genau bestimmt werden, sollten aber gegebenenfalls bei der Bewertung der Ergebnisse berücksichtigt werden.

7.1.2 Bestimmung des ACEs

Um den ACE zu bestimmen, wird, wie bereits in Kapitel 4.2.3 auf Seite 43 beschrieben, der RMSE zwischen generierter und Referenzkarte bestimmt. Da beim Start des Algorithmus' jedoch das Koordinatensystem der Referenzkarte seinen Ursprung dort hat, wo die Drohne sich im Moment des Kartierungsstarts befindet, werden sich in den meisten Fällen die Koordinatenursprünge der beiden Karten um eine Rotation und eine Translation unterscheiden. Es wird somit vor der Auswertung der Karten mit der Software CloudCompare [Clo] die optimale Rotation und Translation der Karten zueinander ermittelt. Als Fehlerfunktion für diese Optimierung wird der RMSE verwendet. In Abbildung 7.1 ist das Ergebnis einer solchen Optimierung dargestellt. Die Transformation wird auf den Rohdaten der vom SLAM-Algorithmus erstellten Karte (+) angewandt. Das Resultat ist die angepasste Karte (●). Die Referenzkarte (×) bleibt unverändert. Bei genauer Betrachtung ist zu erkennen, dass die Punkte der transformierten SLAM-Karte näher an der Referenzkarte liegen, als die der ursprünglichen SLAM-Karte.

Um die Genauigkeit des Algorithmus in verschiedenen Situationen beurteilen zu können, werden in zwei verschiedenen Bewegungsszenarien der Drohne Karten generiert und anschließend mit der Referenzkarte verglichen.
Das erste Szenario sind einfache Runden im Raum. Die Drohne fliegt in einem Rechteck gegen den Uhrzeigersinn durch den Raum und kartiert dabei. Das zweite Szenario sieht einen Start der Kartierung mitten im Raum vor. Die Drohne wird nicht vor- oder seitwärts bewegt, sondern dreht sich lediglich auf der Stelle.
In beiden Szenarien werden jeweils 15 mal drei Runden/Drehungen ausgeführt, wobei die Karte nach einer, zwei und drei Runden/Drehungen abgespeichert wird. Für jedes Szenario werden demnach $15 \cdot 3 = 45$ Karten abgespeichert und ausgewertet.
Diese beiden Szenarien werden gewählt, da sie zwei gegensätzliche Ansätze der Drohnenbewegung bei der Kartierung darstellen. Für eine Raumrunde legt die Drohne eine vergleichsweise große Distanz zurück. Bei der Drehung hingegen bleibt die Drohne größtenteils auf einer Koordinate im Raum. In beiden Szenarien wird nach [HLN12] erwartet, dass mit steigender Anzahl der Runden/Drehungen die Genauigkeit der Karte steigt und somit der ACE sinkt.

Der Ablauf zum Bestimmen der Genauigkeit der Karte sieht demnach wie folgt aus:

1. Verteilen der ArUco-Marker im Raum.
2. Händisches Einmessen der Referenzkarte.
3. Kartieren des Raumes durch:

   Raumrunde Die Drohne fliegt eine bis drei vorprogrammierte Raumrunden. Nach jeder Runde wird die aktuelle Karte gespeichert.

Drehung Die Drohne dreht sich ein- bis dreimal um $360°$. Nach jeder Drehung wird die aktuelle Karte gespeichert.

4. Optimale Transformation von Referenz- zu erstellter Karte bestimmen und anwenden.
5. ACE bestimmen.

Aus den so ermittelten ACEs wird zur Auswertung der Mittelwert $\overline{ACE}_n$ und die Stichprobenstandardabweichung $\sigma_{ACE,n}$ der n-ten Runde/Drehung berechnet:

$$\overline{ACE}_n = \frac{1}{m} \sum_{i=1}^{m} ACE_{i,n} \tag{7.1}$$

$$\sigma_{ACE,n} = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (\overline{ACE}_n - ACE_{i,n})^2} \ , \tag{7.2}$$

wobei m für die Anzahl der Messungen steht.
Der Mittelwert wird hierbei als Maß für die erreichte Genauigkeit verwendet. Die Standardabweichung kann als Maß für die Streuung der Messwerte angenommen werden, wodurch eine Aussage über die Wiederholbarkeit der Messungen getroffen werden kann.
Zuvor werden jedoch Ausreißer durch die 2-σ Methode[1] eliminiert. Bei den so eliminierten Ausreißern wird davon ausgegangen, dass diese durch fehlerhaft assoziierte Landmarken hervorgerufen wurden. Hierauf wird genauer in Abschnitt 7.1.4 eingegangen.

[1] Bei der 2-σ Methode werden alle Messwerte außerhalb eines Bereichs der zweifachen Standardabweichung um den Mittelwert verworfen.

