Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Thorben Dittmar

Entwicklung eines optischen Systems für den autonomen Landeanflug von unbemannten Luftfahrzeugen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Thorben Dittmar

Entwicklung eines optischen Systems für den autonomen Landeanflug von unbemannten Luftfahrzeugen

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

Betreuender Prüfer: Prof. Dr. Ing. Hensel
Zweitgutachter: Prof. Dr. Neumann

Abgegeben am 20. Oktober 2019

Thorben Dittmar

Thema der Masterthesis

Entwicklung eines optischen Systems für den autonomen Landeanflug von unbemannten Luftfahrzeugen

Stichworte

UAV, autonome Landung, Trajektorien, künstliche Marker, Optik, Bildverarbeitung

Kurzzusammenfassung

In dieser Arbeit wird ein System für die autonome Landung von unbemannten Luftfahrzeugen entwickelt. Dazu wird die First-Person-View Kamera an einem Quadrokopter verwendet und ArUco-Marker, die in der Landezone aufgestellt werden.

Thorben Dittmar

Title of the paper

Development of an optical system for autonomous landing of unmanned aerial vehicles

Keywords

UAV, autonomous landing, trajectories, fidicual marker, optics, image processing

Abstract

In this thesis a system is developed for the autonomous landing of unmanned aerial vehicles. The system use the First-Person-View camera of a quadrocopter and some ArUco-Markers that are placed in the landing zone.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

CCD	Charge Coupled device
CMOS	Complementary Metal-Oxide-Semiconductor
FOV	Field of View
FPV	First Person View
IMU	Inertial Measurement Unit
PAPI	Precision Appraoch Path Indicator
UAV	Unmanned Aerial Vehicle
VASI	Visual Approach Slope Indiator

Symbolverzeichnis

u, v, w	Bildkoordinaten
x_c, y_c, z_c	Kartesische Koordinaten im Kamera-Koordinaten-System
x_w, y_w, z_w	Kartesische Koordinaten im Welt-Koordinaten-System
$\mathbf{A}$	Übergangsmatrix
$\mathbf{B}$	Eingangsmatrix
$\mathbf{C}$	Ausgangsmatrix
$\mathbf{D}$	Durchgangsmatrix
$\mathbf{x}$	Zustandsvektor
$\mathbf{y}$	Ausgangsvektor

1. Einleitung

1.1. Ziel

Ziel dieser Arbeit ist es, ein Landesystem zu entwickeln, welches in der Lage ist, autonome unbemannte Luftfahrzeuge in einem vorgegebenen Gleitwinkel zu landen. Die Luftfahrzeuge sollen die Landebahn selbstständig lokalisieren und dort sicher landen. Primär soll das System für einen autonomen Paraglider entwickelt werden. Es sollen aber prinzipiell alle Arten von unbemannten Luftfahrzeugen, die in Deutschland ohne Sondererlaubnis geflogen werden dürfen, mit diesem System ausgerüstet werden können. Als Prototyp soll das System mit einem Quadrokopter umgesetzt werden.

1.2. Motivation

Heutzutage gibt es eine Vielzahl von Anwendungsfällen von unbemannten Luftfahrzeugen. Autonome Luftfahrzeuge sollen beispielsweise für das Ausliefern von Paketen oder das Orten von hilfsbedürftigen Menschen im Meer eingesetzt werden. Damit die Luftfahrzeuge zum Paket abliefern oder nach der Erfüllung ihrer Aufgabe wieder autonom landen können, braucht das Luftfahrzeug ein System, mit dem eine sichere Landung möglich ist.
Viele Landesysteme, die für autonome Luftfahrzeuge präsentiert wurden [BD13, LSY+14], beziehen sich auf Multikopter, die in der Lage sind senkrecht zu landen. Daraus folgt die Motivation ein System zu entwickeln, welches auch horizontal landende Luftfahrzeuge sicher landen kann. Aufgrund der fehlenden GPS-Signale in Innenräumen und der möglichen Ungenauigkeiten von GPS-Signalen, ist es erstrebenswert ein System zu entwickeln, das in der Lage ist ohne GPS-Signale zu landen.

1.3. Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen zu den Themen digitaler Bildgewinnung, zum Kalman-Filter und zu den Flugphasen erläutert. In Kapitel 3 wird dargelegt, wie Piloten heutzutage in

der zivilen Luftfahrt bei der Landung unterstützt werden. Außerdem werden andere Arbeiten zu diesem Thema vorgestellt und das Vorgehen bei Trajektorienregelungen erläutert. Die Anforderungen werden in Kapitel 4 aufgeführt. Die Konzeption und Auswahl eines Systems erfolgt in Kapitel 5. In Kapitel 6 folgt die Umsetzung der Konzeption und in Kapitel 7 werden die Resultate der Umsetzung dargelegt. Ein Ausblick und ein Fazit werden in Kapitel 8 formuliert.

2. Grundlagen

2.1. Flugphasen

Laut der internationalen zivilen Luftfahrtorganisation kann ein Flug in die unterschiedlichsten Flugphasen unterteilt werden. In Abbildung 2.1 sind die unterschiedlichen Flugphasen laut der Definition von The Boeing Company dargestellt. Aufgrund der Möglichkeit, die Flugphasen in ganz unterschiedliche Bereiche zu unterteilen, wird für diese Arbeit die im Folgenden beschriebene Aufteilung angenommen.
Grundlegend kann der Flug in Start, Reiseflug und Landung unterteilt werden. Je nach Luftfahrzeug kann die Unterteilung noch detaillierter erfolgen. Da sich diese Arbeit nur mit der Landung von Luftfahrzeugen befasst, wird der Start und der Reiseflug in dieser Arbeit nicht weiter definiert. Die erste Phase der Landung wird als Sinkflug bezeichnet. Dabei wird das Luftfahrzeug aus der Reiseflughöhe in eine Flughöhe gebracht, aus der der Landeanflug beginnen kann. Im Anschluss an diese Phase folgt der eigentliche Landeanflug. In dieser Phase wird versucht, das Luftfahrzeug auf einen vorgegebenen Gleitweg zu steuern und dort zu halten. Ist es in dieser Phase nicht möglich das Luftfahrzeug bis zu einem gewissen Punkt auf Kurs zu bringen, muss die Landung abgebrochen werden und gegebenenfalls durchgestartet werden. Im Anschluss an den Landeanflug folgt der Endanflug. In dieser Phase ist ein Abbruch der Landung nicht mehr möglich. Ist diese Phase erreicht, erfolgt das Aufsetzen auf den Boden. Im Anschluss an das Aufsetzen erfolgt das Rollen auf dem Rollfeld.
Je nach Luftfahrzeug ist es notwendig während der Landung einen anderen Gleitwinkel zu wählen. So können Multikopter sogar senkrecht landen.

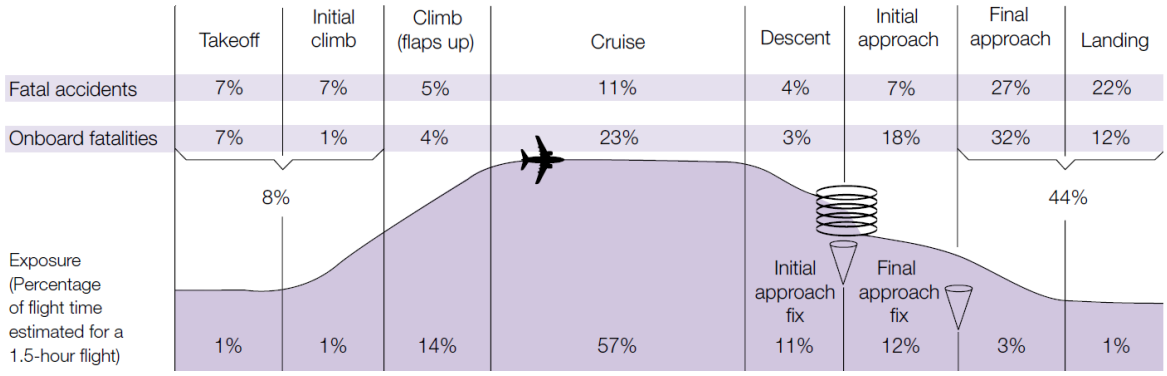
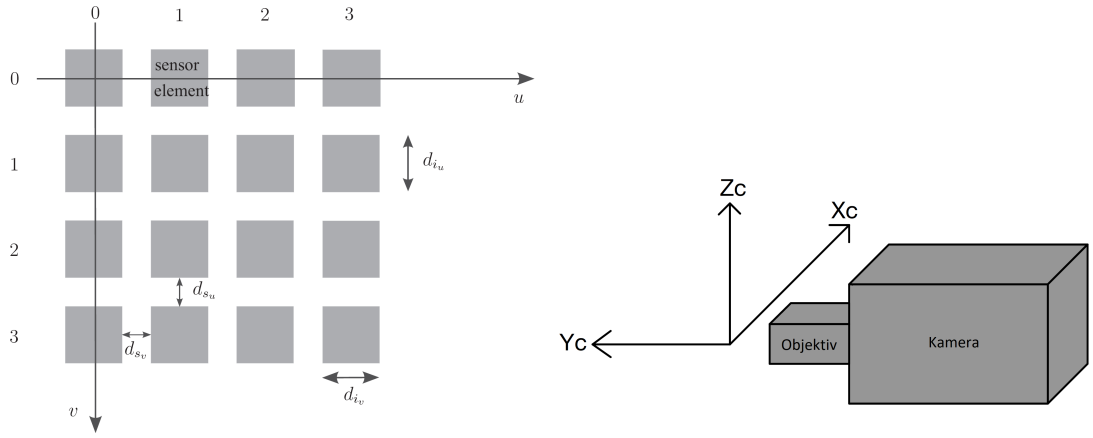


Abbildung 2.1.: Darstellung der Flugphasen [Air17]

2.2. Koordinatensysteme

In diesem Abschnitt wird die in dieser Arbeit verwendete Darstellung und Notation der Position und Orientierung im Raum und im Bild erläutert.
Die Koordinaten in einem Bild werden mit den Formelzeichen u, v, w für Breite, Höhe, Farbkanal beschrieben. Bei einem Bild mit nur einem Farbkanal entfällt der Parameter w. In Abbildung 2.2(a) ist ein Koordinatensystem im Bild dargestellt. Der Ursprung des Bildkoordinatensystems wird in dieser Arbeit in der linken oberen Ecke des Bildes angesetzt.
Für die Koordinaten im Raum werden grundsätzlich die Parameter für die Position mit X, Y, Z angegeben. Mit dem Index c werden die Koordinaten im Kamerakoordinatensystem beschrieben. Weltkoordinaten werden mit dem Index w versehen. In dieser Arbeit wird das Kamerakoordinatensystem wie in Abbildung 2.2(b) dargestellt angenommen. Die Y-Achse ist frontal aus der Kamera heraus gerichtet, die X-Achse aus Sicht der Kamera nach rechts und die Z-Achse ist nach oben gerichtet.
Um eine Orientierung im Raum zu beschreiben, wird grundsätzlich die Darstellung als extrinsische Kardan-Winkel mit der xyz-Konvention verwendet. Dies bedeutet, dass die Orientierung einer Kamera in einem Koordinaten-System mittels der drei Rotationen um die Achsen des Bezugssystems beschrieben wird. Für die Rotationen um die Achsen werden die Formelzeichen α, β, γ verwendet. Eine Drehung um die X-Achse, die mit einem positiven Winkel angegeben wird, bedeutet in dieser Arbeit, dass die Y-Achse in Richtung der Z-Achse gedreht wird.



(a) Bildkoordinaten [Han11] (b) 3D Weltkoordinaten

Abbildung 2.2.: Darstellung der Koordinaten

2.3. Digitale Bildgewinnung

In diesem Abschnitt wird auf die für diese Arbeit relevanten Funktionsweisen von Bildsensoren und auf die mathematischen Grundlagen zur Modellierung von Kameras eingegangen.

2.3.1. Bildsensoren

Für das grundlegende Verständnis der digitalen Bildgewinnung wird an dieser Stelle kurz erläutert, wie die gängigsten Verfahren funktionieren, die Licht in eine digitale Information umwandeln. Grundlegend kann man zwischen Zeilen und Flächenkameras unterscheiden. Zeilenkameras besitzen dabei als Sensor nur eine Zeile von photosensitiven Elementen. Diese Kameras werden hauptsächlich in der Industrie zur Prüfung von Endlosprodukten auf Fließbändern eingesetzt [BLF16]. Für diese Arbeit relevanter sind Flächenkameras. Bei diesen Kameras besteht der Bildsensor aus einer zweidimensionalen Fläche von photosensitiven Elementen. Jedes Element dieses Sensors bildet dabei einen Pixel. Für die photosensitiven Elemente gibt es heute hauptsächlich zwei Technologien. Zum einen gibt es die CCD-Technologie (charge-coupled-device), bei der in einer Halbleiterschicht durch Auftreffen von Photonen Elektronen-Loch-Paare erzeugt werden. Die so entstehende Ladung ist ein Maß für die Anzahl an Photonen, die auf den Pixel getroffen sind. In Abhängigkeit zur Belichtungsdauer kann so auf die Lichtintensität geschlossen werden. Die Umsetzung in ein digitales Signal erfolgt dann durch eine Analog-Digital-Wandler-Schaltung.
Die zweite Technologie beruht auf der CMOS-Technologie (complementary metal-oxide-semiconductor), die ein weit verbreitetes Prinzip für die Konstruktion von Halbleiterschal-

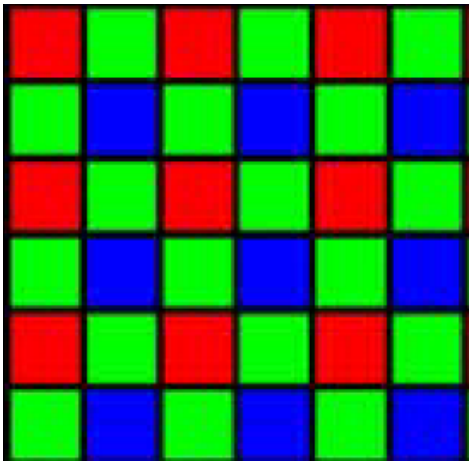


Abbildung 2.3.: Bayer-Pattern [HW08]

tungen ist. Die Detektion von Photonen beruht dabei auf dem gleichen Prinzip wie bei CCD-Sensoren. Allerdings wird die Schaltung zum Auslesen des Ladungsspeichers direkt in den Pixel integriert. Durch die weite Verbreitung der CMOS-Technologie sind diese Sensoren sehr günstig herzustellen und gewinnen immer größere Marktanteile [Fin18].
Monochrome Kameras, die das für den Menschen sichtbare Spektrum abbilden sollen, besitzen häufig keine Filter oder lediglich über der gesamten Pixelfläche einen Infrarot-Sperr-Filter, der die Wellenlänge des Lichtes auf den sichtbaren Bereich beschränkt. Mit diesen Kameras kann aber keine Aussage über die Intensität für bestimmte Wellenlängen getroffen werden. Bei einfachen Farbkameras werden vor den Pixeln unterschiedliche Filter platziert. Dadurch gelangt nur Licht aus einem bestimmten Spektrum an einen Pixel. Am verbreitetsten ist die Unterteilung in die drei Farbkanäle Rot, Grün und Blau und eine Anordnung nach dem Bayer-Pattern [HW08]. In Abbildung 2.3 ist das Bayer-Pattern dargestellt. Nach der Bildaufnahme werden vier Pixel (ein roter, ein blauer und zwei grüne) verwendet, um die Farbinformationen für einen Makropixel zu berechnen. Dieser Vorgang wird als Demosaicing bezeichnet. Die digitale Darstellung der Bilder erfolgt anschließend häufig über ein dreidimensionales Array. Für jedes Pixel wird dabei die Information über die Helligkeit der drei Farbkanäle gespeichert. Nachfolgend werden kurz die wichtigsten Begriffe erläutert, die in Zusammenhang mit der digitalen Bildgewinnung für diese Arbeit von Bedeutung sind.

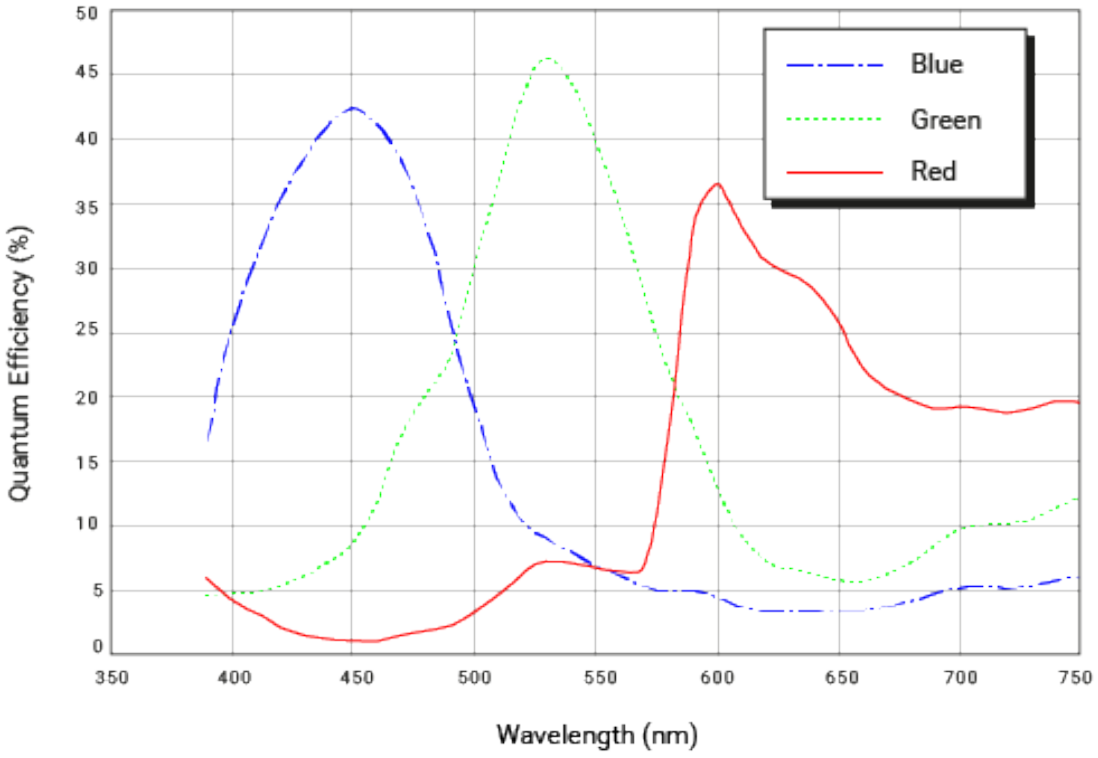


Abbildung 2.4.: Empfindlichkeitskurve Basler 2500-14gc [AG19]

Dynamikbereich

Der Dynamikbereich eines Bildsensors beschreibt den linearen Bereich zwischen dem Punkt mit einem Signal-Rausch-Verhältnis von 1 bis zur Sättigung des Sensors. [G+16]

Füllfaktor

Der Füllfaktor gibt an, wie groß der Anteil der lichtempfindlichen Fläche auf dem Sensorchip ist. [BLF16]

Quantenwirkungsgrad

Der Quantenwirkungsgrad gibt das Verhältnis von erzeugten Elektronen zu auf einen Pixel auftreffenden Photonen an. [BLF16]

Weiß-Abgleich

Bei einem Weiß-Abgleich werden die einzelnen Farbkanäle eines Bildes unterschiedlich gewichtet, um unterschiedliche Beleuchtungsspektren auszugleichen. [BLF16]

2.3.2. Optik

Neben dem Bildsensor benötigt eine Kamera für die Bildgewinnung eine Optik [BLF16], die alle relevanten Objektpunkte auf der Sensorfläche abbildet. Im Folgenden wird kurz erklärt, was bei der Wahl und der Einstellung eines Objektives beachtet werden muss.

In einem Objektiv sind eine oder mehrere Linsen und Blenden enthalten. Die Aufgabe der Linse ist, eine möglichst große Menge von Lichtstrahlen von einem Objekt-Punkt auf den Bildpunkt abzubilden. In Abbildung 2.5 ist die Abbildung mittels einer dünnen Linse dargestellt. Ob ein Bild scharf abgebildet wird, kann über das Abbildungsgesetz mit den Parametern f für die Brennweite der Linse, g für den Abstand des Gegenstands zur Hauptebene der Linse und b für den Abstand des Bildes zur Hauptebene der Linse bestimmt werden. Das Abbildungsgesetz

$$\frac{1}{f} = \frac{1}{g} + \frac{1}{b} \tag{2.1}$$

kann zu

$$f = \frac{g \cdot b}{g + b} \tag{2.2}$$

umgestellt werden, um anzugeben, welche Brennweite bei einem Objektiv eingestellt werden muss, um ein Objekt scharf abzubilden.
Aufgrund der Tatsache, dass das Bild durch die Pixel in diskrete Bereiche unterteilt wird, gibt es einen Toleranzbereich, in dem ein Objekt scharf dargestellt wird. Ein Objekt wird demnach dann scharf abgebildet, wenn der Radius der Zerstreuung der Lichtstrahlen vor oder nach dem Brennpunkt nicht größer ist als die Seitenlänge eines Pixels.
Durch den Einsatz einer Blende ist es möglich, die Schärfentiefe einzustellen. Abbildung 2.6 zeigt die Darstellung des Strahlengangs durch eine Linse mit einer Aperturblende vor der Linse. Die Aperturblende begrenzt die Lichtstrahlen, die auf die Linse treffen. Dadurch wird beeinflusst, welche Strahlen auf den Sensor treffen. Je kleiner das Strahlenbündel ist, dass auf die Linse trifft, desto kleiner ist der Zerstreuungsradius auf der Bildebene. Somit muss für das Einstellen des Objektivs ein Kompromiss zwischen Lichtausbeute und Schärfentiefe gefunden werden.

2.3.3. Projektion von Welt- zu Bildkoordinaten

Für die Abbildung von Welt- in Bildkoordinaten wird eine Projektionsmatrix $\boldsymbol{P}$ gesucht. Im Folgenden wird erläutert, wie diese Matrix bestimmt werden kann.
Eine idealisierte und vereinfachte Darstellung einer Kamera ist die Lochkamera [BLF16]. Mit

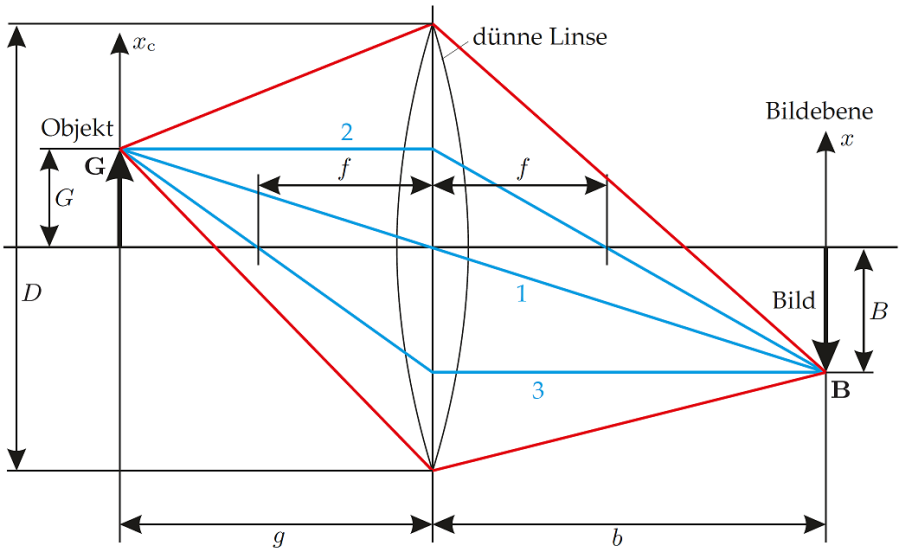


Abbildung 2.5.: Abbildung mittels einer Linse [BLF16]

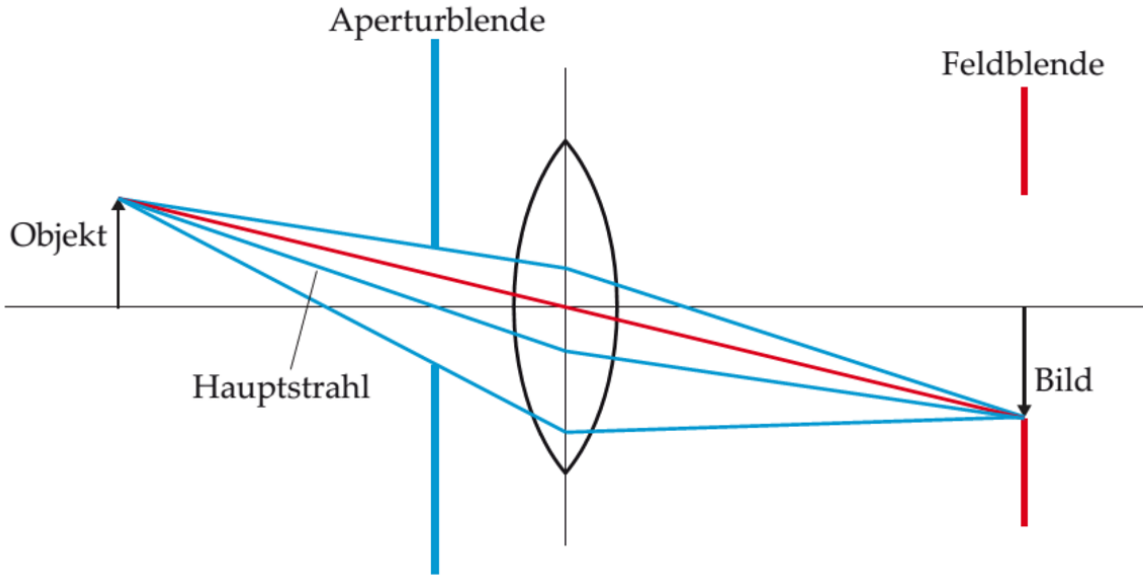


Abbildung 2.6.: Aperturblende [BLF16]

diesem Modell lässt sich die Abbildung eines Punktes im Raum auf die Bildebene beschreiben. Dabei wird angenommen, dass die Lochkamera aus der Bildebene und einer Blende mit einem infinitesimal kleinem Loch besteht, welches nur einen Lichtstrahl passieren lässt. Der Abstand zwischen Blende und Bildebene wird dabei als die Bildweite b bezeichnet. Der Ursprung des Kamerakoordinatensystems liegt dabei im Zentrum der Blende. Die Abbildung der Kamerakoordinaten (x_c, y_c, z_c) auf die Bildkoordinaten (u, y) kann nun mit Hilfe des Strahlensatzes über die Gleichung

