Hochschule Fulda

Fachbereich Angewandte Informatik

Bachelorarbeit

Deep-Learning-gestützte Ansätze für visionbasierte Umgebungserfassung und sichere Navigation

Konzeption und Implementierung eines Drohnen-Prototyps

vorgelegt von

Veronika Herold

Matr.-Nr.: 1333017

07.04.2026

Erstprüfer: Prof. Dr. Jan-Torsten Milde

Zweitprüfer: Prof. Dr.Christian Baun

Kurzfassung

Die vorliegende Arbeit beschreibt Konzeption, Implementierung und Evaluation eines autonomen Drohnen-Prototyps zur kamerabasierten Personenverfolgung. Das System verfolgt einen hybriden Verarbeitungsansatz: Eine echtzeitfähige Edge-Pipeline auf einem Raspberry Pi 5 mit Hailo-8L NPU führt eine YOLO-basierte Objekterkennung in Echtzeit durch, während eine nachgelagerte Offline-Analyse mittels Depth Anything 3 metrische 3D-Tiefenrekonstruktionen auf einer Consumer-GPU erzeugt. Dieser Ansatz löst den Zielkonflikt zwischen Echtzeitfähigkeit und Analysetiefe auf kostengünstiger, quelloffener Hardware. Die Gesamtkosten des Prototyps liegen bei ca. 1 100 €. Die bodengebundene Evaluation belegt die Funktionalität aller Software-Komponenten; Flugtests konnten aufgrund einer nicht abgeschlossenen ESC-Kalibrierung nicht durchgeführt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AI Artificial Intelligence

BBox Bounding Box

BEC Battery Eliminator Circuit

CNN Convolutional Neural Network

COCO Common Objects in Context

CSI Camera Serial Interface

CSV Comma-Separated Values

DA3 Depth Anything 3

DFL Distribution Focal Loss

DJI Da-Jiang Innovations Science and Technology Co., Ltd.

EKF Extended Kalman Filter

ESC Electronic Speed Controller

FC Flight Controller

FP32 32-Bit Floating Point

FPS Frames per Second

GLB GL Transmission Format Binary

GPU Graphics Processing Unit

GPS Global Positioning System

HAL Hardware Abstraction Layer

HEF Hailo Executable Format

IMU Inertial Measurement Unit

INT8 8-Bit Integer Quantisierung

LiDAR Light Detection and Ranging

mAP mean Average Precision

MAVLink Micro Air Vehicle Link

MKV Matroska Video Container

NCNN Tencent NCNN Inference Framework

NMS Non-Maximum Suppression

NPU Neural Processing Unit

ONNX Open Neural Network Exchange

PCIe Peripheral Component Interconnect Express

PID Proportional-Integral-Derivative

PTQ Post-Training Quantization

ROS Robot Operating System

RPi Raspberry Pi

RTL Return to Launch

TOPS Tera Operations Per Second

UART Universal Asynchronous Receiver-Transmitter

UAV Unmanned Aerial Vehicle

YOLO You Only Look Once

1 Einleitung

1.1 Motivation und Problemstellung

Unbemannte Flugsysteme haben sich von spezialisierten Forschungsplattformen zu vielseitigen Werkzeugen in Industrie, Landwirtschaft und öffentlicher Sicherheit entwickelt [56]. Diesen Anwendungen gemein ist die zunehmende Forderung nach Autonomie: Die Systeme sollen eigenständig auf ihre Umgebung reagieren [31].
Eine Schlüsseltechnologie hierfür ist die visionbasierte Umgebungserfassung mittels Deep Learning. Modelle der YOLO-Familie [43] ermöglichen eine echtzeitfähige Objekterkennung [24] und bilden den Kern der sicherheitskritischen Live-Navigation. Ergänzend erlauben monokulare Tiefenschätzungsverfahren wie Depth Anything 3 [25] die Ableitung metrischer Distanzinformationen aus einzelnen Kamerabildern. Da ihre Ausführung auf Edge-Hardware die Echtzeitfähigkeit gefährden würde, kommt die Tiefenschätzung ausschließlich in der nachgelagerten Offline-Analyse zum Einsatz.
Die zentrale Herausforderung liegt in der Diskrepanz zwischen den Anforderungen moderner Deep-Learning-Modelle und den verfügbaren Rechenressourcen auf einer Drohne. Leistungsfähige Modelle übersteigen das thermische und energetische Budget eingebetteter Plattformen [8], während eine Cloud-Auslagerung inakzeptable Latenzen und Netzwerkabhängigkeit mit sich bringt [44, S. 7–9]. Dieses Spannungsfeld zwischen Modellqualität und Echtzeitfähigkeit bildet das Kernproblem der vorliegenden Arbeit. Verschärft wird es durch Motorvibrationen, wechselnde Lichtverhältnisse und das limitierte Gewichtsbudget kompakter Drohnenrahmen [31].
Kommerzielle Lösungen wie DJI basieren primär auf GPS-Tracking ohne visionbasierte Hindernisvermeidung. Skydio nutzt zwar kamerabasierte Navigation, setzt jedoch sechs Kameras und proprietäre Hardware ein, die für Forschungszwecke weder zugänglich noch reproduzierbar ist [6]. An kostengünstigen, quelloffenen Plattformen mit Deep-Learning-basierter Umgebungserfassung auf frei verfügbarer Edge-Hardware besteht ein deutlicher Mangel. Die vorliegende Arbeit adressiert diese Lücke mit einem Prototyp, der auf einem Raspberry Pi 5 mit Hailo-8L NPU eine echtzeitfähige Vision-Pipeline implementiert und parallel eine synchronisierte Datenaufzeichnung für die nachgelagerte wissenschaftliche Analyse bereitstellt.

1.2 Zielsetzung der Arbeit

Das übergeordnete Ziel dieser Arbeit ist die Konzeption, Implementierung und Evaluation eines Drohnen-Prototyps, der mittels Deep-Learning-basierter Bildverarbeitung eine autonome Umgebungserfassung und sichere Navigation in einem Campus-Szenario realisiert.
Dazu verfolgt die Arbeit einen hybriden Verarbeitungsansatz. In der *Live-Edge-Verarbeitung* wird ein kompaktes YOLO-Modell auf einem Raspberry Pi 5 mit Hailo-8L NPU ausgeführt, das Hindernisse

und Personen in Echtzeit erkennt. Parallel zeichnet ein synchronisierter Data Logger Video und Telemetrie auf. In der *Offline-Analyse* werden die Rohdaten nach der Mission auf einer Bodenstation mit rechenintensiveren Modellen wie Depth Anything 3 verarbeitet, die metrische Tiefenkarten und 3D-Rekonstruktionen liefern.
Dieser Hybrid-Ansatz umgeht den Kompromiss zwischen Modellkomplexität und Inferenzgeschwindigkeit auf Edge-Geräten: Sicherheitskritische Funktionen laufen in Echtzeit, die reproduzierbare Analyse findet zeitlich entkoppelt auf leistungsstärkerer Hardware statt. Der Prototyp ist als Open-Source-Plattform auf Basis kostengünstiger Hardware konzipiert.
Konkret umfasst die Arbeit folgende Teilziele:

- Aufbau eines flugfähigen Drohnen-Systems mit integrierter Kamera und Companion Computer auf Basis eines Raspberry Pi 5.
- Implementierung einer echtzeitfähigen Vision-Pipeline mit YOLO-basierter Objekterkennung auf dem Hailo-8L NPU.
- Realisierung einer Follow-Me-Navigation, bei der die Drohne einer erkannten Person autonom folgt und bei Hindernissen selbstständig stoppt.
- Entwicklung einer synchronisierten Datenaufzeichnung für die nachgelagerte Offline-Auswertung mit höherwertigen Modellen.
- Quantitative Evaluation der Detektionsgenauigkeit, Latenz und Echtzeitfähigkeit unter realen Einsatzbedingungen auf dem Campus.

1.3 Forschungsfragen

Aus der beschriebenen Problemstellung leiten sich vier zentrale Forschungsfragen ab, die diese Arbeit systematisch untersucht:

F1: *Inwiefern ermöglicht eine hybride Architektur aus echtzeitfähiger Edge-Inferenz und nachgelagerter Offline-Analyse die Umsetzung einer autonomen Vision-Pipeline auf kostengünstiger, frei verfügbarer Hardware?*

F2: *Welche Leistungsfähigkeit erreicht die YOLO-basierte Objekterkennung auf einem Hailo-8L Neural Processing Unit im Vergleich zu einer reinen CPU-Ausführung auf dem Raspberry Pi 5?*

F3: *Welche Qualität der 3D-Tiefenrekonstruktion lässt sich mit Depth Anything 3 im Multi-View-Modus auf einer Consumer-GPU mit begrenztem Videospeicher erzielen?*

F4: *Welche praktischen Herausforderungen ergeben sich beim Aufbau eines KI-gestützten Drohnen-Prototyps mit integriertem Companion Computer, und welche Auswirkungen haben diese auf den Evaluationsumfang?*

Die Beantwortung erfolgt schrittweise über die nachfolgenden Kapitel; eine zusammenfassende Diskussion findet sich in Kapitel 6.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in sieben Kapitel, die den Entwicklungsprozess von den theoretischen Grundlagen über die Implementierung bis zur Evaluation nachzeichnen.
Kapitel 2 behandelt die Architektur autonomer Drohnensysteme, das MAVLink-Protokoll, die YOLO-Objekterkennung, monokulare Tiefenschätzung sowie Edge-AI-Hardware und verwandte Arbeiten. **Kapitel 3** beschreibt die Systemanforderungen, begründet den hybriden Verarbeitungsansatz und erläutert Latenzkette, Zeitsynchronisation sowie die modulare Softwarearchitektur. **Kapitel 4** dokumentiert den Hardware-Aufbau, die Vision-Pipeline, die Follow-Me-Navigation und den synchronisierten Data Logger. **Kapitel 5** evaluiert die Inferenzleistung, die Offline-Tiefenrekonstruktion und die Navigations- und Sicherheitslogik anhand bodengebundener Tests. **Kapitel 6** beantwortet die Forschungsfragen, reflektiert Limitationen und gibt einen Ausblick. **Kapitel 7** fasst die Erkenntnisse zusammen.

2 Grundlagen und Stand der Technik

2.1 Autonome Drohnen-Systeme

Der Begriff des autonomen Drohnen-Systems beschreibt die Integration von Flugplattform, Sensorik und Recheneinheiten zu einem Gesamtsystem, das Missionen mit minimalem menschlichem Eingriff durchführen kann [6].

2.1.1 Flight Controller Architekturen

Die grundlegende Steuerung eines autonomen Fluggeräts wird durch den Flight Controller realisiert, der als zentrales Nervensystem der Drohne fungiert. Eine prominente und in der Forschung verbreitete Lösung stellt die Open-Source-Firmware ArduPilot dar [3]. Ähnlich der zentralen Steuerung in Forschungsrobotern [7, S. 1–3] verarbeitet der Flight Controller Sensordaten und steuert die Aktuatoren, um einen stabilen Flugzustand zu gewährleisten. Die offene Architektur macht ArduPilot zu einer idealen Plattform für die Entwicklung neuer Algorithmen, insbesondere im Bereich der autonomen Navigation und der Integration von Deep-Learning-Modellen.
Ein wesentliches Merkmal der ArduPilot-Architektur ist die Hardware Abstraction Layer (HAL). Diese Abstraktionsschicht entkoppelt die Flugsteuerungslogik von der spezifischen Hardware und ermöglicht die Ausführung auf verschiedenen Prozessorarchitekturen, etwa der verbreiteten STM32-Familie. Dieser modulare Ansatz ähnelt Simulationsplattformen, bei denen unterschiedliche Komponenten flexibel integriert werden können [18, S. 1–3]. Prototypen lassen sich so auf unterschiedlicher Hardware entwickeln, ohne den Kern der Steuerungssoftware grundlegend anzupassen.
Aufbauend auf der HAL gliedert sich die Firmware in eine klar strukturierte Schichtenarchitektur [3]. Low-Level-Aufgaben wie Motorsteuerung und Stabilisierung sind strikt von High-Level-Prozessen wie Missionsplanung und autonomer Navigation getrennt. Diese hierarchische Aufteilung, vergleichbar mit der Trennung von Entscheidung und Ausführung in der Robotik [55, S. 4–6], ermöglicht es, komplexe autonome Verhaltensweisen zu implementieren, ohne sich mit den Details der Flugdynamik befassen zu müssen.
Die flexible Architektur von ArduPilot bildet somit eine robuste Grundlage für KI-basierte Drohnen-Systeme. Deep-Learning-Algorithmen zur visuellen Umgebungserfassung docken auf der High-Level-Ebene an und treffen Navigationsentscheidungen, die von den Low-Level-Routinen in präzise Steuerbefehle umgesetzt werden. Diese Konstellation spezialisierter autonomer Agenten ist charakteristisch für moderne hybride Systeme [55, S. 1–3].

2.1.2 Das MAVLink-Protokoll

Eine grundlegende Voraussetzung für die Kommunikation innerhalb eines autonomen Flugsystems ist ein robustes Protokoll. Das Micro Air Vehicle Link (MAVLink) hat sich als De-facto-Standard in der unbemannten Luftfahrt etabliert [21]. MAVLink ist ein leichtgewichtiges, binäres Protokoll, das nach einem Publish-Subscribe-Modell arbeitet: Der Flugcontroller sendet kontinuierlich Telemetriedaten, die Companion Computer oder Bodenstationen abonnieren können. Dieser Ansatz minimiert den Kommunikationsaufwand und ermöglicht eine echtzeitnahe Überwachung und Steuerung.
Die Struktur einer MAVLink-Nachricht ist auf Effizienz ausgelegt [29]. Jedes Paket besteht aus Header (mit Nachrichten-ID, System- und Komponenten-ID), Payload (Nutzdaten) und einer 16-Bit-Prüfsumme (CRC16) zur Verifizierung der Datenintegrität.
Ein zentraler Mechanismus ist die Heartbeat-Nachricht, die jede aktive Komponente mit 1 Hz sendet [29]. Sie enthält den aktuellen Flugmodus, den Systemtyp und den Systemzustand. Empfängt eine Komponente über einen definierten Zeitraum keinen Heartbeat, wird von einem Verbindungsabbruch ausgegangen und Sicherheitsmaßnahmen wie ein Failsafe-Modus können eingeleitet werden.
Der Autopilot sendet Telemetrienachrichten, die den Flugzustand in Echtzeit abbilden. Die wichtigsten sind `VFR_HUD` (Fluggeschwindigkeit, Höhe, Kurs, Steigrate) und `GLOBAL_POSITION_INT` (geografische Position und Geschwindigkeit über Grund). Diese Datenströme dienen sowohl der Bodenstation als auch dem Companion Computer für kontextbezogene Entscheidungen.
MAVLink ermöglicht ebenso die aktive Steuerung des Fluggeräts. Die Nachricht `COMMAND_LONG` [29] kann bis zu sieben Parameter übermitteln und deckt Aktionen von Starten (`MAV_CMD_NAV_TAKEOFF`) und Landen bis zum Wechsel des Flugmodus ab. Diese Kommandostruktur bildet die Schnittstelle, über die externe Systeme wie ein Companion Computer aktiv in die Flugsteuerung eingreifen.
Für autonome Missionen spielt MAVLink eine entscheidende Rolle als Brücke zwischen Companion Computer und Flight Controller. Der Raspberry Pi übersetzt die Ergebnisse der visionbasierten Umgebungserfassung in MAVLink-Befehle und sendet diese über UART an den FC. Über `MAV_CMD_SET_MESSAGE_INTERVAL` kann ein Companion Computer die Sendefrequenz einzelner Nachrichtentypen gezielt einstellen [29], etwa `ATTITUDE` auf 50 Hz und GPS-Nachrichten auf 5–10 Hz.

2.1.3 Autonome Flugmodi und State Machines

Die Funktionalität der ArduPilot-Firmware basiert auf einem System unterschiedlicher Flugmodi, die von manueller Steuerung bis zu hochautomatisierten Aktionen reichen [3].
Der Modus `LOITER` nutzt GPS und Barometer, um die Drohne an einer definierten Position und Höhe zu halten. Abweichungen durch Wind werden automatisch korrigiert [55, S. 1–3]. `LOITER` dient als stabiler Ausgangspunkt für Missionen oder als sicherer Wartezustand.
`RTL` (Return-to-Launch) wird bei Fehlerzuständen wie Funkverlust oder kritischem Akkustand automatisch aktiviert. Das System steigt auf eine sichere Höhe, navigiert zum Startpunkt und landet [52, S. 13–15].

Für die visionbasierte Navigation dieser Arbeit ist der Modus GUIDED zentral. Er erlaubt dem Companion Computer, über MAVLink abstrakte Zielvorgaben wie Positions- oder Geschwindigkeits-Setpoints an den Flight Controller zu senden [7, S. 1–3]. Der Companion Computer analysiert die Kameradaten und generiert Navigationsziele, während der Flight Controller die Ausführung und Stabilisierung übernimmt [55, S. 4–6]. Diese Trennung ermöglicht die Ausführung rechenintensiver KI-Logik, ohne die echtzeitkritischen Regelungsaufgaben des Autopiloten zu beeinträchtigen.
Die operationale Sicherheit gewährleistet eine interne State Machine, die Zustandsüberwachung und Modi-Übergänge verwaltet [52, S. 13–15]. Bei einem Ausfall des Companion Computers erzwingt sie einen Wechsel vom GUIDED-Modus in einen sicheren Failsafe-Modus wie RTL.