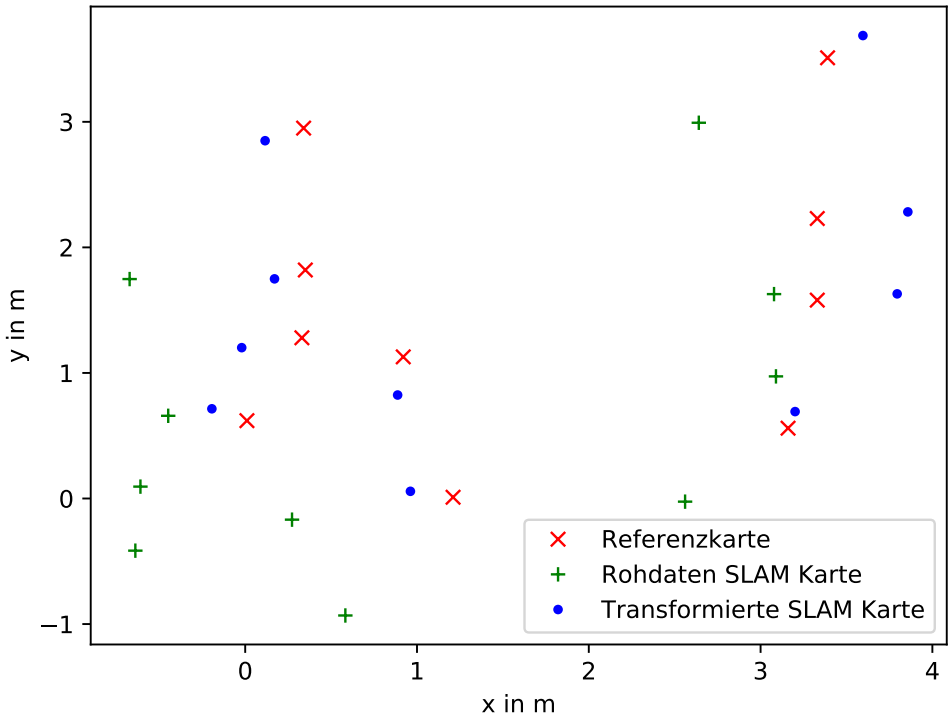


Abbildung 7.1: Ergebnis einer Optimierung der Rotation und Translation von Referenz- zu erstellter Karte

ACE bei Raumrunden

In der Tabelle 7.1 sind die Mittelwerte und Stichprobenstandardabweichungen der einzelnen Messreihen für die Raumrunden der Drohne aufgeführt. Die Abbildung 7.2 stellt diese nochmals grafisch dar.
Es ist zu erkennen, dass der mittlere ACE entgegen der Erwartung nicht mit steigender Anzahl an Raumrunden abnimmt. Vielmehr bleibt dieser fast unverändert. Ähnlich verhält es sich mit der Streuung der Messwerte. Es ist kein Trend zu erkennen.

Tabelle 7.1: Varianz und Mittelwert des ACEs bei unterschiedlicher Anzahl an Raumrunden der Drohne

Anzahl Raumrunden	Mittelwert	Stichprobenstandardabweichung
1	0,28 m	0,062 m
2	0,27 m	0,080 m
3	0,26 m	0,057 m

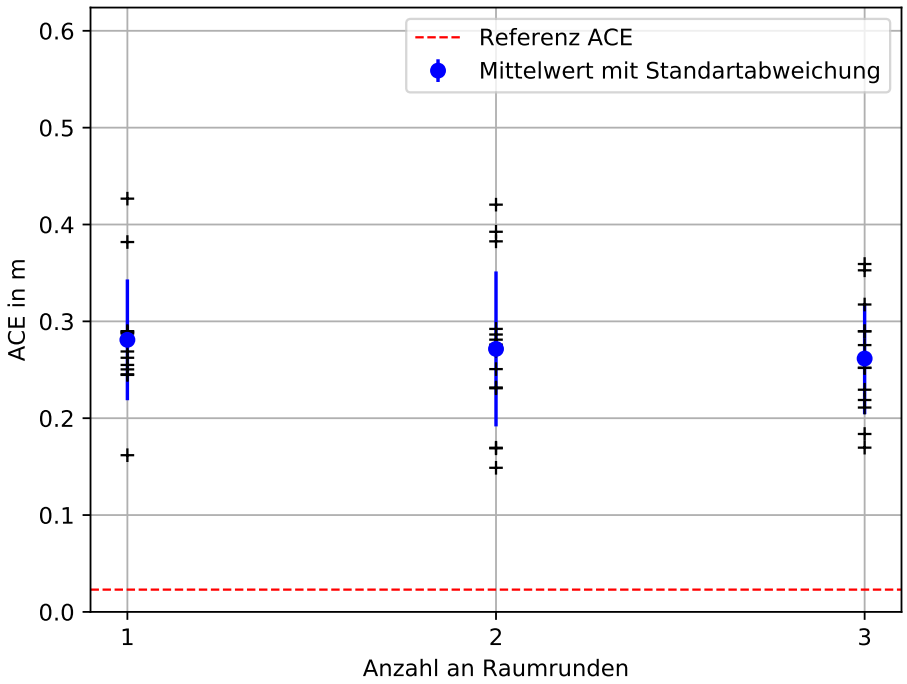


Abbildung 7.2: Verlauf des ACEs bei Raumrunden der Drohne

ACE bei reiner Drehung

Analog zu der Messung des ACEs bei Raumrunden werden für die Drehungen ebenfalls die Mittelwerte und Stichprobenstandardabweichungen in Tabelle 7.2 festgehalten. Die entsprechende Grafik ist in Abbildung 7.3 dargestellt.
Ebenso wie bei der Messreihe des ACEs bei Raumrunden, ist auch hier kein eindeutiger Trend zu erkennen. Der mittlere ACE und die Standardabweichung verhalten sich fast konstant bezüglich der Anzahl der Drehungen. Lediglich bei drei Drehungen ist eine leichte

Verbesserung des ACEs zu erkennen, welche sich aber weiterhin in der Stichprobenstandardabweichung der vorherigen Messungen befindet.