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\frac{b}{z_c} \begin{pmatrix} x_c \\ y_c \end{pmatrix} \tag{2.3}$$

beschrieben werden. Um diese Projektion als Matrixmultiplikation darstellen zu können, werden die kartesischen Koordinaten x_c, y_c, z_c bzw. u, v als homogene Koordinaten x_c', y_c', z_c', h_c bzw. u', v', h dargestellt. Dazu wird ein weiterer Parameter h eingeführt. Die Abbildung von homogenen zu kartesischen Koordinaten erfolgt dabei durch Dividieren von x_c', y_c', z_c' bzw. u, v durch h_c bzw. h. Damit lässt sich die Projektion von Kamerakoordinaten in Bildkoordinaten durch

$$\begin{pmatrix} u' \\ v' \\ h \end{pmatrix} = \begin{pmatrix} -b & 0 & 0 & 0 \\ 0 & -b & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_c' \\ y_c' \\ z_c' \\ h_c \end{pmatrix} \tag{2.4}$$

beschreiben. Alle Parameter, die die Projektion von Kamera in Bildkoordinaten beschreiben, werden als intrinsische Parameter bezeichnet. Deshalb wird die Abbildungsmatrix nachfolgend als Projektionsmatrix $\boldsymbol{P}_i$ bezeichnet.
Um die Position auf der Bildebene in Pixeln zu beschreiben, muss der Bildsensor, wie er in Kapitel 2.3.1 beschrieben wird, in der Projektion mit modelliert werden. Dazu muss die Bildweite b durch die Pixelgröße s geteilt werden. Dabei wird bei rechteckigen Pixelflächen zwischen der Pixellänge in x und y Richtung unterschieden. Des Weiteren ist der Ursprung des Bildkoordinatensystems nicht zwingend im Ursprung des Kamerakoordinatensystems. Diese Verschiebung wird über den Parameter c in Pixeln angegeben. Für einen Sensor, dessen Zentrum auf der optischen Achse liegt und bei dem der Bildpunkt (0,0) in der linken oberen Ecke des Bildes liegt, ist c die Hälfte der jeweiligen Seitenlänge des Sensors. Mit der Annahme, dass das Bildkoordinatensystem nicht schiefwinklig zum Kamerakoordinatensystem liegt, kann die Projektion vom Kamerakoordinatensystem zum Bildpunkt nun durch

$$\begin{pmatrix} u' \\ v' \\ h \end{pmatrix} = \begin{pmatrix} \frac{-b}{s_x} & 0 & c_x & 0 \\ 0 & \frac{-b}{s_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_c' \\ y_c' \\ z_c' \\ h_c \end{pmatrix} \tag{2.5}$$

beschrieben werden.
Um die Abbildung von Weltkoordinaten zu Bildpunkten zu beschreiben werden außerdem die extrinsischen Parameter benötigt. Diese Parameter beschreiben eine Rotation $\boldsymbol{R}$ und Translation $\boldsymbol{t}$. Für homogene Koordinaten kann dies wieder als eine Projektionsmatrix, die mit $\boldsymbol{P}_e$ bezeichnet wird, dargestellt werden. Die Projektion von homogenen Weltkoordinaten zu homogenen Bildkoordinaten kann somit durch

$$\begin{pmatrix} u' \\ v' \\ h \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{-b}{s_x} & 0 & c_x & 0 \\ 0 & \frac{-b}{s_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} & \boldsymbol{R} & & \boldsymbol{t} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{P}} \begin{pmatrix} x'_w \\ y'_w \\ z'_w \\ h_w \end{pmatrix} \tag{2.6}$$

angegeben werden. Die Multiplikation der Matrizen $\boldsymbol{P}_i$ und $\boldsymbol{P}_e$ ergibt die Projektionsmatrix $\boldsymbol{P}$.

2.4. Kalman-Filter

Der nach seinem Entwickler benannte Kalman-Filter [MD17] ist ein Verfahren, mit dem aus verschiedenen Messungen die Zustände eines Systems geschätzt werden können. Besonders häufig wird der Kalman-Filter angewendet, wenn die Postion von Objekten und Fahrzeugen geschätzt werden soll.
Bei der Messung von Zuständen ist es in der Regel nicht möglich einen Zustand ohne Rauschen zu messen. Durch einen Tiefpassfilter kann Rauschen auf dem Sensorsignal abgeschwächt werden. Allerdings eilt das gefilterte Sensorsignal dadurch dem eigentlichen Signal zeitlich nach. Bei der Verwendung eines Kalman-Filters wird deshalb das ungefilterte Sensorsignal verwendet, um die Zustände zu schätzen.
Der Kalman-Filter nutzt das diskrete mathematische Modell mit der Übergangsmatrix **A**, der Eingangsmatrix **B**, der Ausgangsmatrix **C** und der Durchgangsmatrix **D** des Systems, um iterativ die Zustände **x** des Systems zu schätzen. Ein Iterationsschritt setzt sich dabei aus zwei einzelnen Schritten zusammen. Der erste Schritt ist die Prädiktion der aktuellen Zustände zum nächsten Abtastschritt mit den gegebenen Eingangssignalen **u** und dem mathematischen Modell. Die Prädiktion erfolgt durch die Berechnung von

$$\hat{\mathbf{x}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{x}}(k) + \mathbf{B} \cdot \mathbf{u}(k) \tag{2.7}$$

$$\hat{\mathbf{P}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{P}}(k) \cdot \mathbf{A}^T + \mathbf{Q}(k). \tag{2.8}$$

Das Ergebnis sind die aus dem Modell geschätzten Zustände $\hat{\mathbf{x}}$ und die zur Schätzung gehörende Kovarianzmatrix $\hat{\mathbf{P}}$. Die Matrix **Q** spiegelt dabei die Ungenauigkeit der Modellbildung

durch eine Kovarianzmatrix wieder. Nach der Prädiktion findet die Korrektur mit den gemessenen Werten statt. Dafür wird zuerst die Verstärkung **K** des Kalman-Filters durch

$$\mathbf{K}(k) = \hat{\mathbf{P}}(k) \cdot \mathbf{C}^T \cdot (\mathbf{C} \cdot \hat{\mathbf{P}}(k) \cdot \mathbf{C}^T + \mathbf{R}(k))^{-1} \tag{2.9}$$

berechnet. **R** ist dabei die Kovarianzmatrix des Messrauschens. Es wird beim Kalman-Filter angenommen, dass das Messrauschen durch die Varianz vollständig beschrieben werden kann. Das bedeutet, dass das Messrauschen als mittelwertfrei angenommen wird. Im nächsten Schritt werden die Zustände und die Kovarianzmatrix durch die Berechnung von

$$\tilde{\mathbf{x}}(k) = \hat{\mathbf{x}}(k) + \mathbf{K}(k) \cdot (\mathbf{y}(k) - \mathbf{C} \cdot \hat{\mathbf{x}} - \mathbf{D} \cdot \mathbf{u}(k)) \tag{2.10}$$

$$\tilde{\mathbf{P}}(k) = (\mathbf{I} - \mathbf{K}(k) \cdot \mathbf{C}) \cdot \hat{\mathbf{P}}(k) \tag{2.11}$$

korrigiert.
Bei der Verwendung des Kalman-Filters ist es schwierig die Varianzen des System- und Messrauschens zu bestimmen [MD17]. Das Messrauschen kann häufig durch eine vorherige Messung abgeschätzt und im Anschluss als konstant betrachtet werden. Sind das Mess- und das Systemrauschen nicht konstant, dann kann ein adaptiver Kalman-Filter implementiert werden. Die Vorgehensweise dafür wird hier nicht weiter betrachtet, weil in dieser Arbeit das Rauschen als konstant angenommen wird. Das vorher erläuterte Verfahren ist nur für lineare Systeme geeignet, wie es in dieser Arbeit auch verwendet wird. Wenn ein nichtlineares System mit einem Kalman-Filter beobachtet werden soll, dann können Extended- oder Unscented-Kalman-Filter verwendet werden.

3. Stand der Technik

3.1. Landesysteme an Flugplätzen

In der Luftfahrt gibt es verschiedene Landesysteme an Flugplätzen, die einem Piloten die Landung eines Luftfahrzeugs erleichtern. Unterschieden wird dabei in optische Systeme, die den Sichtflug unterstützen oder Instrumentenlandesysteme, die mit Funk-Signalen funktionieren [Men14]. Beim Sichtflug unterstützen optische Markierungen und Signale am Flugplatz den Piloten beim Navigieren des Luftfahrtzeugs. Beim Instrumentenflug erhält der Pilot die Informationen über Instrumente in seinem Cockpit.
Die optischen Systeme können dabei unterteilt werden in Markierungen und Befeuerungen. Markierungen haben den Nachteil, dass sie selbst kein Licht ausstrahlen und somit bei dunklen Lichtverhältnissen nicht gesehen werden können. Befeuerungen hingegen sind Lichtsignale, die von Lampen oder Ähnlichem ausgehen und somit auch bei schwierigen Lichtverhältnissen vom Piloten gesehen werden können. [Men13]
Bei Flugzeugen ist es von Bedeutung, in welchem Winkel die Landebahn angeflogen wird. Damit der Pilot in der Lage ist, den Gleitwinkel optimal zu verfolgen, gibt es verschiedene Systeme, die ihn dabei unterstützen. Im Folgenden werden beispielhaft verschiedene Landehilfen für den Sicht- und Instrumentenanflug beschrieben, die die internationale Zivilluftfahrtorganisation standardisiert hat.

3.1.1. Instrumentenlandesystem

Bei einem Instrumentenlandesystem [Men14] wird der Pilot über Instrumente in seinem Cockpit darüber informiert, ob er sich auf dem richtigen Landeanflugweg befindet. Um den Gleitpfad vorzugeben, werden am Flugplatz Antennen installiert. Für die horizontale und die vertikale Führung werden jeweils zwei Antennen benötigt, die eine modulierte Strahlung erzeugen. Eine Antenne sendet dabei Strahlung mit einer Modulationsfrequenz von 90 Hz und die andere Antenne von 150 Hz aus. In Abbildung 3.1 ist ein Modulationsdiagramm dargestellt. Der Flugpfad wird dabei durch die Strahlungsdiagramme festgelegt. Befindet sich das Flugzeug genau auf dem Gleitpfad, dann ist die Modulationsgrad-Differenz der beiden Frequenzen gleich 0. Je nachdem welche der beiden Frequenzen überwiegt, kann der Pilot

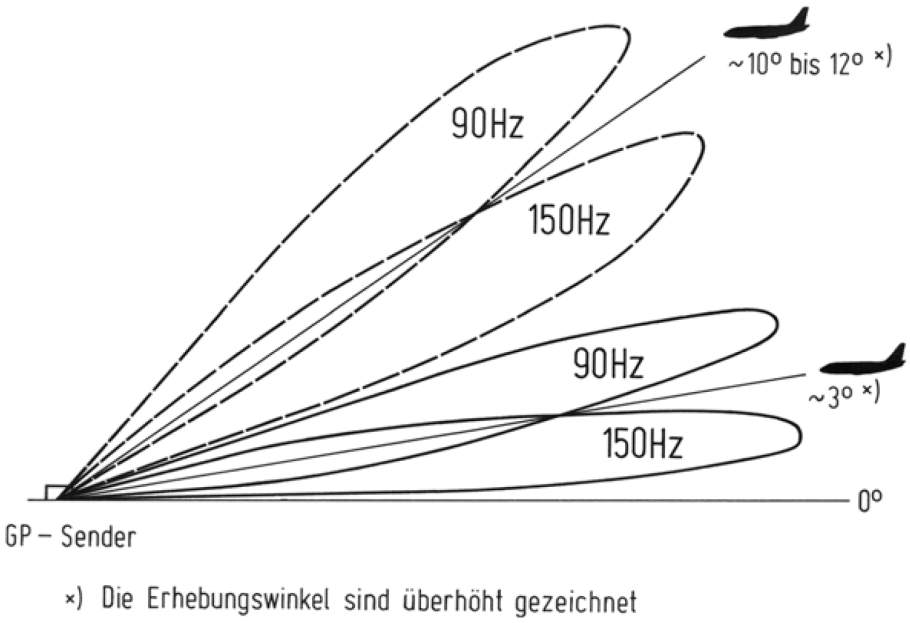
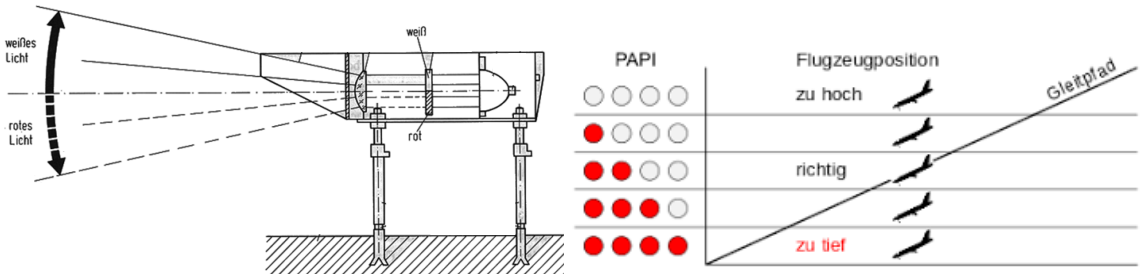


Abbildung 3.1.: Modulationsdiagramm eines Gleitwegsenders [Men14]

dann das Flugzeug in die eine oder die andere Richtung steuern. Die Anzeige im Cockpit erfolgt durch ein Abspielen der Frequenzen im Kopfhörer oder durch die Anzeige an den Instrumenten.

3.1.2. Befeuerungssysteme

An Flugplätzen gibt es häufig Lichtsignale, die dem Piloten die Richtung weisen. Für den richtigen Gleitpfad gibt es das verbreitete Visual Approach Slope Indicator (VASI) System. Dabei wird dem Piloten über unterschiedliche Farben von einem Lichtsignal mitgeteilt, ob er zu hoch oder zu niedrig auf dem Landeanflug ist. Erreicht wird dies durch eine rote und eine weiße Lichtquelle, die so angeordnet werden, dass man von oben nur das weiße und von unten nur das rote Licht sieht. Wie dies umgesetzt werden kann, ist in Abbildung 3.2 schematisch dargestellt. Beim VASI werden zwei Reihen von Lampen verwendet, die so angeordnet werden, dass der Pilot auf dem korrekten Anflug eine weiße und eine rote Reihe von Lichtern sieht. Eine Weiterentwicklung des VASI ist das Precision Approach Path Indicator (PAPI) System. Dabei werden den Piloten vier Lichter gezeigt. Das hat den Vorteil, dass der Pilot eine genauere Angabe darüber bekommt, wie stark er vom Kurs abgewichen ist. Dies hat zu einem deutlich verbesserten Anflugverhalten von Flugzeugen geführt [SJ76].



(a) Schema einer PAPI Lampe (b) Darstellung der Lichter, die der Pilot sieht

Abbildung 3.2.: Funktionsweise des PAPI-Systems [Men14]

3.2. Verwandte Veröffentlichungen

In der Literatur finden sich viele verschiedene Arbeiten zum Thema der Regelung und der automatischen Landung von UAV mittels Bilddaten von einer monokularen Kamera, die am UAV montiert ist. In [ZQQ18] ist beschrieben, wie es mit Hilfe von mehreren Markern, die auf dem Boden verteilt sind, möglich ist, eine Indoor Lokalisierung von einem UAV zu erstellen, die ausreichend gute Ergebnisse liefert, um das UAV mit der resultierenden Positionsschätzung zu regeln. In [SK17] wird ein ähnlicher Marker wie in [ZQQ18] verwendet. Allerdings wird in dieser Arbeit lediglich ein einzelner Marker verwendet, der auf dem Boden platziert ist. Dabei wurde gezeigt, dass es ausreicht, die Positionsschätzung von einem einzelnen Marker zu verwenden, um das UAV zu landen, wenn die Inertial Measurement Unit (IMU) des UAV in die Positionsschätzung miteinbezogen wird. Die IMU sorgt dafür, dass die Positionsschätzung valide bleibt, wenn der Marker nicht im Bild detektiert werden kann. Die Landung in dieser Arbeit erfolgt dabei durch einen waagerechten Anflug bis das UAV über dem Marker in der Luft steht. Anschließend erfolgt ein senkrechter Sinkflug auf den Marker.
Ein anderer Ansatz wird in [LSY+14] beschrieben. Darin wird untersucht, wie durch den Einsatz eines zweiten höher fliegenden UAV die Lokalisierung in einem großen Bereich ohne GPS möglich ist. Die senkrechte Landung erfolgt anschließend, wie in den anderen Arbeiten, mit einem planaren Marker, der auf dem Boden liegt. Die Regelung in dieser Veröffentlichung basiert dabei auf einer Kombination von Fuzzy-Logik und einem neuronalen Netz.
In [BD13] ist beschrieben, wie durch einen Extended Kalman Filter (EKF) die Daten von einer IMU und einem SLAM Algorithmus dazu verwendet werden, um ein UAV mittels der Time-to-Contact Methode zu regeln. Die Landung über eine effizientere Trajektorie wird in der Veröffentlichung [GSS17] beschrieben. Dabei wird versucht die Landung nicht senkrecht erfolgen zu lassen, sondern durch einen Gleitflug, wie er bei Flugzeugen üblich ist. Verwendet wird dazu ein runder farbiger Marker, der auf dem Boden platziert wird.
Einen anderen Schwerpunkt hat die Veröffentlichung [AK12]. Dabei geht es um die Lokalisierung eines UAV mittels einer nicht kalibrierten Kamera anhand der Landebahn. Der einzige

Parameter, der für diese Methode bekannt sein muss, ist die Position der Markierungen auf der Landebahn.
Die verschiedenen Ansätze aus den beschriebenen Veröffentlichungen liefern die Grundlagen für die in Kapitel 5 erstellte Konzeption für ein Landesystem. Dabei werden die in den Veröffentlichungen häufig verwendeten quadratischen Marker, sowie die in Kapitel 3.1 beschriebenen Landesysteme berücksichtigt.

3.3. 3D-Rekonstruktion

3.3.1. Schätzung der intrinsischen Parameter

Die in Abschnitt 2.3.3 erläuterten intrinsischen Parameter werden für die Berechnung von 3D Daten aus den Bildern benötigt. Die Matrix $\boldsymbol{P}_i$ kann bei bekannten Kameraparametern berechnet werden. In einigen Fällen kann es aber vorkommen, dass die notwendigen Parameter des Kamerasystems nicht vorhanden sind. In so einem Fall kann durch eine Menge n von korrespondierenden Punkten $\boldsymbol{w}_k$ im Kamerakoordinatensystem und Bildpunkten $\boldsymbol{i}_k$ eine Matrix $\tilde{\boldsymbol{P}}_k$ bestimmt werden, sodass

$$\boldsymbol{i}_k = \tilde{\boldsymbol{P}}_i \, \boldsymbol{w}_k \qquad k = 0, 1, \ldots n \tag{3.1}$$

gilt. [BLF16, Han11]
Die Bestimmung der Matrix $\tilde{\boldsymbol{P}}_i$ erfolgt dabei durch die Minimierung einer Abbildungs-Fehlerfunktion. In [Han11] sind verschiedene Möglichkeiten beschrieben wie diese Fehlerfunktion aussehen kann.
Für eine erfolgreiche Bestimmung von $\tilde{\boldsymbol{P}}_i$ ist es erforderlich, dass die Punkte nicht in einer Ebene liegen, da das Gleichungssystem ansonsten unterbestimmt ist [BLF16]. Aufgrund der Tatsache, dass bei der Bestimmung der Bildkoordinaten und bei der Vermessung der Weltkoordinaten Fehler auftreten können, ist es von Vorteil, mehr als die minimal erforderlichen sechs Punkte-Paare zu verwenden. Des Weiteren sollte der gesamte Bildbereich abgedeckt werden und eine große Variation in der Entfernung zur Kamera vorhanden sein [Han11]. Da dies mit einem einzelnen Bild von einem dreidimensionalem Kalibrierkörper nicht einfach umzusetzen ist, bietet es sich an, die Kalibrierung mit einem planaren Kalibrierkörper durchzuführen, von dem mehrere Bilder in unterschiedlichen Posen vor der Kamera gemacht werden [Zha00]. Häufig werden dafür schachbrettartige Muster verwendet, da es damit relativ einfach möglich ist, die Bildpunkte automatisch zu bestimmen.
Da es bei einem solchen Vorgehen nicht mehr möglich, ist die Punkte des Schachbrettmusters in einem Kamerakoordinatensystem zu bestimmen, werden die extrinsischen Parameter

mit in das Optimierungsproblem einbezogen. Es wird nun eine Lösung für die Gleichung

$$\boldsymbol{i}_k = \tilde{\boldsymbol{P}}_i \, \tilde{\boldsymbol{P}}_{e,k} \, \boldsymbol{w}_k \qquad k = 0, 1, \ldots n \tag{3.2}$$

gesucht. Dabei wird für jedes Bild ein eigener Satz an extrinsischen Parametern bestimmt. Aufgrund der Tatsache, dass sich die intrinsischen Parameter einer Kamera im laufenden Betrieb nicht verändern, solange keine Parameter am Kamerasystem verändert werden, ist es möglich, die Bestimmung der intrinsischen Parametern offline, also vor der Verwendung des Systems, durchzuführen. Deshalb ist es meistens nicht zeitkritisch, die intrinsischen Parameter zu bestimmen. Daher ist es möglich, die Kalibrierung mit einer großen Menge von Bilder durchzuführen, die das Kalibrierpattern in vielen verschieden Posen abbildet, um gute Ergebnisse zu erhalten.

3.3.2. Schätzung der extrinsischen Parameter

Die Schätzung der extrinsischen Parameter einer Kamera ist in vielen Anwendungen der Augmented Reality von Bedeutung [MDAM10]. Dabei ist häufig das Ziel, zu ermitteln, in welcher Pose die Kamera sich zu einem Referenzsystem befindet. Abbildung 3.3 zeigt die Beziehung zwischen einem Koordinatensystem eines Markers und dem Kamerakoordinatensystem. Bei der Schätzung der extrinsischen Parameter ist es im Gegensatz zu den intrinsischen meistens erforderlich, dass die Berechnung in Echtzeit und nur aufgrund eines Bildes erfolgt [GPK09].
In der Literatur wird das Problem in einer Vielzahl von Abhandlungen [GPK09, Lu18, SP06, GHTC03] adressiert. Unter dem Begriff Perspektive-n-Punkte-Problem oder Perspektive-n-Linien-Problem werden verschiedene Lösungsansätze vorgestellt, um aus einer bestimmten Anzahl an Referenzpunkten oder Linien die Perspektive zu berechnen. Grundlegend kann das Problem über iterative, nicht-iterative, lineare und nicht-lineare Verfahren [Lu18] gelöst werden. Wie bei der Schätzung der intrinsischen Parameter ist es möglich, das Problem über eine Fehlerfunktion zu optimieren. Dabei wird genau wie bei den intrinsischen Parametern eine Fehlerfunktion minimiert. In der Veröffentlichung [SP06] wird gezeigt, dass so eine Fehlerfunktion zwei lokale Minima besitzt, wenn man vier Punktpaare verwendet. Damit mit diesem Verfahren eine gute Schätzung erreicht wird, muss entweder die Optimierung ausreichen gut initialisiert werden oder das Ergebnis muss anschließend einer zusätzlichen Prüfung unterzogen werden.
Weit verbreitete und robuste Algorithmen sind unter den Namen P3P [GHTC03] und EPnP [QL99] bekannt. Der P3P-Algorithmus ist besonders interessant bei der Verwendung von planaren Markern wie sie in Abschnitt 3.3.3 beschrieben werden, weil er vier Punktpaare für die Berechnung der Pose benötigt. Bereits 1903 wurde bewiesen, dass es bei einer kalibrierten Kamera und drei bekannten Punktpaaren genau vier Lösungen für die Kamera-Pose gibt. Diese vier Lösungen werden beim P3P-Algorithmus berechnet und mittels des vierten

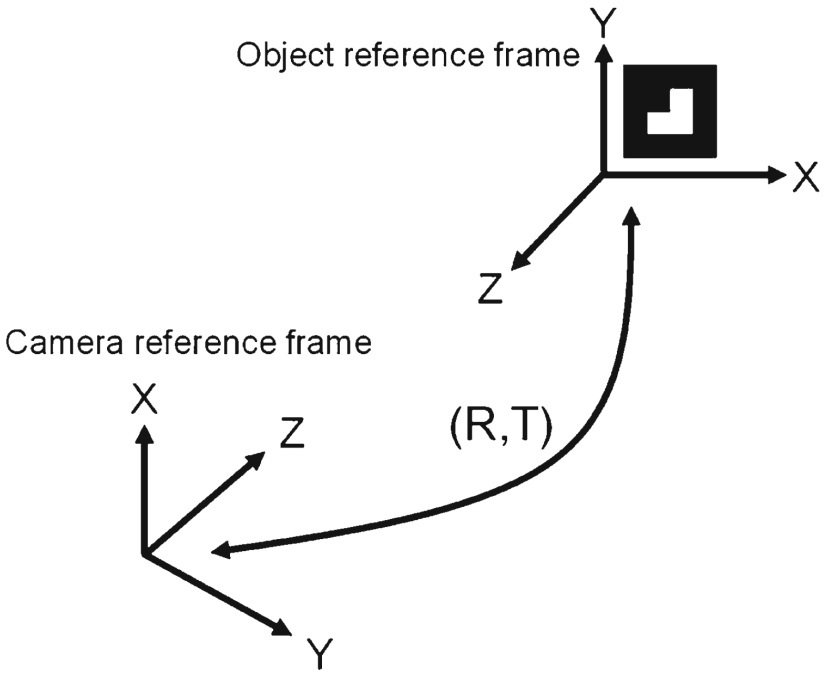


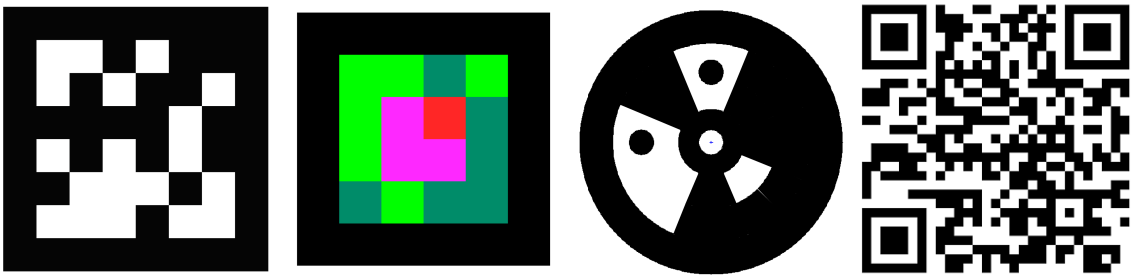
Abbildung 3.3.: Kamera- und Markerkoordinatensystem [MDAM10]

Punktes wird die beste Lösung ausgewählt.
Durch Rauschen oder andere Fehler bei der Bildung von Punktpaaren ist es häufig wünschenswert das Ergebnis der Schätzung zu verbessern, indem die Anzahl der Punkte erhöht wird. Bei den meisten PnP-Algorithmen ist die Komplexität $\mathcal{O}(n^2)$ oder schlechter. Der EPnP-Algorithmus hat eine Komplexität von $\mathcal{O}(n)$, indem die gegebenen n Punkte über eine gewichtete Summe zu vier Punkten zusammengefasst werden.