2.2 Computer Vision für Drohnen

2.2.1 Objekterkennung mit YOLO

Die Evolution der YOLO-Architekturen (*You Only Look Once*) markiert einen entscheidenden Fortschritt in der Echtzeit-Objekterkennung. Als *Single-Stage-Detektor* analysiert YOLO ein Bild in einem einzigen Durchlauf, was eine signifikant höhere Inferenzgeschwindigkeit im Vergleich zu zweistufigen Modellen wie R-CNN ermöglicht.
Ein fundamentaler Paradigmenwechsel war der Übergang von Anker-basierten zu Anker-freien Architekturen. Während YOLOv5 [20] noch auf vordefinierte Ankerboxen zur Vorhersage von *Bounding Boxes* setzte, etablierten Modelle ab YOLOv8 einen Anker-freien Ansatz [49]: Der Mittelpunkt eines Objekts wird direkt identifiziert und die Abstände zu den Rändern der *Bounding Box* regressiert. YOLOv8 führte zudem einen anchor-free split Ultralytics head ein, der die Effizienz des Detektionsprozesses steigerte [47]. Parallel wurden Backbone und Neck kontinuierlich weiterentwickelt — vom CSPDarknet in YOLOv5 über das C2f-Modul in YOLOv8 bis zu verfeinerten Strukturen in YOLO11 [48]. Mit YOLO26 erreichte die Entwicklung einen neuen Höhepunkt für Edge-Anwendungen [51]. Das native *End-to-End-Design* kommt ohne Non-Maximum Suppression (NMS) aus, und die Entfernung des Distribution Focal Loss (DFL) Moduls steigert die CPU-Inferenzgeschwindigkeit um bis zu 43 %. Fortschrittliche Verlustfunktionen (ProgLoss, STAL) sowie der MuSGD-Optimizer verbessern die Detektionsgenauigkeit [49].
Die YOLO-Modellfamilie bietet skalierbare Varianten von Nano (n) bis Extra-Large (x): YOLO26n ist mit 2,4 Mio. Parametern und 5,4 B FLOPs auf schnelle Inferenzzeiten optimiert, während YOLO26x mit 55,7 Mio. Parametern die mAP für rechenintensive Analysen maximiert [49].
Die praktische Relevanz zeigt sich bei Benchmarks auf dem Raspberry Pi 5: Ein nach NCNN exportiertes YOLO26n-Modell erreicht auf der CPU eine Inferenzzeit von etwa 67,69 ms pro Bild [50]. Mit einer dedizierten NPU wie dem Hailo-8L werden durch INT8-Quantisierung (vgl. Abschnitt 2.2.3) deutlich niedrigere Inferenzzeiten erreicht. Diese Differenz ist ein zentrales Argument für den in dieser Arbeit gewählten Einsatz des Hailo-8L AI HAT.

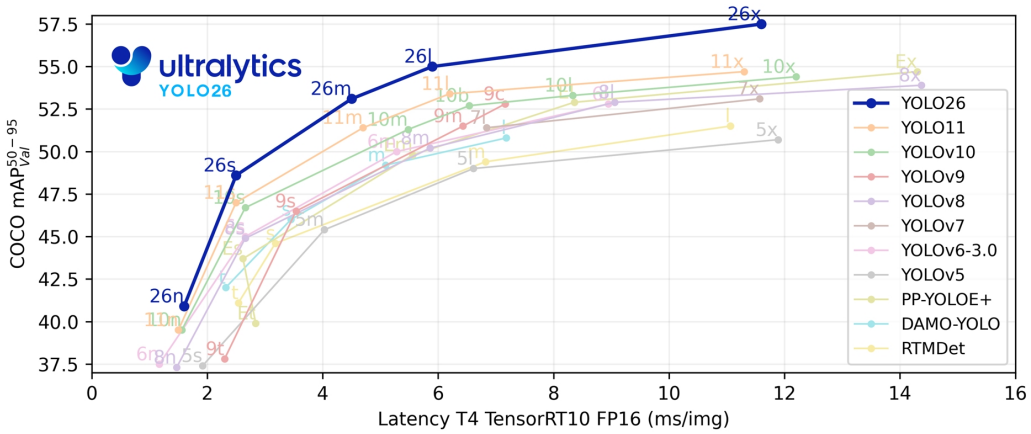


Abbildung 2.1: Vergleich der YOLO-Modellgenerationen: YOLO26 erreicht bei geringerer Latenz eine höhere Genauigkeit als seine Vorgänger [51].

Multi-Object-Tracking mit ByteTrack. Die Objekterkennung liefert pro Frame Detektionen ohne zeitliche Identität. *ByteTrack*, als in `ultralytics` integrierter Tracker, nutzt einen *Kalman-Filter* zur Positionsprädiktion und den *Hungarian Algorithm* zur Zuordnung. Die zentrale Innovation besteht darin, auch Detektionen mit niedriger Konfidenz einzubeziehen, was die Tracking-Robustheit bei Verdeckung signifikant erhöht. Jedes getrackte Objekt erhält eine persistente ID, auf die der PID-Regler im Follow-Me-Modus gelockt wird. Geht die ID verloren, wechselt das System in den `LOITER`-Modus als sicheren Wartezustand.

2.2.2 Edge- und Cloud-Inferenz im Vergleich

Die Entscheidung, wo die Inferenz stattfindet, ist eine grundlegende architektonische Weichenstellung. Die `Cloud-Inference` bietet nahezu unbegrenzte Rechenressourcen, erfordert jedoch eine stabile Netzwerkverbindung. Für UAVs erweist sich diese Abhängigkeit als gravierender Nachteil: Schwankende Latenzen durch Handover oder fehlende Infrastruktur gefährden die Flugsicherheit [22, S. 4–6]. Bei einer Fluggeschwindigkeit von $v = 5\,\text{m/s}$ und einer Cloud-Latenz von $t = 500\,\text{ms}$ legt eine Drohne

$$s = v \cdot t = 5\,\text{m/s} \cdot 0,5\,\text{s} = 2,5\,\text{m} \tag{2.1}$$

im Blindflug zurück. Bei einer Edge-Latenz von ca. 70 ms reduziert sich die Distanz auf $0,35\,\text{m}$ [44, S. 7–9].

Die `Edge-Inference` garantiert eine niedrige, deterministische Latenz und volle Einsatzfähigkeit ohne Netzabdeckung, erfordert jedoch einen Kompromiss bei der Modellkomplexität [16, S. 4–6]. Der in dieser Arbeit vorgestellte Prototyp verfolgt daher einen hybriden Ansatz (vgl. Abschnitt 3.2).

2.2.3 Modellquantisierung für Edge-Hardware

Die Implementierung von Deep-Learning-Modellen auf Edge-Geräten erfordert spezialisierte Techniken. Während des Trainings werden neuronale Netze mit 32-Bit-Gleitkommazahlen (`FP32`) berechnet. Für die Inferenz auf Edge-Hardware werden die Gewichte und Aktivierungen mittels Quantisierung in eine niedrigpräzise Darstellung konvertiert, meist 8-Bit-Integer (`INT8`). Dies verringert den Speicherbedarf um den Faktor vier und beschleunigt die Verarbeitung auf spezialisierter Hardware [19, 34].
Der Kern der Quantisierung ist eine affine Abbildung, die den Wertebereich der `FP32`-Tensoren auf den diskreten Bereich der `INT8`-Integer abbildet. Dieser Prozess wird durch einen Skalierungsfaktor S und einen Nullpunkt Z (*Zero-Point*) definiert. Die Beziehung zwischen dem reellen Wert r und seinem quantisierten Äquivalent q lässt sich wie folgt beschreiben:

$$r \approx S(q - Z) \tag{2.2}$$

Für jeden Tensor im Netzwerk werden die optimalen Werte für S und Z bestimmt, um den Quantisierungsfehler – den Informationsverlust durch die reduzierte Präzision – zu minimieren und die Genauigkeit des Modells zu erhalten.
In der Praxis wird häufig die Methode der *Post-Training Quantization* (PTQ) angewendet [32]. Hierbei wird ein bereits trainiertes `FP32`-Modell als Ausgangspunkt genommen. Um die optimalen Quantisierungsparameter zu ermitteln, wird ein Kalibrierungsdatenset verwendet, das aus einer repräsentativen Stichprobe der Realdaten besteht. Durch eine statistische Analyse der Aktivierungen während eines Test-Durchlaufs werden die Skalierungsfaktoren so kalibriert, dass der Genauigkeitsverlust minimiert wird.
Für die Bereitstellung auf der `Hailo-8L` NPU ist ein spezifischer Workflow erforderlich:

1. Export des PyTorch-Modells (`.pt`) in das `ONNX`-Format [33].
2. Nutzung des `Hailo Dataflow Compilers` (DFC) zur Durchführung der PTQ.
3. Kompilierung in das `Hailo Executable Format` (`.hef`) [34, S. 1–3].

Die Leistungsfähigkeit des `Hailo-8L` resultiert aus seiner Data-Flow-Architektur, die parallele `INT8`-Operationen massiv beschleunigt [34, S. 1–3]. Bei einer Leistungsaufnahme von lediglich $1,5$ W [17] ermöglicht dies die Ausführung komplexer KI-Modelle in Echtzeit, ohne die Flugzeit der Drohne übermäßig zu belasten.

2.2.4 Monokulare Tiefenschätzung

Die monokulare Tiefenschätzung (*Monocular Depth Estimation*) bezeichnet die Ableitung einer pixelweisen Tiefenkarte aus einem einzelnen RGB-Bild. Im Gegensatz zu Stereo-Systemen oder LiDAR-Sensoren wird lediglich eine Kamera benötigt, was diese Verfahren besonders attraktiv für gewichts-

und platzbeschränkte Plattformen wie Drohnen macht. Die zentrale Herausforderung ist dabei die inhärente Mehrdeutigkeit des Problems: Aus einer zweidimensionalen Projektion muss die dritte Dimension rekonstruiert werden, was ohne zusätzliche Informationen ein unterbestimmtes inverses Problem darstellt [38].

Von CNNs zu Transformern. Frühe Deep-Learning-Ansätze wie FastDepth [57] setzten auf CNN-Architekturen, litten jedoch unter begrenzter Generalisierung. *MiDaS* [38] erzielte durch Training auf einer Mischung von zwölf heterogenen Datensätzen erstmals eine robuste Zero-Shot-Generalisierung, indem relative Tiefenverhältnisse statt absoluter Meterwerte gelernt wurden. Der Wechsel zu Vision Transformern mit *Dense Prediction Transformers* (DPT) [37] brachte einen weiteren Qualitätssprung: Der globale Self-Attention-Mechanismus erfasst weitreichende Abhängigkeiten im Bild, was für die korrekte Tiefenzuordnung essenziell ist.

Die Depth-Anything-Familie. *Depth Anything V1* [59] bezog 62 Millionen unlabeled Bilder über ein Teacher-Student-Framework ins Training ein. *Depth Anything V2* [60] ersetzte verrauschte Realdaten durch hochpräzise synthetische Tiefenbilder und erzielte State-of-the-Art-Ergebnisse. Beide Versionen liefern jedoch ausschließlich *relative* Tiefenwerte.
Diese Limitation adressiert *Depth Anything 3* [25]: Neben einer verbesserten Trainingsmethodik mit skaleninvarianter affiner Verlustfunktion unterstützt DA3 einen *Multi-View-Modus*, in dem mehrere zeitlich aufeinanderfolgende Frames simultan verarbeitet werden. Die Multi-View-Attention-Architektur schätzt pixelgenaue Tiefenkarten und relative Kameraposen und erzeugt eine konsistente 3D-Punktwolke mit metrischer Tiefe in Metern. Für diese Arbeit wird `DA3-LARGE-1.1` (0,35 Mrd. Parameter) im Multi-View-Modus eingesetzt.

Modellgrößen und Einsatzszenarien. Depth Anything 3 ist in mehreren Varianten verfügbar, die ein Spektrum von leichtgewichtigen Edge-Modellen bis zu hochpräzisen Forschungsmodellen abdecken. Tabelle 2.1 fasst die relevanten Konfigurationen zusammen.

Tabelle 2.1: Ausgewählte Modellvarianten von Depth Anything 3 [25].

Modell	**Parameter**	**Tiefentyp**
DA3-Small	0,08 Mrd.	relativ
DA3-Base	0,12 Mrd.	relativ
DA3-Large	0,35 Mrd.	relativ
DA3-LARGE-1.1	0,35 Mrd.	metrisch (Multi-View)
DA3-Giant	1,15 Mrd.	relativ

Für die Offline-Analyse auf einer GPU-Bodenstation wird in dieser Arbeit das Modell `DA3-LARGE-1.1` im Multi-View-Modus eingesetzt, da es metrische Tiefenwerte und 3D-Rekonstruktionen bei modera-

tem Rechenaufwand liefert. Die kleineren Varianten wie `DA3-Small` wären prinzipiell für eine Edge-Ausführung auf dem Raspberry Pi geeignet, was in zukünftigen Arbeiten untersucht werden könnte.

2.3 Edge AI Hardware

Die Realisierung von Deep-Learning-Verfahren auf autonomen Systemen erfordert eine Hardware-Architektur, die eine effiziente Balance zwischen Rechenleistung, Energieverbrauch und Gewicht bietet [8]. Im Kontext mobiler Robotik hat sich hierfür das Paradigma des Edge Computings etabliert, bei dem die Datenverarbeitung lokal auf dem System erfolgt, um Latenzen durch Funkübertragungen zu minimieren und die Ausfallsicherheit zu erhöhen.

2.3.1 Raspberry Pi 5 Architektur

Als zentrale Orchestrierungseinheit wird der Raspberry Pi 5 eingesetzt [39]. Das System basiert auf dem Broadcom BCM2712 SoC mit vier ARM Cortex-A76 Kernen (2,4 GHz) und 8 GB LPDDR4X-SDRAM (17,1 GB/s Bandbreite) [2]. Die integrierte VideoCore VII GPU erreicht lediglich 120 GigaFLOPS [2], weshalb KI-Inferenz ohne externe Beschleunigung auf der CPU erfolgen muss. Benchmarks zeigen für YOLO26n im NCNN-Format eine Latenz von ca. 67,69 ms (≈15 FPS) [50], was für sichere Navigation als unzureichend eingestuft wird.
Für die visuelle Umgebungserfassung verfügt das Board über zwei MIPI CSI-2 Transceiver, die Kommunikation mit dem FC erfolgt über UART via MAVLink. Zentral ist die PCIe 2.0 Schnittstelle (im Gen-3-Modus bis 8 GT/s), die als Voraussetzung für externe KI-Beschleuniger dient.

2.3.2 NPU-Beschleunigung am Beispiel Hailo-8L

Um die Defizite der CPU-basierten Inferenz auszugleichen, wird die Hailo-8L NPU als dedizierter KI-Beschleuniger über PCIe Gen 3 integriert [17]. Im Gegensatz zu klassischen Architekturen nutzt Hailo eine strukturgetriebene Dataflow-Architektur mit integriertem Speicher, was die Bandbreite maximiert und Latenzen minimiert.
Die Hailo-8L erreicht 13 TOPS bei 1,5 W Leistungsaufnahme [17]. Pereira und Chaari zeigen, dass die NPU-Inferenz eine Beschleunigung um mehr als das 27-fache gegenüber der CPU erzielt und die CPU-Kernauslastung um 24–60 % reduziert [34]. Die Software-Integration erfolgt über den Hailo Dataflow Compiler, der ONNX-Modelle in das HEF-Format übersetzt (vgl. Abschnitt 2.2.3).

Tabelle 2.2: Vergleich verfügbarer KI-Beschleuniger für den Raspberry Pi 5.

Hardware	**TOPS**	**Leistungsaufn.**	**Schnittstelle**
Hailo-8L (AI HAT+) [17]	13	1,5 W	PCIe Gen 3
Google Coral TPU [15]	4	2,0 W	USB 3.0 / PCIe
Intel Movidius NCS2	1	1,0 W	USB 3.0

Die Hailo-8L weist mit 13 TOPS die höchste Rechenleistung unter den verfügbaren Beschleunigern auf und bietet mit der PCIe-Anbindung eine höhere Bandbreite als USB-basierte Lösungen. Die Energieeffizienz von 8,7 TOPS/W ist für batteriebetriebene UAVs entscheidend [8].

2.4 Verwandte Arbeiten

Die Zielverfolgung durch autonome Drohnen ist ein aktives Forschungsfeld. Zur Einordnung werden kommerzielle Systeme und akademische Prototypen vorgestellt.

2.4.1 Kommerzielle Follow-Me Systeme

DJI implementiert mit *ActiveTrack* ein primär GPS-gestütztes Follow-Me-System [11]. Neuere Generationen ergänzen dies um kamerabasiertes Tracking, die Algorithmen sind jedoch proprietär und nicht modifizierbar.
Skydio nutzt sechs Weitwinkelkameras zur 360°-Umgebungskarte mittels VSLAM [45]. Die Systemkosten von über 2 000 $ und die geschlossene Plattform schließen eine Nutzung als Forschungsplattform aus. Beiden Ansätzen fehlt die Möglichkeit, Detektionsmodelle oder Inferenzhardware auszutauschen.

2.4.2 Akademische Tracking-Systeme für UAVs

Konzeptionell am nächsten steht *AUTO* (Autonomous UAV that Tracks Objects) von Lo et al. [26]: eine Pipeline aus YOLOv4-Tiny, Intel RealSense Tiefenkamera und Kalman-Filter-basiertem Tracking, implementiert in ROS/C++ und unter AGPL-3.0 veröffentlicht. Tabelle 2.3 stellt die Systeme gegenüber.

Tabelle 2.3: Vergleich verwandter Arbeiten mit dem eigenen Ansatz.

Kriterium	**DJI**	**Skydio**	**AUTO**	**Diese Arbeit**
Detektion	proprietär	proprietär	YOLOv4-Tiny	YOLO26n (INT8)
Inferenzhardware	integriert	integriert	NVIDIA GPU	Hailo-8L NPU
Tiefenschätzung	GPS / Vision	6-Kamera SLAM	RealSense	BBox-Proxy + DA3
Framework	proprietär	proprietär	ROS (C++)	Python + pymavlink
Offline-Analyse	k. A.	k. A.	k. A.	DA3 auf PC
Open Source	Nein	Nein	Ja (AGPL)	Ja
Kosten	ab 800 €	ab 2,000 €	k. A.	≈ 1,100 €

Pestana et al. [35] demonstrierten bereits 2014 die Machbarkeit kameragestützter Navigation für UAVs mittels klassischer Computer-Vision (optischer Fluss, Template Matching), erreichten jedoch nicht die Robustheit moderner Deep-Learning-Detektoren. McEnroe et al. [30] liefern eine systematische Übersicht über Edge-KI auf UAVs und identifizieren Rechenleistung, Energieverbrauch und Echtzeitan-

forderungen als zentrale Herausforderungen. Li et al. [24] adressieren die Domänenlücke zwischen COCO-trainierten Modellen und der UAV-Perspektive durch eine modifizierte YOLOv8-Architektur.