Tabelle 7.2: Varianz und Mittelwert des ACEs bei unterschiedlicher Anzahl an Drehungen der Drohne

Anzahl Drehungen	Mittelwert	Stichprobenstandardabweichung
1	0,31 m	0,095 m
2	0,31 m	0,102 m
3	0,26 m	0,094 m

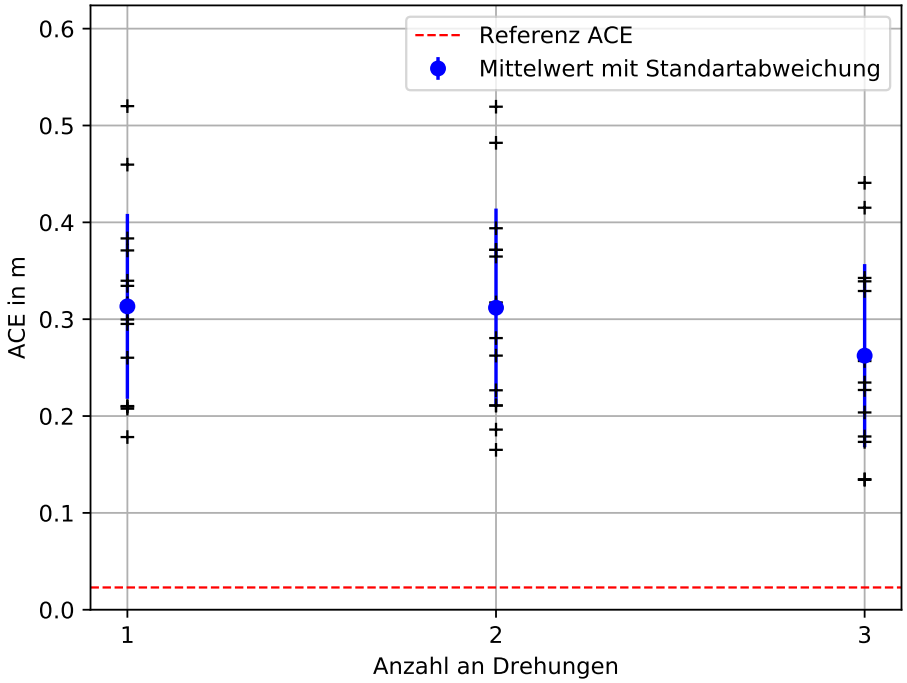


Abbildung 7.3: Verlauf des ACEs bei reinen Drehungen der Drohne

7.1.3 Auswertung der Messergebnisse

Im Folgenden werden die Messergebnisse ausgewertet und gegebenenfalls die Ursachen für fehlgeschlagene Kartierungen erörtert.

7.1.4 Auftreten von Ausreißern in den Messergebnissen

Wie bereits zuvor erwähnt, sind vor der Berechnung des Mittelwertes und der Stichprobenstandardabweichung gegebenenfalls Ausreißer aus den Messergebnissen eliminiert worden. In Abbildung 7.4 ist die Karte inklusive Referenzkarte einer solchen fehlgeschlagenen Kartierung abgebildet. In der rechten Hälfte des XY-Plots sind die Referenzkarte (×), sowie die vom Algorithmus erstellte Karte (+) zu erkennen. Bereits mit bloßem Auge ist zu sehen, dass die Karten bis auf einen Versatz der rechten oberen 5 Landmarken zu den unteren 3 Landmarken relativ gut zueinander passen. Der Grund für diesen Versatz ist die fehlerhaft Erkannte Landmarke in dem unteren linken Bereich der Karte. Hier wurde vom Bildverarbeitungssystem ein ArUco-Marker detektiert, welcher in der Realität nicht vorhanden ist. Eine solche fehlerhafte Detektion ist im Durchschnitt bei einer von 15 Messungen aufgetreten.
Die erste Ursache für eine solche Fehldetektion ist ein nicht optimal parametrisierter Algorithmus zur Detektion der ArUco-Marker, was dazu führt, dass rechteckige Strukturen im Raum, welche keine ArUco-Marker sind, als solche erkannt werden. Dies tritt insbesondere bei dunkleren Lichtverhältnissen im Raum auf, da die Tello EDU hierbei einen höheren Gain-Faktor bei der Bildaufnahme verwendet, was zwangsläufig zu einem erhöhten Rauschen im Bild führt.