3.3.3. Planare Marker

In diesem Abschnitt werden die Marker aufgezeigt, die in Anwendungen der Augmented Reality und der Selbstlokalisierung weit verbreitet sind. Es gibt verschiedene Verfahren, die aufgrund von Kamerabildern eine Lokalisierung vornehmen. Dazu müssen in der Umgebung Punkte gefunden werden, die über mehrere Bilder getrackt werden. Es gibt eine Vielzahl von Abhandlungen, die sich damit beschäftigen, die Punkte in der natürlichen Umgebung zu finden. Dieser Bereich der Lokalisierung wird in dieser Arbeit aber nicht weiter betrachtet.
Wenn es möglich ist, in der Umgebung einen künstlichen Marker zu platzieren, dann kann dadurch der Rechenaufwand stark vermindert werden [LDMC+16]. Außerdem ist es mit einem künstlichen Marker einfacher, eine globale Position zu erhalten, wenn man die Position

des Markers im Weltkoordinaten-System kennt.
Zu den künstlichen Markern gibt es ebenfalls eine Vielzahl an Abhandlungen [WO16, DBH17, NF02, SP10, LDMC+16], die sich mit der Form und Beschaffenheit der Marker befassen. In Abbildung 3.4 sind einige Beispiele abgebildet. Jeder Marker hat Vor- und Nachteile und es muss für die individuelle Anwendung entschieden werden, welcher Marker am besten passt.
AprilTag und ChromaTag sind entwickelt worden, um möglichst effizient und robust den Marker zu detektieren. Die Berechnung der Pose bei diesen Markern erfolgt dabei über die äußeren Ecken und die Kodierung im Inneren der Marker. Dabei ist es möglich durch die Kodierung verschiedene Marker in einem Bild zu unterscheiden.
Bei dem Intersense Marker ist es hingegen nicht möglich, die Marker voneinander zu unterscheiden. Runde Marker sind besonders robust gegen Verzerrungen und sind präziser bei der Lokalisierung. Der QR-Code hingegen wurde nicht dazu entwickelt eine Pose zu bestimmen, sondern um eine möglichst große Anzahl an Informationen zu speichern. Der QR-Code wird heute häufig verwendet, um Informationen über ein Smartphone einzulesen.
Unabhängig von der Wahl des Markers ist für die Performance des Markers entscheidend, wie groß er gewählt wird und wie die Umgebungsbedingungen bei der Bildaufnahme sind [LDMC+16]. Die Open-Source Bildverarbeitungssoftware OpenCV hat eine Vielzahl von Funktionen implementiert, die sich auf die AprilTag ähnlichen ArUco Marker beziehen. Dadurch gibt es viele Abhandlungen [TH17, SK17] zur Lokalisierung mittels ArUco-Makern. Die ArUco Bibliothek ermöglicht es, die Marker selbst zu generieren. So ist es möglich den Code im Inneren des Markers selbst festzulegen und je nach Anwendung einen Code mit 4x4, 6x6, usw. Rechtecken zu definieren. Dadurch ist es möglich für jede Anwendung das optimale Verhältnis zwischen Detektionsgeschwindigkeit und Markeranzahl zu wählen.
Für das Verständnis der Vorteile von ArUco-Markern wird im Folgenden kurz erläutert, wie eine Detektion der ArUco-Marker [RRMSMC18] abläuft. Der erste Schritt ist eine Segmentierung des Bildes. Dazu wird auf dem gesamten Bild ein lokaler adaptiver Schwellwert angewendet. Damit wird ein Pixel zu einer 0 bzw. einer 1, wenn er dunkler bzw. heller ist als der Mittelwert der Pixel um ihn herum. Die Größe des Fenster ist dabei frei wählbar. Das schwarze Quadrat auf dem weißen Hintergrund beim ArUco-Marker ist besonders hilfreich für den adaptiven Schwellwert, weil die Kanten des Markers dadurch den größten möglichen Kontrast zum Hintergrund haben. Im zweiten Schritt werden die Konturen im binären Bild nach ihrer Ähnlichkeit mit Rechtecken gefiltert. Alle Konturen, die nicht einen bestimmten Wert überschreiten, werden in den nachfolgenden Schritten nicht berücksichtigt. Anschließend erfolgt die Analyse des inneren Codes des Markers.
Dazu werden die Perspektiven der Konturen korrigiert. Als Beispiel wird angenommen, dass eine Kontur einen Marker zeigt, der um 45 Grad gedreht ist. Dann wird die Region in der Kontur aus dem originalen Bild so abgebildet, dass eine Drehung um -45 Grad erfolgt. Somit ist das Innere der Kontur ohne Drehungen und Verzerrungen dargestellt. Auf diesem entzerrten Bildausschnitt wird wiederum ein Schwellwert mit dem Verfahren von Otsu angewendet. Weil



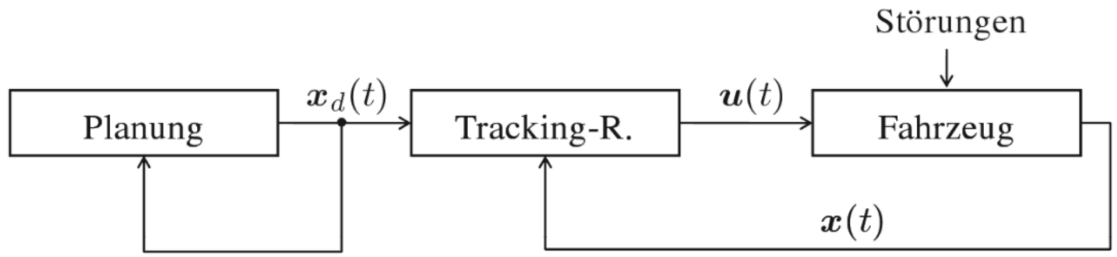
(a) AprilTag [WO16] (b) ChromaTag [DBH17] (c) Intersense [NF02] (d) QR-Code [SP10]

Abbildung 3.4.: Beispiele für künstliche Marker

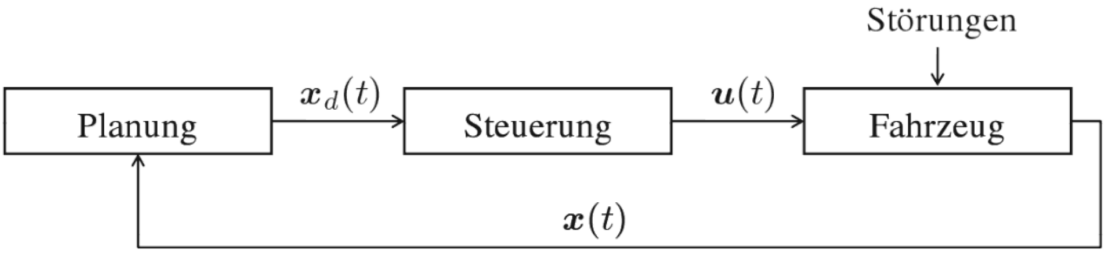
die Kantenlänge der Kontur und die Art der Codierung (4x4, 6x6, etc.) bekannt ist, kann auf die segmentierte Region ein Raster gelegt werden. Sind in einem Rasterpunkt mehr Pixel mit dem Wert 0 bzw. 1, dann wird der Rasterpunkt als eine 0 bzw. eine 1 gewertet. Die Auswertung ergibt damit einen Binär-Code, der mit den Codes der validen Marker abgeglichen werden kann. Wichtig ist dabei, dass alle vier Rotationsmöglichkeiten abgeglichen werden.

3.4. Trajektorien

Um ein Fahrzeug jeglicher Art auf einer Route von einem Punkt A über Punkt C nach Punkt B zu bewegen, ist es notwendig, eine Trajektorie zu bestimmen, auf der sich das Fahrzeug bewegen kann. Die Berechnung der Routenpunkte wird hier nicht behandelt, da es für einen Landeanflug meistens eine feste Route gibt.
Je nach Anwendungsfall können unterschiedliche Verfahren eingesetzt werden, um eine Trajektorie zu erzeugen. Dabei gibt es zwei grundlegend unterschiedliche Ansätze, wann die Berechnung einer Trajektorie erfolgt. Der eine Ansatz ist laut [Wer11] eine Trajektorien-Planung mit einem Low-Level Regler. Dabei wird, wie in Abbildung 3.5(a) dargestellt, die Trajektorien-Planung vollständig vor dem Beginn der Regelung fertiggestellt. Anschließend wird die gesamte Trajektorie abgefahren. Wenn während des Abfahrens der Trajektorie Störungen auftreten, die das Fahrzeug von der Trajektorie entfernen, oder Hindernisse umfahren werden müssen, dann kann es notwendig sein, dass die Trajektorie während des Abfahrens neu berechnet werden muss. Dieses Vorgehen ist in Abbildung 3.5(b) dargestellt. Wenn zu erwarten ist, dass auf dem Pfad der Trajektorie Hindernisse auftauchen oder verschwinden können oder starke Störungen auf das Fahrzeug wirken können, dann ist dieses Vorgehen zwingend notwendig. Der Nachtteil liegt allerdings in dem zusätzlichen Rechenaufwand. Mit diesem Verfahren ist es deshalb schwieriger eine Echtzeitanwendung zu erstellen.



(a) Low-Level Stabilisierung mit Tracking-Regler

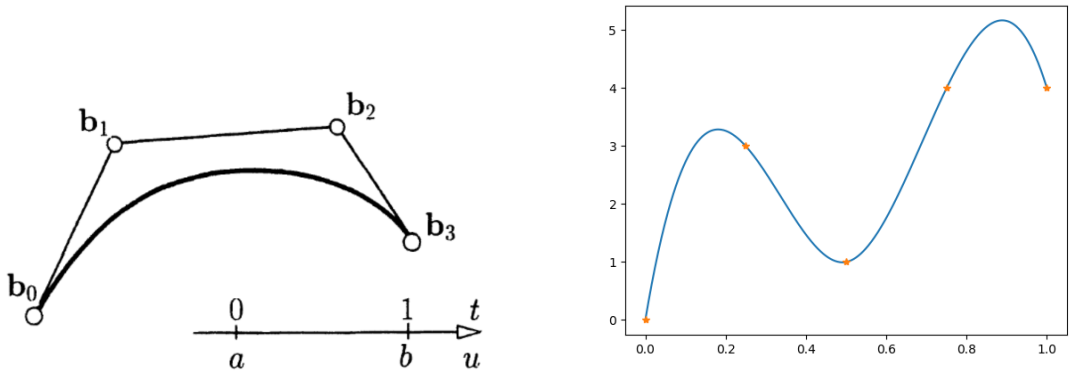


(b) High-Level Stabilisierung

Abbildung 3.5.: Vergleich von unterschiedlichen Zustandsrückführungen [Wer11]

3.4.1. Interpolation der Strecke

Für die Berechnung der Trajektorie ist es notwendig, den Pfad zwischen den Punkten einer Route zu interpolieren. Bei einer Punkt-zu-Punkt Trajektorie erfolgt dies meist durch ein Fitting einer Geraden. Wenn die Route aus einer Vielzahl von Punkten besteht, dann ist es theoretisch möglich, die Trajektorie durch ein Fitting von einem Polynom größerer Ordnung zu interpolieren. Ein Nachteil dabei ist, dass Polynome größerer Ordnung zum Überschwingen neigen. Daher haben sich verschiedene andere Methoden bei der Interpolation von Pfaden durchgesetzt [PBP13, CCE08, WH90, DLLO91].
In der Computergrafik werden häufig Bezíer-Kurven verwendet, um zwischen Punkten zu interpolieren. Die Bezíer-Kurven basieren auf den Bernstein-Polynomen und werden durch die Stützpunkte definiert. Je mehr Stützpunkte es gibt, um so höher ist der Grad der Bezíer-Kurve. Der Vorteil der Bezíer-Kurven ist, dass sie immer innerhalb der konvexen Hülle der Stützpunkte verlaufen und somit nicht überschwingen. Eine Bezíer-Kurve ist dabei immer definiert auf dem Intervall von 0 bis 1 und verläuft durch den Anfangs- und den Endpunkt, aber nicht durch die Stützpunkte dazwischen. In Abbildung 3.6(a) ist eine kubische Bezierkurve dargestellt. Die Berechnungsvorschrift für eine allgemeine Bezíer-Kurve durch n Stützstellen



(a) Kubische Bezíer-Kurve [PBP13] (b) Kubische Spline Interpolation

Abbildung 3.6.: Vergleich von Bezíer-Kurve und Spline

$\mathbf{b}_i$ lautet

$$\mathbf{x} = \mathbf{b}(t) \;=\; \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \mathbf{b}_i, \quad 0 \leq t \leq 1. \tag{3.3}$$

Je nach Anwendung ist es entweder wünschenswert, dass die Stützpunkte einer Trajektorie wie bei den Bezíer-Kurven überschliffen werden, oder dass jeder Stützpunkt exakt durchfahren wird. Wenn jeder Stützpunkt durchfahren werden soll, eignet sich die verbreitete Spline-Interpolation. Ein Spline ist eine Funktion, die abschnittsweise durch unterschiedliche Polynome definiert ist. Im Gegensatz zu den Bezíer-Kurven verlaufen die Splines durch alle Stützpunkte. Die Anzahl der Stützpunkte ist dabei nicht ausschlaggebend für den Grad der Polynome. In Abbildung 3.6(b) ist ein kubischer Spline durch fünf Punkte dargestellt. In der Abbildung ist deutlich zu erkennen, dass der Spline nicht innerhalb der komplexen Hülle der Stützpunkte bleibt.
Um den Verlauf eines Splines zu beeinflussen, können hermitische Splines verwendet werden. Bei diesen Splines ist es möglich, zu jedem Stützpunkt auch eine Tangente vorzugeben. Durch das Umrechnen der Stützpunkte lassen sich die Splines auch als Bezíer-Kurve darstellen und umgekehrt. Ein Vorteil den Splines gegenüber Bezíer-Kurven haben, ist die Eigenschaft der Lokalität. Wenn in einer Bezíer-Kurve ein Stützpunkt verschoben wird, dann hat das einen Einfluss auf die komplette Trajektorie. Bei einem Spline hingegen werden nur die Streckenabschnitte beeinflusst, die unmittelbar mit dem Stützpunkt zusammenhängen.

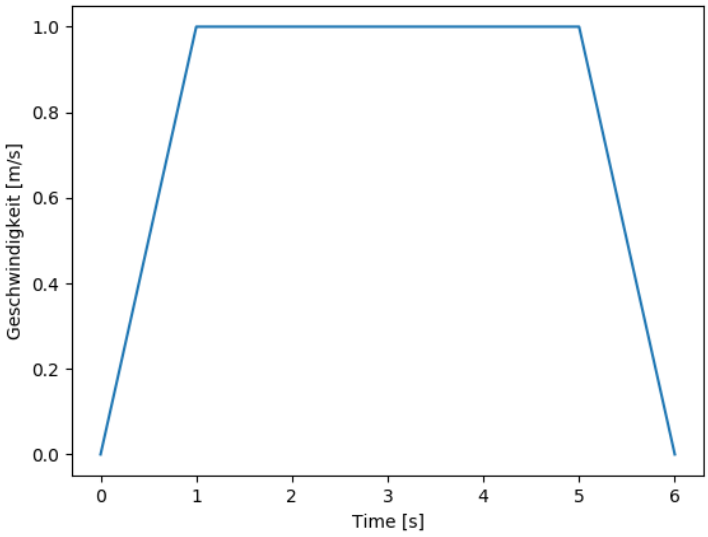


Abbildung 3.7.: Fiktives zeitoptimales Geschwindigkeitsprofil

3.4.2. Geschwindigkeitsprofil

Durch die erläuterten Interpolationsarten ist es lediglich möglich, die Strecke auf dem Intervall von 0 bis 1 zu interpolieren. Für eine vollständige Trajektorie ist es aber zusätzlich notwendig festzulegen, zu welchem Zeitpunkt das Fahrzeug an welcher Stelle der Trajektorie sein soll und wie groß die anderen Zustände wie Geschwindigkeit oder Beschleunigung sein sollen. Eine verbreitete Methode [Web19] dafür ist, die gesamte Länge der Strecke zu errechnen und dafür ein Geschwindigkeitsprofil zu erstellen, welches die Grenzen des Fahrzeugs einhält. In Abbildung 3.7 ist ein fiktives zeitoptimales Geschwindigkeitsprofil dargestellt, für eine maximale Geschwindigkeit von 1 $\frac{m}{s}$ und einer maximalen Beschleunigung von 1 $\frac{m}{s^2}$. Als zeitoptimal wird dieses Profil betrachtet, weil mit der maximal möglichen Beschleunigung auf die maximale Geschwindigkeit beschleunigt wird und anschließend mit der maximalen Bremsbeschleunigung gebremst wird. Damit wird die Strecke in der kürzest möglichen Zeit zurückgelegt.
Mit dem Profil erhält man die Zeit, in der die Strecke zurückgelegt wird, und über das Integral der Geschwindigkeit erhält man die zurückgelegte Strecke zu bestimmten Zeitpunkten. Über eine Look-Up Tabelle ist es nun möglich, die Zeitpunkte für die Positionen auf der Strecke zu bestimmen.

4. Analyse der Anforderungen

Die Anforderungen an das zu entwickelnde System ergeben sich aus unterschiedlichen Quellen. Professor Dr. Hensel hat als Auftraggeber für diese Arbeit das Ziel ausgegeben, ein System zu entwickeln, welches in der Lage ist, einen Landeanflug für Starrflügler zu unterstützen. In dieser Arbeit soll gezeigt werden, wie es möglich ist, einen präzisen Gleitflug während einer Landung zu ermöglichen. Nachträgliche Arbeiten sollen auf der Grundlage dieser Arbeit in der Lage sein, eine vollständig autonome Landung mit einem Starrflügler durchführen zu können. Eingegrenzt werden die Anforderungen durch den gesetzlichen Rahmen, der in Deutschland den Betrieb von unbemannten Luftfahrzeugen reguliert.

4.1. Anwendungsfälle

In diesem Abschnitt werden die Anwendungsfälle für das Landesystem dargelegt. In Abbildung 4.1 sind die Anwendungsfälle in einem UML-Diagramm dargestellt. Grundlegend gibt es zwei Anwender, für die das System konzipiert werden soll. Zum einen kann der automatisierte Landeanflug von einem menschlichen Piloten angefordert werden und zum anderen kann dies durch einen anderen Autopiloten erfolgen. Als Landeanflug gilt hier die in Kapitel 2.1 beschriebene Definition vom Landeanflug.
Bevor die Landung durchgeführt werden kann, ist es notwendig, dass der Landeanflug vom Anwender parametriert wird. Dieser Fall wird in der Tabelle 4.3 beschrieben.
Der Anwendungsfall der automatischen Landung ist in Abbildung 4.1 dargestellt. Wichtig dabei ist, dass die automatische Landung vom Anwender angefordert werden muss, damit sie ausgeführt wird.
Der Abbruch eines Landeanflugs kann dabei intern eingeleitet werden oder vom Anwender. Die Beschreibung dazu ist in Abbildung 4.2 aufgelistet. Ob ein Abbruch noch erfolgreich durchgeführt werden kann, wird durch das System entschieden.

Tabelle 4.1.: Anwendungsfall: Automatische Landung

Beschreibung	Das System soll die Landung eines UAV übernehmen. Damit die Landung gestartet wird, muss sie von der aktuellen Flugkontrolle angefordert werden.
Vorbedingung	Die Landeroute ist festgelegt und parametriert.
Nachbedingung	Das UAV ist gelandet oder der Landeversuch ist abgelehnt worden.
Beteiligungen	Flugkontrolle, UAV
Auslöser	Die aktuelle Flugkontrolle fordert eine Landung beim Landesystem an.
Standardvorgehen	1. Die aktuelle Flugkontrolle fordert eine Landung an. 2. Das System prüft die Möglichkeit einer Landung. 3. Der Landevorgang wird gestartet. 4. Die Landung ist erfolgreich durchgeführt.

Tabelle 4.2.: Anwendungsfall: Abbruch der Landung

Beschreibung	Intern: Das Landesystem hat ein Problem beim Landeanflug erkannt und gibt die Kontrolle über das UAV zurück. Extern: Der Anwender fordert den Abbruch der Landung an.
Vorbedingung	Der Landeanflug ist gestartet.
Nachbedingung	Das Landesystem hat keine Kontrolle mehr über das UAV.
Beteiligungen	Flugkontrolle
Auslöser	Intern: Das Landesystem löst einen Abbruch aus. Extern: Der Anwender löst den Abbruch aus.
Standardvorgehen	1. Das Landesystem/der Anwender hat ein Problem festgestellt. 2. Das System bringt das UAV in eine sichere Fluglage. 3. Der Anwender übernimmt.

Tabelle 4.3.: Anwendungsfall: Parametrieren

Beschreibung	Der User oder ein anderes Programm übergibt die Parameter für die Landung an das Landesystem.
Vorbedingung	-
Nachbedingung	Das Landesystem hat alle nötigen Parameter, um eine Landung durchzuführen.
Beteiligungen	Flugkontrolle
Auslöser	Der User startet Parametrierung.
Standardvorgehen	1. Eingabe aller Parameter über GUI oder anderes Interface. 2. Prüfen der Daten auf Tauglichkeit für die Landung.

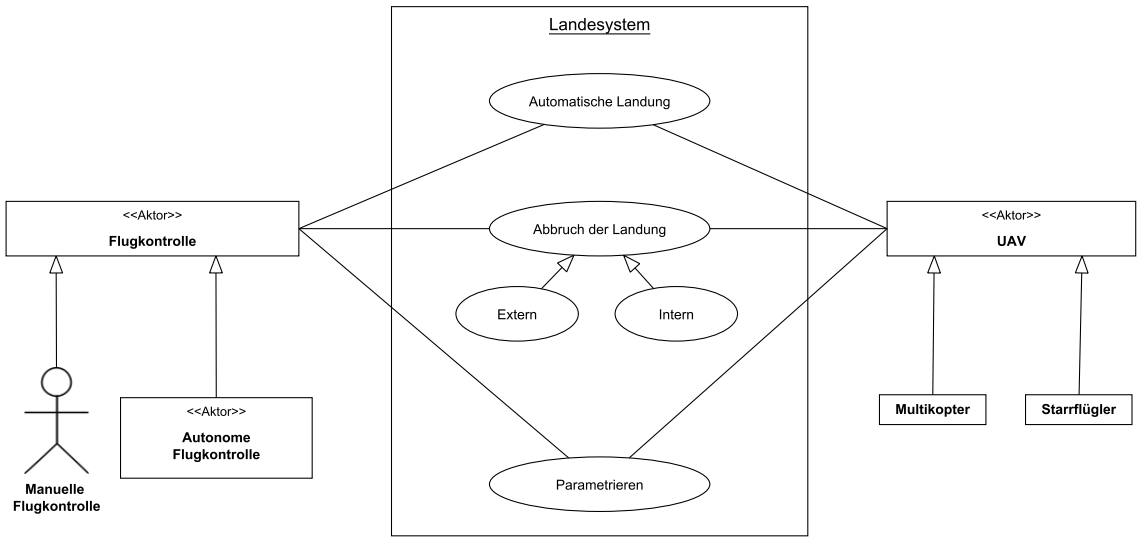


Abbildung 4.1.: Anwendungsfall Diagramm

4.2. Rechtliche Einschränkungen

In Deutschland wird das Fliegen von UAV in § 21 der Luftverkehrs-Ordnung reglementiert [luf19]. Anhand des gesetzlichen Rahmens werden die Anforderungen an das zu entwickelnde Landesystem eingegrenzt, da das zu entwickelnde System nur unter den rechtlichen Bedingungen verwendet werden darf. Im Folgenden werden relevante Bedingungen aufgeführt und die dadurch entstehenden Anforderungen zusammengefasst.
Das Fliegen eines UAV darf nur erfolgen, wenn das UAV in Sichtweite des Piloten ist [luf19]. Bei einem Landeanflug durch das automatische System ist der Pilot im rechtlichen Sinne immer noch der Mensch, der die Verantwortung für den Flug trägt. Deshalb ist es jederzeit erforderlich, dass der menschliche Pilot das UAV sehen kann. Außerdem ist der Flug nur tagsüber erlaubt. Das Fliegen von einem UAV ist außerdem nur gestattet, wenn die Witterungsbedingungen einen sicheren Flug zulassen. Aus den genannten Punkten wird geschlossen, dass beim Verwenden des Landesystems die Sichtbedingungen ausreichen, um die Landebahn beim Start des Landeanflugs zu sehen. Eine Landung bei Nacht oder schlechten Sichtbedingungen muss demnach nicht ermöglicht werden.
Ohne Sondererlaubnis ist das Überschreiten des Abfluggewichtes von 5 kg nicht zulässig. Dies bedeutet, dass das zusätzliche Gewicht des zu entwickelnden Systems an Bord des UAV begrenzt ist. Des Weiteren ist für einen sicheren Flug vom Piloten dafür zu sorgen, dass der Flugbereich gesichert ist und dass die Lande- und Startflächen angemessen in Fläche und Beschaffenheit sind. Dementsprechend kann davon ausgegangen werden, dass die Anforderung an eine Landung nur erfolgt, wenn eine angemessene Landefläche vorhan-

den ist. Das System muss daher bei einer Kennzeichnung der Landebahn durch den Piloten nicht in der Lage sein, die Landezone selbständig auf die Beschaffenheit zu überprüfen.
Für die Entwicklung des System ist es außerdem erforderlich, dass die geltenden Regeln der Bundesnetzagentur für die Frequenznutzung [fre19] eingehalten werden.