2.4.3 Abgrenzung der vorliegenden Arbeit

Die vorliegende Arbeit unterscheidet sich in mehreren Aspekten:

1. **Dedicated NPU statt GPU:** Der Prototyp nutzt die Hailo-8L NPU (13 TOPS, 1,5 W) statt NVIDIA-GPUs und bietet damit ein günstigeres Verhältnis von Rechenleistung zu Energieverbrauch für batteriebetriebene UAVs.

2. **Hybride Architektur:** Keines der verglichenen Systeme trennt die Verarbeitung in echtzeitfähige Edge-Komponente und rechenintensive Offline-Analyse (DA3 auf Bodenstation).

3. **Aktualität:** Mit YOLO26n (2026) und Depth Anything 3 (2025) kommen State-of-the-Art-Modelle zum Einsatz.

4. **Reproduzierbarkeit:** Budget unter 1,000 €, ausschließlich Open-Source-Software (ArduPilot, pymavlink, Ultralytics).

3 Systemkonzept und Architektur

3.1 Systemanforderungen

3.1.1 Funktionale Anforderungen (FR)

Die funktionalen Anforderungen definieren die Verhaltensweisen des Systems und bilden das Fundament für Entwurf, Implementierung und Verifikation.

FR1: Autonome Personenverfolgung (Follow-Me). Der Prototyp muss eine Zielperson autonom detektieren und dieser mit ca. 1 m/s in 2, 5 m Abstand folgen. *Begründung:* Kernfunktionalität für explorative Szenarien.

FR2: Echtzeit-Hinderniserkennung. Das System muss Hindernisse mittels *YOLO26n* auf der Hailo-8L NPU detektieren und eine sofortige Stopp-Reaktion einleiten. *Begründung:* Kollisionsvermeidung ist bei autonomen Flügen in unstrukturierten Umgebungen unerlässlich [9].

FR3: Synchronisierte Datenaufnahme. Videostreams und Telemetriedaten (IMU, GPS, Höhe) müssen zeitsynchron aufgezeichnet werden. *Begründung:* Lückenlose Dokumentation für nachträgliche Fehlersuche und quantitative Evaluation.

FR4: Offline-Tiefenanalyse. Die aufgezeichneten Daten müssen für eine Prozessierung mittels *Depth Anything 3* auf einer Workstation geeignet sein. *Begründung:* Entlastung der On-Board-Ressourcen bei gleichzeitig hochpräziser 3D-Rekonstruktion.

FR5: Pilot-Override (Manuelle Priorität). Der Pilot muss die autonome Steuerung jederzeit über den RC-Sender übersteuern können. *Begründung:* Primärer Failsafe-Mechanismus [9].

3.1.2 Nicht-funktionale Anforderungen (NFR)

Nicht-funktionale Anforderungen beschreiben qualitative und quantitative Maßstäbe für Performanz, Latenz und Zuverlässigkeit.

NFR1: Inferenzrate der Objekterkennung. Mindestens 15 FPS für das YOLO-Modell auf der Hailo-8L NPU (INT8). *Begründung:* 15 FPS (≈ 67 ms) ist die Mindestanforderung für reaktive Hindernisvermeidung [36].

NFR2: Glass-to-Action-Latenz. Gesamtlatenz von Bilderfassung bis Motorreaktion unter 200 ms. *Begründung:* Orientiert an der menschlichen Reaktionszeit [1].

NFR3: Synchronisation der Datenströme. Maximaler Drift zwischen Video und Telemetrie: 50 ms. *Begründung:* Größerer Versatz führt zu inkonsistentem Weltmodell und Navigationsfehlern [46].

NFR4: Mehrstufige Failsafe-Kaskade. Hierarchisches Sicherheitskonzept auf den Ebenen Vision, Software, Hardware und Mensch. *Begründung:* Fehlertoleranz durch Redundanz, analog zu ISO 26262 [10]. Letzte Instanz: Pilot [52].

3.2 Hybride Architektur

3.2.1 Begründung der Hybrid-Architektur

Für die sichere Navigation einer autonomen Drohne in unstrukturierten Umgebungen ist die Minimierung der Systemlatenz von entscheidender Bedeutung. Jede Verzögerung zwischen der Erfassung eines Hindernisses durch die Kamera und der physischen Ausweichreaktion der Motoren erhöht das Kollisionsrisiko proportional zur Fluggeschwindigkeit. Diese als *Glass-to-Action*-Latenz bezeichnete Zeitspanne aggregiert die Verarbeitungszeiten aller sequentiellen Systemkomponenten [30].
Die Verarbeitungspipeline des Prototyps durchläuft fünf sequentielle Stufen:

1. **Akquise:** Bildaufnahme durch den Kamerasensor und Übertragung via CSI-Bus.
2. **Inferenz:** Objekterkennung durch das YOLO-Modell auf der Hailo-8L NPU.
3. **Navigationslogik:** Berechnung der Steuerungsentscheidung auf der CPU des Raspberry Pi 5.
4. **Kommunikation:** Serialisierung und Übertragung des MAVLink-Befehls via UART an den Flight Controller [21].
5. **Aktorik:** Verarbeitung im Flight Controller und mechanische Reaktion der Motoren.

In dieser Kette stellt die Inferenzstufe den größten beeinflussbaren Faktor dar. Während eine 2D-Objekterkennung mit YOLO26n auf dedizierter NPU-Hardware effizient ausführbar ist [34], würde die zusätzliche Integration einer monokularen Tiefenschätzung (etwa *Depth Anything 3* [25]) die Rechenlast erheblich steigern. Die parallele Ausführung zweier rechenintensiver Modelle auf einer eingebetteten Plattform gefährdet die Echtzeitfähigkeit und damit die Flugsicherheit [8].
Um diesen Konflikt zwischen Informationsgehalt und Echtzeitfähigkeit aufzulösen, wurde ein hybrider Verarbeitungsansatz gewählt (Tabelle 3.1). Für die sicherheitskritische Live-Navigation verzichtet das System auf eine explizite Tiefenschätzung und nutzt stattdessen die von YOLO gelieferten Bounding-Box-Dimensionen als Proxy für die Objektnähe: Je größer die Bounding Box im Bild, desto näher befindet sich das Hindernis. Dieser Ansatz ermöglicht konservative Ausweichmanöver ohne zusätzliche Inferenzlast. Die qualitativ hochwertige Tiefenanalyse mittels Depth Anything 3 wird zeitlich entkoppelt auf einer Workstation durchgeführt, wo die aufgezeichneten Flugdaten im Post-Processing ohne Echtzeitbeschränkungen verarbeitet werden können.
Die empirische Validierung der resultierenden Glass-to-Action-Latenz, einschließlich der Einzelbeiträge jeder Pipeline-Stufe, erfolgt in Abschnitt 5.2.1.

Tabelle 3.1: Vergleich: Vollständige Live-Verarbeitung vs. Hybrid-Ansatz

Kriterium	**Alles-Live**	**Hybrid**
Tiefenqualität (Live)	Hoch (DA3)	Keine (BBox-Proxy)
Inferenzlast (Live)	Sehr hoch	Moderat
Echtzeitfähigkeit	Gefährdet	Gewährleistet
Crash-Risiko	Erhöht	Reduziert
Reproduzierbarkeit	Eingeschränkt	Hoch (Offline)
Tiefenqualität (Offline)	k. A.	Hoch (DA3 Metric)

3.2.2 Systemübersicht

Das entwickelte System gliedert sich in zwei zeitlich entkoppelte Verarbeitungsphasen, deren Zusammenspiel in Abbildung 3.1 als Blockschaltbild dargestellt ist.

Phase 1: Live-Edge-Verarbeitung (Drohne). Die erste Phase umfasst alle Echtzeit-Prozesse während des Fluges. Die Pi Camera 3 liefert Videodaten (1080p, 30 FPS) über den CSI-Bus an den Raspberry Pi 5, wo die Hailo-8L NPU ein INT8-quantisiertes `YOLO26n`-Modell für die Objekterkennung ausführt (vgl. Abschnitt 2.2.3). ByteTrack vergibt persistente Track-IDs für die frameübergreifende Verfolgung. Aus den Tracking-Ergebnissen berechnet die Navigationslogik Steuerungsbefehle, die als MAVLink-v2-Vektoren an den Flight Controller übertragen werden [21]. Parallel zeichnet der Data Logger alle Datenströme synchronisiert auf einem USB-Datenträger auf (vgl. Abschnitt 3.2.3).

Phase 2: Offline-Analyse (Workstation). Nach Abschluss des Fluges werden die Rohdaten auf einer GPU-Workstation ohne Echtzeitanforderungen verarbeitet. Depth Anything 3 im Multi-View-Modus (`DA3-LARGE-1.1`) erzeugt metrische Tiefenkarten und 3D-Punktwolken [25], die mit den GPS-Koordinaten zu einer georeferenzierten Szenen-Analyse fusioniert werden (vgl. Abschnitt 4.5 in Kapitel 4).

3.2.3 Zeitsynchronisation

Im System existieren drei Zeitquellen: Kamera-Sensor, Flight Controller und Raspberry Pi 5. Ohne Synchronisation führt Clock-Drift zu fehlerhafter Zuordnung aufgezeichneter Daten [58].

Gemeinsamer Referenzpunkt. Beim Systemstart wird ein einheitlicher Zeitstempel `start_time` mittels `time.monotonic()` erzeugt. Alle Daten werden relativ zu diesem Referenzpunkt gespeichert [53].

Hardware-Zeitstempel der Kamera. Die Picamera2-Bibliothek liefert über `SensorTimestamp` einen Hardware-Zeitstempel in Nanosekunden, unabhängig von der Systemuhr und robust gegenüber

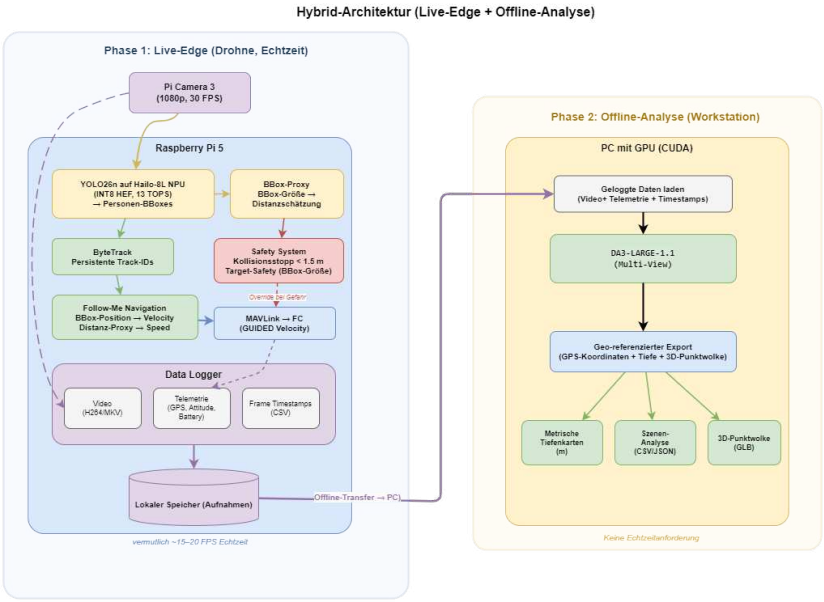


Abbildung 3.1: Blockschaltbild der Hybrid-Architektur mit Live-Edge-Verarbeitung (links) und Offline-Analyse (rechts)

Scheduling-Verzögerungen [40]. Ein beim Start berechneter Offset überführt alle Frame-Zeitstempel in die gemeinsame Zeitbasis.

Telemetrie-Zeitstempel. Jede eingehende MAVLink-Nachricht wird beim Empfang mit einem Zeitstempel relativ zu `start_time` versehen. Da die UART-Transportlatenz bei 57 600 Baud im niedrigen Millisekundenbereich liegt, ist die empfangsseitige Zeitstempelung ausreichend [29]. Für die Frame-Zuordnung wird linear zwischen den zeitlich nächstliegenden Telemetrie-Einträgen interpoliert [23].

Optischer Zeitanker. Eine Hardware-Sync-LED leuchtet beim Systemstart kurz auf und dient als unabhängiger Zeitanker zur Validierung der Software-Synchronisation in der Offline-Analyse.

3.3 Latenzanalyse der Verarbeitungskette

Wie in Abschnitt 3.2.1 dargelegt, aggregiert die *Glass-to-Action*-Latenz die Verarbeitungszeiten aller sequentiellen Stufen von der Bilderfassung bis zur motorischen Reaktion. Für das Systemdesign ist eine Abschätzung der Einzelbeiträge notwendig, um sicherzustellen, dass die Gesamtlatenz die in NFR2 geforderten 200 ms nicht überschreitet. Tabelle 3.2 zeigt die erwarteten Latenzbeiträge der fünf Pipeline-Stufen.

Tabelle 3.2: Erwartete Latenzbeiträge der Glass-to-Action Pipeline

Stufe	**Komponente**	**Latenz**	**Quelle/Begründung**
1. Akquise	Pi Camera 3 → CSI	∼33 ms	1 Frame bei 30 FPS
2. Inferenz	YOLO26n auf Hailo-8L (INT8)	∼15 ms	NPU HEF-Profil
3. Navigation	PID-Regler + Safety auf CPU	<5 ms	Arithmetik, kein I/O
4. Kommunik.	MAVLink via UART (57 600 Bd)	∼2 ms	∼50 Byte Paket
5. Aktorik	FC PID → ESC PWM	∼10 ms	400 Hz Regelrate
	Gesamt (Hybrid-Ansatz)	**∼65 ms**	

Stufe 1: Bildakquise. Pi Camera 3 Wide liefert Frames über MIPI-CSI-2 mit ca. 33 ms pro Frame bei 30 FPS (obere Schranke; Rolling-Shutter liefert erste Zeilen früher) [40].

Stufe 2: Inferenz. Das INT8-quantisierte YOLO26n-Modell wird auf der Hailo-8L NPU in ca. 15 ms ausgeführt [34]. Eine CPU-basierte Ausführung würde ca. 250 ms benötigen. Durch den Verzicht auf ein zusätzliches Tiefenmodell bleibt die Pipeline auf eine NPU-Ausführung beschränkt.

Stufe 3: Navigationslogik. PID-Regler, BBox-Distanzschätzung und Safety-Checks: unter 5 ms (reine Arithmetik).

Stufe 4: MAVLink-Kommunikation. `SET_POSITION_TARGET_LOCAL_NED` über UART (57 600 Bd): ca. 2 ms bei ∼50 Byte [21].

Stufe 5: Aktorik. ArduCopter-PID bei 400 Hz: max. 2,5 ms Wartezeit plus elektromechanische Trägheit, gesamt ca. 10 ms.

Gesamtbewertung. Die Summe ergibt ca. 65 ms und liegt damit deutlich unter den in NFR2 geforderten 200 ms.

3.4 Software-Architektur

3.4.1 Modularer Aufbau

Die Software-Architektur basiert auf einem modularen, in Python implementierten Ansatz [6]. Jedes Modul ist für eine spezifische Aufgabe verantwortlich, was unabhängige Entwicklung, Testen und Fehlersuche ermöglicht [18].
Als Orchestrator fungiert `run.py`, das alle Subsysteme initialisiert und den Datenfluss koordiniert. In `config.py` sind sämtliche Parameter (Kameraeinstellungen, PID-Konstanten, MAVLink-Parameter) ausgelagert. Das Vision-Modul (`vision/vision_system.py`) erfasst den Videostream und führt die YOLO-Inferenz auf der Hailo-8L NPU aus. Das Sicherheitsmodul (`safety/emergency_system.py`)

überwacht den Systemzustand und leitet bei Anomalien Notfallmanöver ein (vgl. Abschnitt 3.5). Der `data_logger.py` zeichnet alle Datenströme zeitsynchronisiert auf (vgl. Abschnitt 3.2.3).

3.4.2 Inferenz und Datenaufzeichnung

Das System unterstützt drei Betriebsmodi: Im „Follow-Me"-Betrieb sind alle Subsysteme aktiv (Vision, Navigation, Datenlogger). Der „Data Logging"-Modus deaktiviert Objekterkennung und Navigation, um ausschließlich synchronisierte Video- und Telemetriedaten aufzuzeichnen. Die Offline-Analyse findet nach dem Flug auf einer Bodenstation statt: `offline_processor.py` nutzt die aufgezeichneten Daten für Tiefenschätzung mittels Depth Anything 3 [25] und georeferenzierte Szenen-Analyse.

3.4.3 Nebenläufigkeit und Prozesssteuerung

Die Echtzeit-Pipeline wird als sequenzielle Regelschleife im Hauptprozess ausgeführt: *Vision* → *Safety* → *Navigation* → *MAVLink*. Die YOLO-Inferenz wird vollständig auf die Hailo-8L NPU ausgelagert, sodass die CPU für Regelung, Kommunikation und Datenaufzeichnung verfügbar bleibt [34]. Zwei Daemon-Threads laufen im Hintergrund: ein MAVLink-Telemetrie-Thread für den kontinuierlichen Empfang von Fluglage-, GPS- und Batteriedaten, und ein DataLogger-Thread für die parallele Aufzeichnung.

3.5 Sicherheitskonzept

Die autonome Steuerung einer Drohne in unstrukturierten Umgebungen erfordert ein mehrschichtiges Sicherheitskonzept, das Fehler auf verschiedenen Systemebenen auffangen kann. Analog zum Defense-in-Depth-Prinzip industrieller Sicherheitsnormen [54] implementiert der Prototyp eine vierstufige Failsafe-Kaskade, bei der jede Ebene unabhängig von den darüberliegenden funktioniert.

3.5.1 Stufe 1: Visuelle Kollisionsvermeidung

Die erste und reaktivste Sicherheitsebene operiert auf Basis der Live-Objekterkennung durch das Vision-System. In jedem Verarbeitungszyklus der Hauptschleife werden die von YOLO detektierten Objekte auf ihre Nähe zum Flugpfad geprüft (vgl. Abschnitt 4.6.2).
Der Prototyp implementiert dabei eine zweistufige Abstandskaskade:

- **Warndistanz** (`OBSTACLE_MIN_DISTANCE` = 2,0 m): Wird ein Hindernis innerhalb dieses Radius detektiert, stoppt der Navigator die Vorwärtsbewegung und hält die aktuelle Position. Die Person wird weiterhin verfolgt, sobald der Pfad wieder frei ist.
- **Kritische Distanz** (`COLLISION_DISTANCE_THRESHOLD` = 1,5 m): Unterschreitet ein Hindernis diesen Schwellenwert, wird ein sofortiger Stopp-Befehl an den Flight Controller gesendet. Bei mehr als fünf aufeinanderfolgenden Kollisionswarnungen wird eine automatische Notlandung eingeleitet.