7.1.5 Auswertung der ACEs

Wie bereits zuvor erwähnt, sinkt der mittlere ACE entgegen der Erwartungen nicht mit steigender Anzahl an Runden/Drehungen, sondern bleibt fast gänzlich unverändert. Es kann lediglich gesagt werden, dass der resultierende ACE bei den Raumrunden der Drohne leicht geringer ausfällt, als bei den Drehungen.
Im Vergleich zu dem Referenz-ACE von $ACE = 0{,}023\,\mathrm{m}$ (vgl. Kapitel 4.2.3 auf Seite 43) liegt der in dieser Arbeit erreichte ACE durchweg deutlich darüber. In den Abbildungen 7.2 und 7.3 ist dieser Referenz-ACE als gestrichelte Linie dargestellt. Der Referenzwert an das System beträgt $ACE = 0{,}023\,\mathrm{m}$ (vgl. Kapitel 4). Die in dieser Arbeit erreichten ACE Werte sind um ein Vielfaches von dem Referenzwert entfernt. Im Folgenden werden die Ursachen für diese Ungenauigkeit diskutiert und gegebenenfalls überprüft.

Ungenaue Modellierung der Drohne

Wie bereits in Kapitel 6.1 auf Seite 56 erwähnt, ist eine genaue Modellbildung der Drohne zur Bewegungsschätzung unabdingbar für die Genauigkeit des daraus resultierenden EKF-SLAMs. Dies kann insbesondere bei Abschnitten einer Kartierung zu Fehlern führen, in

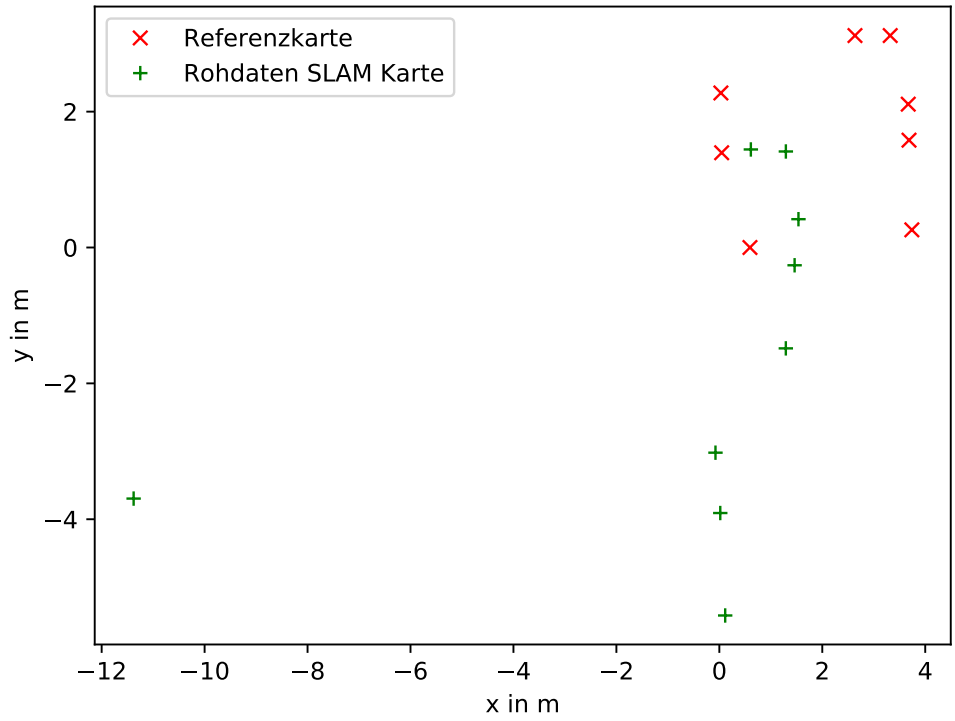


Abbildung 7.4: Ergebnis einer fehlerhaften Kartierung basierend auf Assoziationsfehlern

denen die Kameradaten keine ArUco-Marker enthalten. In diesen Abschnitten wird die Bewegung der Drohne alleine anhand der Bewegungsmodells prädiziert. Fehler in der Modellierung der Drohne führen hier zwangsläufig zu einem Fehler in der Bewegungsschätzung. Wird nach so einem Abschnitt anschließend ein ArUco-Marker entdeckt, welcher noch nicht in der Karte vorhanden ist, wird dieser, ausgehend von der falschen Positionsschätzung der Drohne, fehlerhaft in der Karte initialisiert.
Ein Grund für diese unzureichende Modellierung sind zum einen fehlerbehaftete Modellparameter der Drohne. Ein weiterer maßgeblicher Punkt hierbei ist jedoch das „unberechenbare“ Verhalten der Tello EDU. Bei einigen Kartierungen ist aufgefallen, dass die interne Regelung der Tello EDU teilweise nach geradlinigen Flügen in eine Richtung beim Wegnehmen des Vorschubs eigenständig ein Gegenschubmanöver durchführt, um auf der Stelle stehen zu bleiben. Leider konnte dieses Verhalten nicht mit modelliert werden, da hierbei keine Gesetzmäßigkeit festgestellt werden konnte.