4.3. Referenz-UAV

Ziel dieser Arbeit ist es, ein System zu entwickeln, welches eine möglichst große Anzahl von unterschiedlichen UAV landen kann. Für die Entwicklung des Prototypen wird eine Tello EDU verwendet. Dieses Modell ist für den Gebrauch in Innenräumen geeignet. Durch eine verfügbare Python Programmierschnittstelle ist das Entwerfen einer Steuerungssoftware für die Tello EDU mit einem geringeren Aufwand verbunden als bei vergleichbaren Multikoptern. In der Tabelle 4.4 sind die wichtigsten Daten der Tello EDU aufgeführt. Mit der verbauten Steuerung ist die Tello EDU bereits in der Lage, selbständig vertikal zu landen. Des Weiteren bietet die Tello EDU die Möglichkeit, das entwickelte System in einem gesicherten Umfeld zu testen. Aus diesen Gründen wird die Tello EDU als Demonstrator für die Funktionsfähigkeit des Systems verwendet.
Außerdem bietet die Tello die Möglichkeit, die entworfene Steuerung ohne Gefahren für Personen zu testen, indem ein Sicherheitskäfig über der gesamten Drohne montiert wird. In Abbildung 4.2 ist ein Bild der Tello EDU mit dem Sicherheitskäfig abgebildet.

Tabelle 4.4.: Daten des Referenz-UAV [Tel]

Kategorie	Tello EDU
Gewicht	87 g
Zuladung	-
Kamera	720p bei 30 Hz
Geschwindigkeit	8 m/s (28,8 km/h)

Abbildung 4.2.: Tello EDU mit Sicherheitskäfig

4.4. Das Anforderungsprofil

In der Tabelle 4.5 sind die Anforderungen an das System zusammengefasst und kurz beschrieben. Einer der wichtigsten Punkte ist das Gewicht der Komponenten an Bord des Luftfahrzeugs. Je geringer das zusätzliche Gewicht am Luftfahrzeug ist, desto länger kann das Luftfahrzeug in der Luft sein. Da UAV häufig an verschiedenen Orten verwendet werden, soll das System so konzipiert werden, dass es möglichst einfach transportiert und an verschiedenen Orten eingesetzt werden kann. Für die Kalkulation der Kosten soll nicht nur das System für ein spezielles UAV betrachtet werden. Stattdessen soll berücksichtigt werden, dass dieses System von mehreren UAV verwendet werden kann. Deshalb ist darauf zu achten, welche Komponenten mehrfach benötigt werden und welche gleichzeitig von mehreren UAV verwendet werden können.
Die Skalierbarkeit des Systems zielt darauf ab, dass verschiedenste Luftfahrzeuge mit diesem System gelandet werden können, ohne dass grundlegende Anpassungen vorgenommen werden müssen. Die Robustheit des Systems beschreibt, unter welchen Bedingungen das System noch einwandfrei funktioniert. Dieser Punkt ist mit der Priorität *Muss* eingestuft, weil dieses System nur in den gesetzlichen Grenzen aus Abschnitt 4.2 betrieben wird. Zusätzlich werden die Luftfahrzeuge nur betrieben, wenn alle Bedingungen für einen sicheren Betrieb gewährleistet sind. Die Zuverlässigkeit ist insofern wichtig, als dass das System keinen kompletten Systemausfall haben darf, in dem das UAV in einen unsicheren Zustand gerät. Ein fehlgeschlagener Landeanflug, der zum Durchstarten des UAV führt, ist erlaubt.

Tabelle 4.5.: Anforderungen an das Landesystem

Priorität	Kriterium	Beschreibung
Muss	Gewicht	Die Komponenten des Systems, die an Bord des UAV sind, sollen das Abfluggewicht des UAV nicht deutlich erhöhen. Das Abfluggewicht von 5 kg darf nicht überschritten werden.
Muss	Mobilität	Das Gesamtsystem soll so konstruiert werden, dass ein Verlegen der Landebahn möglich ist.
Muss	Zuverlässigkeit	Das System muss in der Lage sein, den Gleitpfad vorzugeben und ein Abweichen davon zu erkennen und zu korrigieren.
Muss	Skalierbarkeit	Das System soll für verschiedene Gleitwinkel anwendbar sein. Insbesondere die für Verkehrsflugzeuge üblichen Winkel von ungefähr 3 Grad [Men14] sollen angeflogen werden können.
Muss	Kosten	Die Kosten für das System sollen das Budget dieser Arbeit von 500 Euro nicht überschreiten.
Kann	Energieverbrauch	Die Elektronik der Komponenten an Bord des UAV darf die maximale Flugzeit durch den Energieverbrauch nicht beeinflussen.
Kann	Robustheit	Das System muss in allen Umgebungsbedingungen, die im Rahmen der gesetzlichen Vorschriften liegen, operieren können. Dazu zählen insbesondere verschiedene Beleuchtungsszenarien.

5. Konzeption und Design

In diesem Kapitel werden verschiedene Konzepte entwickelt und anhand der Anforderungen aus Kapitel 4 bewertet. Anschließend wird ein Konzept ausgewählt und eine Detaillösung entworfen.

5.1. Vergleich verschiedener Konzepte

Der Vergleich von verschiedenen Konzepten erfolgt in diesem Kapitel in zwei Stufen. Es wird zuerst ein optisches mit einem funkbasierten System verglichen. Anschließend wird das Konzept konkretisiert. Für jedes Kriterium erhalten die Konzepte eine Bewertung von „+“ (positiv), „o“ (neutral) oder „-“ (negativ).

Instrumentensystem versus optisches System

Als erstes werden zwei Möglichkeiten der Informationsübermittlung verglichen. In Abschnitt 3.1 werden die heute gängigen Landesysteme, wie sie an Flugplätzen eingesetzt werden, kurz vorgestellt. Landesysteme können dabei über optische Signale oder Funksignale dem Piloten anzeigen, ob er sich auf dem richtigen Gleitweg befindet.
Ein Instrumentenlandesystem kann zum Beispiel wie an realen Flugplätzen mittels einer Frequenzmodulation [Men14] erfolgen. Dafür müssen alle UAV mit einer Einheit ausgerüstet werden, die es ermöglicht, die verschiedenen Frequenzmodulationen auszuwerten. Die Bodenstation benötigt dabei Antennen, mit denen die Funksignale ausgesendet werden können. Bei der Auslegung des Systems muss darauf geachtet werden, dass die rechtlichen Gegebenheiten bezüglich des Aussendens von Funksignalen eingehalten werden. Der Vorteil dieses System liegt in der Robustheit bezüglich der Witterungsbedingungen, weil dieses System sowohl bei schlechter Sicht als auch in der Nacht verwendet werden kann. Außerdem sind die Informationen schnell in einem digitalen Format vorhanden und können direkt verwendet werden.
Bei einem optischen System wird eine Kamera verwendet, um den Gleitweg zu bestimmen. Da viele unbemannte Luftfahrzeuge über eine Kamera verfügen, ist es in vielen Fällen nicht notwendig, einen zusätzlichen Sensor am UAV zu verbauen. Dies bedeutet für ein UAV mit

Tabelle 5.1.: Bewertung von optischen und funkbasierten Systemen

Priorität	Kriterium	optisches Sys.	Instrumenten-Sys.
Muss	Gewicht	+	-
Muss	Mobilität	o	-
Muss	Zuverlässigkeit	o	o
Muss	Skalierbarkeit	+	+
Muss	Kosten	+	-
Kann	Energieverbrauch	+	-
Kann	Robustheit	-	+
	Gewählt	x	

einer Kamera, dass Kosten gespart werden können und kein zusätzliches Gewicht transportiert werden muss.
Die Mobilität des optischen Systems hängt stark von der verwendeten Bodenstation ab und wird deshalb in der Tabelle 5.1 als neutral bewertet. Die Wahl zwischen einem optischen und einem Instrumentensystem hat keinen grundlegenden Einfluss auf die Zuverlässigkeit. In beiden Fällen ist die Umsetzung entscheidend. Deshalb werden auch hier beide Systeme neutral bewertet.
Bezüglich der Robustheit hat das optische System den Nachteil, dass die Bildaufnahme stark von der Beleuchtung abhängt. Aus den in Kapitel 4 genannten Gründen ist dieses Kriterium mit der Priorität niedrig versehen und hat deshalb nur einen geringen Einfluss auf die Wahl des Systems.
Ausgewählt wird ein optisches System, weil eine große Anzahl an UAV schon über eine Kamera verfügen und dies zu Gewichts- und Kosteneinsparungen führen kann. Außerdem ist es bei vielen Standardfluggeräten nicht möglich, eine Hardware-Konfiguration vorzunehmen.

Künstliche versus natürliche Marker versus Befeuerungssysteme

Bei der Verwendung von Kamerabildern zur Positionierung können Marker verwendetet werden, die vorher in der Umgebung platziert wurden, oder es wird versucht in der Umgebung markante Punkte zu finden, die als natürliche Marker dienen. Zu beiden Themengebieten gibt es eine Vielzahl von Beispielen, in denen diese Techniken eingesetzt werden. Für die Kamera ergibt sich kein Unterschied bei der Wahl zwischen den Markern, wodurch die Kriterien des Gewichts und des Energieverbrauchs neutral bewertet werden. Als dritte Alternative bietet sich die Möglichkeit einen aktiven Marker wie die in Kapitel 3.1.2 vorgestellten Befeuerungssysteme zu verwenden. Diese ist hinsichtlich des Energieverbrauchs insofern nachteilig, als dass die Bodenstation mit Energie versorgt werden muss. Dies hat auch einen negativen Einfluss auf die Mobilität des Befeuerungssystems.

Die Mobilität ist im Falle von künstlichen Markern beschränkt, weil die Marker transportiert und platziert werden müssen. Natürliche Marker werden hingegen aus den Gegebenheiten der Umgebung gesucht. Dies führt zu keinem erhöhten Aufwand für den Transport bei der Verlegung des Standortes. Allerdings ist es für eine Lokalisierung relativ zur Landebahn notwendig, Referenzpunkte zu kennen, die eventuell vorher bestimmt werden müssen. Dies führt bei einem Verlegen der Landebahn zu einem zusätzlichen Aufwand. Deshalb wird die Mobilität negativ bewertet.
Aufgrund der Tatsache, dass eine Landebahn aus einer großen ebenen Fläche besteht und es im Modellflugbereich nicht unbedingt Markierungen auf der Landebahn gibt, kann es unter Umständen dazu kommen, dass es nicht möglich ist eine große Anzahl an markanten Stellen für die natürlichen Marker in der Umgebung zu bestimmen. Daher wird die Zuverlässigkeit und die Robustheit negativ bewertet. Künstliche Marker hingegen können sehr zuverlässig erkannt werden und sind im Vergleich zu natürlichen Markern sehr robust. Ein Befeuerungssystem hat den Vorteil, dass es aktiv leuchtet und so auch in der Dunkelheit gesehen werden kann. Durch die rechtliche Einschränkung, dass keine Flüge in der Nacht erlaubt sind, ist dies im Zusammenhang mit dieser Arbeit jedoch nicht als Vorteil zu betrachten. Durch das aktive Leuchten des Befeuerungssystems ist es notwendig, die Stärke der Beleuchtung zu regulieren, da ansonsten die Kamera des UAV sättigen kann oder das Licht viel zu schwach ist, um auf dem Bild detektiert zu werden. Passive Marker leuchten hingegen immer so hell wie ihre Umgebung. Solange keine stark reflektierenden Objekte in der Umgebung sind, ist es deshalb relativ einfach möglich, die Helligkeit vom gesamten Bild über die Belichtungszeit oder die Verstärkung der Kamera zu regeln. In Abbildung 5.1 ist ein Befeuerungssystem abgebildet. Darin ist zu erkennen, dass bei voller Sonneneinstrahlung die Kamera mit aktiver Helligkeitsregelung im Bereich der Lichtquelle immer noch sättigt. Eine Regelung nur auf den Bereich der Lichtquelle wäre theoretisch möglich, allerdings muss dafür die Position im Bild bekannt sein und diese müsste vorher auf dem gesamten Bild gesucht werden.
Künstliche Marker und Befeuerungssysteme müssen hergestellt werden und verursachen deshalb zusätzliche Kosten, die bei natürlichen Markern nicht entstehen. Alle Systeme sind sowohl in Innenräumen als auch im Freien gut einzusetzen und können für alle UAV eingesetzt werden, die ein ausreichendes Kamerasystem besitzen. Deshalb ist die Skalierbarkeit positiv bewertet. Allerdings kann das PAPI-System nur angeben, ob das UAV auf dem aktuell eingestellten Gleitpfad ist oder nicht. Es ist nicht möglich, ohne ein Verstellen der PAPI Lichter verschiedene Gleitwinkel anzufliegen.
Die Tabelle 5.2 fasst die Resultate des Vergleichs zusammen. Gewählt wird ein System mit künstlichen Markern, da sie zuverlässig erkannt werden können und die Landebahn dadurch markiert werden kann.

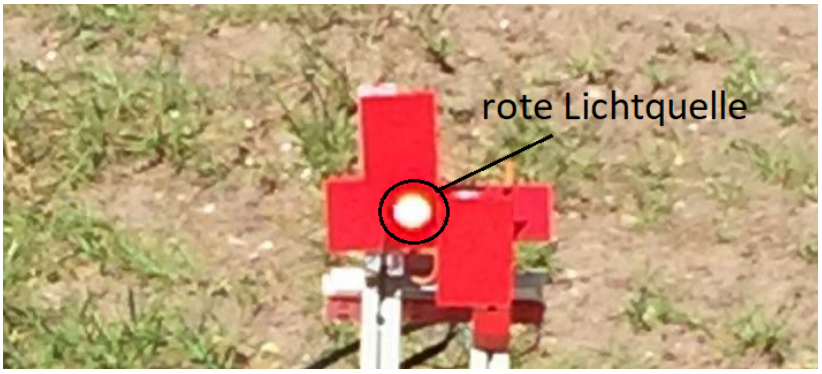


Abbildung 5.1.: Fotografie einer roten Lichtquelle. Darin ist zu erkennen, dass trotz aktiver Helligkeitsregelung der Bereich der Lichtquelle sättigt und eine Unterscheidung zwischen weiß und rot nicht mehr möglich ist.

Tabelle 5.2.: Bewertung von verschiedenen optischen Systemen

Priorität	Kriterium	künstliche M.	natürliche M.	Befeuerungssystem
Muss	Gewicht	o	o	o
Muss	Mobilität	-	-	-
Muss	Zuverlässigkeit	+	-	o
Muss	Skalierbarkeit	+	+	o
Muss	Kosten	-	+	-
Kann	Energieverbrauch	o	o	-
Kann	Robustheit	+	-	-
	gewählt	x		

5.2. Design der Detaillösung

In diesem Abschnitt wird dargelegt, wie ein Landeanflug mit einem Marker im Detail ablaufen kann und wie der Marker, der verwendet wird, aussehen soll. Außerdem wird die Architektur der Software beschrieben.

5.2.1. Aufbau der Landezone

Für die Qualität der Detektion der Marker ist es entscheidend, wie die Positionen der Marker zur Kamera sind. Deshalb wird in diesem Abschnitt kurz erläutert, wie die Marker für dieses System platziert werden sollen.
Damit das System vielseitig einsetzbar ist, können beliebig viele Marker in der Landezone platziert werden. Die Positionen der Marker müssen vor der Landung im System hinterlegt werden. In Abbildung 5.2 ist ein Beispiel für eine Platzierung der Marker dargestellt. Aufgrund der Tatsache, dass sowohl eine große Entfernung als auch ein großer Winkel zwischen Marker- und Kamerakoordinatensystem zu einer schlechteren Positionsschätzung führen, werden die Marker an unterschiedlichen Positionen und mit unterschiedlichen Winkeln entlang der Landebahn platziert.
Der erste Marker ist dabei so ausgerichtet, dass er im besten Fall orthogonal zur Lande-Trajektorie steht, da er damit den geringsten Winkel zu einer FPV-Kamera des UAV hat. Dieser Marker ist hauptsächlich für die Positionsschätzung während des Gleitfluges wichtig, weil er den geringsten Abstand zur Trajektorie des Gleitflugs hat. Da im Bereich des Aufsetzens auf der Landebahn der erste Marker eventuell nicht mehr im Field of View (FOV) der Kamera ist, wird ein zweiter Marker weiter hinten platziert. Dieser Marker ist damit hauptsächlich für den Endanflug und das Aufsetzen notwendig. Es ist vorteilhaft diesen zweiten Marker in der Mitte zu platzieren, weil die Schätzung der Position genauer erfolgen kann, als wenn lediglich am Ende der Landebahn ein Marker platziert wird. Das Ende der Landebahn wird von einem dritten Marker markiert. Dieser steht senkrecht zur Landebahn und ist hauptsächlich für die Positionsschätzung auf der Landebahn wichtig.
Während des Landeanfluges können aber alle Marker, die im Bild detektiert worden sind, dazu verwendet werden, um die Positionsschätzung zu verbessern. Deshalb kann die Schätzung der Position auch noch weiter verbessert werden, indem noch zusätzliche Marker aufgestellt werden.

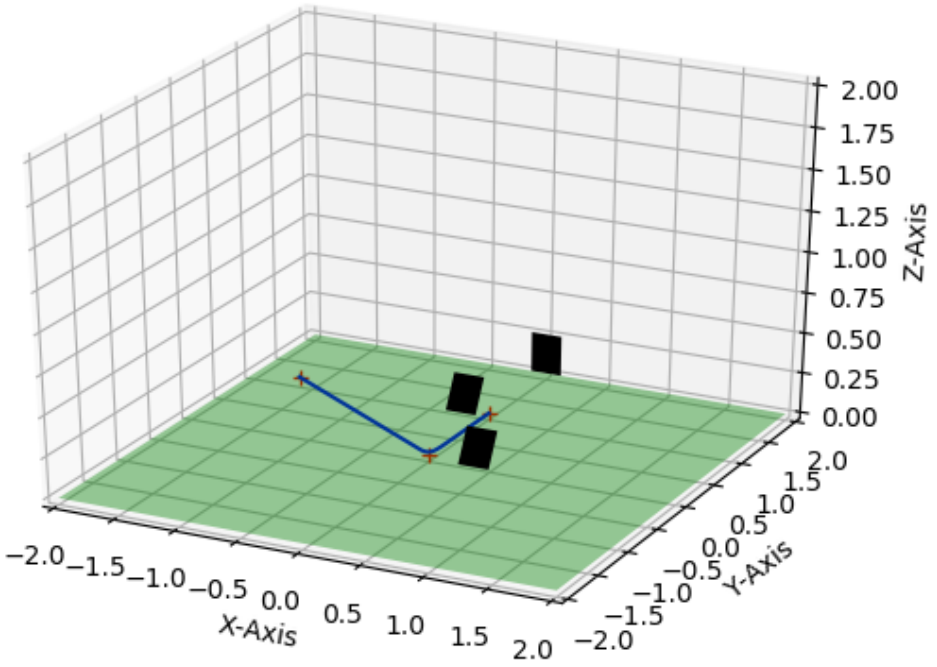


Abbildung 5.2.: Position der Marker in der Landezone. Der erste Marker ist mit einem Winkel von 55 Grad aufgerichtet auf Höhe des Aufsetzpunktes platziert. Der zweite Marker ist ebenfalls mit 55 Grad aufgerichtet und steht in der Mitte der Landebahn. Der dritte Marker steht senkrecht am Ende der Landebahn. Die blaue Linie stellt eine mögliche Lande-Trajektorie dar.

5.2.2. Markerauswahl

In diesem Abschnitt wird erläutert, welcher Marker für das Landesystem verwendet wird und aus welchen Gründen die Wahl auf diesen Marker fällt. Bereits in Abschnitt 3.3.3 wurden eine Vielzahl von Markern vorgestellt. Für das Landesystem ist von Bedeutung, dass sich die Position des Markers in Weltkoordinaten aus dem Bild herleiten lässt. Dazu müssen, wie in Abschnitt 3.3.2 erläutert, zu mindestens vier Punkten im Bild die Weltkoordinaten bekannt sein. Rechteckige Marker eignen sich besser als runde Marker, da durch die Detektion der Kanten bereits vier Punkte im Bild detektiert sind.
Für eine schnelle Detektion ist es von Vorteil, wenn die Klassifikation der Marker nicht durch Template-Matching erfolgt, sondern wie beispielsweise bei den ArUco-Markern [RRMSMC18] die Codierung der Marker durch einen Schwellwert ausgewertet werden kann. Die detaillierte Beschreibung des verwendeten Verfahrens ist im Abschnitt 3.3.3 auf Seite 29

(a) Marker 0 (b) Marker 1 (c) Marker 2 (d) Marker 3 (e) Marker 4

Abbildung 5.3.: Beispiele für ArUco-Marker

beschrieben.
Ein weiterer wichtiger Punkt ist die Unterstützung durch vorhandene Software-Bibliotheken. Dadurch ist es schneller und einfacher möglich, verschiedene Funktionalitäten zu implementieren. In der weit verbreiteten Bildverarbeitungsbibliothek OpenCV sind viele Funktionalitäten zu den ArUco-Markern implementiert. Daher werden für diese Arbeit die ArUco-Marker verwendet. Gewählt wird das in OpenCV standardmäßig implementierte Dictionary „4X4_50“. Es beinhaltet 50 unterschiedliche Marker mit einer 4x4 Codierung. In dem Dictionary sind die Marker so angelegt, dass bei der Verwendung von den ersten Markern, die größtmögliche Hamming-Distanz zwischen den Codewörtern der Marker vorhanden ist. Dadurch ist die maximal mögliche Sicherheit gegen eine Verwechslung der Marker gegeben. Je mehr Marker verwendet werden, desto geringer ist die Hamming-Distanz. In Abbildung 5.3 sind die Marker 0 bis 4 dargestellt.

5.2.3. Ablaufschema

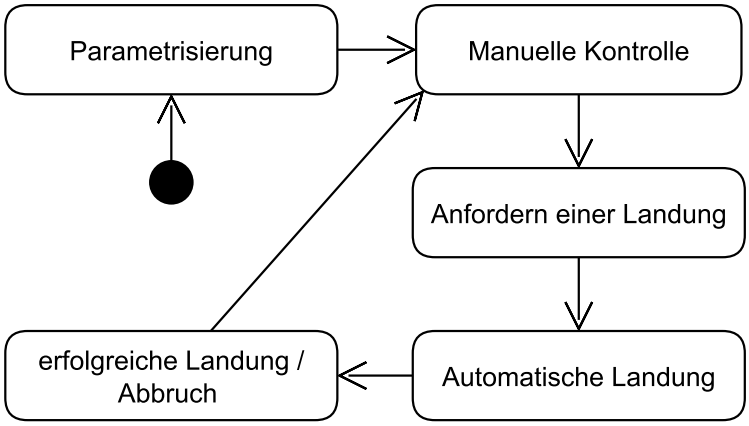
Der grundlegende Ablauf, der notwendig ist, um eine erfolgreiche Landung zu ermöglichen, ist in Abbildung 5.4(a) dargestellt. Es beginnt mit der Parametrierung des Systems. Dazu gehört das Eingeben der Marker-Positionen und der Landeroute. Anschließend kann der manuelle Flug durch den Anwender oder eine andere Steuerung erfolgen. Wird ein automatisierter Landeanflug angefordert, erfolgt der Start der automatischen Landung. Nach einer erfolgreichen Landung, oder durch einen Abbruch der Landung, erfolgt wieder die manuelle Kontrolle. Das Vorgehen während der automatischen Landung ist detailliert in Abbildung 5.4(b) dargestellt.
Die zwei wichtigsten asynchronen Prozesse in diesem System sind die Positionsschätzung und die Regelung. Die Positionsschätzung sucht auf jedem eintreffenden Kamerabild die zuvor parametrierten Marker und schätzt die Position von allen ermittelten Markern. Ist die Positionsschätzung erfolgt, wird auf das nächste Bild gewartet und die Auswertung beginnt von vorne.
Die Positionsdaten werden anschließend verwendet, um die Trajektorie zu planen. Nach dem Planen der Trajektorie wird entschieden, ob es möglich ist, von der aktuellen Position aus den eingestellten Gleitwinkel zu verfolgen. So kann es aufgrund der aktuellen Höhe und Entfernung zur Landebahn eventuell nicht möglich sein, den gewünschten Winkel abzufliegen. Je nachdem wird dann ein Anflug gestartet oder der Landeanflug abgebrochen. Wenn der Anflug gestartet wird, dann wird die Trajektorie abgeflogen, indem die Positionsdaten aus den Kamerabildern für die Zustandsregelung verwendet werden. Dabei ist es nicht unbedingt notwendig, dass für jeden Iterationsschritt der Regelung eine Positionsschätzung aus den Kamerabildern vorhanden ist. Ist die Trajektorie beendet, die Positionsschätzung zu ungenau oder ein anderer Fehler aufgetreten, dann wird die automatische Landung beendet. Die detaillierte Umsetzung erfolgt in Kapitel 6.