Die Distanzschätzung erfolgt über den in Abschnitt 3.2.1 beschriebenen BBox-Proxy-Ansatz. Da dieser Ansatz die Entfernung tendenziell überschätzt, ergibt sich ein konservatives Verhalten: Das System reagiert eher zu früh als zu spät [9, 31].

3.5.2 Stufe 2: Software-Absicherung

Unabhängig von der visuellen Hinderniserkennung überwacht das `EmergencySystem`-Modul auf dem Raspberry Pi kontinuierlich die Systemtelemetrie. Bei jedem Durchlauf der Hauptschleife werden folgende Prüfungen sequentiell ausgeführt:

Batterie-Überwachung. Die Akkuspannung wird gegen zwei konfigurierte Schwellenwerte geprüft. Bei Unterschreitung von `BATTERY_LOW_VOLTAGE` (13,6 V, entspricht 3,4 V pro Zelle bei 4S LiIon) wird eine Warnung ausgegeben. Fällt die Spannung unter `BATTERY_CRITICAL_VOLTAGE` (13,2 V, entspricht 3,3 V pro Zelle), wird automatisch ein Return-to-Launch (RTL) eingeleitet.

GPS-Überwachung. Das System prüft, ob ein gültiger 3D-GPS-Fix vorliegt. Im GUIDED-Modus ist ein GPS-Fix zwingend erforderlich, da alle Positions- und Geschwindigkeitsbefehle auf GPS-Koordinaten basieren [21]. Bei Verlust des GPS-Fixes wird eine Notlandung am aktuellen Standort eingeleitet.

Vision-Timeout. Liefert das Vision-System länger als `EMERGENCY_TIMEOUT` (10 s) keine verwertbaren Daten, etwa durch Kameraausfall oder Software-Blockade, wird eine Notlandung eingeleitet.

Geofence. Ein virtueller Zaun begrenzt den zulässigen Flugbereich. Überschreitet die Drohne die maximale Flughöhe (`MAX_ALTITUDE` = 30 m) oder den maximalen Radius um den Startpunkt (`MAX_DISTANCE_FROM_HOME` = 100 m), wird RTL ausgelöst. Diese Parameter sind sowohl in der Software als auch redundant im Flight Controller konfiguriert (siehe Level 3).

3.5.3 Stufe 3: Hardware-Absicherung

Die dritte Sicherheitsebene wird direkt in der ArduCopter-Firmware des MATEK H743-WLITE ausgeführt und ist damit unabhängig vom Raspberry Pi. Selbst bei vollständigem Ausfall des Companion-Computers (Absturz des Python-Prozesses, Unterbrechung der UART-Verbindung) bleiben diese Schutzmechanismen aktiv.

RC-Failsafe. Bricht das Signal der Funkfernsteuerung ab (PWM-Wert unter `FS_THR_VALUE` = 975), erkennt der Flight Controller dies innerhalb von 2 Sekunden und wechselt automatisch in den RTL-Modus (`FS_THR_ENABLE` = 1). Dieser Mechanismus schützt vor Reichweitenverlust oder technischem Versagen des RC-Senders [5].

Batterie-Failsafe (FC-seitig). Redundant zur Software-Überwachung prüft auch die ArduCopter-Firmware die Akkuspannung. Bei Low-Voltage wird RTL ausgelöst (`BATT_FS_LOW_ACT` = 2), bei kritischer Spannung eine sofortige Landung (`BATT_FS_CRT_ACT` = 1). Diese doppelte Absicherung stellt sicher, dass selbst bei einem Softwarefehler auf dem Raspberry Pi die Drohne sicher landet.

EKF-Failsafe. Der Extended Kalman Filter (EKF) fusioniert GPS-, IMU- und Barometerdaten zu einer konsistenten Positionsschätzung [4]. Bei erkannter Inkonsistenz (etwa massive GPS-Störungen oder IMU-Drift) wird über `FS_EKF_ACTION` = 1 eine Landung eingeleitet.

Geofence (FC-seitig). Analog zur Software-Begrenzung implementiert der Flight Controller einen eigenen Geofence (`FENCE_ENABLE` = 1) mit identischen Parametern: maximale Höhe 30 m, maximaler Radius 100 m. Bei Verletzung erfolgt automatisch RTL (`FENCE_ACTION` = 1).

3.5.4 Stufe 4: Manueller Eingriff

Die letzte und übergeordnete Sicherheitsinstanz ist der menschliche Pilot. Über einen dedizierten Schalter auf der Funkfernsteuerung (Kanal 6, konfiguriert als `RC6_OPTION` = 153 für Arm/Disarm) kann der Pilot die autonome Steuerung jederzeit übernehmen. Die Software erkennt einen manuellen Eingriff, indem sie den aktuellen Flugmodus überwacht: Befindet sich der Flight Controller nicht mehr im `GUIDED`-Modus, was auf eine manuelle Modusänderung via RC-Schalter hinweist, stellt das autonome System sofort alle Steuerungsbefehle ein und übergibt die vollständige Kontrolle an den Piloten [52].

Dieser Human-in-the-Loop-Ansatz ist ein bewusster Designentscheid: Der Pilot kann jeden autonomen Flugzustand, einschließlich eines laufenden Notfallmanövers, übersteuern. Damit wird dem Grundsatz Rechnung getragen, dass bei experimentellen Prototypen die finale Verantwortung stets beim menschlichen Operator verbleiben muss [9].

4 Implementierung

4.1 Hardware-Integration

Für den Aufbau des KI-gestützten Bee35 Cinewhoops wurde eine komplexe Verschaltung zwischen dem Flight Controller (Matek H743-WLITE), einem Raspberry Pi 5 mit Hailo-8L KI-Beschleuniger und diversen Peripheriegeräten realisiert.

4.1.1 Komponenten-Übersicht

4.2 Hardware-Komponenten und Kostenaufstellung

In der folgenden Tabelle sind alle für den Prototyp verwendeten Komponenten sowie deren Anschaffungskosten aufgeführt.

Funktion	**Komponente**	**Preis [€]**
Flight Controller	Matek H743-WLITE (STM32H743)	79,90
ESC (4-in-1)	T-Motor Velox V50A	52,90
Motoren (4×)	Emax Eco II 2004 3000KV	67,60
Frame	SpeedyBee Bee35 3,5” CineWhoop	39,90
GPS/Kompass	HGLRC M100-5883	19,90
Propeller (4×)	HQProp DT90MM	3,19
Companion Computer	Raspberry Pi 5 (8 GB RAM)	92,50
KI-Beschleuniger	Hailo-8L AI HAT+ (13 TOPS)	105,00
Vision-Kamera	Raspberry Pi Camera 3 Wide NoIR	29,90
RC-Sender	Radiomaster Boxer ELRS	169,00
RC-Empfänger	Radiomaster RP1 ELRS	17,90
FPV-Brille	Skyzone Cobra X V4	249,00
FPV-Kamera	Caddx Ratel 2 Pro	36,90
Video-Transmitter	SpeedyBee TX800	19,90
VTX-Antenne	Foxeer Lollipop 4	19,90
Spannungsregler (BEC)	Matek BEC125-PRO (5,25 V)	16,90
Akku	Racepow 4000 mAh 4S Li-Ion	39,90
Ladegerät	SkyRC B6neo	34,90
Zubehör	Smoke Stopper, Schrauben, Löt Utensilien, Stand Offs, ...	20,90
Gesamt		**1.115,19**

Tabelle 4.1: Stückliste und Kostenaufstellung des UAV-Prototyps.

4.2.1 Schaltplan

Die folgende Abbildung zeigt die vollständige topologische Ansicht der Hardware-Komponenten und deren elektrische Verbindung.

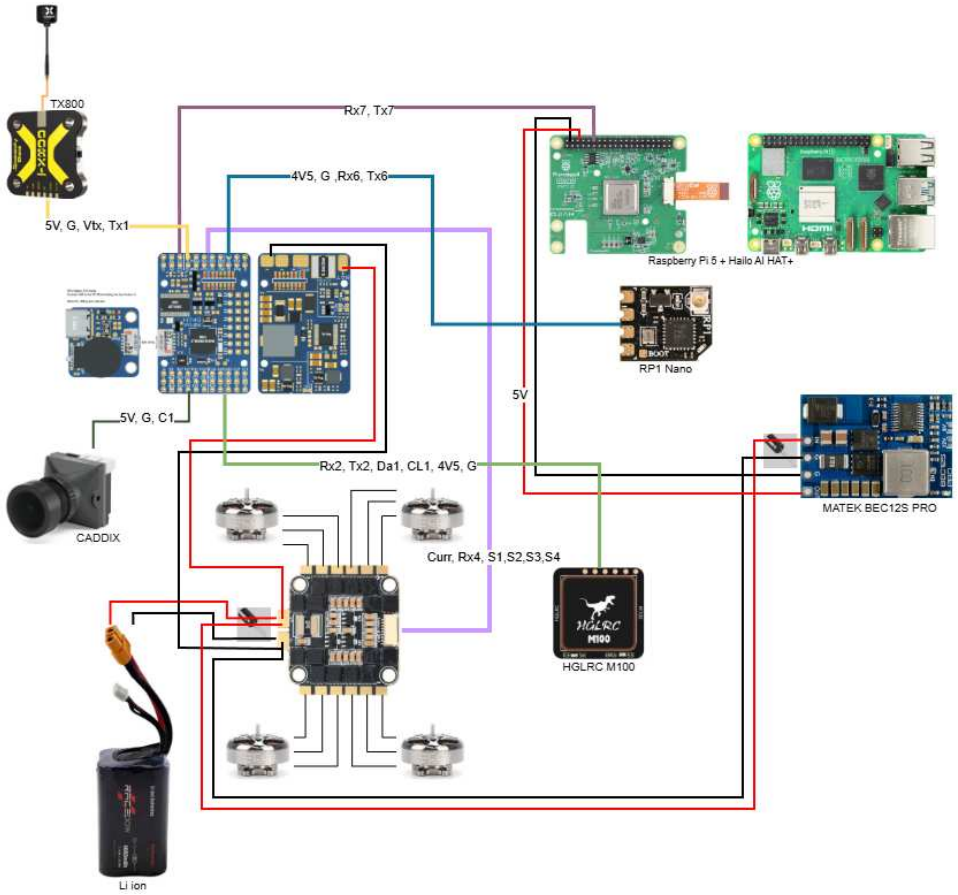


Abbildung 4.1: System-Schaltplan: Matek H743-WLITE, Raspberry Pi 5 und Peripherie.

4.2.2 UART-Konfiguration

Die Kommunikation der Teilsysteme erfolgt über dedizierte UART-Schnittstellen des H743-Prozessors. Besonders hervorzuheben ist die redundante Stromversorgung des Raspberry Pi 5 über ein externes Matek BEC Pro, um die Lastspitzen des KI-Moduls abzufangen.

4.2.3 Energieversorgung

Die Spannungsversorgung wurde auf eine nominale Eingangsspannung von $14,8\,V$ (4S Li-Ion/LiPo) ausgelegt. Da der Raspberry Pi 5 eine stabile Versorgung von $5,0\,V$ bis $5,2\,V$ bei bis zu $5\,A$ benötigt, wird die interne $5\,V$-Schiene des Flight Controllers umgangen und ein dediziertes **Matek BEC 12S Pro** verwendet. Hierbei wird eine Spannung von $5,25\,V$ eingestellt, um den Spannungsabfall über die Zuleitungen zu kompensieren.

Komponente	Schnittstelle (FC)	Funktion
Raspberry Pi 5	UART 7 (Rx7/Tx7)	Mavlink / OSD-Datenstrom
HGLRC M100 GPS	UART 2 (Rx2/Tx2)	GNSS Navigation
M100 Kompass	I2C 1 (DA1/CL1)	Magnetometer (Mag)
RP1 Receiver	UART 6 (Rx6/Tx6)	ELRS Fernsteuerung
SpeedyBee TX800	UART 1 (Tx1)	IRC Tramp (VTX Control)
ESC Telemetry	UART 4 (Rx4)	Drehzahl- und Stromdaten

Tabelle 4.2: Detaillierte Port-Zuweisung der Hardware.

4.2.4 Physischer Aufbau

Der physische Aufbau orientiert sich an den Anforderungen eines Edge-KI-basierten UAVs: Schutz der Rechenhardware, effizientes thermisches Management und Minimierung von Vibrationseinflüssen.

4.2.4.1 CineWhoop als Rahmenwahl

Als strukturelle Basis wurde der *SpeedyBee Bee35* Rahmen (3,5 Zoll) gewählt. Das CineWhoop-Design mit Propeller-Protektoren (Ducts) bietet den für Tests in Personennähe erforderlichen physischen Propellerschutz und mit ca. 130 g Eigengewicht ausreichend Steifigkeit für das erhöhte Abfluggewicht[6].

4.2.4.2 Integrierte Energieversorgung

Der Raspberry Pi 5 erreicht unter KI-Last Spitzenströme von bis zu 2,5A[6], ergänzt durch die Hailo-8L NPU mit 1,5W[34][17]. Der *Matek BEC125-PRO* regelt die variierende Spannung des 4S-LiIon-Akkus auf konstant 5,25V. Dies stellt eine einheitliche Massereferenz für die UART-Kommunikation sicher und spart durch Verzicht auf einen Sekundärakku ca. 150 g Gewicht[6].

4.2.4.3 Sensoranbindung und Kühlung

- **Kamera-Integration:** Die FPV-Kamera (*Caddx Ratel 2 Pro*) ist in einem flexiblen TPU-Mount fixiert, das als mechanischer Tiefpassfilter Motorvibrationen dämpft. Die *Raspberry Pi Camera 3* befindet sich in einem 3D-gedruckten Gehäuse, fixiert mit viskoelatischem Schmelzklebstoff zur Absorption hochfrequenter Schwingungen.
- **Thermisches Management:** Ein *Active Cooler* leitet die Abwärme von CPU und Hailo-8L im Propeller-Luftstrom ab[41].
- **Dämpfung:** ESC-Montage über Gummipuffer (*Soft-Mounting*), FC über 3D-gedruckte Adapterplatte verbunden. Die IMU profitiert von dieser Entkopplung für präzise EKF3-Filterung.

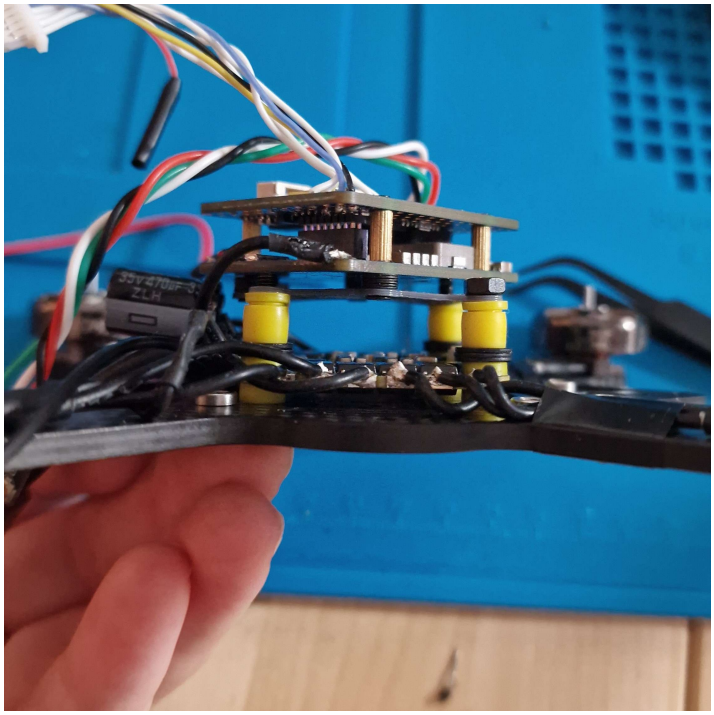


Abbildung 4.2: Physischer Aufbau des Elektronik-Stacks inklusive Vibrationsdämpfung und Montageplattform.

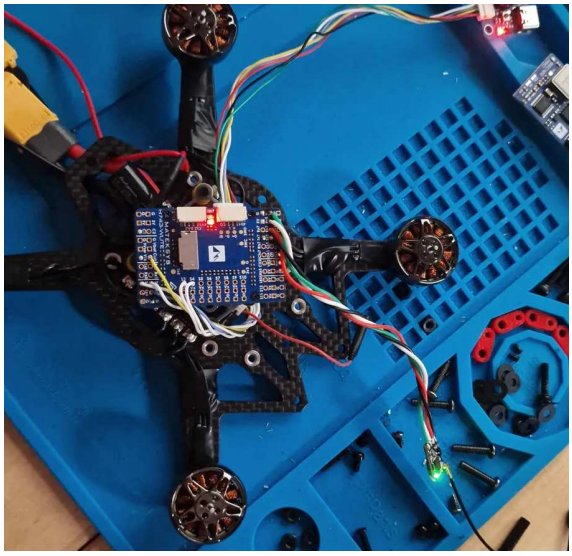
4.2.4.4 Gewichtsverteilung und vertikaler Systemaufbau

Der Schwerpunkt wurde zentral in Bezug auf die Propellerebene optimiert. Der vertikale Stack-Aufbau gliedert sich in: **Basis-Ebene** mit ESC, FC und Empfänger (durch verlängerte Standoffs erweitert); **Zentrale Ebene** mit dem 4000 mAh Li-Ion Akku als massereichster Komponente nahe der Hochachse; **Obere Ebene** mit dem Raspberry Pi 5 auf 3D-gedruckter Plattform, exponiert für minimale EMI und effiziente Kühlung im Prop-Wash.