Weitere Fehlerquellen

Die bereits zuvor erwähnte Referenzkarte ist eine weitere Fehlerquelle für die Ungenauigkeit des Systems. Der beim Einmessen der Karte entstehende Fehler kann leider nicht genau bestimmt werden. Es wird jedoch angenommen, dass dieser deutlich über dem Messfehler des verwendeten Messgeräts von $\pm 1{,}5\,\mathrm{mm}$ liegt.
Ein weiterer Faktor hierbei ist die Parametrisierung des EKF-SLAM Algorithmus. Insbesondere das Prozessrauschen kann hierbei nicht gemessen, sondern nur empirisch ermittelt werden.
Eine fehlerhafte Kamerakalibrierung wird hierbei als Fehlerquelle weitestgehend ausgeschlossen, da in [Dit19] das Verfahren zum bestimmen der intrinsischen Kameraparameter validiert wurde.
Des Weiteren können Fehler im Ablauf des SLAM-Algorithmus' ebenfalls weitestgehend ausgeschlossen werden, da die erreichte Genauigkeit in der Simulation deutlich bessere Ergebnisse erzielt, welche annähernd an den Referenzwerten liegen.

Es sei zudem an dieser Stelle noch gesagt, dass der Referenzwert von $ACE = 0{,}023\,\mathrm{m}$ ein sehr gutes Ergebnis ist. Wie bereits in Kapitel 3.3 erwähnt, unterliegt das System aus [MS+18] jedoch der großen Einschränkung, dass sich immer mindestens zwei ArUco-Marker im Bild befinden müssen, anhand derer die Bewegung der Kamera bestimmt werden kann.

7.2 Funktionalität der Anlage

Abschließend wird in diesem Kapitel noch auf die in Abschnitt 4 gestellten Anforderungen eingegangen. Hierbei wird die in der Analyse der Anforderungen aufgestellte Struktur beibehalten.
Da viele der aufgestellten Anforderungen rein funktional sind, kann die Erfüllung dieser Anforderungen nicht mit Messwerten belegt werden. Das Erreichen dieser Anforderungen geht hierbei aus der Konzeption, der Softwarearchitektur oder der Anwendung des Systems hervor.

Funktionale Anforderungen

Die funktionalen Anforderungen aus Tabelle 4.7 sind aus den Anwendungsfällen abgeleitet. In der Tabelle 7.3 ist festgehalten, ob diese Anforderungen erfüllt wurden oder nicht.
Die Forderung nach einem MMI wird hierbei als „zum Teil erfüllt“ bewertet, da die Schnittstelle theoretisch vorhanden ist und auch vom HMI genutzt wird, es jedoch noch keine Tests bezüglich des MMI gab.

Tabelle 7.3: Erfüllung der funktionalen Anforderungen

Anforderung	Kommentar	Erfüllt
HMI	Alle Anforderungen an das HMI sind erfüllt. Das System lässt sich über die Tastatur bedienen. Während der Kartierung werden die aktuelle Karte sowie der Videostream auf dem Bildschirm angezeigt.	✓
MMI	Anstelle des HMI kann ein MMI das System bedienen. Dies wurde jedoch nicht getestet und wird deshalb als „zum Teil erfüllt“ gekennzeichnet.	○
Start	Der Befehl Start wird von der Drohne unterstützt. Erst nach dem Start können Befehle an die Drohne gesendet werden.	✓
Landen	Der Befehl Landen wird von der Drohne unterstützt. Erst nach dem Start können Befehle an die Drohne gesendet werden.	✓
Notstopp	Der Befehl Notstopp wird von der Drohne unterstützt. Die Aktorik wird ausgeschaltet und die Kartierung gestoppt.	✓
Steuerbefehle	Das System unterstützt die Befehle Starten, Landen, Kartierung starten, Kartierung stoppen und Notstopp.	✓
Kartierung	Die Kartierung kann unabhängig vom Flugzustand der Drohne gestartet und gestoppt werden.	✓
Speichern der Karte	Die Karte wird als *csv*-Datei abgespeichert.	✓

Anforderungen bezüglich der Drohne

In der Tabelle 7.4 wird auf die drohnenbezogenen Anforderungen eingegangen.

Tabelle 7.4: Erfüllung der drohnenbezogenen Anforderungen

Anforderung	Kommentar	Erfüllt
Tello EDU	Für das vorliegende System wird die Tello EDU verwendet.	✓
Schnittstelle der Drohne	Wie dem Klassendiagramm zu entnehmen ist, werden die geforderten Funktionalitäten Kommunikationsaufbau, Start, Landung, Bewegung und Notstopp unterstützt.	✓
Verwendete Sensorik	Das System verwendet nur die drohneninterne Sensorik.	✓

Allgemeine Anforderungen an das System

Zu den allgemeinen Anforderungen an das System gehört unter anderem das Erreichen einer Ausführungsfrequenz von größer $1{,}6\,\mathrm{Hz}$. Für diese Ausführungsfrequenz wird als worst-case Szenario angenommen, dass die Karte bereits aus allen verfügbaren Landmarken besteht und eine hohe Anzahl an Landmarken im aktuellen Bild zu sehen ist. Die gemessene mittlere Ausführungszeit beträgt $0{,}076\,\mathrm{s}$, was zu einer Ausführungsfrequenz von $f_s = 13{,}16\,\mathrm{Hz}$ führt. Die angestrebte minimale mittlere Ausführungsfrequenz von $1{,}6\,\mathrm{Hz}$ wird demnach erreicht.
Das Kriterium zur Datenassoziation des Systems wird als „zum Teil erfüll“ bewertet, da es bei keiner Kartierung zu einer Verwechslung von Landmarken gekommen ist, jedoch in der Umgebung Landmarken entdeckt wurden, welche eigentlich gar nicht vorhanden sind.