Das Verhalten des Systems ist als Zustandsdiagramm in Abbildung 5.5 dargestellt. Die Transitionen dazu sind in der Tabelle 5.3 erläutert. Die einzelnen Zustände des Systems geben an, wie das UAV gesteuert wird. Während des manuellen Fluges werden lediglich die Steuersignale von einem Anwender oder von einer alternativen Steuerung an das UAV weitergereicht. Durch die Transition mit der Bedingung b wird der Zustand „Initialisierung der Autokontrolle“ erreicht. In diesem Zustand wird versucht, die automatische Landung zu starten. Solange dies nicht möglich ist, wird in diesem Zustand verblieben und die Steuerung erfolgt wie bei einem manuellen Flug. Ist der automatische Landeanflug gestartet, erfolgt die Übergabe der Kontrolle über das UAV nur nach einer erfolgreichen Landung oder nach einer Anforderung zum Abbruch der Landung. Dabei wird im Zustand „Abbruch des Anfluges“ sichergestellt, dass die Übergabe in einer sicheren Fluglage erfolgt. Erfolgt der Abbruch des Anfluges durch das System selbst und der Anwender gibt aufgrund eines Verbindungsabbruchs oder Ähnlichem nicht das Signal, dass er bereit ist die Steuerung zu

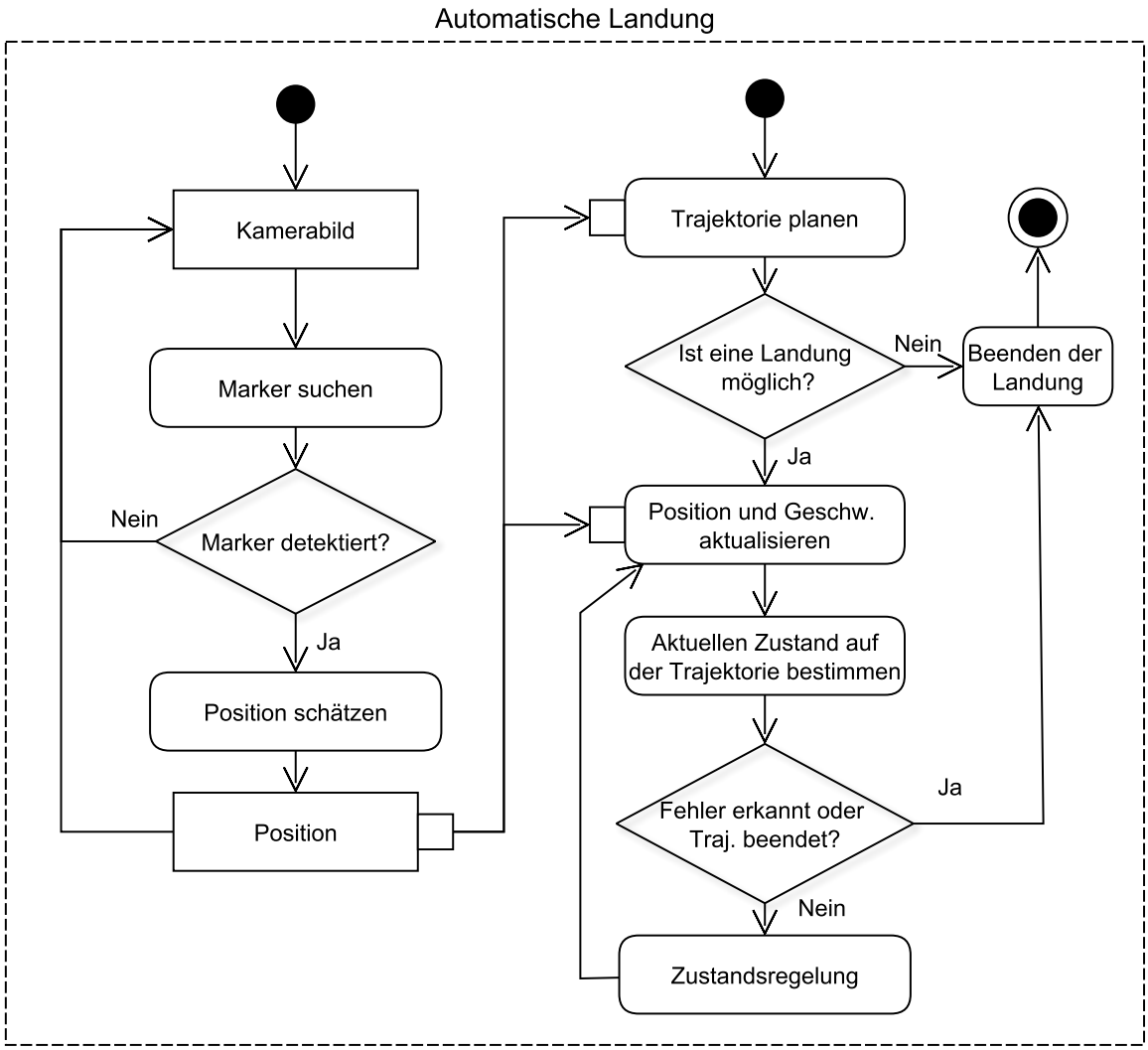
übernehmen, dann wird im Zustand „Fehler“ ein Notfallprotokoll ausgeführt. Dieses Protokoll ist für jedes UAV individuell zu bestimmen. Bei der Tello wird in so einem Fall in den Schwebeflug übergegangen.

5.2.4. Software-Architektur

Bei der Definition der Klassen für dieses System ist das Ziel, die asynchronen Prozesse innerhalb des Systems zu kapseln. Das zentrale Element des Systems bildet die Klasse FlightControl. Sie beinhaltet die in Abschnitt 5.2.3 beschriebene Zustandsmaschine und bedient die Funktionalitäten des Systems.
Die abstrakte Klasse *UavInterface* definiert alle Funktionalitäten, die notwendig sind, um ein UAV zu steuern. Die Implementierung erfolgt je nach UAV. In Abbildung 5.6 ist das Klassendiagramm für die Tello EDU dargestellt. *FlightControl* erbt von der Klasse *TelloControl* um die Funktionalitäten mit der automatischen Steuerung, wie dem Regelalgorithmus, zu erweitern. Die Klasse *FrameGrabber* implementiert alle Funktionalitäten, die notwendig sind, um die Bilder vom UAV zu erhalten und stellt sie in einer Deque für die anderen Prozesse bereit. Da die Bilder asynchron vom UAV geliefert werden, wird der *FrameGrabber* in einem eigenen Thread gestartet. Die Schätzung der Positionsdaten anhand der Bilddaten erfolgt in der Klasse *MarkerPositioning*. Das Tracking der Marker ist besonders rechenaufwendig und benötigt je nach Bild unterschiedliche Laufzeiten. Deshalb wird auch dieser Prozess asynchron in einem Thread ausgeführt und die Übermittlung der ermittelten Positionen erfolgt über eine Deque.
Für die Regelung ist es notwendig, ein festes Intervall einzuhalten. Da der *FrameGrabber* und das *MarkerPositioning* unregelmäßig und asynchron zum Regelalgorithmus Daten liefern, ist es notwendig, einen Beobachter zu implementieren, mit dem die Daten für den Regelalgorithmus verwendbar gemacht werden. Alle notwendigen Funktionalitäten für den Beobachter sind in der Klasse *Observer* implementiert. Die Berechnung der Flugbahn erfolgt in der Klasse *TrajectoryPlaner*. Die Flugplanung und der Beobachter laufen synchron mit dem Regelalgorithmus und laufen deshalb nicht in einem eigenen Thread, sondern werden in der Klasse *FlightControl* bedient.
Für die manuelle Steuerung ist eine *GUI* vorgesehen, die die Bilder für ein First Person View (FPV) über eine Deque vom FrameGrabber erhält. Um die Position und die Orientierung im Raum zu beschreiben, werden die beiden Klassen *Point3D* und *Orientation3D* definiert.



(a) Aktivitätsdiagramm des Landesystems



(b) Detaillierung der automatischen Landung

Abbildung 5.4.: Aktivitätsdiagramme

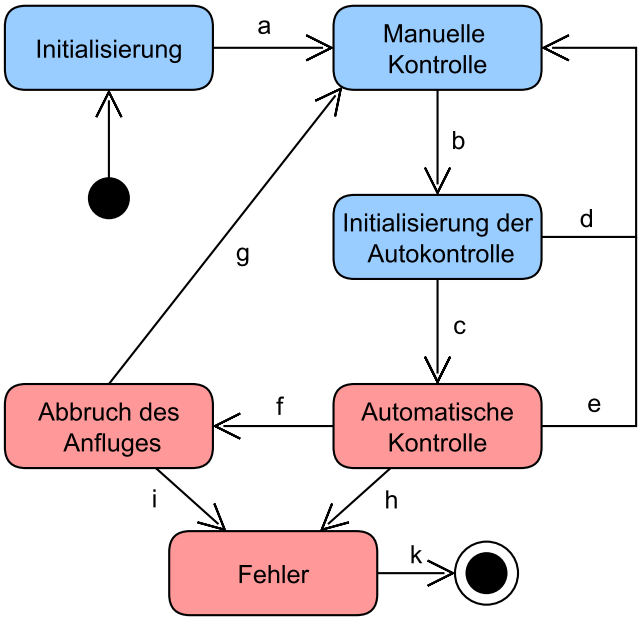


Abbildung 5.5.: Zustandsdiagramm des Landesystems. Die mit rot hinterlegten Zustände sind Zustände, in denen das Landesystem die Kontrolle über das UAV hat. In den blauen Zuständen ist die Kontrolle beim Anwender.

Tabelle 5.3.: Transitionen im Zustandsdiagramm

Variable	Art	Beschreibung
a	Bedingung	Wenn die Verbindung zum UAV hergestellt wurde, dann wird zum manuellen Flug übergegangen.
b	Event	Der Anwender stellt die Anfrage einer Landung.
c	Bedingung	Die Landung ist vom aktuellen Zustand aus möglich.
d	Event	Der Anwender bricht die Landung ab. / Nach einer bestimmten Zeit der nicht erfolgreichen Initialisierung wird die Landung abgebrochen.
e	Bedingung	Die Landung ist erfolgreich beendet.
f	Event/Bedingung	Der Anwender fordert den Abbruch der Landung an. / Das System muss den Landeanflug abbrechen.
g	Bedingung	Der Anwender/ Die alternative Steuerung ist bereit die Kontrolle über das UAV zu übernehmen und das UAV befindet sich in einer sicheren Fluglage.
h	Bedingung	Das UAV ist nicht mehr zu kontrollieren und es muss das Notfallprotokoll ausgeführt werden.
i	Event	Ist der Anwender nach einer bestimmten Zeit nicht bereit die Steuerung zu übernehmen, erfolgt das Umschalten in den Fehlerbetrieb.
k	Bedingung	Das Notfallprotokoll wurde durchgeführt.

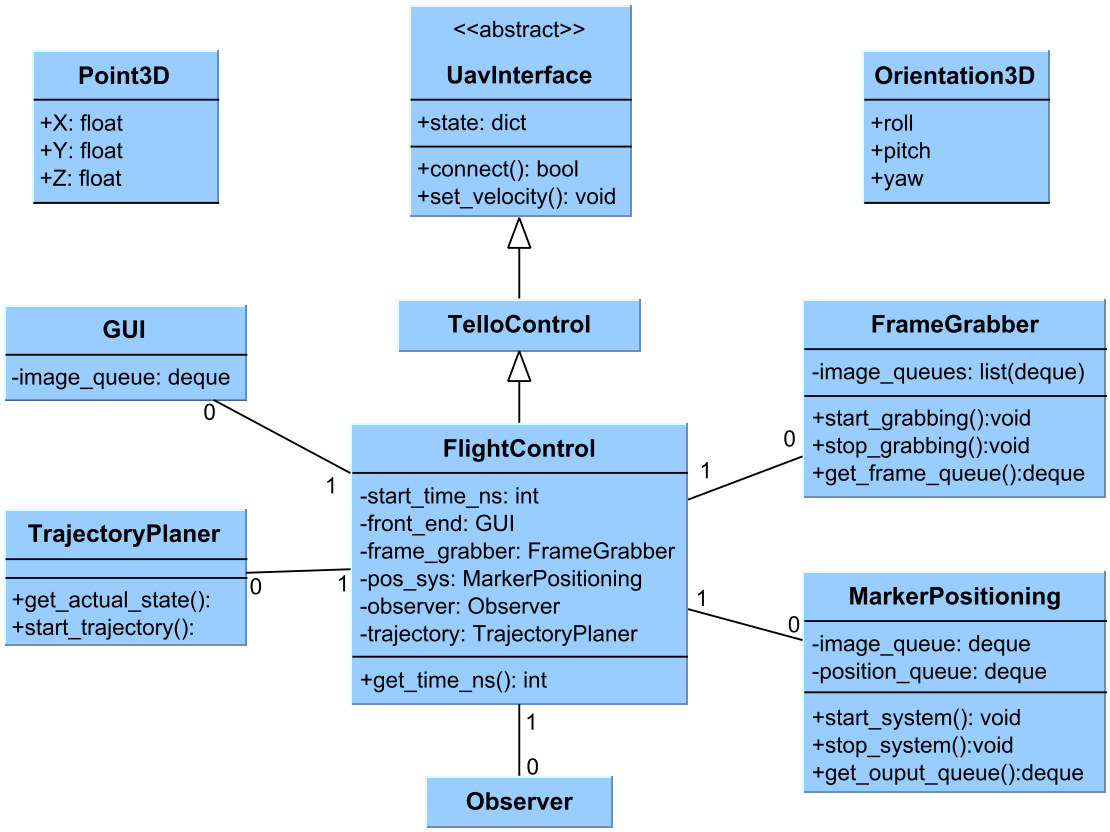


Abbildung 5.6.: Klassendiagramm des Landesystems. Die Klassen *Point3D* und *Orientation3D* werden in fast allen anderen Klassen verwendet.

6. Umsetzung

In diesem Abschnitt wird die Umsetzung der Konzeption aus Kapitel 5 mit der Tello EDU als UAV beschrieben.

6.1. Kamerakalibrierung

Wie in Abschnitt 2.3.3 erläutert, muss für eine Positionsschätzung anhand von Bildpunkten die Matrix der intrinsischen Kameraparameter bekannt sein. Die Bestimmung der Parameter für die Tello EDU wird mit Hilfe des ChArUco-Boards aus Abbildung 6.1 durchgeführt. Ein ChArUco-Board ist ein Brett mit dem gleichen schwarz-weißen Muster wie ein Schachbrett, wobei die weißen Flächen mit ArUco-Markern belegt sind. Für die Kamerakalibrierung hat das ChArUco-Board die gleiche Funktion wie ein Schachbrett. Die Kalibrierung erfolgt auf den Ecken der weißen und schwarzen Rechtecke. Der Vorteil des ChArUco-Boards liegt darin, dass durch die Codierung eine eindeutige Zuordnung der detektierten Ecken im Bild zu den Ecken des ChArUco-Boards erfolgen kann. Dies ist bei der automatisierten Bestimmung der Ecken in den Bildern von Vorteil, weil durch die ArUco-Marker sichergestellt werden kann, dass die detektierten Ecken zum ChArUco-Board und nicht zur Umgebung gehören. Außerdem ist es möglich, bei einem teilweise verdeckten Board die Punkte zu extrapolieren und so mehr Punktpaare zu erhalten.
Um das Verfahren der Kamerakalibrierung zu validieren, wird die Kalibrierung mit einer Referenz-Kamera durchgeführt. Von der Referenz-Kamera sind alle Daten des Sensors und des Objektives bekannt, damit die intrinsischen Parameter rechnerisch bestimmt und anschließend mit den gemessenen Werten verglichen werden können. Verwendet wird die Flächenkamera acA2500-14gc von der Basler AG. Die Daten der Kamera sind in der Tabelle 6.1 aufgelistet.
Für die acA2500-14gc wird die Berechnung der intrinsischen Parameter im Folgenden dargelegt. Der Parameter b ergibt sich aus der Brennweite des Objektives zu -35 mm. Die Pixel in diesem Kameramodell sind quadratisch. Damit ergibt sich

$$s_x = s_y = 2,2 \frac{\mu\text{m}}{\text{px}}. \tag{6.1}$$

Für die Parameter c_x und c_y wird angenommen, dass die Mitte des Sensors auf der optischen Achse liegt. Daraus resultiert für c_x = 1295 px und für c_y = 971 px. Für die berechnete intrinsische Projektionsmatrix $P_{i,calc}$ erhält man

$$\boldsymbol{P}_{i,calc} = \begin{pmatrix} \frac{-b}{s_x} & 0 & c_x & 0 \\ 0 & \frac{-b}{s_x} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \frac{35\ \text{mm}\cdot\text{px}}{2{,}2\ \mu\text{m}} & 0 & 1295\ \text{px} & 0 \\ 0 & \frac{35\ \text{mm}\cdot\text{px}}{2{,}2\ \mu\text{m}} & 871\ \text{px} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 15909\ \text{px} & 0 & 1295\ \text{px} & 0 \\ 0 & 15909\ \text{px} & 971\ \text{px} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{6.2}$$

Die berechnete Projektionsmatrix stellt den Erwartungswert für die Bestimmung der Kameraparameter mit dem Verfahren aus Abschnitt 3.3.1 auf Seite 28 dar. Die Kalibrierung wird mit 51 Bildern durchgeführt. Die Bilder zeigen das ChArUco-Board in drei unterschiedlichen Entfernungen. In jeder der drei Entfernungen wird das ChArUco-Board in unterschiedlichen Posen fotografiert.
In jedem Bild werden dann mit der in OpenCV implementierten Funktion *aruco.detectMarkers()* die Marker gesucht und die Ecken der Marker ausgegeben. Mit Hilfe der detektierten Ecken und dem bekannten Board werden anschließend die Bildpunkte $\mathbf{i}_k$ der Ecken des Schachbrettmuster bestimmt. Diese Punkte werden verwendet, um mit Hilfe der Weltkoordinaten $\mathbf{w}_k$ die Fehlerfunktion

$$err = \sqrt{\sum_{k=0} (\mathbf{i}_k - \tilde{\boldsymbol{P}}_i\ \tilde{\boldsymbol{P}}_{e,k}\ \boldsymbol{w}_k)^2} \tag{6.3}$$

zu minimieren. Die Werte für die Parameter c_x und c_y sind vor der Minimierung als Startwert vorgegeben worden. Der Startwert ist die Hälfte der Pixel auf der jeweiligen Sensorseite.
Das Ergebnis der Minimierung ist die gemessene Projektionsmatrix

$$\boldsymbol{P}_{i,meas} = \begin{pmatrix} 15564\ \text{px} & 0 & 1294\ \text{px} & 0 \\ 0 & 15517\ \text{px} & 971\ \text{px} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{6.4}$$

Mit den Projektionsmatrizen $\mathbf{P}_{i,calc}$ und $\mathbf{P}_{i,meas}$ wird anschließend eine Positionsbestimmung, wie sie auch in der späteren Implementierung verwendet wird, durchgeführt. Die Differenz der beiden Schätzungen ist bei einer Entfernung von 5 Metern kleiner als ein halber Zentimeter. Die Abweichung beider Schätzungen vom Ground-Truth beträgt dabei mehrere Zentimeter. Aus diesem Grund wird die Schätzung der intrinsischen Parameter als ausreichend betrachtet und dasselbe Verfahren mit der Kamera der Tello EDU vorgenommen.

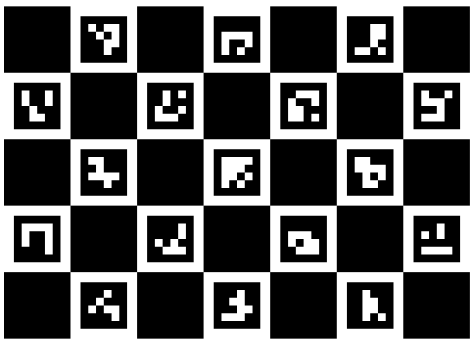


Abbildung 6.1.: ChArUco-Board für die Kamera-Kalibrierung

Tabelle 6.1.: Daten der Basler acA2500-14gc mit Objektiv

Brennweite f	35 mm
max. Auflösung (H x V)	2590 px x 1942 px
Pixel Größe	2,2 μm x 2,2 μm
Mono/Color	Color

Das Resultat ist die Matrix

$$P_{i,Tello} = \begin{pmatrix} 912\text{ px} & 0 & 480\text{ px} & 0 \\ 0 & 912\text{ px} & 360\text{ px} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{6.5}$$

Mit diesem Ergebnis ist die Kamerakalibrierung abgeschlossen. Die Parameter für die Verzerrungen durch das Objektiv werden vernachlässigt, weil die Ergebnisse der Positionsschätzung mit und ohne Berücksichtigung der ermittelten Verzerrungen keinen signifikanten Unterschied gezeigt haben.

6.2. Marker

Für die Verwendung der ArUco-Marker als Kennzeichnungen, werden die Marker auf Hartschaumplatten gedruckt. Für den Gebrauch in Innenräumen werden die Marker auf Platten der Größe 25 cm x 25 cm gedruckt. Der Marker an sich hat damit eine Kantenlänge von ca. 18,8 cm, weil um den Marker herum ein weißer Rand gedruckt wird, damit der Marker besser detektiert werden kann. Als Mindestgröße für die Marker werden die Kantenlängen aus [WO16] angenommen, weil in dieser Veröffentlichung mit einer Kantenlänge von 16,7 cm gute Positionsschätzungen in 15 m Entfernung erreicht worden sind. Für die Umsetzung im Freien werden die Marker doppelt so groß bedruckt.

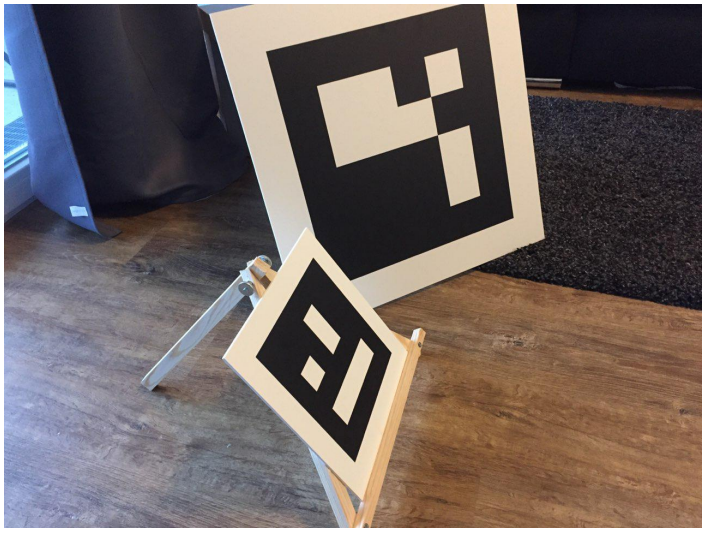


Abbildung 6.2.: Fotografie der Staffelei mit Marker

Für das Aufstellen wird eine Staffelei ähnliche Konstruktion entworfen, mit der es möglich ist, den Winkel der Markers stufenlos einzustellen. In Abbildung 6.2 sind die Marker mit der entworfenen Staffelei abgebildet.

6.3. Modellbildung und Reglerauslegung

Für die Regelung der Flugbahn ist es notwendig ein Modell vom UAV aufzustellen und dafür einen Regler auszulegen. In diesem Abschnitt wird die Modellbildung für die Tello EDU beschrieben und der Regler ausgelegt.
Das Modell wird bei der Tello EDU für die Flugrichtungen in X, Y und Z und den Yaw-Winkel bestimmt. Für die anderen Achsen wird kein Modell bestimmt, weil diese nicht zu steuern sind, sondern vom Lageregler der Tello EDU geregelt werden. Weil die Tello EDU bereits einen Lageregler beinhaltet, wird angenommen, dass die einzelnen Achsen der Drohne entkoppelt sind. Somit wird jede Achse einzeln betrachtet und das in Abbildung 6.3 dargestellte Modell für jede Achse parametriert. Als Steuersignal wird hier ein Wert von -1 bis 1 angenommen. Dieser Wert wird in der Implementierung der UAV-Ansteuerung auf die maximalen und minimalen Werte Tello EDU abgebildet.
Die Übertragungstotzeit der Steuersignale wird durch das Messen der Antwortzeit auf Steuersignale bestimmt. Diese ist für alle Achsen identisch und liegt im Mittel bei $T_t = 0.5$ s. Der Verstärkungswert K_P wird für jede Achse durch das Messen der Geschwindigkeit bei voller Ansteuerung bestimmt. Die Zeitkonstante T wird anhand einer Sprungantwort ermittelt. Die Resultate sind in der Tabelle 6.2 aufgelistet. Als Zustandsraummodell ergibt sich mit den

Tabelle 6.2.: Parameter Tello EDU-Modell

Achse	K_P	T	K_1	K_2
X-Achse	$1\ \frac{\text{m}}{\text{s}}$	1 s	1.8	1.7
Y-Achse	$1\ \frac{\text{m}}{\text{s}}$	1 s	1.8	1.7
Z-Achse	$1\ \frac{\text{m}}{\text{s}}$	1 s	1.8	1.7
Rotation	$1.28\frac{\text{rad}}{\text{s}}$	1 s	1.4	1.3

Werten aus Tabelle 6.2 für jede Achse die Matrizen

$$A = \begin{pmatrix} 0 & 1 \\ 0 & \frac{-1}{T} \end{pmatrix}$$
$$B = \begin{pmatrix} 0 \\ \frac{K_P}{T} \end{pmatrix}$$
$$C = \begin{pmatrix} 1 & 0 \end{pmatrix}$$
$$D = \begin{pmatrix} 0 \end{pmatrix}.$$

Für dieses Modell wird ein Zustandsregler mittels Polplatzierung ausgelegt. Die Polvorgabe erfolgt anhand der Übertragungstotzeit. Die Dynamik des Gesamtsystems muss für eine stabile Regelung langsamer als die größte Totzeit im System sein. Deshalb werden die Pole auf einen Wert von

$$p_{1,2} = -\frac{1}{1.5 \cdot T_t} \tag{6.6}$$

gelegt. Das Resultat sind die beiden Reglerparameter K_1 und K_2 aus der Tabelle 6.2.
Weil die Zustände nicht direkt gemessen werden können, wird ein Kalman-Filter für die Zustandsschätzung verwendet. Die Implementierung des Kalman-Filters ist in Kapitel 6.4.5 dargelegt.
Damit die Kameradaten als Messdaten verwendet werden können, wird die Totzeit der Bildübertragung bestimmt. Die Messung ergab dabei eine mittlere Totzeit von $T_{t,Video} = 0.5$ s von der Aufnahme des Bildes bis zum Empfang mit dem *FrameGrabber*.
In Abbildung 6.4 ist das Blockschaltbild der Regelung, mit allen Elementen die für die Umsetzung implementiert werden, dargestellt.