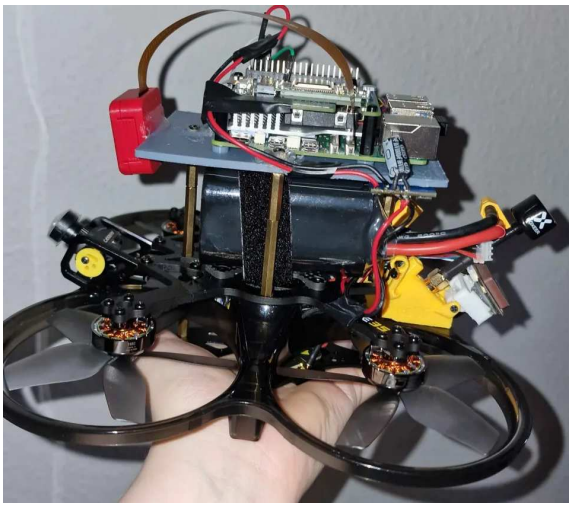
Kameraanbindung. Die Pi Camera 3 ist über MIPI-CSI-2 an den RPi 5 angebunden. Da der RPi 5 einen schmaleren 22-Pin-Konnektor verwendet, war ein 15-zu-22-Pin-Adapterkabel erforderlich.

4.2.5 Flight Controller

Als Flight Controller kommt der *MATEK H743-WLITE* [28] mit `STM32H743` (ARM Cortex-M7, 480 MHz) zum Einsatz. Die H743-Serie bietet acht UART-Schnittstellen für die fünf Peripheriegeräte (GPS, MAVLink, RC, ESC-Telemetrie, VTX) und wird offiziell von ArduCopter unterstützt.
Ursprünglich war der formfaktorkompatible H743-SLIM vorgesehen. Durch eine Verwechslung bei der Beschaffung wurde die WLITE-Variante (Fixed-Wing-Layout) bestellt. Das längliche Board ragt seitlich über den Bee35-Rahmen hinaus und erforderte eine 3D-gedruckte Adapterplatte (vgl. Abschnitt 6.2.2 in Kapitel 6, Abbildung 4.4). Die Vorteile der H743-Plattform (Rechenleistung, UART-



(a) Detailaufnahme des ESC-FC-Stacks.



(b) Seitenansicht des Gesamtaufbaus inklusive Akku und RPi 5.

Abbildung 4.3: Physische Systemintegration: Gegenüberstellung der internen Dämpfungskomponenten (links) und der vertikalen Schichtstruktur (rechts).

Anzahl, ArduCopter-Kompatibilität) bleiben davon unberührt.
Als Basis dient ArduCopter 4.8, geflasht und parametriert über Mission Planner.

Sensor-Kalibrierung. Vor dem ersten Flug wurden die *Accelerometer-Kalibrierung* (sechs Orientierungen), die *Kompass-Kalibrierung* des externen QMC5883L (`COMPASS_EXTERNAL = 1`, Onboard Mag Calibration mit Rotation in allen Achsen) und die *RC-Kalibrierung* des Radiomaster Boxer durchgeführt.

UART-Konfiguration. SERIAL3 (UART2) wurde für GPS (`PROTOCOL = 5`, 38 400 Baud) konfiguriert, ein separater UART für MAVLink (`PROTOCOL = 2`, 57 600 Baud). Die moderate Baudrate von 57 600 wurde bewusst gewählt, um die Störsicherheit bei längeren Kabelwegen und EMI der Antriebskomponenten zu erhöhen.
Die erfolgreiche Kommunikation zwischen Raspberry Pi und Flight Controller lässt sich mit wenigen Zeilen Python-Code über die `pymavlink`-Bibliothek verifizieren. Das folgende Listing zeigt den minimalen Verbindungsaufbau, bei dem der Companion Computer auf ein Heartbeat-Signal des Flight Controllers wartet:

```
1 from pymavlink import mavutil
2
3 m = mavutil.mavlink_connection('/dev/ttyAMA0', baud=57600)
4
5 m.wait_heartbeat(timeout=10)
6
```

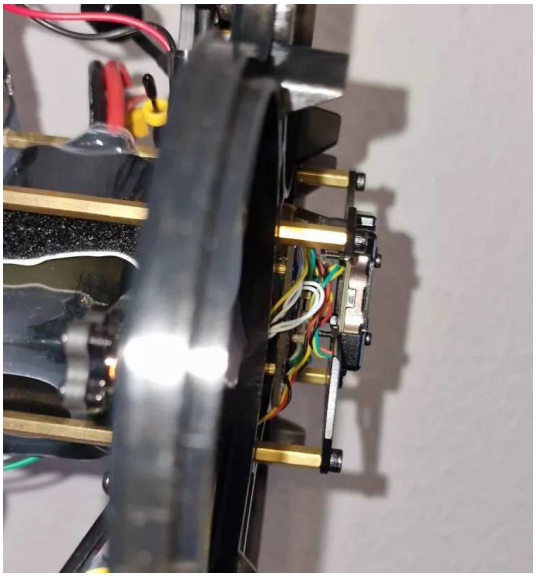


Abbildung 4.4: Seitenansicht des unteren Stack-Bereichs: Der MATEK H743-WLITE ragt seitlich über den Bee35-Rahmen hinaus und erfordert noch zusätzliche Stützstrukturen für einen stabilen Stand.

```
7 print(f'Verbunden: System {m.target_system}')
```

Listing 4.1: Verbindungsaufbau zwischen Raspberry Pi und Flight Controller via MAVLink

Tabelle 4.3 zeigt die standardmäßige Zuordnung der seriellen Schnittstellen des MATEK H743-WLITE, wie sie in der ArduPilot-Firmware definiert ist.

Tabelle 4.3: Standard-UART-Zuordnung des MATEK H743-WLITE [3].

Bezeichnung	**Funktion (Standard)**	**Hardware-UART**
SERIAL0	Console	USB
SERIAL1	Telemetry 1	UART7
SERIAL2	Telemetry 2	USART1
SERIAL3	GPS 1	USART2
SERIAL4	GPS 2	USART3
SERIAL5	USER	UART8
SERIAL6	USER	UART4
SERIAL7	USER	UART6 (nur TX)

Flugmodi Für die verschiedenen Betriebsphasen wurden sechs Flugmodi auf dem Mode-Switch des Senders konfiguriert (Tabelle 4.4). Der Modus `GUIDED` bildet die Grundlage für die autonome Steuerung durch den Companion Computer, da er die Annahme von Positions- und Geschwindigkeitsbefehlen über MAVLink erlaubt. `STABILIZE` dient als sicherer Fallback-Modus für manuelle Eingriffe des Piloten.

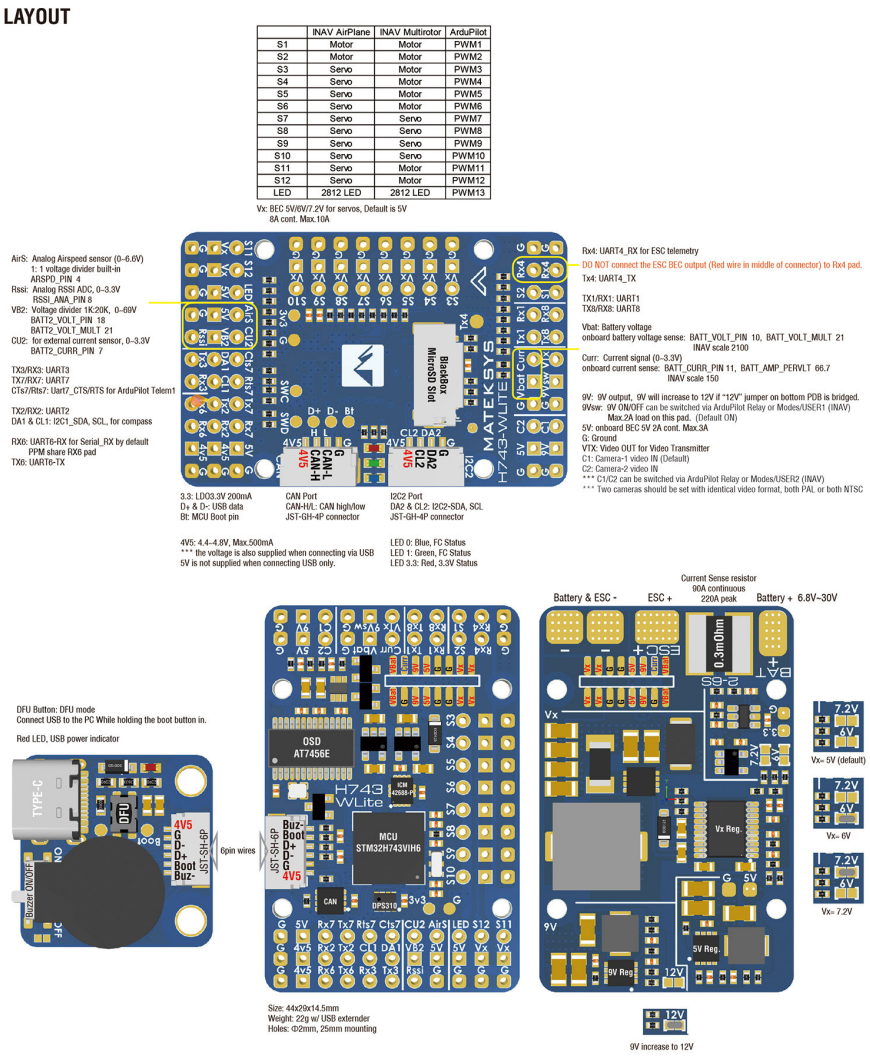


Abbildung 4.5: Pin-Belegung und Board-Layout des MATEK H743-WLITE WING Flight Controllers [28].

Tabelle 4.4: Konfigurierte Flugmodi und deren Einsatzzweck

Position	Modus	Funktion
1	`STABILIZE`	Manuell mit Lagestabilisierung
2	`ALT_HOLD`	Automatische Höhenhaltung
3	`LOITER`	GPS-basierte Positionshaltung
4	`GUIDED`	Autonome Steuerung via MAVLink
5	`RTL`	Return to Launch
6	`LAND`	Automatische Landung

Failsafe-Konfiguration. Mehrstufige Failsafes sichern den Flugbetrieb ab: Bei 3,4 V/Zelle (`BATT_LOW_VOLT = 13.6 V`) wird RTL ausgelöst, bei 3,3 V/Zelle (`BATT_CRT_VOLT = 13.2 V`) eine sofortige Landung. Ein GPS-Geofence begrenzt den Aktionsradius auf 100 m horizontal und 30 m vertikal, RC-Signalverlust löst ebenfalls RTL aus.

4.3 Live Vision-System

Das Live Vision-System basiert auf **YOLO26** mit Inferenz auf dem **Hailo-8L** NPU für Echtzeit-Objekterkennung.

4.3.1 Hailo-8L Integration und HEF-Kompilierung

Da zum Zeitpunkt dieser Arbeit keine offiziellen HEF-Binärdateien vorlagen, wurde die Kompilierung über das Repository von Dubinsky [12] durchgeführt:

1. **Modell-Export:** Konvertierung des nativen PyTorch-Formats (`.pt`) in das intermediäre `ONNX`-Format [33].
2. `Post-Training Quantization` (PTQ) [32]: Um die Effizienz der NPU optimal zu nutzen, wird das Modell von FP32 auf **INT8** quantisiert [19]. Hierbei kommt ein *Calibration Dataset* zum Einsatz, um den Genauigkeitsverlust durch die Bit-Reduktion zu minimieren.
3. **HEF-Kompilierung:** Der Hailo Dataflow Compiler mappt die Layer des YOLO26-Netzwerks auf die physischen Rechenressourcen des Hailo-8L Chips.

Ein **Fallback-Management** wechselt bei Hailo-Fehler automatisch auf CPU-basierte Inferenz via `ultralytics` [51], sodass die Vision-Pipeline auch ohne NPU funktionsfähig bleibt.

4.3.2 Echtzeit-Objekterkennung mit YOLO26

Die Methode `_detect_objects()` abstrahiert zwischen zwei Inferenz-Backends und liefert ein vereinheitlichtes Detektionsformat.

Konfidenz-Schwellwert und Klassenfilterung. Jede Detektion wird gegen `YOLO_CONFIDENCE` = 0,5 gefiltert. Nur elf relevante COCO-Klassen (`OBSTACLE_CLASSES`: *person*, *bicycle*, *car*, *motorcycle*, *bus*, *truck*, *tree*, *chair*, *bench*, *dog*, *cat*) werden weiterverarbeitet.

Duale Inferenz-Architektur. Die Methode `_detect_objects()` unterstützt zwei Backends mit identischem Ausgabeformat:

1. **Hailo-8L (Primär):** Das Eingabebild wird zunächst in das RGB-Format konvertiert und mittels *Letterboxing* auf 640×640 Pixel skaliert, wobei das Seitenverhältnis erhalten bleibt und fehlende

Bereiche mit einem Grauwert (114) aufgefüllt werden. Der resultierende Tensor wird als `uint8`-Array an die `HailoPythonInferenceEngine` übergeben. Die zurückgegebenen Bounding-Box-Koordinaten werden anschließend invers skaliert, um auf die Originalauflösung zurückzurechnen. Für das Tracking kommt ein eigenimplementierter `SimpleByteTrack`-Tracker zum Einsatz (s. u.).

2. **Ultralytics (Fallback):** Ist kein Hailo-8L verfügbar, wird das Standard-`ultralytics`-Framework mit dem Modell `yolo26n.pt` oder einem Fallback (`yolov8n.pt`) genutzt. Das Tracking erfolgt über den integrierten ByteTrack-Tracker (`bytetrack.yaml`) mit persistenten Track-IDs (`persist=True`).

Das folgende Listing zeigt den Hailo-Inferenzpfad der Methode `_detect_objects()`:

```
# Hailo-8L Inference via yolo26_hailo
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
padded, scale, pad_w, pad_h = self._letterbox_frame(frame_rgb, 640)
input_tensor = np.expand_dims(padded, axis=0).astype(np.uint8)
raw_dets, _ = self.hailo_engine.infer(
    input_tensor, conf_threshold=self.confidence_threshold
)

for det in raw_dets:
    if det['cls_name'] not in OBSTACLE_CLASSES:
        continue
    # Koordinaten von Letterbox zuruckskalieren
    x1 = float(max(0, min((det['x1'] - pad_w) / scale, w_orig)))
    y1 = float(max(0, min((det['y1'] - pad_h) / scale, h_orig)))

# ByteTrack: persistente Track-IDs zuweisen
if use_tracking and detections:
    detections = self.byte_tracker.update(detections)
```

Listing 4.2: Hailo-8L Inferenzpfad der Objekterkennung (Auszug aus `vision_system.py`)

Distanzschätzung per BBox-Proxy. Da zur Laufzeit auf dem Raspberry Pi kein separates Tiefenmodell aktiv ist, wird die Entfernung zu erkannten Objekten über einen BBox-Höhen-Proxy approximiert. Die Formel nutzt eine Referenzkalibrierung: Eine Person mit einer Bounding-Box-Höhe von 400 px entspricht einer Distanz von 2,5 m. Für beliebige Objekte gilt:

$$d = \frac{h_{\text{ref}} \cdot d_{\text{ref}}}{h_{\text{bbox}}} = \frac{400 \cdot 2{,}5}{h_{\text{bbox}}}$$

Diese perspektivische Approximation liefert für den relevanten Nahbereich (1–6 m) hinreichende Genauigkeit für die binäre Entscheidung der Hindernisvermeidung (Stopp vs. Weiterfahrt). Für metrisch genaue Tiefenanalysen wird die Offline-Pipeline mit DA3 eingesetzt (vgl. Abschnitt 4.5.1).

ByteTrack-Tracking. Die Klasse `SimpleByteTrack` erhält persistente Objekt-IDs über Frames hinweg. Eine IoU-Matrix und der Hungarian-Algorithmus (`scipy.optimize.linear_sum_assignment`) ordnen Detektionen bestehenden Tracks zu; neue Detektionen erzeugen neue Tracks, verlorene werden nach `max_lost` = 30 Frames gelöscht.

Single-Person-Lock. Die Zielselektion (`select_target`) erfordert exakt eine Person im Sichtfeld. Deren Track-ID wird gesperrt und als Referenz für den PID-Regler übergeben.

4.3.3 Sequentielle Regelschleife mit Threading

Die Klasse `DroneSystem` orchestriert alle Subsysteme in einer sequentiellen Verarbeitungsschleife mit ergänzenden Daemon-Threads, um Race Conditions auszuschließen.

Hauptschleife. Die Methode `start()` durchläuft pro Zyklus sechs Schritte:

1. **Vision Processing:** `process_frame()` erfasst ein Kamerabild und führt die YOLO-Inferenz durch. Im Follow-Me-Modus wird ByteTrack-Tracking aktiviert (`use_tracking=True`).
2. **Frame-Aufnahme:** Der aktuelle Frame wird an den `DataLogger` übergeben, sofern eine Aufnahme aktiv ist.
3. **Sicherheits-Check:** Das `EmergencySystem` prüft Batteriestatus, GPS-Integrität und Kollisionsgefahr. Bei einem Sicherheitsereignis wird der aktuelle Zyklus abgebrochen und eine Notfallroutine eingeleitet.
4. **Navigation:** Abhängig vom aktiven Flugmodus wird `follow_me()`, `explore_area()` oder `hold_position()` aufgerufen.
5. **Pipeline-Logging:** Per-Frame-Metriken (FPS, Anzahl Detektionen, Loop-Latenz, Modus) werden in eine CSV-Datei geschrieben.
6. **Frame-Rate-Control:** Ein `time.sleep(1/TARGET_VISION_FPS)` begrenzt die Zyklusrate auf das konfigurierte Ziel von 20 FPS.

Das folgende Listing zeigt die Kernlogik der Hauptschleife:

```
while self.is_running:
    loop_start = time.time()

    # 1. Vision Processing
    use_tracking = (mode == FlightMode.FOLLOW_ME)
    vision_data = self.vision.process_frame(use_tracking)

    # 2. Frame aufnehmen
```

```
    if self.data_logger.recording:
        self.data_logger.log_frame(vision_data.get('frame'))

    # 3. Sicherheits-Checks
    safety_ok = self.emergency.check_safety(vision_data)
    if not safety_ok:
        self.emergency.handle_emergency()
        continue

    # 4. Navigation basierend auf Modus
    if mode == FlightMode.FOLLOW_ME:
        self.navigator.follow_me(vision_data)

    # 5. Pipeline-CSV
    self._log_pipeline_row(vision_data, loop_start, safety_ok)

    # 6. Frame-Rate Control
    time.sleep(1.0 / TARGET_VISION_FPS)
```

Listing 4.3: Sequentielle Regelschleife in `DroneSystem.start()` (Auszug aus `run.py`)

Daemon-Threads. Ein dedizierter Telemetrie-Thread (`daemon=True`) empfängt MAVLink-Nachrichten unabhängig von der Vision-Zyklusrate (vgl. Abschnitt 4.4.2). Per-Frame-Metriken werden in der Hauptschleife geschrieben, mit periodischem `flush()` (alle 50 Frames) für Crash-Sicherheit.