Tabelle 7.5: Erfüllung der allgemeinen Anforderungen

Anforderung	**Kommentar**	**Erfüllt**
SLAM Algorithmus	Es wird ein EKF-SLAM Algorithmus verwendet.	✓
Ausführungsfrequenz	Es wird eine Ausführungsfrequenz von $f_s = 13{,}16\,\mathrm{Hz} > 1{,}6\,\mathrm{Hz}$ erreicht.	✓
Datenauswertung	Durch den in Abschnitt 6.3 verwendeten FIFO-Speicher, welcher bei vollem Speicher die ältesten Bilder löscht, werden nur die neuesten Bilddaten verarbeitet.	✓
3D-SLAM	Der Algorithmus liefert eine dreidimensionale Karte. Die enthaltenen Landmarken werden durch ihre x-, y- und z-Koordinaten angegeben.	✓
Datenassoziation	Es treten keine Verwechslungen von Landmarken auf. Teilweise werden jedoch Landmarken dort erkannt, wo gar keine sind.	○

Anforderungen an den SLAM Algorithmus

Auf die Anforderungen bezüglich des SLAM Algorithmus' wird in Tabelle 7.6 eingegangen. Das Kriterium der Genauigkeit ist nicht Erfüllt. Die erreichte ACE liegt deutlich über dem Referenz-ACE.
Der Messaufwand für die Referenzkarte fällt zwar gering aus, jedoch ist die nicht zu bestimmende Messungenauigkeit ein großer Nachteil. Da diese Lösung nicht optimal ist, wird der Messaufwand mit „zum Teil erfüllt“ bewertet.

Tabelle 7.6: Erfüllung der Anforderungen an den SLAM Algorithmus

Anforderung	**Kommentar**	**Erfüllt**
Genauigkeit	Der beste erreichte ACE des Systems liegt mit $0{,}26\,\mathrm{m}$ deutlich über dem Referenz-ACE von $0{,}023\,\mathrm{m}$	×
Messaufwand Referenzkarte	Der Messaufwand der Referenzkarte ist zwar gering, jedoch kann die Messgenauigkeit bei diesem Verfahren nicht genau bestimmt werden.	○
Implementierungsaufwand	Das System ist in der vorgegebenen Zeit implementiert worden.	✓

8 Fazit und Ausblick

In dieser Arbeit wurde ein System zur Kartierung von Innenräumen anhand der Bilddaten eines Quadrocopters entwickelt. Der Innenraum wird mit Hilfe von ArUco-Markern diskret Kartiert.
Die mit dem System aufgenommenen Karten weisen hierbei nur eine geringe Genauigkeit auf. Diese geringe Genauigkeit ist jedoch, mit Bezug auf die eingangs aufgeführte Motivation einer mobilen Navigationsapp für Gebäude, noch akzeptabel. Um eine Person innerhalb eines Gebäudes zu lokalisieren und ihr den Weg zu weisen reicht es theoretisch schon aus, die Blickrichtung und den Raum zu identifizieren, in dem sich die Person befindet. Anhand dieser Informationen können weitere Anweisungen bezüglich des Weges gegeben werden.
Abgesehen von der Genauigkeit der Karte ist die Lösung mit den ArUco-Markern als Landmarken in der echten Anwendung nur bedingt verwendbar. Für eine flächendeckende Karte eines Gebäudes müssten unzählige Marker verteilt werden. Zum einen führt dies zu einem recht hohen Hardwareaufwand. Zum anderen kommt noch ein ästhetischer Aspekt hinzu: Ein Gebäude, in dem an den Wänden unzählige künstliche Marker hängen, wirkt befremdlich. Eine denkbarer Kompromiss wären hierbei Marker in einer Farbe außerhalb des für den Menschen sichtbaren Spektrums.
Beim Debuggen des Systems hat sich die mitentwickelte Simulationsumgebung als äußerst nützlich erwiesen. Durch diese ist es möglich, neben den vom EKF-SLAM geschätzten Zuständen auch die echten Systemzustände einzusehen, was bei der Fehlersuche eine große Hilfe ist. Ein weiterer Vorteil hierbei ist, dass die Entwicklung unabhängig von dem Akkuladezustand des Quadrocopters durchgeführt werden kann. Die Simulationsumgebung bietet eine solide Infrastruktur für weitere Arbeiten an dem Thema.