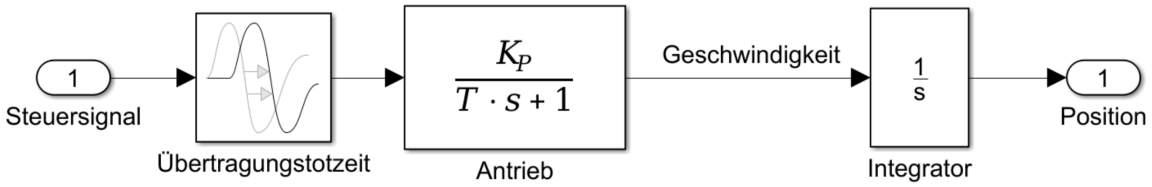


Abbildung 6.3.: Modell einer Flugachse der Tello EDU

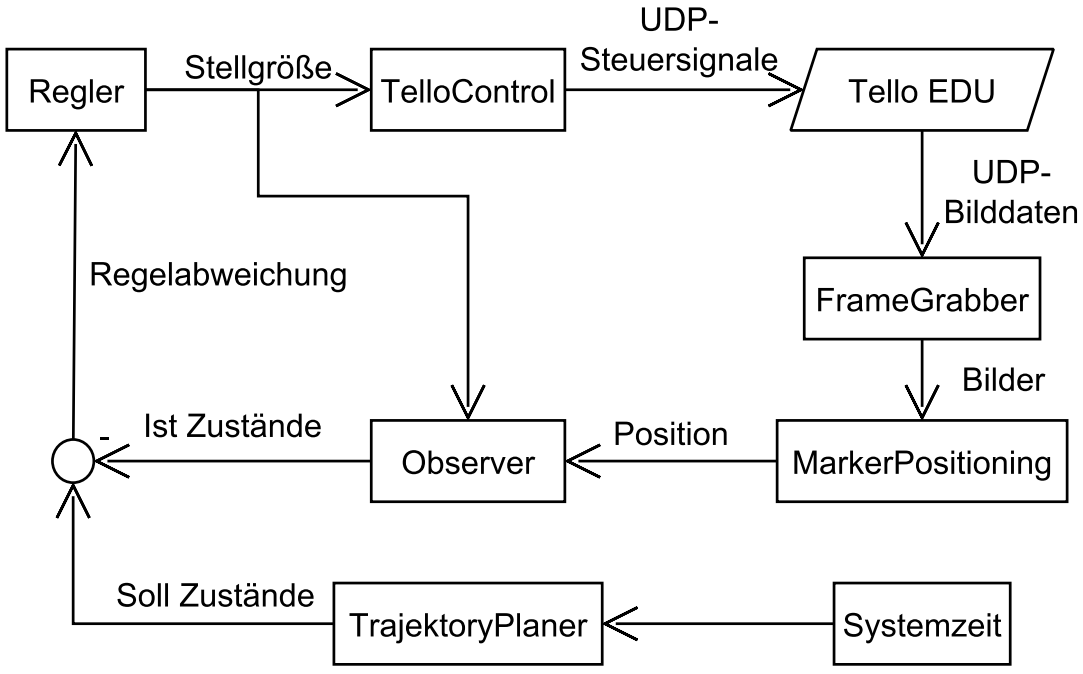


Abbildung 6.4.: Blockschaltbild des gesamten Landesystems mit Tello EDU

6.4. Implementierung

In diesem Abschnitt wird die Implementierung der einzelnen Klassen aus dem in Kapitel 5.2.4 dargestellten Klassendiagramm erläutert. Die Implementierungen befinden sich im Anhang. Für die Implementierung wird die Programmiersprache Python verwendet.

Systemzeit

Damit im gesamten System die selbe Zeitrechnung verwendet wird, wird die Klasse *TimedSystem* implementiert. Diese Klasse hat lediglich die Funktion „get_time“ als Member. Die Funktion verweist in dieser Implementierung auf die Funktion *perf_counter_ns()* aus der *time* Bibliothek. Der Wert, der zurückgegeben wird, ist der aktuelle Wert von dem genauest möglichen Zähler im System. Dabei ist zu beachten, dass der Wert nur relativ zu einem vorherigen Aufruf der Funktion valide ist. Der absolute Wert des Zählers ist nicht zur Verwendung geeignet, weil je nach System andere Zähler verwendet werden und diese nicht zu einem festen Punkt im Programmablauf gestartet werden. Alle Klassen, die eine Systemzeit benötigen, erben von der Klasse *TimedSystem* und verwenden damit denselben Zähler zur Zeitmessung.

Kommunikation

Die nachfolgend vorgestellten Klassen beinhalten teilweise Funktionalitäten, die nebenläufig ausgeführt werden müssen. Damit die einzelnen Abläufe nicht für den Datenaustausch synchronisiert werden müssen, erfolgt die Übergabe von Daten zwischen den Klassen immer über Queues. Der Datentyp *Deque* aus der *collections*-Bibliothek ist bereits so implementiert, dass er asynchron von unterschiedlichen Abläufen bedient werden kann. Die *Deque* ist ein First-in-First-out (FIFO) Speicher mit einer festen Größe.
Die Größe kann bei der Initialisierung der *Deque* festgelegt werden. Wird versucht ein Objekt einer vollen *Deque* hinzuzufügen, wird das älteste Objekt entfernt. Dies hat den Vorteil, dass der Verbraucher nur mit den aktuellsten Daten arbeitet, wenn der Erzeuger zu schnell Daten liefert.
In dieser Anwendung gibt es *Deques* für das Übergeben der Bilddaten an die *GUI* und an das *MarkerPositioning*. Eine *Deque* ist für die Übergabe der Positionsdaten an die *FlightControl* notwendig und eine weitere übergibt die User-Eingaben an die *FlightControl*.

6.4.1. FlightControl

Die Klasse *FlightControl* ist das zentrale Element des Landesystems. Darin ist die State-Maschine aus Abbildung 5.5 implementiert. Außerdem beinhaltet die *FlightControl* Instanzen der im Folgenden beschriebenen Klassen und verwendet deren Funktionalitäten. Für die Kommunikation mit der Tello EDU erbt die Klasse *FlightControl* alle Funktionalitäten von der Klasse *TelloControl*.
Bei der Initialisierung werden die Objekte der anderen Klassen erzeugt und die Übergabe der *Deques* erfolgt. Am Ende der Initialisierung *FlightControl* wird ein Thread gestartet, der die Funktion *__run_state_machine()* aus dem Listing 6.1 ausführt. Darin wird der Status der State-Machine aktualisiert und die Übergabe vom Anwender verarbeitet. Im manuellen Flug (Zeile 12) und während der Initialisierung der Auto-Kontrolle (Zeile 15) wird das Steuersignal vom Anwender an das UAV durchgereicht. Während der Initialisierung der Auto-Kontrolle wird der Beobachter instanziiert, der Regler durch die Pol-Platzierung ausgelegt und das *MarkerPositioning* gestartet. Der Übergang zur aktiven Auto-Kontrolle erfolgt sobald der Beobachter eine Positionsbestimmung liefert, bei der die Varianz kleiner als ein bestimmter Grenzwert ist. Die Varianz der Positionsbestimmung kann in diesem Fall als Quadrat der mittleren Abweichung der realen Position vom Schätzwert betrachtet werden. Der Grenzwert ist dabei für jeden Anwendungsfall spezifisch zu wählen, weil es für einen Landeanflug im Freien aus 30 m Entfernung unter Umständen zu Beginn nicht möglich ist, die gleiche Genauigkeit zu erzielen, wie zum Beispiel bei einem Landeanflug aus 5 m in einem Innenraum. Für die Verwendung mit einer Tello EDU eignet sich für die Varianz ein Wert von $0,25\ m^2$. Ist die Positionsbestimmung erfolgreich, wird die Trajektorie im *TrajectoryPlaner* berechnet. Wenn die Berechnung erfolgreich ist und die Trajektorie von der aktuellen Position gestartet werden kann, erfolgt die Auto-Kontrolle.
Während der Auto-Kontrolle wird bei einer vorhandenen Positionsschätzung der Korrekturschritt des Kalman-Filters durchgeführt. Aufgrund der Tatsache, dass die Bilder von der Kamera verzögert eintreffen und die Bildverarbeitung eine gewisse Zeit beansprucht, gibt es zwei Instanzen des Beobachters. Die Hauptinstanz wird dazu verwendet, um mit den aktuellen Steuersignalen den Prädiktionsschritt durchzuführen und so auch ohne aktuelle Positionsschätzung eine Regelung zu ermöglichen. Die Nebeninstanz ist eine Kopie der Hauptinstanz, die erstellt wird, nachdem ein Korrekturschritt erfolgt ist. Alle Steuersignale, die für die Prädiktion in der Hauptinstanz verwendet wurden, werden gespeichert. Ist eine Positionsschätzung vorhanden, werden die Prädiktionsschritte bis zum Zeitstempel der Positionsschätzung durchgeführt. Anschließend erfolgt die Korrektur mit den Positionsdaten und die verbleibenden Steuersignale werden dafür aufgewendet, um die Prädiktion zum aktuellen Zeitpunkt durchzuführen. Die Nebeninstanz wird damit zur Hauptinstanz und für die Nebeninstanz wird eine neue Kopie angelegt. Dadurch ist es möglich, die verzögerten Positionsdaten im Korrekturschritt des Kalman-Filters zu verwenden. In Abbildung 6.5 ist ein Vergleich der Zustandsschätzung für die Postion auf der Y-Achse mit und ohne einer Kompensation

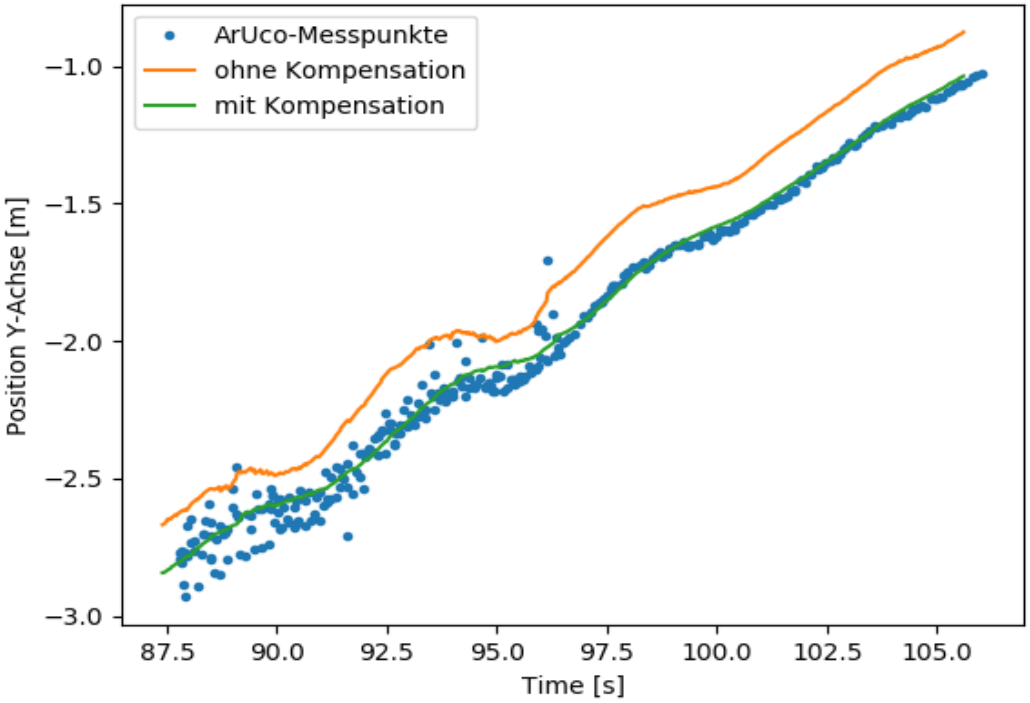


Abbildung 6.5.: Kalman-Schätzung mit und ohne Kompensation

der Verzögerung abgebildet. Darin ist deutlich zu erkennen, dass die Zustandsschätzung mit dem Kalman-Filter durch die Kompensation verbessert werden konnte.
Mit den Zustandsschätzungen des Kalman-Filters, den aktuellen Zuständen der Trajektorie und den Reglerparametern aus Kapitel 6.3 wird anschließend das neue Steuersignal berechnet. Wenn nach einer gewissen Zeit keine neuen Positionsdaten mehr eintreffen, dann wird die Varianz der Positionsschätzung immer größer. Ab einem bestimmten Wert wird der Zustand *AutoControl* verlassen und der Abbruch der Landung eingeleitet. Das Vorgehen zum Abbruch der Landung muss ebenfalls je nach UAV definiert werden. Im Fall der Tello EDU ist für den Abbruch vorgesehen, dass die Drohne in den schwebenden Flug übergeht und dort verweilt, bis die manuelle Kontrolle übernimmt oder nach einer gewissen Zeit die Notabschaltung erfolgt.

Listing 6.1: Implementierung der State-Machine

```
1  def __run_state_machine(self):
2      """
3      Run the StateMachine
4      :return:
5      """
6      while self.__running:
7
8          user_input = self.__front_end.get_actual_user_input()
9          if self.__current_state == self.States.Init:
10             self.__current_state = self.__state_machine_init(user_input)
11
12         elif self.__current_state == self.States.Manual:
13             self.__current_state = self.__state_machine_manual(user_input)
14
15         elif self.__current_state == self.States.Init_AutoControl:
16             self.__current_state = self.__state_machine_initautocontrol(user_input)
17
18         elif self.__current_state == self.States.AutoControl:
19             self.__current_state = self.__state_machine_autocontrol(user_input)
20
21         elif self.__current_state == self.States.Cancel:
22             self.__current_state = self.__state_machine_cancel(user_input)
23
24         elif self.__current_state == self.States.Failure:
25             self.__current_state = self.__state_machine_failure(user_input)
26
27         else:
28             self.__current_state = self.States.Failure
29         self.state['StateMachine'] = self.__current_state
```

6.4.2. FrameGrabber

In der Klasse *FrameGrabber* sind alle Funktionalitäten implementiert, die für das Erhalten eines Bildes notwendig sind. Die wichtigsten Member der Klasse sind der String des zu lesenden Video-Streams, der zu verwendende Modus zum Abrufen der Bilder und die Liste der Queues, die als Ausgang verwendet werden.
Für das Abrufen der Bilder sind zwei unterschiedliche Modi implementiert. Im *FreeRun* werden die Bilder so schnell vom Stream gelesen, wie der Stream die Bilder liefert. Im Modus *GrabOneByOne* wird erst ein neues Bild gelesen, wenn die Bilder aus den vorhandenen Queues von den anderen Prozessen ausgelesen sind. Abhängig davon, aus welcher Quelle und zu welchen Zweck die Bilder verwendet werden, wird der passende Modus ausgewählt. Der Video-Stream muss in der Initialisierung des Objektes des *FrameGrabbers* übergeben

Listing 6.2: Implementierung des Bildeinzugs

```
1  def __grab_freerun(self):
2      while self.is_grabbing:
3          (grabbed, grab) = self.__video_capture.read()
4          if grabbed:
5              self.__last_grab = self.get_time() - self.video_timeout
6              if self.image_format_transform:
7                  self.__frame = cv2.cvtColor(grab, self.image_format_transform)
8              else:
9                  self.__frame = grab
10             for queue in self.__output_queues:
11                 queue.append(Frame(self.__frame, self.__last_grab))
12     self.__video_capture.release()
```

werden. Mit der Klassen-Funktion *start_grabbing()*, wird das kontinuierliche Lesen der Bilder vom Stream gestartet. Das Lesen der Bilder vom Stream läuft asynchron zu den anderen Funktionen des Gesamtsystems. Deshalb wird für das Lesen der Bilder ein Thread gestartet, der die Funktion *__grab_freerun()* ausführt, bis das Lesen der Bilder durch den Aufruf von *stop_grabbing()* angehalten wird.
Bevor das Lesen des Streams gestartet werden kann, müssen eine oder mehrere Ausgangs-Queues erzeugt werden, die an die anderen Objekte übergeben werden. Durch diese Queues kann eine asynchrone Übergabe der Bilder an andere Prozesse erfolgen.
Die Implementierung zum Lesen der Bilder ist im Listing 6.2 abgebildet. In Zeile 3 wird die OpenCV Funktion *read()* vom VideoCapture Objekt des Video-Streams aufgerufen. Ist das Lesen des Bildes erfolgreich, wird der Zeitstempel für das Bild berechnet. Dazu wird in 5 die Funktion *get_time()* aufgerufen und die Verzögerung der Videoübertragung subtrahiert. Die Verzögerung muss durch den Anwender des *FrameGrabbers* vor dem Starten des Threads gesetzt werden. Dies wird in Abschnitt 6.4.1 beschrieben.
In 7 wird, bei einer definierten Format-Transformation, das Format des Bildes zum Beispiel von einem RGB-Bild in ein Grauwert-Bild transformiert. Das Array des Bildes wird anschließend zusammen mit dem Zeitstempel verwendet, um ein Objekt der Klasse *Frame* zu erstellen. Die Klasse *Frame* beinhaltet die Member *array* und *timestamp*. Dies dient der festen Zuordnung von Zeitstempel und Bild. Die erzeugten Frames werden anschließend in Zeile 11 an jede der vorhandenen Queues angehängt. Anschließend erfolgt der nächste Schleifendurchlauf.

6.4.3. TelloControl

Die Klasse *TelloControl* beinhaltet alle Funktionen, die notwendig sind, um die Tello EDU zu steuern. Die Grundlage dafür bietet die abstrakte Klasse *UavInterface*. Durch die abstrakte Klasse soll es später einfacher möglich sein, ein anderes UAV mit diesem System zu landen. In *UavInterface* sind Funktionen wie *set_velocity()* und *stream_on()* definiert. Außerdem wird definiert, wie das mathematische Modell hinterlegt werden muss und welche Status vom UAV für die *FlightControl* bereitgestellt werden müssen. Das mathematische Modell wird über die vier Matrizen *A, B, C, D* definiert. Zusätzlich muss angegeben werden, welche Pole das geregelte System haben soll, damit der Zustandsregler in *FlightControl* ausgelegt werden kann. In Kapitel 6.3 ist die Bestimmung der Parameter für die Tello beschrieben.
Für die Steuerung der Tello wird die bereitgestellte API verwendet. Mit der API kann die Tello durch Senden und Empfangen von Strings über das User Datagram Protocol (UDP) gesteuert werden.
Für das Empfangen der Status-Daten von der Tello EDU wird ein Thread gestartet, der in einer Schleife die UDP-Pakete von der Tello empfängt und den String verarbeitet. Für das Senden der Kontrollkommandos wie *connect* oder *stream on* sind einzelne Funktionen implementiert, die den gewünschten String an den zugehörigen UDP-Port der Tello schicken. Diese Funktionen liefern als Rückgabewert den Wert *True* wenn die Drohne den Erhalt des Kommandos mit dem String *ok* quittiert. Andernfalls wird nach einer bestimmten Zeit der Wert *False* zurückgeliefert.
Die Steuerung des Fluges erfolgt bei der Tello EDU in einem gesonderten Thread. Da die Tello nach einer gewissen Zeit, in der die Tello EDU keine Signale mehr erhalten hat, automatisch landet, wird in einer Schleife alle 20 ms ein neuer Steuerbefehl an die Tello EDU gesendet. Dies geschieht auch wenn der gleiche Befehl mehrmals gesendet wird. Der aktuelle Steuerbefehl wird dazu von *FlightControl* mit der Funktion *set_velocity()* in Variablen hinterlegt und dort vom Thread abgerufen und an die Tello gesendet. Die Quittierung über den Erhalt der Befehle erfolgt hierbei nicht.
Für die Messung der Zeiten erbt die Klasse *TelloControl* von der Klasse *TimedSystem*.

6.4.4. MarkerPositioning

Initialisierung und Schnittstellen

In der Klasse *MarkerPositioning* ist die Positionsschätzung des UAV anhand der Kamerabilder implementiert. Damit dies möglich ist, muss bei der Initialisierung von einem Objekt dieser Klasse der Name des UAV übergeben und in dem Projektverzeichnis unter *config/* eine *.npz* mit diesem Namen abgelegt sein. Diese Datei muss die Kalibrierung der Kamera

enthalten. Für die Übertragung der Bilder muss eine *Deque*, wie sie in 6.4 dargestellt wurden, von einem *FrameGrabber*-Objekt übergeben werden.
Zusätzlich muss eine Liste aller verwendeten Marker mit deren Eckpunkten in Weltkoordinaten übergeben werden. Anhand dieser Liste wird ein *cv2.aruco.Dictionary* mit allen verwendeten Markern erstellt. Nach der erfolgreichen Initialisierung kann die kontinuierliche Detektion mit *start_system()* gestartet werden.
Für die Detektion wird ein Thread gestartet, der sobald ein *Frame* in der *Deque* vorhanden ist, die Funktion *get_position()* aufruft. Zurückgegeben werden zwei Listen mit den Ergebnissen der Positions- und Orientierungsbestimmungen. Für jeden detektierten Marker ist eine Schätzung in den Listen enthalten. Zusätzlich wird der Zeitstempel des Bildes zurückgegeben, von dem die Schätzungen stammen. Die Daten werden anschließend zum Datentyp *Data* zusammengefasst und an die Ausgangs-Deque des Objektes angehängt. Der Datentyp *Data* hat dabei die gleiche Funktionalität wie *Frame* bei den Bildern und sorgt lediglich für eine feste Verkettung von Daten und Zeitstempel.

Detektion und Positionsschätzung

In Listing 6.3 ist das Vorgehen für die Ermittlung der Positionsdaten dargestellt. Für die Detektion der Marker wird ein *Frame*-Objekt von der *Deque* entnommen. Mit der in OpenCV implementierten Funktion *cv2.aruco.detectMarkers()* wird in dem Bild nach den im *Dictionary* hinterlegten Markern gesucht. Das Resultat ist ein Array mit den vier Ecken pro gefundenen Marker in Bildkoordinaten und die dazugehörigen Marker-IDs. Dieses Vorgehen ist in der Funktion *find_marker()* implementiert und wird in Zeile 8 aufgerufen. Das Detektieren der Marker erfolgt nach dem Verfahren, das in Abschnitt 3.3.3 auf Seite 29 beschrieben ist.
In Zeile 10 wird die Positionsschätzung für das aktuelle Bild abgebrochen, wenn kein Marker im Bild detektiert worden ist. Ansonsten wird in Zeile 13 über alle gefundenen Marker-IDs und die Bildpunkte der Ecken iteriert. In Zeile 14 werden dazu die Weltkoordinaten des aktuellen Markers aus dem Dictionary *self.__marker_points* in die Variable *obj_points* geladen. Die Bestimmung der Weltkoordinaten der Marker ist in Kapitel 6.4.7 beschrieben.
Die Welt- und Bildkoordinaten eines Markers bilden zusammen ein Punktepaar, welches notwendig ist, um ein PnP-Problem darzustellen, wie es in Kapitel 3.3.2 beschrieben wird. Durch die OpenCV-Funktion *cv2.solvePnP()* wird die Fehlerfunktion (6.3) durch ein iteratives Verfahren minimiert. Allerdings wird im Gegensatz zur Kamerakalibrierung nicht der Parameter $\boldsymbol{P}_i$ optimiert, sondern mit dem durch die Kamerakalibrierung festgelegten $\boldsymbol{P}_i$ eine optimale Matrix $\boldsymbol{P}_e$ bestimmt.
Das Resultat von *cv2.solvePnP()* sind die zwei Vektoren $\boldsymbol{t}_{cam \rightarrow world}$ und $\boldsymbol{r}_{cam \rightarrow world}$, die die Translation und Rotation der Landebahn im Kamerakoordinaten-System beschreiben. Weil die Trajektorie in den Weltkoordinaten beschrieben wird, ist es einfacher, die Regelung mit Daten zu realisieren, die die Translation und Rotation der Kamera im Weltkoordinatensys-

Listing 6.3: Bestimmung der Kamera-Position

```
1  def __get_camera_position(self):
2      """
3      returns a list of camera position estimations for each detected
4      :return: a list of rotation and translation vectors and the ts of the image
5      """
6      rvec_list = []
7      tvec_list = []
8      corners, ids, ts = self.find_marker()
9      if ids is None:
10         return None, None, None
11     ids = np.squeeze(np.asarray(ids), 1)
12     corners = np.squeeze(np.asarray(corners), 1)
13     for _id, img_points in zip(ids, corners):
14         obj_points = self.__marker_points[_id]
15         rvec, tvec = self.__estimate_pose_pnp(obj_points, img_points)
16
17         rvec_list.append(rvec)
18         tvec_list.append(tvec)
19     return rvec_list, tvec_list, ts
```

tem beschreiben. Deshalb wird aus $\boldsymbol{r}_{cam \to world}$ die Rotationsmatrix $\boldsymbol{R}_{cam \to world}$ bestimmt und damit

$$\boldsymbol{t}_{world \to cam} = -\boldsymbol{R}^{-1}_{cam \to world} \cdot \boldsymbol{t}_{cam \to world} \tag{6.7}$$

berechnet. Die Rotation $\boldsymbol{r}_{cam \to world}$ wird in die Darstellung als Euler-Winkel transformiert. Durch Negation der Winkel erhält man die Rotation $\boldsymbol{r}_{world \to cam, euler}$ der Kamera im Weltkoordinatensystem der Landebahn. Die beschriebenen Berechnungen sind in der Funktion *__estimate_pose_pnp()* implementiert und werden in Zeile 15 aufgerufen.
Die Ergebnisse der Berechnungen werden in zwei Listen gesammelt und anschließend zusammen mit dem Zeitstempel an die aufrufende Funktion zurückgegeben.