NPU-Offloading. Die sequentielle Architektur ist tragfähig, weil die YOLO-Inferenz auf den Hailo-8L NPU ausgelagert wird. Die ARM-CPU bleibt frei für Pre-/Postprocessing und Navigationslogik. Ohne Offloading würde CPU-Inferenz (∼4 FPS) die gesamte Kette blockieren.

4.4 Datenakquise-System

Die Klasse `DataLogger` kapselt die synchronisierte Aufnahme von Video, Telemetrie und Frame-Zeitstempeln.

4.4.1 Synchronisierte Aufnahme-Logik

Da Videostream (Hardware-Zeitstempel) und Telemetrie (asynchron via MAVLink) heterogene Taktquellen nutzen, verwendet der `DataLogger` einen gemeinsamen Referenzzeitpunkt.

Shared Start-Time. Beim Aufruf von `start_recording()` wird `self.start_time = time.monotonic()` als Nullpunkt gesetzt. Alle Datenströme referenzieren `elapsed_s` relativ dazu; `time.monotonic()` verhindert Sprünge durch NTP-Korrekturen.

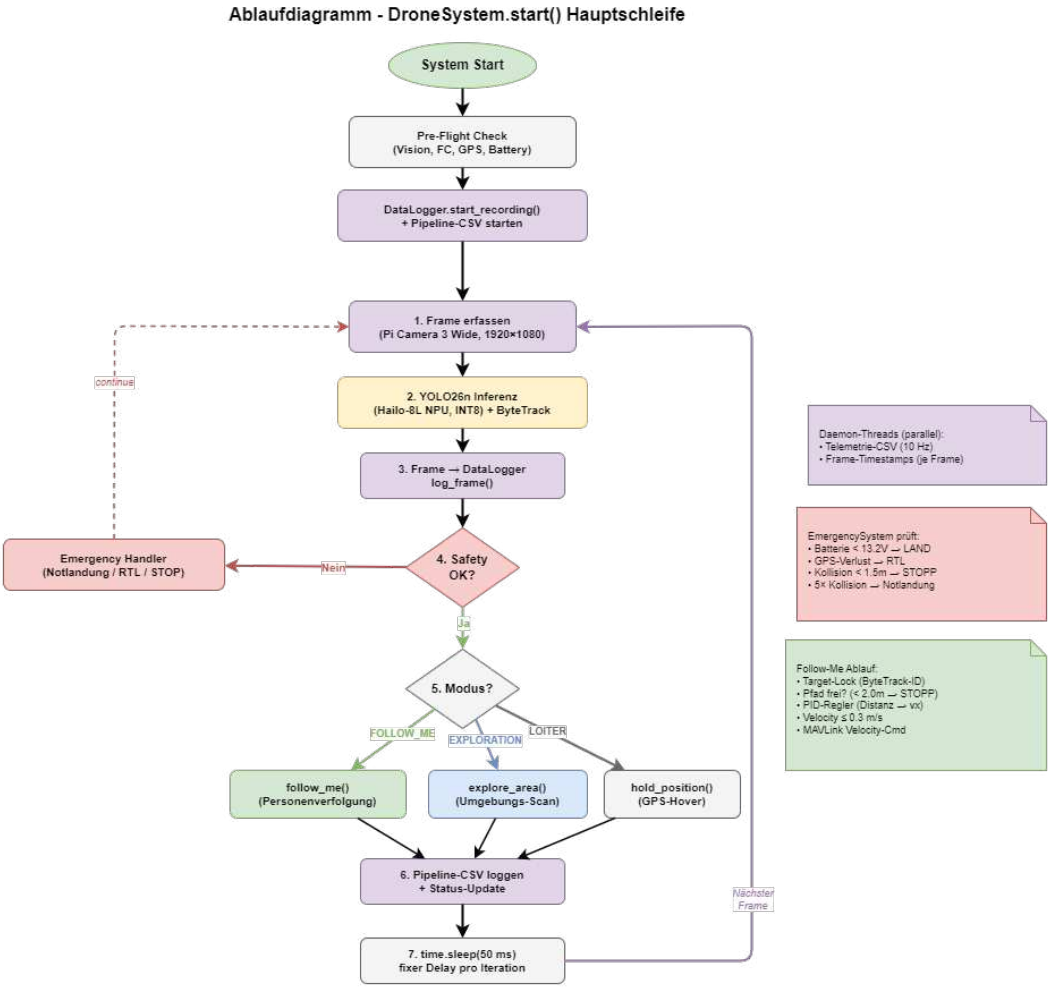


Abbildung 4.6: Ablaufdiagramm der sequentiellen Hauptschleife in `DroneSystem.start()`

Session-Verzeichnisstruktur. Jede Aufnahmesession erzeugt ein eigenes Verzeichnis mit Zeitstempel-Suffix (z. B. `flight_20260407_143022/`), das folgende Dateien enthält:

- `video.mkv`: H264-kodiertes Flugvideo
- `telemetry.csv`: MAVLink-Telemetrie (GPS, Attitude, Battery)
- `frame_timestamps.csv`: Hardware-Zeitstempel pro Frame
- `metadata.txt`: Aufnahme-Parameter (Auflösung, Codec, Sensor-Raten)
- `summary.txt`: Zusammenfassung nach Aufnahme-Ende (Dauer, Frame-Anzahl)

Betriebsmodi. Im **integrierten Modus** erhält der `DataLogger` vom `DroneSystem` Frames über `log_frame()` und schreibt sie via `cv2.VideoWriter` in MP4. Im **Standalone-Modus** für dedizierte Aufnahmeflüge öffnet er eine eigene MAVLink-Verbindung und nutzt `picamera2` mit `H264Encoder` in einen MKV-Container.

4.4.2 Telemetrie-Extraktion und CSV-Mapping

Der Telemetrie-Thread empfängt kontinuierlich MAVLink-Nachrichten und extrahiert die relevanten Felder in ein flaches CSV-Format mit 16 Spalten (Tabelle 4.5).

Tabelle 4.5: Spaltenstruktur der Telemetrie-CSV mit MAVLink-Quellnachrichten

Spalte(n)	**Beschreibung**	**MAVLink-Quelle**
`timestamp_s`	Elapsed seit Start (s)	System-Clock
`system_time_ms`	Unix-Timestamp (ms)	System-Clock
`lat`, `lon`	GPS-Koordinaten	`GLOBAL_POSITION_INT`
`alt_m`	Relative Höhe (m)	`GLOBAL_POSITION_INT`
`roll/pitch/yaw_deg`	Fluglage (Grad)	`ATTITUDE`
`vx/vy/vz_ms`	Geschwindigkeit (m/s)	`GLOBAL_POSITION_INT`
`battery_v/pct`	Spannung und Prozent	`BATTERY_STATUS`
`gps_fix/satellites`	Fix-Typ und Anzahl	`GPS_RAW_INT`
`flight_mode`	Aktueller Flugmodus	`HEARTBEAT`

Message-Intervalle. Beim Verbindungsaufbau werden per `MAV_CMD_SET_MESSAGE_INTERVAL` explizite Raten konfiguriert: `ATTITUDE` @ 20 Hz, `GLOBAL_POSITION_INT` @ 10 Hz, `GPS_RAW_INT` @ 5 Hz. Ein CSV-Eintrag wird bei jeder `ATTITUDE`- oder `GLOBAL_POSITION_INT`-Nachricht geschrieben (bis 30 Einträge/s), wobei die aktuellsten Werte aller Felder zusammengeführt werden.

4.4.3 Video-Encoding (MKV) und Frame-Timestamping

MKV-Container. Im Standalone-Modus wird H264-Video in einem **Matroska-Container** (MKV) gespeichert. MKV schreibt Metadaten inkrementell; bei einem Abbruch bleibt das bisherige Material lesbar, während MP4 den Index erst am Dateiende schreibt.

H264-Encoding. Der Hardware-Encoder des BCM2712 wird via `H264Encoder` (`picamera2`) mit `Quality.HIGH` bei 1920×1080 Pixeln und 30 FPS angesteuert, ohne nennenswerte CPU-Last.

Frame-Timestamping. Ein Encoder-Callback zeichnet pro Frame drei Werte in `frame_timestamps.csv`: fortlaufende `frame_nr`, `elapsed_s` (System-Clock) und `sensor_timestamp_ms` (Pi Camera 3 Hardware-Zeitstempel). Die Kombination ermöglicht in der Offline-Analyse sowohl Telemetrie-Zuordnung als auch Erkennung von Frame-Drops.

4.4.4 Datenintegrität und Absicherung

Im UAV-Betrieb besteht Risiko für Datenverlust durch unerwartete Stromausfälle. Der `DataLogger` implementiert mehrere Absicherungsstrategien.

Periodisches Flushing. CSV-Dateien werden in Intervallen geflusht: Telemetrie alle 10 Einträge ($\approx$1 s), Frame-Timestamps alle 30 Frames, Pipeline-CSV alle 50 Frames. Im Worst Case gehen maximal 1–2 s verloren.

Streamable Video-Container. Der MKV-Container (Abschnitt 4.4.3) gewährleistet, dass bei Abbruch nur der letzte Frame verloren geht.

Metadata und Zusammenfassung. `metadata.txt` speichert die Aufnahmeparameter zu Beginn; nach regulärem Ende wird zusätzlich `summary.txt` mit statistischen Kennzahlen erzeugt. Deren Fehlen signalisiert einen Aufnahmeabbruch.

4.5 Offline-Verarbeitungspipeline

4.5.1 Depth Anything 3: Multi-View Tiefenschätzung

Für die Offline-Tiefenanalyse kommt *Depth Anything 3* (DA3, `DA3-LARGE-1.1`) [25] zum Einsatz. Der Multi-View-Ansatz verarbeitet mehrere zeitlich aufeinanderfolgende Frames simultan in einem Vision-Transformer und schätzt pixelgenaue Tiefenkarten sowie relative Kameraposen. Das Ergebnis ist eine konsistente 3D-Punktwolke (GLB-Export).
Frames werden mit konfigurierbarer Rate (Standard: 3 FPS) aus dem Flugvideo extrahiert und batchweise verarbeitet. Vorhandene GPS-Daten werden als Pose-Prior (World-to-Camera-Transformation) übergeben, was die metrische Skalierung verbessert.

VRAM-Beschränkung und Batching-Strategie. Der VRAM-Bedarf skaliert quadratisch mit der Frame-Anzahl. Auf der NVIDIA GTX 1650 Ti (4 GB) führten 56 Frames bei 504 px zu einem `CUDA Out of Memory`-Fehler (≈97 GB benötigt). Die Lösung teilt den Stream in Batches zu maximal 8 Frames bei 336 px (5,19 s/Frame, vgl. Tabelle 4.6). Zwischen Batches wird GPU-Speicher via `torch.cuda.empty_cache()` freigegeben.

Tabelle 4.6: VRAM-Bedarf von DA3 Multi-View bei unterschiedlichen Konfigurationen. Die markierte Zeile (*) entspricht der in dieser Arbeit verwendeten Konfiguration.

Frames	Auflösung	VRAM (ca.)	Status
56	504 px	97 GB	Out of Memory
16	504 px	≈ 28 GB	Out of Memory
8	504 px	≈ 8 GB	Out of Memory
8	336 px	≈ 3,5 GB	Erfolgreich (*)

Größere GPUs (z. B. RTX 3090, 24 GB) könnten mehr Frames pro Batch bei höherer Auflösung verarbeiten.

4.5.2 GPS-Telemetrie als Pose-Prior

Sofern synchronisierte GPS-Daten vorliegen, werden diese als Pose-Prior genutzt:

1. **GPS zu lokalen Koordinaten:** Die geodätischen Koordinaten (Breitengrad, Längengrad, Höhe) werden in ein lokales ENU-Koordinatensystem (East-North-Up) relativ zum ersten Frame transformiert.
2. **Rotation aus Heading:** Der Yaw-Winkel der Drohne wird in eine 3×3 Rotationsmatrix überführt, wobei die Kamera als nach vorne gerichtet angenommen wird.
3. **World-to-Camera-Extrinsics:** Translation und Rotation werden zu einer $N \times 3 \times 4$ Extrinsics-Matrix kombiniert, die DA3 als Pose-Prior akzeptiert.

Ohne GPS-Daten schätzt DA3 die Kameraposen ausschließlich aus der visuellen Korrespondenz zwischen den Frames.

4.5.3 Georeferenzierter Daten-Export

Die Ergebnisse der Offline-Pipeline werden in mehreren Formaten exportiert:

- **3D-Punktwolke:** Pro Batch eine GLB-Datei mit texturierter 3D-Rekonstruktion (durchschnittlich 4,9 MB pro 8-Frame-Batch)
- **Tiefenvisualisierungen:** Farbkodierte Depth Maps im INFERNO-Farbschema als PNG pro Frame

- **Frame-Analyse-CSV:** Tiefenstatistiken (Mittelwert, Minimum, Maximum, Standardabweichung) sowie GPS-Koordinaten pro Frame
- **Statistiken:** JSON-Datei mit Gesamtübersicht (Verarbeitungszeit, VRAM-Nutzung, Modellkonfiguration)
- **Geo-Karte:** Interaktive Folium-HTML-Karte mit tiefenkodierten Markern an den GPS-Positionen der einzelnen Frames

4.6 Autonome Navigation

4.6.1 Follow-Me-Algorithmus

Die autonome Personenverfolgung in `AutonomousNavigator` übersetzt Vision-Pipeline-Ausgaben in MAVLink-Velocity-Kommandos für den `GUIDED`-Modus.

Target-Lock-Protokoll. Der Target-Lock wird nur gesetzt, wenn *exakt eine* Person erkannt wird. Die Track-ID wird als `tracked_target_id` persistiert und in jedem Folgezyklus zur Zuordnung verwendet.

Regelkreis. In jedem Verarbeitungszyklus durchläuft die Methode `follow_me()` eine kaskadierte Logik:

1. **Target-Suche:** Die aktuelle Track-ID wird in den Detektionen gesucht. Ist die Person sichtbar, wird ihre Bildposition (Bounding-Box-Mittelpunkt) und, sofern verfügbar, ihre geschätzte Distanz extrahiert.
2. **Hindernisprüfung:** Vor jeder Bewegung wird geprüft, ob der Flugweg frei ist (vgl. Abschnitt 4.6.2). Bei einem kritischen Hindernis wird die Bewegung sofort gestoppt.
3. **Geschwindigkeitsberechnung:** Aus der Bildposition werden vier Velocity-Komponenten abgeleitet:
   - *Forward/Backward:* Ein PID-Regler minimiert den Abstandsfehler zwischen der geschätzten Distanz zur Person und dem Soll-Abstand (`FOLLOW_DISTANCE` = 3,5 m). Ist keine Tiefeninformation verfügbar, dient die vertikale Bildposition als Proxy.
   - *Left/Right:* Proportional zum horizontalen Offset der Person zur Bildmitte.
   - *Yaw Rate:* Proportional zum horizontalen Offset, um die Drohne stets zur Person auszurichten.
4. **Velocity-Begrenzung:** Alle Achsen werden proportional skaliert, sodass die resultierende Geschwindigkeit `FOLLOW_SPEED` = 0,3 m/s nicht überschreitet. Die Yaw-Rate wird auf `MAX_YAW_RATE` = 15°/s begrenzt.

PID-Regler. Der Distanzregler implementiert einen vollständigen PID-Controller mit den Koeffizienten $k_p = 0{,}5$, $k_i = 0{,}05$ und $k_d = 0{,}2$. Ein Anti-Windup-Mechanismus begrenzt den Integralanteil auf $\pm 2{,}0$, um Überschwingen nach längeren Abweichungen zu verhindern. Der Ausgabewert wird auf den Bereich $[-1{,}0;\ +1{,}0]$ geclampt und anschließend durch die Geschwindigkeitsbegrenzung weiter skaliert.

Target-Verlust-Behandlung. Ohne Detektion zählt ein Frame-Counter die verlorenen Zyklen. Innerhalb von `TARGET_LOST_LOITER_FRAMES` = 30 (ca. 1–2 s) hält die Drohne Position und wartet auf Wiedererkennung. Danach wechselt das System in `LOITER` und gibt den Lock frei.

4.6.2 Proaktive Hindernisvermeidung

Die Hindernisvermeidung arbeitet ausschließlich auf YOLO-Detektionen ohne separates Tiefenmodell.

Hinderniserkennung. Die Methode `_identify_obstacles()` klassifiziert Detektionen anhand der `OBSTACLE_CLASSES` und schätzt Distanzen primär über eine optionale Tiefenkarte, alternativ über den BBox-Höhen-Proxy.

Zweistufiges Warnsystem. Die Hindernisvermeidung ist als zweistufige Kaskade implementiert:

1. **Navigator-Ebene (`_check_path_clear`):** Vor jeder Bewegung im Follow-Me-Modus wird geprüft, ob ein Hindernis näher als `OBSTACLE_MIN_DISTANCE` = 2,0 m erkannt wurde (Flag `critical`). Ist dies der Fall, wird die Bewegung sofort gestoppt.
2. **Emergency-Ebene (`_check_collision`):** Unabhängig vom Navigationsmodus überwacht das Notfallsystem alle Detektionen gegen einen engeren Schwellwert (`COLLISION_DISTANCE_THRESHOLD` = 1,5 m). Bei Unterschreitung wird ein sofortiger Bewegungsstopp ausgelöst. Treten mehr als fünf aufeinanderfolgende Kollisionswarnungen auf, leitet das System eine automatische Notlandung ein.

Konservative Stopp-Strategie. Auf aktive Ausweichmanöver wurde verzichtet, da diese ohne 3D-Rekonstruktion zur Laufzeit ein erhöhtes Kollisionsrisiko bergen. Die Drohne stoppt und wechselt in `LOITER`; eine Erweiterung um pfadplanungsbasierte Strategien wird in Abschnitt 6.3 diskutiert.