Als verbesserungswürdig an dem System wird das händische Einmessen der Referenzkarte angesehen. Der hierbei auftretende Messfehler kann nur schwer eingeschätzt werden, was eine exakte Beurteilung der Genauigkeit des Systems erschwert. Für zukünftige Arbeiten wäre es denkbar, zum Erstellen der Referenzkarte das System aus [MSMC18] zu verwenden.
Wie bereits zuvor erwähnt, ist die Lösung zur Kartierung eines Raumes durch ArUco-Marker für die Praxis eher unbrauchbar. Die Alternative hierzu ist die Verwendung von natürlichen Landmarken. Da diese jedoch, anders als die ArUco-Marker, keine Tiefeninformationen zur Verfügung stellen, müsste hierbei das Verfahren der *inversen Tiefen-Parametrisierung* aus Kapitel 3.1.3 angewandt werden. Denkbar wäre auch eine Kombination aus natürlichen und künstlichen Landmarken, bei dem die ArUco-Marker nur zum

Einmessen der Karte als Stützpunkte im Raum verteilt werden. Mit der Verwendung von natürlichen Landmarken steigt auch potentiell die Anzahl der Landmarken im Raum. Da diese Anzahl quadratisch in den Rechenaufwand des EKF-SALMs eingeht, müsste dieser durch den weitaus performanteren Fast-SLAM Algorithmus ausgetauscht werden.

Literatur

[Ack16] Evan Ackerman. „Cheap Lidar: The Key to Making Self-Driving Cars Affordable“. In: *IEEE Spectrum* (2016). URL: https://spectrum.ieee.org/transportation/advanced-cars/cheap-lidar-the-key-to-making-selfdriving-cars-affordable.

[Anaa] *GroundSdk Android*. Parrot. 2019. URL: https://developer.parrot.com/docs/groundsdk-android-samples/index.html.

[Anab] *OlympeSDK*. Parrot. 2019. URL: https://developer.parrot.com/docs/olympe/.

[BLF16] Jürgen Beyerer, Fernando Puente León und Christian Frese. *Automatische Sichtprüfung: Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung*. Springer-Verlag, 2016.

[Bra00] G. Bradski. „The OpenCV Library“. In: *Dr. Dobb's Journal of Software Tools* (2000).

[CDM08] Javier Civera, Andrew J Davison und JM Martinez Montiel. „Inverse depth parametrization for monocular SLAM“. In: *IEEE transactions on robotics* 24.5 (2008), S. 932–945.

[Clo] *CloudCompare*. Software (GPL License). 2019. URL: https://www.danielgm.net/cc/.

[Dit19] Thorben Dittmar. *Entwicklung eines optischen Systems fÃ¼r den autonomen Landeanflug von unbemannten Luftfahrzeugen*. 2019.

[EPF14] David Eigen, Christian Puhrsch und Rob Fergus. „Depth map prediction from a single image using a multi-scale deep network“. In: *Advances in neural information processing systems*. 2014, S. 2366–2374.

[ESC14] Jakob Engel, Thomas Schöps und Daniel Cremers. „LSD-SLAM: Large-scale direct monocular SLAM“. In: *European conference on computer vision*. Springer. 2014, S. 834–849.

[FS37] Finsterwalder und Scheufele. *Das Rückwärtsschneiden im Raum*. Herbert Wichmann, 1937.

[Gar+16] Ravi Garg u. a. „Unsupervised cnn for single view depth estimation: Geometry to the rescue“. In: *European Conference on Computer Vision*. Springer. 2016, S. 740–756.

[GJ+14] Sergio Garrido-Jurado u. a. „Automatic generation and detection of highly reliable fiducial markers under occlusion“. In: *Pattern Recognition* 47.6 (2014), S. 2280–2292.

[Han11] Tobias Hanning. *High precision camera calibration*. Springer, 2011.

[Han+14] Ankur Handa u. a. „A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM“. In: *2014 IEEE international conference on Robotics and automation (ICRA)*. IEEE. 2014, S. 1524–1531.

[HLN12] Joachim Hertzberg, Kai Lingemann und Andreas Nüchter. *Mobile Roboter: Eine Einführung aus Sicht der Informatik*. Springer-Verlag, 2012.

[HVV19] HVV. 2019. URL: `https://www.hvv.de/`.

[IIJ16] Jun-Hyuck Im, Sung-Hyuck Im und Gyu-In Jee. „Vertical corner feature based precise vehicle localization using 3D LIDAR in urban area“. In: *Sensors* 16.8 (2016), S. 1268.

[Jäh13] Bernd Jähne. *Digitale Bildverarbeitung*. Springer-Verlag, 2013.

[Jen+06] Patric Jensfelt u. a. „A framework for vision based bearing only 3D SLAM“. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, S. 1944–1950.

[Kae+11] Michael Kaess u. a. „iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering“. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, S. 3281–3288.

[Kal60] Rudolph Emil Kalman. „A new approach to linear filtering and prediction problems“. In: *Journal of basic Engineering* 82.1 (1960), S. 35–45.

[KB99] Hirokazu Kato und Mark Billinghurst. „Marker tracking and hmd calibration for a video-based augmented reality conferencing system“. In: *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. IEEE. 1999, S. 85–94.

[Kra+11] Tomáš Krajník u. a. „AR-drone as a platform for robotic research and education“. In: *International conference on research and education in robotics*. Springer. 2011, S. 172–186.

[KYY12] Zeyneb Kurt-Yavuz und Sirma Yavuz. „A comparison of EKF, UKF, FastSLAM2. 0, and UKF-based FastSLAM algorithms“. In: *2012 IEEE 16th International Conference on Intelligent Engineering Systems (INES)*. IEEE. 2012, S. 37–43.