6.4.5. Observer

In der Klasse *Observer* ist der Algorithmus eines Kalman-Filters implementiert. Die Klasse stellt die beiden Funktionen *predict()* und *correct()* bereit, die die beiden Schritte des Kalman-Filters darstellen, wie sie in Kapitel 2.4 beschrieben wurden. Das Modell des Systems wird in Abschnitt 6.3 bestimmt und hier verwendet.
Im Listing 6.4 ist die Funktion für den Prädiktionsschritt dargestellt. Zu Beginn der Prädiktion

Listing 6.4: Implementierung des Prädiktionsschritts des Kalman-Filters

```
1  def predict(self, timestamp: int, u: np.ndarray):
2      """
3      Prediction step of the Kalman Filter
4      :param timestamp: the actual timestamp
5      :param u: the actual control values
6      :return:
7      """
8      # calculate the delta to last prediction
9      dt = (timestamp - self.__timestamp) * 1e-9
10     # set the actual time as filter time
11     self.__timestamp = timestamp
12     # calculate the discrete space system
13     self.__dss = self.css.to_discrete(dt)
14     # calculate the process noise
15     Gd = self.__dss.A @ self.__G
16     # update the states and the probability matrix
17     self.__x = self.__dss.A @ self.__x + self.__dss.B @ u
18     self.__P = self.__dss.A @ self.__P @ self.__dss.A.T + Gd @ self.__Q @ Gd.T
19     return self.__x, self.__P
```

wird in Zeile 9 die vergangene Zeit seit dem letzten Prädiktionsschritt berechnet. Anschließend wird in Zeile 11 der Zeitstempel des Filters auf die aktuelle Zeit aktualisiert. Mit dem bestimmten Delta *dt* zwischen den Zeitstempeln wird in Zeile 13 das kontinuierliche Modell des Systems, welches bei der Initialisierung übergeben wurde, diskretisiert. Anschließend erfolgt die Berechnung des Messrauschens. Die eigentliche Prädiktion der Zustände erfolgt in Zeile 17 durch die Lösung der Gleichung (2.7) von Seite 22. Die Prädiktion der Kovarianzmatrix erfolgt in Zeile 18 durch die Lösung der Gleichung (2.8).

Mit dem in Listing 6.5 abgebildeten Korrekturschritt werden die Messdaten mit in den Kalman-Filter eingebunden. Bevor die Korrektur stattfindet, müssen die Zustände des Kalman-Filters zum aktuellen Zeitstempel prädiziert werden. Dies erfolgt in Zeile 12. Zu einem Zeitschritt kann es mehrere Messungen geben, weil in einem Bild mehrere ArUco-Marker detektiert werden können. Die Fusion der Messergebnisse erfolgt bei der Korrektur im Kalman-Filter. Dazu wird in Zeile 14 über eine Liste iteriert, die alle zum aktuellen Zeitstempel gehörenden Messergebnisse enthält. Jede Messung wird dann einzeln dazu verwendet, die Prädiktion zu korrigieren, indem in Zeile 19 die aktuelle Verstärkung des Kalman-Filters berechnet wird. Im Anschluss werden die Zustände in Zeile 21 aktualisiert, indem das Ergebnis der Gleichung (2.10) berechnet wird. In Zeile 22 wird abschließend noch die Kovarianzmatrix durch die Lösung von Gleichung (2.11) aktualisiert.

Listing 6.5: Implementierung des Korrekturschritts des Kalman-Filters

```
1  def correct(self,  measurement: [np.ndarray], timestamp: int, u=None):
2      """
3      Correction-Step of the Kalman-Filter with the measurement
4      :param measurement: list of measurements
5      :param timestamp: actual timestamp in ns
6      :param u: the actual control value
7      :return:
8      """
9      # if no control is given set it to zero
10     if u is None:
11         u = np.zeros((self.css.B.shape[1], 1))
12     self.predict(timestamp, u)
13     # iterate over all measurements
14     for meas in measurement:
15         # temporal calculations
16         ep = meas - self.__dss.C @ self.__x + self.__dss.D @ u
17         tmp = self.__dss.C @ self.__P @ self.__dss.C.T + self.__R
18         # calculate the gain
19         K = self.__P @ self.__dss.C.T @ np.linalg.inv(tmp)
20         # update the states and the probability matrix
21         self.__x = self.__x + K @ ep
22         self.__P = self.__P - K @ self.__dss.C @self.__P
23     return self.__x, self.__P
```

Nachdem alle Messungen verarbeitet worden sind, wird der aktuelle Zustand und die aktuelle Kovarianzmatrix zurückgegeben.

6.4.6. TrajectoryPlaner

In der Klasse *TrajectoryPlaner* ist die Berechnung der Trajektorie für den Landeanflug implementiert. Die Berechnung der Trajektorie erfolgt dabei nur einmal zu Beginn der Landephase und folgt damit dem Verfahren aus Abbildung 3.5(a) von Seite 32. Es erfolgt keine Aktualisierung der Trajektorie, weil bei einem Landeanflug eine feste Strecke abgeflogen werden soll. Kleine Störungen können durch den Regler ausgeglichen werden. Bei größeren Störungen sollte der Landeanflug abgebrochen werden, anstatt eine neue Route zu versuchen, weil für eine Landung die für ein UAV geltenden Parameter wie Anflugwinkel und Geschwindigkeit unbedingt eingehalten werden sollten.
Das Geschwindigkeitsprofil wird dabei anhand der übergebenen Parameter bestimmt. Die zu übergebenden Parameter sind in der Tabelle 6.3 aufgeführt. Die Trajektorie wird so berechnet, dass sich ein zeitoptimales Geschwindigkeitsprofil ergibt, wie es in Kapitel 3.4.2 auf

Seite 34 beschrieben wird. Allerdings wird das Verfahren so erweitert, dass es möglich ist, für die Beschleunigung in der Luft und am Boden unterschiedliche Werte anzugeben, weil ein Luftfahrzeug sich unter Umständen in der Luft deutlich anders verhalten kann als am Boden. Zusätzlich kann es wünschenswert sein, dass nach der erfolgreichen Landung das Luftfahrzeug direkt von der Landebahn fahren soll. Aus diesem Grund ist es möglich, eine Endgeschwindigkeit anzugeben, die das UAV am Ende der Trajektorie haben soll.
Der räumliche Verlauf der Trajektorie wird durch den Aufsetzpunkt $\boldsymbol{p}_{td}$, den Endpunkt der Trajektorie $\boldsymbol{p}_{end}$ und den Anflugwinkel α bestimmt. Im Folgenden wird der Vorgang zu Berechnung der Trajektorie beschrieben.
Der Anfangspunkt der Trajektorie wird durch die aktuelle Position des UAV festgelegt und den Endpunkt bildet der Punkt $\boldsymbol{p}_{end}$. Diese beiden Punkte werden explizit angefahren. Dazwischen befinden sich die Punkte $\boldsymbol{p}_{td}$ und $\boldsymbol{p}_{entry}$. Der Punkt $\boldsymbol{p}_{entry}$ beschreibt den Punkt, an dem das UAV vom Reiseflug in den Gleitflug wechselt. Dieser wird beim Start der Trajektorie mit Hilfe des Anflugwinkels α und der aktuellen Flughöhe h zu

$$\boldsymbol{dir}_{x,y} = \frac{\boldsymbol{p}_x - \boldsymbol{p}_y}{|\boldsymbol{p}_x - \boldsymbol{p}_y|} \tag{6.8}$$

$$\boldsymbol{p}_{entry} = \boldsymbol{p}_{td} - \boldsymbol{dir}_{td,end} \cdot \left| \frac{h}{\tan(\alpha)} \right| \tag{6.9}$$

$$\boldsymbol{p}_{entry}.z = \boldsymbol{p}_{entry}.z + h \tag{6.10}$$

berechnet. In 6.9 wird festgelegt, wie die X und Y Koordinaten von $\boldsymbol{p}_{entry}$ lauten sollen. Die Verschiebung auf die aktuelle Flughöhe erfolgt in Gleichung 6.10. Der Punkt $\boldsymbol{p}_{entry}$ befindet sich damit immer auf der aktuellen Flughöhe und in einer Linie mit den Punkten $\boldsymbol{p}_{td}$ und $\boldsymbol{p}_{end}$. Die vier Punkte $\boldsymbol{p}_{start}$, $\boldsymbol{p}_{entry}$, $\boldsymbol{p}_{td}$ und $\boldsymbol{p}_{end}$ werden nachfolgend als die Basis-Punkte der Trajektorie bezeichnet.
Damit der Richtungswechsel durch eine gleichmäßige Bewegungsänderung der Achsen erfolgt, werden die Basis-Punkte $\boldsymbol{p}_{entry}$ und $\boldsymbol{p}_{td}$ überschliffen und nicht exakt durchflogen. Für die Berechnung der Trajektorie werden deshalb die neuen Stützpunkte $\boldsymbol{p}_{entry,1}$, $\boldsymbol{p}_{entry,2}$ und $\boldsymbol{p}_{td,1}$, $\boldsymbol{p}_{td,2}$ berechnet, durch die die Trajektorie verlaufen soll. Die neuen Stützpunkte sind die Schnittpunkte der Geraden zwischen den Basis-Punkten und einer Kugel mit dem Radius r um den zu überschleifenden Basis-Punkt herum. Als Berechnungsvorschrift ergibt sich

somit für die Stützpunkte

$$\boldsymbol{p}_{entry,1} = \boldsymbol{p}_{entry} - \boldsymbol{dir}_{start,entry} \cdot r \tag{6.11}$$
$$\boldsymbol{p}_{entry,2} = \boldsymbol{p}_{entry} + \boldsymbol{dir}_{entry,td} \cdot r \tag{6.12}$$

$$\boldsymbol{p}_{td,1} = \boldsymbol{p}_{td} - \boldsymbol{dir}_{entry,td} \cdot r \tag{6.13}$$
$$\boldsymbol{p}_{td,2} = \boldsymbol{p}_{td} + \boldsymbol{dir}_{td,end} \cdot r. \tag{6.14}$$

Die sechs resultierenden Stützpunkte werden anschließend mit Hilfe einer kubischen Spline-Interpolation interpoliert. Das Resultat ist in Abbildung 6.6 abgebildet. Die Größe des Überschleifradius r kann vom Anwender vorgegeben werden.

Geschwindigkeitsprofil

Anschließend erfolgt die Berechnung des zur räumlichen Trajektorie passenden Geschwindigkeitsprofils. Dazu wird die Trajektorie in die vier Bereiche *Beschleunigung, konstante Geschwindigkeit, Bremsen* und *Landebahn* eingeteilt. Es ist dabei nicht notwendig, dass alle Bereiche in jeder Trajektorie vorkommen. Für jeden Abschnitt wird bestimmt wie viel Zeit notwendig ist, um diesen zu bewältigen.
Die benötigte Zeit auf der Landebahn berechnet sich durch

$$t_{runway} = |\frac{v_{td} - v_{end}}{a_{ground}}|. \tag{6.15}$$

Wobei a_{ground} eventuell verringert wird, wenn nicht die maximale Bremsbeschleunigung auf der Landebahn benötigt wird. Die benötigte Zeit für das Bremsen vor dem Aufsetzen berechnet sich durch

$$t_{dec} = |\frac{v_{max} - v_{td}}{a_{air}}|. \tag{6.16}$$

Die Beschleunigungszeit am Anfang der Trajektorie ergibt sich durch

$$t_{acc} = |\frac{v_{start} - v_{max}}{a_{air}}|. \tag{6.17}$$

Mit den Strecken aus den zuvor beschriebenen Abschnitten und der gesamten Strecke s_{ges} ist es nun möglich, die Strecke für den Abschnitt *konstante Geschwindigkeit*

$$s_{constvel} = s_{ges} - s_{acc} - s_{dec} - s_{runway} \tag{6.18}$$

Tabelle 6.3.: Parameter für die Trajektorienberechnung

Formelzeichen	Einheit	Bedeutung
a_{air}	$\frac{m}{s^2}$	Die Beschleunigung der Geschwindigkeitsänderung in der Luft
a_{ground}	$\frac{m}{s^2}$	Bremsbeschleunigung auf der Landebahn
v_{max}	$\frac{m}{s}$	Die maximale Geschwindigkeit auf der Trajektorie
v_{td}	$\frac{m}{s}$	Die Geschwindigkeit zum Zeitpunkt des Aufsetzens
v_{end}	$\frac{m}{s}$	Die Geschwindigkeit am Ende der Trajektorie
$\boldsymbol{p}_{td}$	m	Die Position des Aufsetzpunktes in den Weltkoordinaten X, Y und Z
$\boldsymbol{p}_{end}$	m	Das Ende der Trajektorie
α	Grad	Der Anflugwinkel

zu bestimmen. Die Zeit für diesen Abschnitt ergibt sich damit aus

$$t_{constvel} = \frac{s_{constvel}}{v_{max}}. \tag{6.19}$$

Mit Hilfe der bestimmten Zeitpunkte lassen sich zwei abschnittsweise definierte Gleichungen für die zurückgelegte Strecke s und die aktuelle Geschwindigkeit v in Abhängigkeit von der Zeit definieren. Diese Gleichungen werden im Listing 6.6 ausgewertet.
In Zeile 6 wird anhand des übergebenen Zeitstempels und der Startzeit der Trajektorie die aktuelle Zeit auf der Trajektorie berechnet. Die berechnete Zeit wird anschließend verwendet, um zu bestimmen, in welcher Phase des Geschwindigkeitsprofils die Trajektorie aktuell ist. In den Zeilen 9 und 10 wird die aktuell zurückgelegte Strecke und die aktuelle Geschwindigkeit für die Beschleunigungsphase berechnet.
In der Variable *self.__s_xyz* ist die kumulative Summe der zurückzulegenden Strecke der gesamten Trajektorie abgelegt. Mit der aktuell zurückgelegten Strecke s wird in Zeile 32 der Index des Wertes aus *self.__s_xyz* gesucht, der der aktuellen Strecke am nächsten kommt. Mit dem Index wird anschließend in Zeile 33 die aktuelle Position aus dem Array *self.__xyz* ermittelt. Das Array *self.__xyz* beinhaltet die abgetasteten Werte aus der Spline-Interpolation.
Die Variable v beinhaltet lediglich den absoluten Wert für die Geschwindigkeit. Für die Bestimmung der Richtung wird in Zeile 34 die Ableitung der Spline-Interpolation *self.__v_xyz* an der Stelle *index* ausgewertet. Die Ableitung der Spline-Interpolation ist in Abbildung 6.8 dargestellt. Dieser Wert wird auf 1 normiert und mit dem Wert v multipliziert. Zurückgegeben werden die aktuelle Position, die aktuelle Geschwindigkeit aller Achsen und ein boolscher Wert, der angibt, ob die Trajektorie beendet ist oder nicht.

Listing 6.6: Bestimmung der aktuellen Trajektorienzustände

```
def get_traj_states(self, timestamp: int) -> (Point3D, np.ndarray, bool):
    """
    Return the actual states
    :return:
    """
    t = (timestamp - self.__t_start_ns) * 1e-9
    # acceleration
    if t <= self.t_acc:
        s = self.__v_start * t + self.__acc * t**2 / 2
        v = self.__v_start + self.__acc * t
    # constant velocity
    elif t <= self.t_full_vel + self.t_acc:
        t -= self.t_acc
        s = self.__velocity * t + self.s_acc
        v = self.__velocity
    # deceleration
    elif t <= self.t_full_vel + self.t_dec + self.t_acc:
        t -= self.t_full_vel + self.t_acc
        s = self.s_acc + self.s_full_vel \
            + self.__velocity * t - self.__deceleration * t**2 / 2
        v = self.__velocity - self.__deceleration * t
    # runway
    elif t <= self.t_full_vel + self.t_dec + self.t_runway + self.t_acc:
        t -= self.t_full_vel + self.t_dec + self.t_acc
        s = self.s_acc + self.s_full_vel + self.s_dec \
            + self.__v_touchdown * t - self.__d_runway * t**2 / 2
        v = self.__v_touchdown - self.__d_runway * t
    # finished
    else:
        return self.__p_end, np.zeros(3), True

    index = np.searchsorted(self.__s_xyz, s, 'right')
    actual_point = Point3D(*self.__xyz[index])
    v_real = self.__v_xyz[index] / np.linalg.norm(self.__v_xyz[index]) * v


    return Point3D(*self.__xyz[index]), v_real, False
```

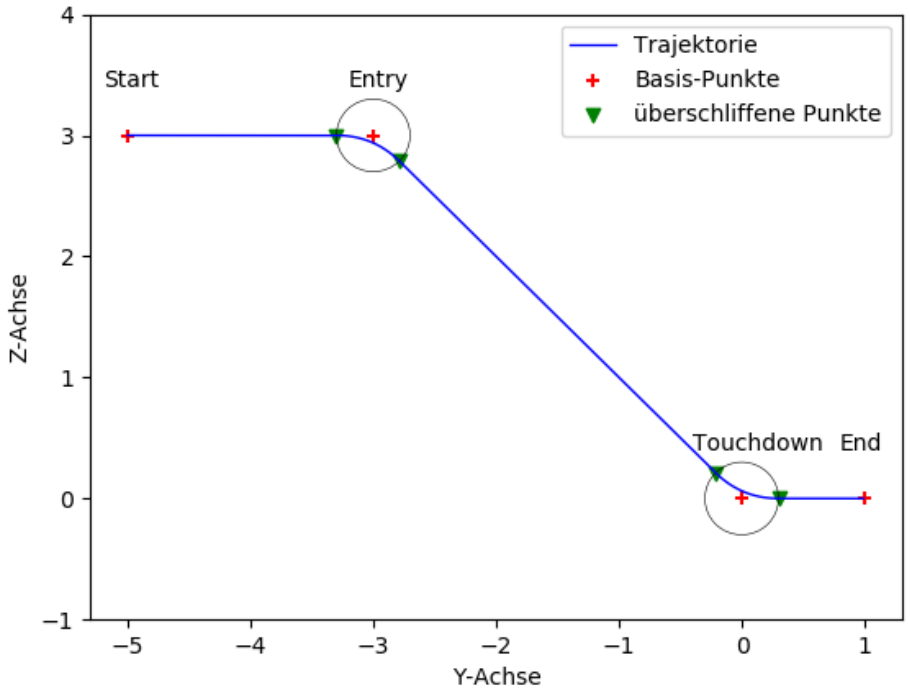


Abbildung 6.6.: Trajektorie im Raum

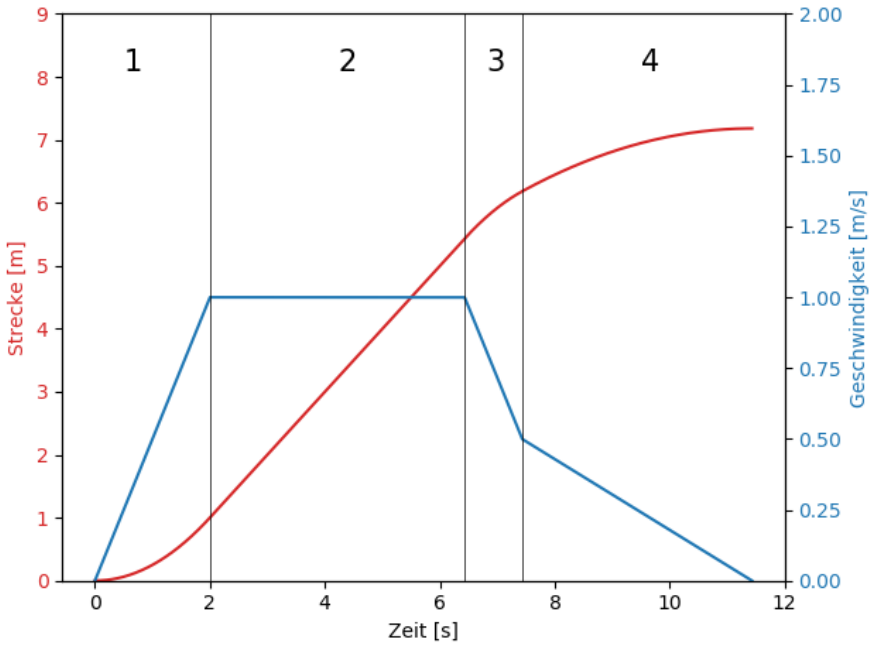


Abbildung 6.7.: Geschwindigkeitsprofil der Trajektorie mit der zurückgelegten Strecke. Abschnitte: 1. Beschleunigung, 2. konstante Geschwindigkeit, 3. Bremsung und 4. Bremsung auf der Landebahn.

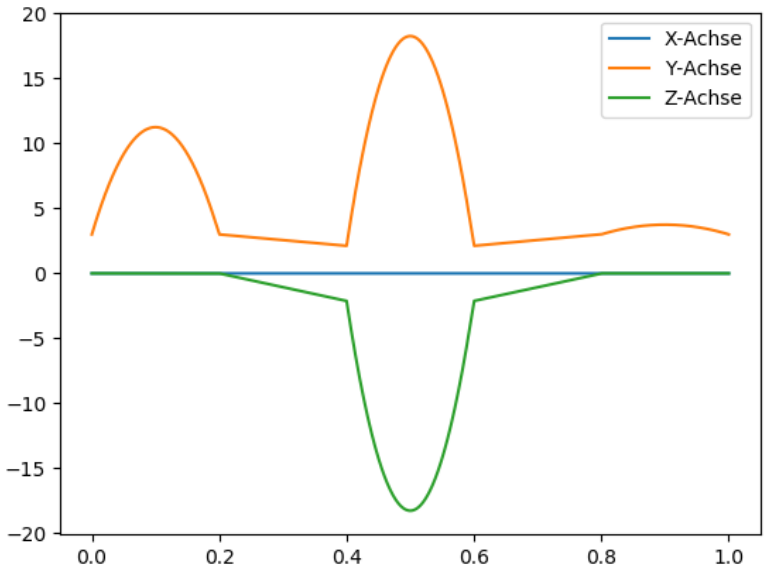


Abbildung 6.8.: Ableitung der Spline-Interpolation

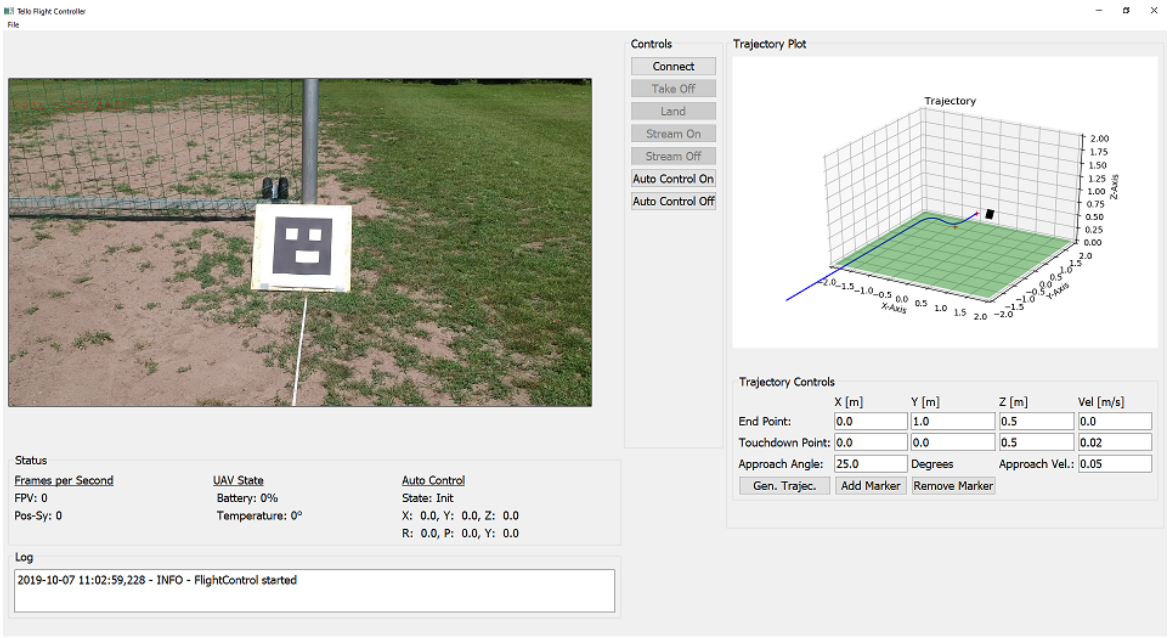


Abbildung 6.9.: Trajektorie im Raum

6.4.7. GUI

Für den Fall, dass das Landesystem ohne ein anderes Programm verwendet wird, wird eine GUI implementiert, die es dem Anwender ermöglicht, den manuellen Flug durchzuführen und die Trajektorie und Landebahn zu parametrisieren. Die GUI beinhaltet einen First-Person-View, einen Plot der Trajektorie mit der Landebahn und Kontrollelemente, mit der das UAV gesteuert werden kann. Eine Abbildung des GUI-Hauptfensters ist in Abbildung 6.9 dargestellt.
Über die GUI können die Positionen der Marker eingegeben werden und es kann festgelegt werden, in welchem Winkel der Landeanflug erfolgen soll. Der Plot der Trajektorie dient der Überprüfung, ob die Marker in der Landetrajektorie stehen oder anderes unerwünschtes Verhalten parametriert wurde.

7. Ergebnisse

In diesem Kapitel wird überprüft, on die Implementierung die in Kapitel 4 gestellten Anforderungen erfüllt.