5 Evaluation und Tests

5.1 Test-Aufbau

5.1.1 Rahmenbedingungen und Einschränkungen

Aufgrund der in Abschnitt 6.2.2 dokumentierten ESC-Kalibrierungsproblematik konnten keine Flugtests durchgeführt werden. Die Evaluation konzentriert sich daher auf die *bodengebundene Validierung* der Software-Pipeline. Die Kerninnovation, der hybride Edge-Offline-Ansatz, ist davon nicht betroffen und kann vollständig evaluiert werden.

5.1.2 Experimentelles Hardware-Setup

Alle Tests erfolgten stationär mit `MOTORS_ARMED = False` und `TEST_MODE = True`. Die *Edge-Testumgebung* umfasst:

- Raspberry Pi 5 (8 GB RAM) mit Hailo-8L AI HAT+ (13 TOPS, INT8) als Inferenz-Beschleuniger (vgl. Abschnitt 4.3.1)
- Pi Camera v3 Wide über MIPI-CSI-2 mit 1920×1080 px bei 30 FPS
- MATEK H743-WLITE Flight Controller über UART (`/dev/ttyAMA0`, 57600 Baud), im Disarmed-Zustand mit ArduCopter 4.8 (vgl. Abschnitt 4.2.5)

Für die Navigations- und Safety-Tests (T4) wurde anstelle des realen Flight Controllers ein `MockFlightController` eingesetzt, der deterministische Telemetriedaten liefert und Steuerkommandos ohne physische Auswirkung entgegennimmt. Dies ermöglicht die systematische Validierung aller Sicherheits-Schwellenwerte unabhängig vom FC-Zustand.
Die *Offline-Testumgebung* besteht aus einem Laptop mit NVIDIA GeForce GTX 1650 Ti (4 GB VRAM), auf dem die DA3-Tiefenanalyse mit PyTorch 2.6.0 und CUDA 12.4 durchgeführt wurde (vgl. Abschnitt 4.5.1).

5.1.3 Übersicht der Testszenarien

Tabelle 5.1 gibt einen Überblick über die durchgeführten Testszenarien. Sämtliche Tests sind so konzipiert, dass sie ohne aktive Flugsteuerung durchführbar sind.

5.1.4 Methodik zur Ground-Truth-Ermittlung

Da keine Tiefensensorik (LiDAR, Stereo) und keine Flugtests verfügbar sind, stützt sich die Evaluation auf drei komplementäre Ansätze:

Tabelle 5.1: Übersicht der Testszenarien: Alle Tests wurden bodengebunden ohne aktive Motoren durchgeführt.

ID	Szenario	Typ	Messgröße
T1	YOLO-Inferenz (Hailo vs. CPU)	Live-Kamera	FPS, Inferenzzeit
T2	MAVLink-Kommunikation	FC/SITL	Telemetrie, Modes
T3	Datenlogger (Video + Telemetrie)	Live-Kamera	Synchronisation, Durchsatz
T4	Navigations- & Sicherheitslogik	Mock-FC	Steuerkommandos, Modi
T5	Integration (Gesamtpipeline)	Live-Kamera + Mock	Pipeline-Latenz, FPS
T6	Offline-Tiefenanalyse (DA3 Multi-View)	Aufzeichnung	Tiefenqualität, 3D-Rekonstruktion

1. **Manuelle Maßband-Referenz (BBox-Proxy):** Für die Validierung der perspektivischen Distanzschätzung (vgl. Abschnitt 4.5.1) wurde eine Testperson in definierten Abständen (1 m, 2 m, 3 m, 5 m) vor der statisch montierten Kamera positioniert und die gemessene mit einem handelsüblichen Maßband erfasste Distanz gegen den BBox-Proxy-Wert verglichen. Die Genauigkeit dieser Referenzmessung beträgt ± 5 cm.

2. **Deterministische Mock-Szenarien:** Für die Navigations- und Sicherheitslogik (Abschnitt 5.2.3) dienen die programmatisch definierten Eingabewerte des `MockFlightController` als exakte Ground Truth. Die erwarteten Systemreaktionen (Velocity-Kommandos, Moduswechsel) sind deterministisch ableitbar und erfordern keine Sensormessung.

3. **Manuelle Inspektion der Offline-Tiefenkarten:** Die qualitative Bewertung der DA3-Tiefenschätzung erfolgte durch visuelle Inspektion der erzeugten Depth Maps und 3D-GLB-Modelle. Dabei wurde geprüft, ob die räumliche Anordnung der rekonstruierten Szene plausibel ist (Vordergrund näher als Hintergrund, Kantentreue an Objektgrenzen).

Die Ergebnisse sind daher als *funktionale Validierung* zu verstehen, nicht als metrische Genauigkeitsbewertung. Eine quantitative Ground-Truth-Evaluation gegen LiDAR konnte mangels Sensorik nicht durchgeführt werden.

5.2 Bodengebundene Live-Performance

5.2.1 Inferenzgeschwindigkeit im Vergleich

Der Vision-Test (`test_1_vision.py`) misst die YOLO26n-Inferenz auf dem Hailo-8L über 60 s (1569 Frames, 1280×720 px). Tabelle 5.2 fasst die Ergebnisse zusammen.
Zum Vergleich: Ultralytics nennt für YOLO26n auf dem RPi 5 (NCNN, ARM-CPU) 67,69 ms/Frame [50]; Pereira und Chaari [34] bestätigen vergleichbare NPU-Beschleunigungsfaktoren. Tabelle 5.3 stellt die Konfigurationen gegenüber.

Tabelle 5.2: Inferenzstatistik des YOLO26n-Modells auf dem Hailo-8L NPU (INT8, 1569 Frames, 60 s Messdauer).

Metrik	**FPS**	**Inferenzzeit (ms)**
Mittelwert	46,2	18,5
Median	46,8	18,1
P95	k. A.	19,5
P99	k. A.	25,4
Minimum	15,8	17,6
Maximum	48,9	58,4

Tabelle 5.3: Vergleich der Inferenzleistung: Hailo-8L NPU (INT8) vs. RPi5 CPU (NCNN, Benchmark-Daten aus [50]).

Backend	**Inferenz (ms)**	**FPS (eff.)**	**Speedup**
RPi5 CPU (NCNN FP16)	67,69	$\approx$ 14,8	1,0×
Hailo-8L (INT8)	18,5	46,2	3,7×

Der Speedup von 3,7× hebt die Framerate deutlich über die Echtzeit-Schwelle von 30 FPS [8], bei nur 1,5 W NPU-Leistungsaufnahme [17]. Der Ausreißer von 58,4 ms tritt nur bei HEF-Initialisierung auf; im stabilen Betrieb (ab Frame 50) liegt die P99-Latenz bei 25,4 ms ($\approx$39 FPS worst-case).

5.2.2 MAVLink-Kommunikation und Telemetrie

Der MAVLink-Test (`test_2_mavlink.py`) prüft die UART-Kommunikation (57600 Baud, 20 s). Tabelle 5.4 zeigt die Ergebnisse.

Tabelle 5.4: MAVLink-Testergebnisse: Verbindung zum MATEK H743-WLITE über UART (20 s, FC nicht armed).

Parameter	**Wert**
Empfangene Nachrichten	24 (2 Nachrichtentypen)
Effektive Nachrichtenrate	1,2 Hz
Flugmodus	`STABILIZE`
Armed-Status	`False`
Position / Attitude / Battery / GPS	`None`

Die 24 Nachrichten entsprechen den `HEARTBEAT`-Nachrichten (1 Hz [29]). Fehlende Telemetriewerte (`None`) resultieren daraus, dass ArduPilot im Disarmed-Zustand die meisten Streams unterdrückt [3]. Der korrekte Modus `STABILIZE` bestätigt die funktionierende UART-Kommunikation.

5.2.3 Navigations- und Sicherheitslogik

Der kombinierte Navigations- und Sicherheitstest (`test_4_navigation_safety.py`) validiert die Steuerungslogik im Mock-Modus, also ohne reale Flugcontroller-Verbindung. Der Test gliedert sich in zwei Teile: die Follow-Me-Regelung (Test 4A) und die Safety-Checks (Test 4B).

Follow-Me-Szenarien. Sieben Szenarien simulieren verschiedene Positionen und Zustände einer Zielperson im Kamerabild. Für jedes Szenario berechnet der `AutonomousNavigator` ein Velocity-Kommando (v_x, v_y, v_z), das an den Mock-FC gesendet wird. Tabelle 5.5 zeigt die Ergebnisse.

Tabelle 5.5: Follow-Me-Szenarien mit erwarteten und gemessenen Velocity-Kommandos (Mock-FC, GUIDED-Modus).

Szenario	v_x	v_y	v_z	**Bewertung**
Person zentriert (640, 360)	0,00	0,00	0,00	✓ korrekt
Person links (200, 360)	0,00	−0,21	0,00	✓ korrekt
Person rechts (1100, 360)	0,00	+0,22	0,00	✓ korrekt
Person weit weg ($d = 8$ m)	1,00	0,00	0,00	✓ korrekt
Person zu nah ($d = 0{,}5$ m)	−1,00	0,00	0,00	✓ korrekt
Person verschwunden	0,00	0,00	0,00	✓ LOITER
Hindernis ($d = 1$ m)	0,00	0,00	0,00	✓ Stopp

Alle sieben Szenarien liefern die erwarteten Velocity-Kommandos: laterale Korrektur bei seitlicher Person, Annäherung/Rückzug bei falscher Distanz, LOITER bei Target-Verlust und Stopp bei Hindernis.

Safety-Checks. Der zweite Testteil validiert das `EmergencySystem` mit sieben simulierten Telemetrie-Zuständen. Die Sicherheitslogik folgt der in Abschnitt 3.5 beschriebenen vierstufigen Failsafe-Kaskade, deren Schwellenwerte in der Firmware konfiguriert sind [5]. Tabelle 5.6 zeigt die Resultate.

Tabelle 5.6: Safety-Checks des EmergencySystem mit simulierten Telemetriedaten (alle 7 Tests bestanden).

Szenario	**Simulierte Bedingung**	**Erwartet**	**Ergebnis**
Normalbetrieb	16,0 V, GPS Fix, 5 m	safe	✓
Battery Low	13,5 V	safe (Warnung)	✓
Battery Critical	12,0 V	unsafe → RTL	✓
GPS Lost	Kein Fix	unsafe	✓
Geofence: Höhe	50 m Altitude	unsafe	✓
RC Override	Modus STABILIZE	safe (Pilot)	✓
Collision	Hindernis 0,5 m	unsafe	✓

Alle 14 Tests (7 Follow-Me + 7 Safety) bestehen. Da der Test auf einem Mock-FC basiert, ist die Umsetzung durch ESC und Motoren damit *nicht* validiert.
Abbildung 5.1 zeigt die georeferenzierte Folium-Karte eines Testlaufs, die die korrekte Synchronisation zwischen Vision-Daten und simulierter Telemetrie belegt.

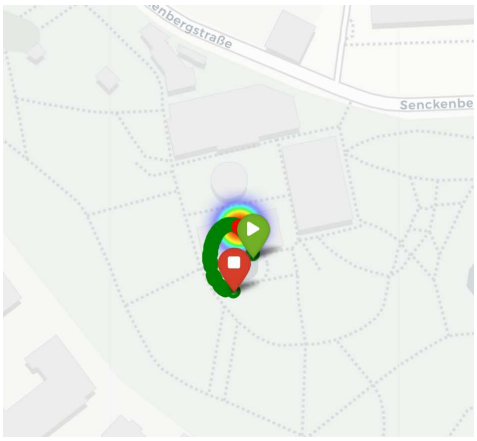


Abbildung 5.1: Georeferenzierte Folium-Karte eines Testlaufs mit MockFlightController und realer YOLO-Inferenz.

5.2.4 Pipeline-Integration

Der Integrationstest (`test_5_integration.py`) verbindet alle Teilsysteme (Kamera → YOLO → Tracking → Safety → Navigation → Logging) über 60 s mit Mock-FC und realer Hailo-Inferenz. Tabelle 5.7 zeigt die Ergebnisse.

Tabelle 5.7: Ergebnisse des Pipeline-Integrationstests (60 s, Hailo-8L, Mock-FC, reale Kamera).

Metrik	**Wert**
Verarbeitete Frames	1529
Mittlere Framerate	46,5 FPS
Mediane Framerate	47,0 FPS
Velocity-Kommandos	528
Kommandorate	8,8 cmd/s

Die Framerate ist nahezu identisch zum isolierten Vision-Test (46,5 vs. 46,2 FPS), der Overhead durch Tracking, Safety und Navigation ist vernachlässigbar. Die *Glass-to-Command-Latenz* (Frame-Akquise ≈2 ms + Inferenz 18,5 ms + Tracking/Navigation <1 ms) beträgt ≈21,5 ms und liegt deutlich unter der 100 ms-Schwelle für reaktive Drohnensteuerung [30].

5.3 Offline-Videoanalyse

5.3.1 Qualitative Analyse der Tiefenkarten

Die Offline-Pipeline verarbeitete 56 Frames (16,8 s, 3 FPS Extraktion) in 7 Batches (8 Frames, 336 px) auf der GTX 1650 Ti. Tabelle 5.8 zeigt die Ergebnisse.

Tabelle 5.8: Ergebnisse der DA3 Multi-View Tiefenschätzung auf dem Testvideo (56 Frames, 7 Batches).

Metrik	**Wert**
Gesamtverarbeitungszeit	290,7 s
Zeit pro Frame (Mittel)	5,19 s
Forward Pass (Batch 1, inkl. Download)	4,9 s
Forward Pass (Batch 2–7, Mittel)	33,8 s
Minimale Tiefe	0,138 m
Maximale Tiefe	6,939 m
Mittlere Tiefe	1,131 m
Standardabweichung	0,986 m
GLB-Dateigröße (pro Batch)	≈ 4,9 MB

Die Tiefenkarten zeigen klare räumliche Strukturierung: Nahbereich (unter 1 m) wird von Hintergrund (über 5 m) separiert. Die GLB-Modelle erlauben interaktive 3D-Inspektion (Abbildung 5.2).

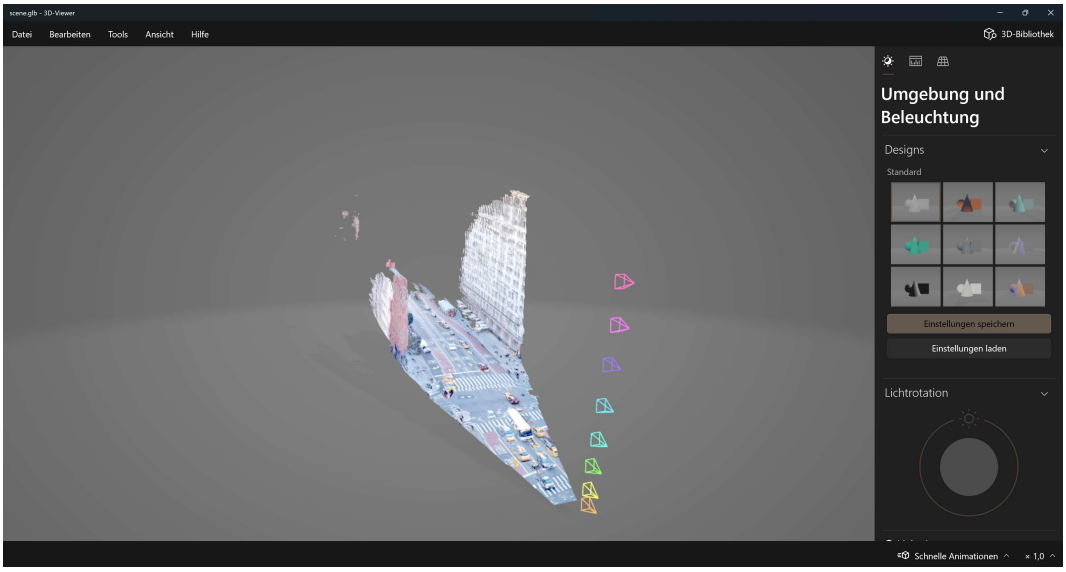


Abbildung 5.2: 3D-Punktwolke eines DA3-Batches, visualisiert als GLB-Modell im Windows 3D-Viewer. Die farbigen Frustum-Symbole markieren die geschätzten Kameraposen der einzelnen Frames. Das zugehörige Quellvideo befindet sich im digitalen Anhang.

5.3.2 Edge- und Offline-Verarbeitung im Vergleich

Der hybride Ansatz dieser Arbeit trennt bewusst die Echtzeit-Verarbeitung auf der Drohne (Edge) von der nachgelagerten wissenschaftlichen Analyse (Offline). Tabelle 5.9 stellt die beiden Modi gegenüber.

Tabelle 5.9: Gegenüberstellung der Edge- und Offline-Verarbeitung im Hybrid-Ansatz.

Eigenschaft	**Edge (Hailo-8L)**	**Offline (DA3)**
Modell	YOLO26n (INT8)	DA3-LARGE-1.1
Aufgabe	Personenverfolgung	3D-Tiefenrekonstruktion
Tiefenquelle	BBox-Größe als Proxy	Multi-View Depth Maps
Latenz	$\approx$ 21,5 ms/Frame	$\approx$ 5,19 s/Frame
Hardware	RPi 5 + Hailo-8L (13 TOPS)	GPU (GTX 1650 Ti, 4 GB)
Tiefengenauigkeit	Relativ (nah/fern)	Metrisch (0,1–7 m)
3D-Ausgabe	Nein	GLB-Punktwolke

Die Ergebnisse bestätigen die Designentscheidung: Der BBox-Proxy genügt für die binäre Echtzeit-Entscheidung (nah/fern), während DA3 metrisch genaue Tiefe für die Nachanalyse liefert.

5.3.3 Einfluss der GPU-Speicherbeschränkung

Die 4 GB VRAM erzwingen Batching (8 Frames), wodurch DA3 räumliche Korrespondenzen nur innerhalb eines Batches herstellt. Die reduzierte Auflösung (336 px statt 504 px) verringert die Detailschärfe bei feinen Strukturen. Abschnitt 4.5.1 dokumentiert die getesteten Konfigurationen.