[Low04] David G Lowe. „Distinctive image features from scale-invariant keypoints“. In: *International journal of computer vision* 60.2 (2004), S. 91–110.

[Lu18] Xiao Xin Lu. „A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation“. In: *Journal of Physics: Conference Series*. Bd. 1087. 5. IOP Publishing. 2018, S. 052009.

[Lun10] Jan Lunze. *Regelungstechnik 2: Mehrgrößensysteme, Digitale Regelung*. Springer-Verlag, 2010.

[Man18] Peter Mandl. *TCP und UDP Internals: Protokolle und Programmierung*. Springer-Verlag, 2018.

[Mar15] Francesco Martinelli. „A robot localization system combining RSSI and phase shift in UHF-RFID signals“. In: *IEEE Transactions on Control Systems Technology* 23.5 (2015), S. 1782–1796.

[MAT17] Raul Mur-Artal und Juan D Tardós. „Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras“. In: *IEEE Transactions on Robotics* 33.5 (2017), S. 1255–1262.

[MD17] Reiner Marchthaler und Sebastian Dingler. *Kalman-Filter*. Bd. 30. Springer, 2017.

[MS+18] Rafael Munoz-Salinas u. a. „Mapping and localization from planar markers“. In: *Pattern Recognition* 73 (2018), S. 158–171.

[MSMC18] Rafael Munoz-Salinas und Rafael Medina-Carnicer. „SPM-SLAM: Simultaneous localization and mapping with squared planar markers“. In: *Pattern Recognition* 86 (2018), S. 156–171.

[MSMC19] Rafael Munoz-Salinas und Rafael Medina-Carnicer. „UcoSLAM: Simultaneous Localization and Mapping by Fusion of KeyPoints and Squared Planar Markers“. In: *arXiv preprint arXiv:1902.03729* (2019).

[MSN11] Amir Monjazeb, Jurek Z Sasiadek und D Necsulescu. „Autonomous navigation among large number of nearby landmarks using FastSLAM and EKF-SLAM-A comparative study“. In: *2011 16th International Conference on Methods & Models in Automation & Robotics*. IEEE. 2011, S. 369–374.

[Nar+15] Luigi Nardi u. a. „Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM“. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, S. 5783–5790.

[NBB16] Michael Neunert, Michael Bloesch und Jonas Buchli. „An open source, fiducial based, visual-inertial motion capture system“. In: *2016 19th International Conference on Information Fusion (FUSION)*. IEEE. 2016, S. 1523–1530.

[PD19] Bernd Pfrommer und Kostas Daniilidis. „TagSLAM: Robust SLAM with Fiducial Markers“. In: *arXiv preprint arXiv:1910.00679* (2019).

[PGB18] Mitesh Patel, Andreas Girgensohn und Jacob Biehl. „Fusing Map Information with a Probabilistic Sensor Model for Indoor Localization Using RF Beacons“. In: *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE. 2018, S. 1–8.

[Sch19] Erik Dominik Schnell. *Entwicklung von Methoden der Regelungstechnik unbemannter Fahrzeuge fÃ¼r die Verfolgung von Soll-Trajektorien*. Aug. 2019.

[SMN08] JZ Sasiadek, A Monjazeb und D Necsulescu. „Navigation of an autonomous mobile robot using EKF-SLAM and FastSLAM“. In: *2008 16th Mediterranean conference on Control and automation*. IEEE. 2008, S. 517–522.

[SR14] Herbert Süße und Erik Rodner. *Bildverarbeitung und Objekterkennung*. Springer, 2014.

[Stu+12] Jürgen Sturm u. a. „A benchmark for the evaluation of RGB-D SLAM systems“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, S. 573–580.

[Stu+17] Lukas von Stumberg u. a. „From monocular SLAM to autonomous drone exploration“. In: *2017 European Conference on Mobile Robots (ECMR)*. IEEE. 2017, S. 1–8.

[TBF05] Sebastian Thrun, Wolfram Burgard und Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[Tel] *Tello SDK*. 1.3.0.0. Ryze Tech. URL: https://terra-1-g.djicdn.com/2d4dce68897a46b19fc717f3576b7c6a/Tello%20%E7%BC%96%E7%A8%8B%E7%9B%B8%E5%85%B3/For%20Tello/Tello%20SDK%20Documentation%20EN_1.3_1122.pdf.

[Wei15] Karsten Weicker. *Evolutionäre algorithmen*. Springer-Verlag, 2015.

[WO16] John Wang und Edwin Olson. „AprilTag 2: Efficient and robust fiducial detection“. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, S. 4193–4198.

[Zho+17] Tinghui Zhou u. a. „Unsupervised learning of depth and ego-motion from video“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, S. 1851–1858.

Anhang

Anhang A1: Thesis als PDF-Datei

Anhang A2: Python Programmcode

Anhang A3: Messergebnisse

Die Anhänge A1 bis A3 sind in elektronischer Form auf einer CD abgelegt und können bei Prof. Dr. -Ing. Marc Hensel oder Prof. Dr. -Ing. Dipl. -Kfm. Jörg Dahlkemper eingesehen werden.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 2. Januar 2020	
Ort, Datum	Unterschrift