7.1. Marker-Detektion

Für eine Bewertung wird überprüft, wie gut die Marker mit dem System detektiert werden können und wie gut die Positionsschätzung dazu ist. Um die Detektionsrate zu testen, werden sowohl im Freien als auch im Innenraum Videos mit einer Länge von 10 Sekunden mit der FPV Kamera einer Drohne aufgenommen. Für den Innenraum wird die Tello EDU verwendet und im Freien eine DJI Spark. Die DJI Spark liefert dabei Bilder in einer Auflösung von 1920 x 1080 Pixeln. Im Innenraum wird in Abständen von einem Meter jeweils ein Video bei schwebender Drohne aufgenommen. Im Freien werden die Videos in Abständen von 3 Metern aufgenommen. Es wird dabei sichergestellt, dass in jedem Bild der Marker zu sehen ist. Anschließend werden die Bilder mit dem *MarkerPositioning* ausgewertet. In Abbildung 7.1 sind die Detektionsraten dargestellt. Darin ist zu erkennen, dass selbst in großer Entfernung die Marker noch sicher detektiert werden. Bei der Überprüfung der detektierten Marker konnten in allen Videos keine Falsch-Positiv-Detektionen entdeckt werden. Die Detektionsraten, die hier erzielt werden, sind für den Gebrauch in Innenräumen ausreichend, da selbst bei 7 Metern Entfernung zwischen dem Marker und der Drohne in jedem Bild der Marker detektiert wurde. Für den Außenbereich ist die Detektionsrate auch bei 20 Metern noch bei 100%. In einer Entfernung von 15 Metern wird auf zwei von 300 Bildern der Marker nicht erkannt. Bei der Untersuchung der Bilder konnte nicht festgestellt werden, aus welchen Gründen die Marker nicht erkannt wurden. In einem weiteren Video ist auch in 15 Metern Entfernung eine Detektionsrate von 100% erreicht worden. Trotzdem soll hier gezeigt werden, dass auch bei gut sichtbaren Markern eine Detektion gelegentlich fehlschlagen kann. Allerdings ist auch bei einer extremen Beleuchtung wie sie in Abbildung 7.2 abgebildet ist, eine Detektionsrate von 100% erreicht worden. Damit kann man mit dem implementierten Vorgehen davon ausgehen, dass die Marker bis auf einzelne Fehler sehr gute Detektionsraten aufweisen.
Zusätzlich zur Detektionsrate ist außerdem entscheidend, wie gut die Positionsschätzung anhand der Marker ist. Dazu werden erneut die Videos ausgewertet und die Ergebnisse der

Positionsschätzung betrachtet. Am wichtigsten ist dabei, dass die Positionsschätzung direkt vor dem Aufsetzen auf der Landebahn eine hohe Genauigkeit hat. Dazu wird die Tello EDU so auf einem Tisch platziert, dass die Kamera der Tello EDU in einem Abstand von 1 Meter zum Marker und auf einer Höhe von 48 cm zentral vor dem Marker steht. In dieser festen Position wird ein Video mit einer Länge von 10 s aufgenommen und die Postion mittels der Implementierung von *MarkerPositioning* bestimmt. Bei einer korrekten Parametrierung der Marker kann mit der in Kapitel 6.1 erstellten Kamerakalibrierung eine maximale Abweichung von unter einem Zentimeter für die Achsen X, Y und Z erreicht werden. In der Veröffentlichung [LCC16] sind ähnlich gute Ergebnisse mit einer geringeren Bild-Auflösung erzielt worden.
Die Messergebnisse für eine Entfernung von 3 Metern im Freien ist in Abbildung 7.3 dargestellt. Auch in dieser Entfernung ist die Positionsschätzung noch zufriedenstellend. Die Messung der absoluten Abweichung ist in diesem Fall nicht so exakt möglich, wie bei der Messung in einem Meter Abstand, aber es gibt keine größeren Messfehler, wie sie in größerer Entfernung auftreten. Ab einer Entfernung von 11 m ist die Schätzung nicht mehr ausreichend, weil ab dieser Entfernung die Optimierung zwei Lösungen enthalten kann. Dieses Problem wird in Kapitel 3.3.2 erwähnt. Die Ergebnisse der Schätzungen in 11 m Entfernung sind in Abbildung 7.4 dargestellt. In diesem Fall ist es hilfreich, einen weiteren Sensor zu verwenden, um zum Beispiel die Höhe zu ermitteln und somit falsche Messungen zu erkennen.

Bei Verwendung eines Laptops (Intel i7 Prozessor, 8 GB Arbeitsspeicher) mit GUI, laufendem FPV und eingeschalteter Regelung ist es möglich, eine Verarbeitungsrate für die Bilder der Tello EDU von bis zu 14 Hz zu erzielen. Diese Rate kann noch minimal gesteigert werden, wenn durch a priori-Wissen über die Marker die Detektorparameter für den in OpenCV implementierten Detektor angepasst werden. Allerdings ist der Anstieg der Rate nur gering und die Detektionsqualität wird dadurch teilweise erheblich schlechter. Aus diesem Grund werden die Standardparameter verwendet. Lediglich für das Verfeinern der Eckendetektion wird das Verfahren aus der Veröffentlichung [WO16] verwendet, weil es zu einer deutlich robusteren Positionsschätzung führt.

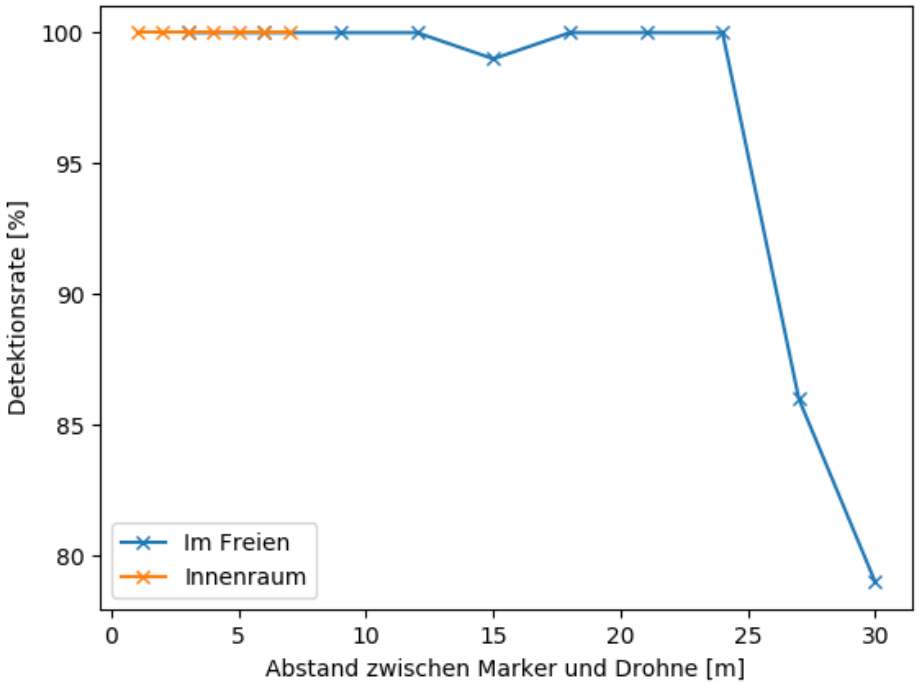


Abbildung 7.1.: Detektionsraten bei unterschiedlichen Entfernungen

Abbildung 7.2.: Beispiel für Beleuchtungsszenario

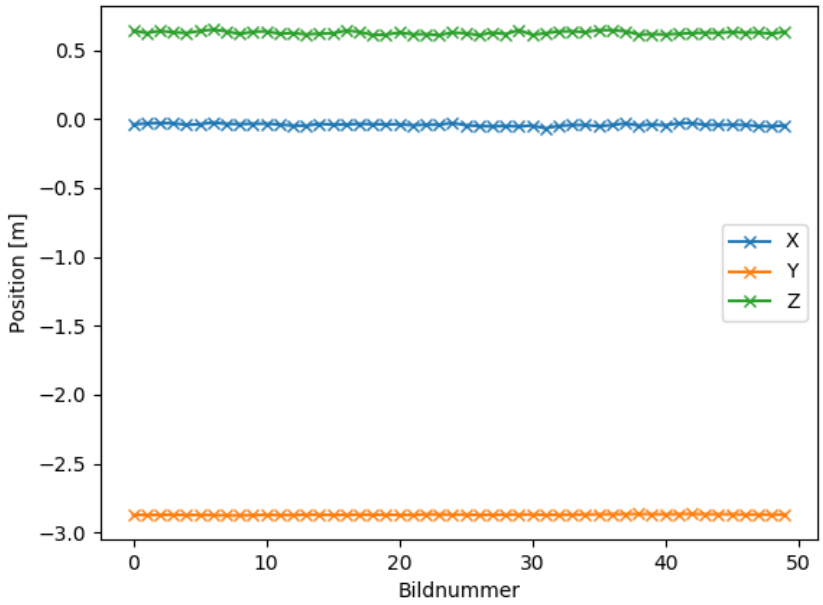


Abbildung 7.3.: Positionsschätzung in 3 Meter Entfernung

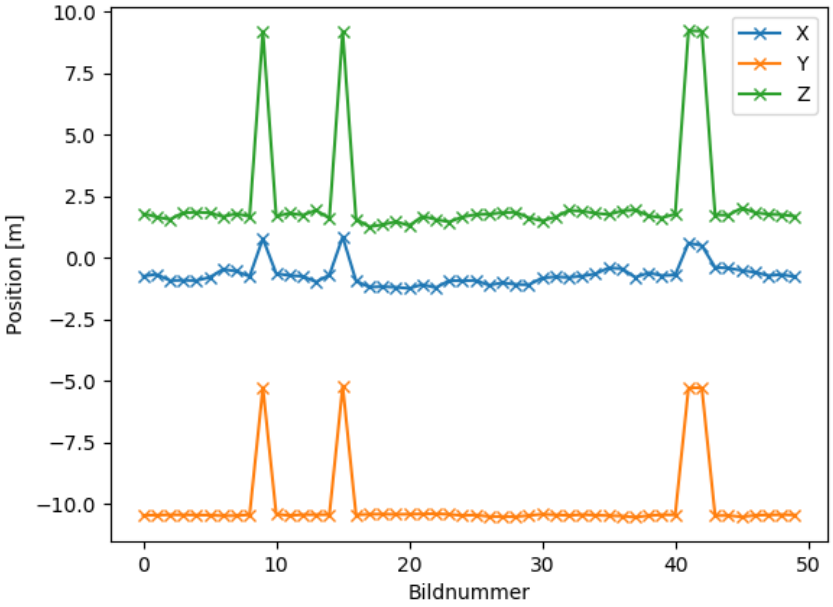


Abbildung 7.4.: Positionsschätzung in 11 Meter Entfernung

7.2. Bewertung

In diesem Abschnitt wird die Umsetzung anhand der in Abschnitt 4 getroffenen Anforderungen überprüft.
An der Tello EDU ist kein zusätzliches Gewicht entstanden. Auch für alle anderen UAV, die eine FPV-Kamera besitzen, ist es nicht notwendig, ein zusätzliches Gewicht zu transportieren. Für eigens konstruierte UAVs ist es relativ einfach, eine kleine Kamera zu befestigen. Somit kann die Anforderung an das Gewicht als erfüllt betrachtet werden. Die Mobilität gilt auch als erfüllt, weil sich die ArUco-Marker beliebig platzieren lassen und sowohl links und rechts als auch vor oder hinter einer Landebahn stehen können. Außerdem sind die Hartschaumplatten relativ leicht und mit der maximalen Größe von 50 x 50 cm sehr mobil.
Die Zuverlässigkeit des Systems ist in soweit erfüllt, als dass während des gesamten Landeanflugs bei korrekt parametrierten Markern eine Positionsbestimmung möglich ist. Außerdem wird der Marker während des Anflugs auf einer Trajektorie geregelt und kann somit auch leichte Störeinflüsse korrigieren. Ein größerer Windstoß, der das UAV weit vom Gleitpfad entfernt, führt zu einem Abbruch der Landung. Dies ist ab einer bestimmten Stärke der Störung auch wünschenswert, weil bei einem starken Abweichen vom Gleitpfad ein erneuter Landeanflug sicherer ist, als zu versuchen, den Landeanflug aus einem anderen Winkel zu beenden. Eine Platzrunde zu fliegen und einen neuen Anflug zu starten, ist alleine mit dem Marker System allerdings nicht möglich.
Die Skalierbarkeit gilt ebenfalls als erfüllt, weil mit einer Ausrichtung der Marker unterschiedliche Winkel angeflogen werden können. Diese müssen lediglich in der Steuerung für das jeweilige UAV hinterlegt sein. Die Kosten des Gesamtsystems belaufen sich exklusive der Drohne auf ca. 80 Euro für das Bedrucken der Marker und das Material für die Staffelei. Damit ist der Rahmen des Budgets eingehalten worden. Bezüglich des Energieverbrauchs kann keine Verkürzung der Flugzeit bei der Tello EDU ausgemacht werden, weil die Kamera Bestandteil der Drohne ist. Für eigens konstruierte UAVs muss die Analyse gesondert erfolgen.
Die Tests des Systems haben gezeigt, dass es selbst bei ungünstigen Sonneneinstrahlungen möglich ist, die Marker zuverlässig zu detektieren. Die Tello EDU verfügt über eine aktive Helligkeitsregelung, was einen großen Vorteil bei der Bildaufnahme bedeutet. Kameramodelle ohne eine Helligkeitsregelung müssen für verschiedene Beleuchtungen durch eine solche Implementierung erweitert werden.
In der Tabelle 7.1 sind die Ergebnisse zusammengetragen.

Tabelle 7.1.: Anforderungen an das Landesystem

Priorität	Kriterium	Ergebnis
Muss	Gewicht	erfüllt
Muss	Mobilität	erfüllt
Muss	Zuverlässigkeit	bedingt erfüllt
Muss	Skalierbarkeit	erfüllt
Muss	Kosten	erfüllt
Kann	Energieverbrauch	für Tello EDU erfüllt
Kann	Robustheit	erfüllt

8. Fazit und Ausblick

Diese Arbeit hat gezeigt, dass es möglich ist, eine Positionsbestimmung mit Hilfe einer FPV-Kamera zu erreichen, die ausreichend ist, um das UAV auf einem vorgegebenen Gleitweg zu regeln. Der entwickelte Demonstrator ist in der Lage, die Tello EDU auf einem vorher definierten Pfad zu landen. Die aktuelle Implementierung kann ohne weitere Modifikationen in Innenräumen verwendet werden um UAVs zu landen.
Bei einer Verwendung im Freien hat sich gezeigt, dass die Detektion der Marker auch in großen Entfernungen noch zuverlässig funktioniert. Allerdings ist die Positionsbestimmung in großen Entfernungen nicht mehr so robust, als dass ein kompletter Landeanflug ausschließlich anhand der Kamerabilder erfolgen kann. Für die Verwendung im Freien bietet sich daher die Einbindung von GPS-Signalen an. GPS-Signale können auf dem gesamten Flug eine relativ genaue Positionsbestimmung liefern. Diese ist für den Endanflug nicht ausreichend genau, aber sie kann zu Beginn des Landeanflugs dazu verwendet werden, um einen Beobachter, wie er in dieser Arbeit verwendet wird, zu initialisieren, bevor die ersten validen Marker-Detektion erfolgen. Im Verlauf der Landung kann die Positionsbestimmung sowohl anhand der GPS-Daten als auch der Marker-Detektion erfolgen. Je näher das UAV der Landebahn kommt, umso besser wird die Bestimmung der Position anhand der Bilddaten. Dies sollte bei einer Fusion der beiden Signale durch eine Anpassung des Messrauschens im Kalman-Filter berücksichtigt werden. Zu Beginn des Anflugs wird den GPS-Signalen mehr vertraut als der Markerdetektion. Während des Anflugs wird dies schrittweise umgekehrt, sodass der Endanflug und das Aufsetzen hauptsächlich durch die Positionsbestimmung mit den Markern erfolgt.
Durch die Verwendung von GPS-Signalen ist es außerdem möglich eine Position zu erhalten, bevor die erste Marker-Detektion erfolgt ist. Anhand dieser Daten kann berechnet werden, wo im Kamerabild die Marker zu erwarten sind. Durch diese Eingrenzung ist es vor allem bei hochauflösenden Kameras möglich, die Detektion der Marker zu beschleunigen.
Durch die Übertragung der Bilddaten über eine WLAN-Verbindung gibt es eine relativ große Verzögerung bis die Position bestimmt ist. Dies hat einen großen Einfluss auf die Dynamik des Regelkreises und somit auf das Ausregeln von Störungen und auf die Geschwindigkeit, mit der die Trajektorie abgeflogen werden kann. Für andere Entwicklungen bietet es sich daher an die komplette Regelung und Marker-Detektion direkt an Board des UAV durchzuführen, sofern es möglich ist.
Bei größeren Störungen kann es auch notwendig sein, dass die Trajektorie während des Lan-

deanflugs immer wieder neu berechnet wird oder dass statt einer Trajektorien-Folgeregelung ein anderes Verfahren wie zum Beispiel eine modellprädiktive Regelung verwendet wird. Diese Arbeit bietet somit ein grundlegendes Prinzip, mit dem es möglich ist verschiedene UAVs sicher zu landen.

Literaturverzeichnis

[AG19] AG, Basler: *ace GigE User Manual*. https://www.baslerweb.com/de/vertrieb-support/downloads/downloads-dokumente/basler-ace-gige-users-manual/, 2019. – Accessed: 10.07.2019

[Air17] AIRPLANES, Boeing C.: Statistical summary of commercial jet airplane accidents worldwide operations. In: *Boeing, Seattle* (2017)

[AK12] ANITHA, G ; KUMAR, RN G.: Vision based autonomous landing of an unmanned aerial vehicle. In: *Procedia Engineering* 38 (2012), S. 2250–2256

[BD13] BI, Yingcai ; DUAN, Haibin: Implementation of autonomous visual tracking and landing for a low-cost quadrotor. In: *Optik-International Journal for Light and Electron Optics* 124 (2013), Nr. 18, S. 3296–3300

[BLF16] BEYERER, Jürgen ; LEÓN, Fernando P. ; FRESE, Christian: *Automatische Sichtprüfung: Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung*. Springer-Verlag, 2016

[CCE08] CHOI, Ji-wung ; CURRY, Renwick ; ELKAIM, Gabriel: Path planning based on bézier curve for autonomous ground vehicles. In: *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008* IEEE, 2008, S. 158–166

[DBH17] DEGOL, Joseph ; BRETL, Timothy ; HOIEM, Derek: ChromaTag: a colored marker and fast detection algorithm. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2017, S. 1472–1481

[DLLO91] DE LUCA, Alessandro ; LANARI, Leonardo ; ORIOLO, Giuseppe: A sensitivity approach to optimal spline robot trajectories. In: *Automatica* 27 (1991), Nr. 3, S. 535–539

[Fin18] FINTEL, Rene v.: *Modern CMOS Cameras as Replacement for CCD Cameras*. Basler AG, 2018

[fre19] Bundesnetzargentur: *Frequenzplan gemäss Telekommunikationsgesetz*. https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Frequenzplan.pdf?__blob=publicationFile&v=8, 2019

[G+16] GROUP, EMVA 1288 W. u. a.: *EMVA Standard 1288–Standard for Characterization of Image Sensors and Cameras, Release 3.1*. 2016

[GHTC03] GAO, Xiao-Shan ; HOU, Xiao-Rong ; TANG, Jianliang ; CHENG, Hang-Fei: Complete solution classification for the perspective-three-point problem. In: *IEEE transactions on pattern analysis and machine intelligence* 25 (2003), Nr. 8, S. 930–943

[GPK09] GREST, Daniel ; PETERSEN, Thomas ; KRÜGER, Volker: A comparison of iterative 2d-3d pose estimation methods for real-time applications. In: *Scandinavian Conference on Image Analysis* Springer, 2009, S. 706–715

[GSS17] GAUTAM, Alvika ; SUJIT, PB ; SARIPALLI, Srikanth: Autonomous Quadrotor Landing Using Vision and Pursuit Guidance. In: *IFAC-PapersOnLine* 50 (2017), Nr. 1, S. 10501–10506

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

[HW08] HIRAKAWA, Keigo ; WOLFE, Patrick J.: Spatio-spectral color filter array design for optimal image recovery. In: *IEEE Transactions on Image Processing* 17 (2008), Nr. 10, S. 1876–1890

[LCC16] LÓPEZ-CERÓN, Alberto ; CANAS, José M: Accuracy analysis of marker-based 3D visual localization. In: *XXXVII Jornadas de Automatica Workshop*, 2016

[LDMC+16] LA DELFA, Gaetano C. ; MONTELEONE, Salvatore ; CATANIA, Vincenzo ; DE PAZ, Juan F. ; BAJO, Javier: Performance analysis of visualmarkers for indoor navigation systems. In: *Frontiers of Information Technology & Electronic Engineering* 17 (2016), Nr. 8, S. 730–740

[LSY+14] LEE, Min-Fan R. ; SU, Shun-Feng ; YEAH, Jie-Wei E. ; HUANG, Husan-Ming ; CHEN, Jonathan: Autonomous landing system for aerial mobile robot cooperation. In: *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)* IEEE, 2014, S. 1306–1311

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

[luf19] Bundesministerium der Justiz und für Verbraucherschutz: *Luftverkehrs-Ordnung (LuftVO)*. https://www.gesetze-im-internet.de/luftvo_2015/__21.html, 2019

[MD17] MARCHTHALER, Reiner ; DINGLER, Sebastian: *Kalman-Filter*. Bd. 30. Springer, 2017

[MDAM10] MAIDI, Madjid ; DIDIER, Jean-Yves ; ABABSA, Fakhreddine ; MALLEM, Malik: A performance study for camera pose estimation using visual marker based tracking. In: *Machine Vision and Applications* 21 (2010), Nr. 3, S. 365–376

[Men13] MENSEN, Heinrich: *Handbuch der Luftfahrt*. Springer-Verlag, 2013

[Men14] MENSEN, Heinrich: *Moderne Flugsicherung: Organisation, Verfahren, Technik*. Springer-Verlag, 2014

[NF02] NAIMARK, Leonid ; FOXLIN, Eric: Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In: *Proceedings of the 1st International Symposium on Mixed and Augmented Reality* IEEE Computer Society, 2002, S. 27

[PBP13] PRAUTZSCH, Hartmut ; BOEHM, Wolfgang ; PALUSZNY, Marco: *Bézier and B-spline techniques*. Springer Science & Business Media, 2013

[QL99] QUAN, Long ; LAN, Zhongdan: Linear n-point camera pose determination. In: *IEEE Transactions on pattern analysis and machine intelligence* 21 (1999), Nr. 8, S. 774–780

[RRMSMC18] ROMERO-RAMIREZ, Francisco J. ; MUÑOZ-SALINAS, Rafael ; MEDINA-CARNICER, Rafael: Speeded up detection of squared fiducial markers. In: *Image and vision Computing* 76 (2018), S. 38–47

[Rue19] RUETHER, Jan: Entwicklung eines Systems zur Kartografie von Innenräumen mittels 2D-Bilddaten eines Quadrocopters. (2019)

[SJ76] SMITH, AJ ; JOHNSON, D: The Precision Approach Path Indicator-PAPI. / ROYAL AIRCRAFT ESTABLISHMENT FARNBOROUGH (ENGLAND). 1976. – Forschungsbericht

[SK17] SANI, Mohammad F. ; KARIMIAN, Ghader: Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors. In: *2017 International Conference on Computer and Drone Applications (ICon-DA)* IEEE, 2017, S. 102–107

[SP06] SCHWEIGHOFER, Gerald ; PINZ, Axel: Robust pose estimation from a planar target. In: *IEEE transactions on pattern analysis and machine intelligence* 28 (2006), Nr. 12, S. 2024–2030

[SP10] SUTHEEBANJARD, Phaisarn ; PREMCHAISWADI, Wichian: QR-code generator. In: *2010 Eighth International Conference on ICT and Knowledge Engineering* IEEE, 2010, S. 89–92

[Tel] *Teschnische Daten der Tello EDU*. https://www.ryzerobotics.com/de/tello/specs, . – Accessed: 2019-10-01

[TH17] TØRDAL, Sondre S. ; HOVLAND, Geir: Relative vessel motion tracking using sensor fusion, aruco markers, and mru sensors. (2017)

[Web19] WEBER, Wolfgang: *Industrieroboter: Methoden der Steuerung und Regelung*. Carl Hanser Verlag GmbH Co KG, 2019

[Wer11] WERLING, Moritz: *Ein neues Konzept für die Trajektoriengenerierung und-stabilisierung in zeitkritischen Verkehrsszenarien*. Bd. 34. KIT Scientific Publishing, 2011

[WH90] WANG, C-H ; HORNG, J-G: Constrained minimum-time path planning for robot manipulators via virtual knots of the cubic B-spline functions. In: *IEEE transactions on automatic control* 35 (1990), Nr. 5, S. 573–577

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

[Zha00] ZHANG, Zhengyou: A flexible new technique for camera calibration. In: *IEEE Transactions on pattern analysis and machine intelligence* 22 (2000)

[ZQQ18] ZHENGLONG, Guo ; QIANG, Fu ; QUAN, Quan: Pose Estimation for Multicopters Based on Monocular Vision and AprilTag. In: *2018 37th Chinese Control Conference (CCC)* IEEE, 2018, S. 4717–4722

A. Anhang

Der Anhang befindet sich auf einer CD und ist bei Prof. Dr.-Ing. Marc Hensel oder Prof. Dr. Heike Neumann einzusehen. Auf der CD befindet sich:

Beschreibung	Speicherort
-Die Arbeit als PDF	Thesis.pdf
-Der verwendete Code	Code
-Die Kalibrierung der Tello EDU und der DJI Spark	Code/config
-Videos vom Marker	videos
-Bilder der Tello-Kalibrierung	videos/Tello_Calib
-Die verwendete Literatur	literatur

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, 20. Oktober 2019	
Ort, Datum	Unterschrift