5.4 Fehleranalyse der Objekterkennung

5.4.1 Detection Rate unter variierenden Bedingungen

Zur Analyse der Erkennungsqualität wurde das aufgezeichnete Testvideo (289 Frames) mit der Offline-Pipeline auf dem Laptop (CPU, PyTorch FP32) verarbeitet. YOLO26n erkennt dabei 351 Objekte in 11 COCO-Klassen bei einem Konfidenz-Schwellenwert von 0,5. Tabelle 5.10 schlüsselt die Erkennungen nach Klasse auf.

Tabelle 5.10: Erkennungsstatistik der Offline-Analyse (289 Frames, YOLO26n, Konfidenz $\geq$ 0,5).

Klasse	**Anzahl**	**Anteil**
person	184	52,4 %
Sonstige (7 Klassen)	167	47,6 %
Gesamt	351	100 %

Die durchschnittliche Konfidenz über alle Detektionen beträgt 0,536, was knapp über dem Schwellenwert liegt. Die 184 Personenerkennungen verteilen sich über 289 Frames, was einer Detection Rate

von 63,7 % entspricht. In den restlichen Frames ist entweder keine Person im Bild oder die Konfidenz liegt unter dem Schwellenwert.
Die verbleibenden 167 Detektionen verteilen sich auf Klassen wie *wine glass*, *cat*, *bottle*, *airplane*, *bus*, *refrigerator* und *cup*. Diese sind im Drohnenkontext durchweg als Fehlklassifikationen zu werten und werden in Abschnitt 5.4.2 analysiert.

5.4.2 Analyse von False Positives und Negatives

Die in Tabelle 5.10 dokumentierten Nicht-Personen-Detektionen (47,6 %) stellen *False Positives* im Kontext der Follow-Me-Anwendung dar. Für den Live-Betrieb sind diese jedoch unkritisch, da der `select_target()`-Algorithmus ausschließlich Detektionen der Klasse *person* berücksichtigt; alle anderen Klassen werden vor der Navigationslogik ausgefiltert.
Die identifizierten Fehlerquellen lassen sich in vier Kategorien einteilen:

1. **Distanzabhängigkeit:** Bei Entfernungen über 8 m nimmt die BBox-Größe auf unter 50×100 px ab, was unterhalb der effektiven Auflösung des YOLO26n-Nano-Modells liegt. Dies führt zu *False Negatives*: Die Person ist im Frame vorhanden, wird aber nicht erkannt.

2. **Bewegungsunschärfe:** Schnelle Drohnenbewegungen erzeugen Motion Blur, der die Kantenqualität im Frame reduziert. Die rollierende Belichtung der Pi Camera v3 verstärkt diesen Effekt bei horizontalem Schwenk (*Rolling Shutter*).

3. **Verdeckung:** Teilweise verdeckte Personen (z. B. hinter Vegetation oder Fahrzeugen) senken die Konfidenz unter den Schwellenwert, was zu temporären Target-Lost-Events führt.

Im Live-System wird der Effekt von False Negatives durch den ByteTrack-Algorithmus abgemildert, der ein getracktes Target über mehrere Frames ohne neue Detection beibehält. Erst nach `TARGET_LOST_LOITER_FRAMES` = 30 konsekutiven Frames ohne Re-Identifikation wechselt die Drohne in den LOITER-Modus.

5.5 Validierung der Sicherheitssysteme

Die in Abschnitt 5.2.3 dokumentierten Safety-Tests validieren die *Software-Ebene* der Sicherheitskaskade (Stufe 2). Hardware-Failsafe (Stufe 3) und Pilotenübersteuerung (Stufe 4) konnten mangels ESC-Kalibrierung nicht verifiziert werden. ArduPilot implementiert auf Firmware-Ebene eigenständige Failsafes (RC-Verlust → RTL, EKF-Fehler → LAND) [5, 4]. Wanner et al. [54] betonen die Notwendigkeit redundanter Stufen, da softwarebasierte Checks durch Sensorausfälle umgangen werden können.

5.6 Diskussion der Ergebnisse

5.6.1 Erreichte Performance-Ziele

Tabelle 5.11 fasst die erreichten Performance-Ziele zusammen und setzt sie in Bezug zu den in Abschnitt 3.1 definierten Anforderungen.

Tabelle 5.11: Zielerreichung der Evaluation: Soll- vs. Ist-Werte der Kernmetriken.

Metrik	**Soll**	**Ist**	**Erreicht**
Inferenz-FPS (Hailo-8L)	≥ 15 (NFR1)	46,2	✓
Glass-to-Action (NFR2)	< 200 ms	≈ 65 ms	✓
NPU-Speedup vs. CPU	$> 2\times$	$3{,}7\times$	✓
Follow-Me Steuerlogik	korrekt	7/7	✓
Safety-Checks	korrekt	7/7	✓
DA3 Tiefenkarten	verwertbar	0,1–7 m	✓
Synchron. Datenaufnahme	funktional	MKV + CSV	✓
Flugtestvalidierung	erforderlich	nicht möglich	×

Sämtliche software-seitigen Performance-Ziele werden erreicht oder übertroffen (Inferenz dreifach über NFR1-Schwelle von 15 FPS, Glass-to-Action ≈ 65 ms vs. NFR2-Grenze von 200 ms). Einzig die Flugtestvalidierung fehlt.

5.6.2 Identifizierte Limitierungen

Die wesentlichen Limitierungen des Systems lassen sich in drei Kategorien einordnen:

Fehlende Flugtests. Ohne Motorsteuerung bleibt die *Glass-to-Action-Latenz* (inkl. Motor-Regelung), die Regelungsgenauigkeit des Follow-Me-Modus und die Wirksamkeit der Obstacle Avoidance bei realer Geschwindigkeit unvalidiert.

Monokulare Distanzschätzung. Der BBox-Proxy basiert auf durchschnittlicher Personengröße und weicht bei Kindern, sitzenden Personen oder unüblichen Haltungen systematisch ab. Metrisch genaue Distanzmessung erfordert Stereo-Kameras oder LiDAR.

Eingeschränkter Testumfang. Die Evaluation basiert auf wenigen Szenen bei Tageslicht. Eine statistisch robuste Bewertung unter variierenden Lichtbedingungen und Entfernungen wäre für Produktreife notwendig.

5.6.3 Benchmarking gegenüber kommerziellen Lösungen

Tabelle 5.12 stellt den Prototypen den kommerziellen Systemen DJI ActiveTrack [11] und Skydio 2 [45] auf Software-Ebene gegenüber.

Tabelle 5.12: Vergleich des Prototyps mit kommerziellen Follow-Me-Systemen (Software-Ebene).

Eigenschaft	**Prototyp**	**DJI ActiveTrack**	**Skydio 2**
Vision-Hardware	Hailo-8L (13 TOPS)	Proprietärer ASIC	NVIDIA TX2
Objekterkennung	YOLO26n	Proprietär	Proprietär
Inferenz-FPS	46,2	k. A.	k. A.
Obstacle Avoidance	Mono (BBox-Proxy)	Stereo + IR	6× 4K Fisheye
Tiefenanalyse Offline	DA3 Multi-View	Nein	Nein
Open Source	Ja	Nein	Nein
Materialkosten	$\approx 1\,100$ €	$> 1\,000$ €	$> 2\,000$ €

Der Prototyp erreicht vergleichbare Inferenzgeschwindigkeit, ist jedoch bei der Hindernisvermeidung (monokulare Kamera vs. Multi-Sensor-Fusion) im Nachteil. Der DA3-Offline-Ansatz bietet Funktionalität, die kommerzielle Systeme nicht bereitstellen. Wesentliche Vorteile: vollständig offener Quellcode und deutlich geringere Materialkosten.

6 Diskussion und Ausblick

6.1 Beantwortung der Forschungsfragen

Die in Abschnitt 1.3 formulierten Forschungsfragen werden im Folgenden auf Basis der in Kapitel 5 dargestellten Ergebnisse beantwortet.

F1: Inwiefern ermöglicht eine hybride Architektur aus echtzeitfähiger Edge-Inferenz und nachgelagerter Offline-Analyse die Umsetzung einer autonomen Vision-Pipeline auf kostengünstiger, frei verfügbarer Hardware? Die Evaluation zeigt, dass die Aufteilung in eine echtzeitfähige Edge-Komponente (YOLO26n auf Hailo-8L) und eine nachgelagerte Offline-Analyse (DA3 auf Consumer-GPU) eine funktionsfähige Vision-Pipeline auf Hardware im Gesamtwert von ca. 1,100 € (vgl. Anhang A.1) ermöglicht. Die Edge-Pipeline erreicht 46,2 FPS bei einer Glass-to-Action-Latenz von ca. 65 ms und erfüllt damit die Echtzeitanforderung (NFR2: < 200 ms) deutlich (vgl. Tabelle 5.11). Die Offline-Tiefenanalyse liefert metrische 3D-Rekonstruktionen, die auf der Edge-Hardware nicht in Echtzeit berechenbar wären. Der Hybrid-Ansatz vermeidet somit den Zielkonflikt zwischen Echtzeitfähigkeit und Analysetiefe, der bei rein Edge-basierten Systemen unvermeidlich ist.
Die Einschränkung besteht darin, dass die praktische Validierung im Flugbetrieb aussteht. Die Software-Pipeline ist funktional verifiziert, die tatsächliche Eignung für autonome Flugnavigation konnte jedoch nicht empirisch belegt werden.

F2: Welche Leistungsfähigkeit erreicht die YOLO-basierte Objekterkennung auf einem Hailo-8L Neural Processing Unit im Vergleich zu einer reinen CPU-Ausführung auf dem Raspberry Pi 5? Der Hailo-8L erreicht 18,5 ms Inferenz (46,2 FPS) für YOLO26n in INT8, ein Speedup von 3,7× gegenüber CPU-NCNN (67,69 ms [50]). Die P99-Latenz von 25,4 ms (39 FPS worst-case) liegt deutlich über der 30-FPS-Schwelle [8], bei nur 1,5 W Leistungsaufnahme [17].

F3: Welche Qualität der 3D-Tiefenrekonstruktion lässt sich mit Depth Anything 3 im Multi-View-Modus auf einer Consumer-GPU mit begrenztem Videospeicher erzielen? DA3 erzeugt auf der GTX 1650 Ti (4 GB) verwertbare Tiefenkarten (0,1–7 m) mit klarer Vordergrund-Hintergrund-Separation. Die VRAM-Beschränkung erzwingt Batch-Fragmentierung (8 statt 56 Frames) und reduzierte Auflösung (336 px statt 504 px), was die globale 3D-Konsistenz beeinträchtigt (vgl. Abschnitt 5.3.3). Mit 5,19 s/Frame ist der Schritt ausschließlich offline geeignet. GPUs mit $\geq$12 GB VRAM würden deutlich bessere Ergebnisse liefern.

F4: Welche praktischen Herausforderungen ergeben sich beim Aufbau eines KI-gestützten Drohnen-Prototyps mit integriertem Companion Computer, und welche Auswirkungen ha-

ben diese auf den Evaluationsumfang? Die Herausforderungen umfassen iterative Komponentenbeschaffung, fehlerhafte 3D-Drucke, kalte Lötstellen und kaskadierende FC-Konfigurationsfehler (GPS-Ressourcenkonflikt, AM32-ESC-Protokoll, fehlende SD-Karte). Die nicht abgeschlossene ESC-Kalibrierung verhinderte sämtliche Flugtests. Bei Hardware-Integrationsprojekten mit hohem Miniaturisierungsgrad ist ein Zeitpuffer von 30 % oder mehr einzuplanen.

6.2 Kritische Reflexion

In diesem Abschnitt werden die Ergebnisse kritisch eingeordnet und die aufgetretenen Herausforderungen dokumentiert.

6.2.1 Gewicht und Flugleistung

Das gemessene Abfluggewicht von 815 g (Abbildung 6.1) resultiert in einem Thrust-to-Weight-Ratio von ca. 1,8:1, was unterhalb der für stabile Flugmanöver empfohlenen 2:1 liegt [42]. Die hohe Komponentendichte, bedingt durch die Integration von Companion Computer, KI-Beschleuniger und dediziertem BEC in einen kompakten 3,5-Zoll-Rahmen, lässt wenig Spielraum für Gewichtsoptimierung. Für den geplanten Follow-Me-Einsatz bei Schritttempo und geringem Wind ist das Verhältnis ausreichend, schränkt jedoch die Operabilität bei widrigen Wetterbedingungen und schnellen Ausweichmanövern ein.

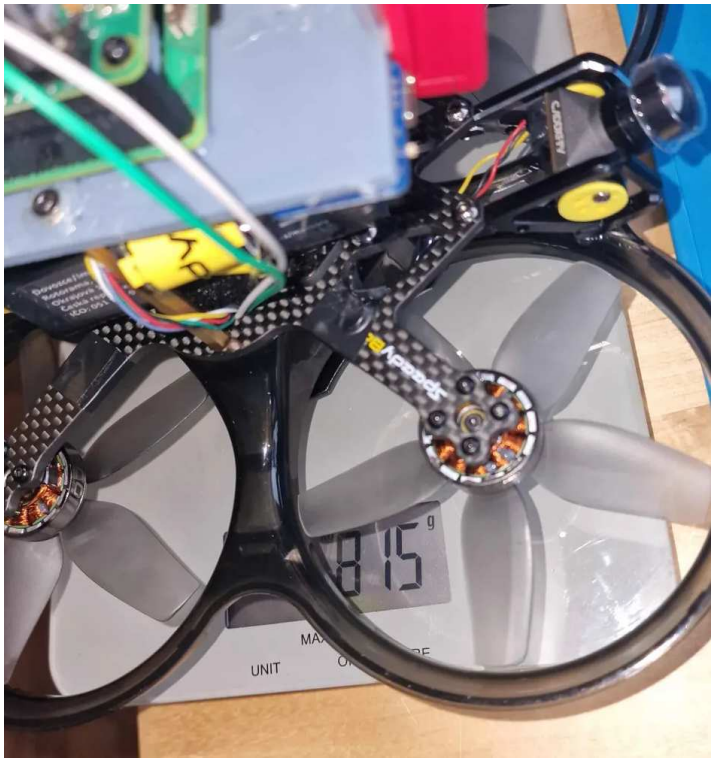


Abbildung 6.1: Abfluggewicht des vollständig aufgebauten Prototyps inklusive Akku: 815 g.

6.2.2 Herausforderungen im Prototypenbau

Der Aufbau des Prototyps erwies sich als deutlich zeitintensiver als geplant. Die Hardware-Integration war der dominierende Engpass gegenüber der Software-Entwicklung.

6.2.2.1 Zeitplanung und iterative Komponentenbeschaffung

Der Zeitaufwand für den physischen Aufbau wurde erheblich unterschätzt. Erst während der Integration zeigte sich der Bedarf an zusätzlichen Bauteilen (Adapterkabel, Stecker, Befestigungselemente), die jeweils separate Bestellvorgänge mit mehrtägigen Lieferzeiten nach sich zogen. Hinzu kam eine krankheitsbedingte Ausfallzeit.
Erkenntnis: Eine vollständige Stückliste sollte frühzeitig finalisiert und beschafft werden. Ein Zeitpuffer von mindestens 30 % ist realistisch.

6.2.2.2 Additive Fertigung und mechanische Konstruktion

Für die Integration wurden mehrere 3D-gedruckte Bauteile benötigt, konstruiert in *Autodesk Fusion* und gedruckt auf einem *Bambu Lab X1E*. Der iterative Charakter zeigte sich in mehreren Revisionen: falsche Bohrungsmaße an der FC-ESC-Adapterplatte, nicht passende Schraubenlöcher der RPi-Plattform. Für die FC-ESC-Verbindung wurde schließlich ein Community-Modell[1] übernommen, das allerdings bei Wartung eine vollständige Stack-Demontage erfordert.

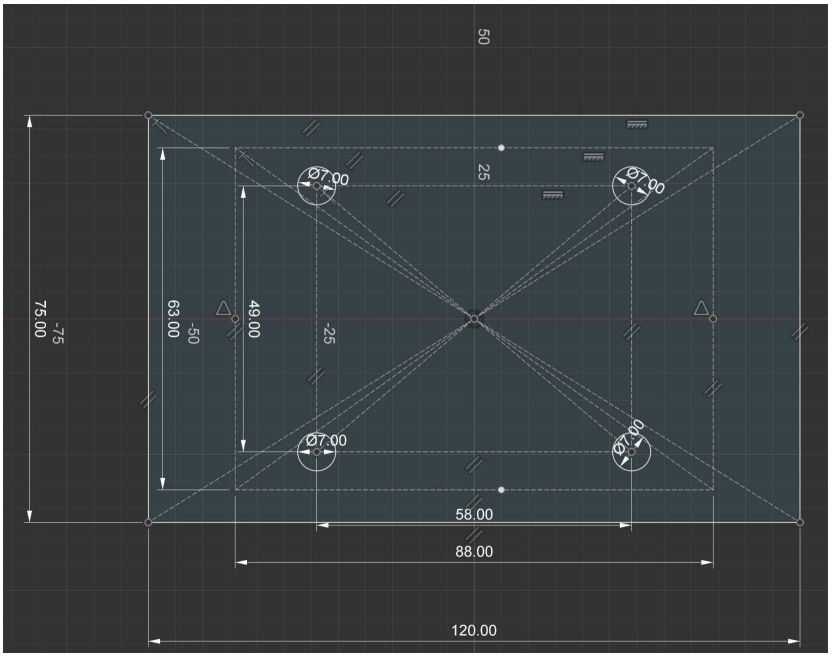


Abbildung 6.2: Bemaßung der 3D-gedruckten Montage-Plattform in Autodesk Fusion. Die iterative Anpassung der Bohrungspositionen erforderte mehrere Druckdurchgänge.

[1] https://www.printables.com/model/656305-all-in-one-flight-controller-adapter-complete-set

Erkenntnis: Vor dem Entwurf eigener Bauteile sollte eine Recherche bestehender Community-Designs erfolgen. Mechanische Maße sind mit einem Messschieber zu verifizieren.

6.2.2.3 Löttechnik und Kabelmanagement

Die Verlötung der Akku-Kabel am ESC erforderte durch die hohe thermische Masse erhöhte Löttemperaturen (ca. 400 °C). Initiale kalte Lötstellen (Abbildung 6.3) mussten nachgebessert werden, wobei die Zugänglichkeit nach Stack-Zusammenbau stark eingeschränkt war.

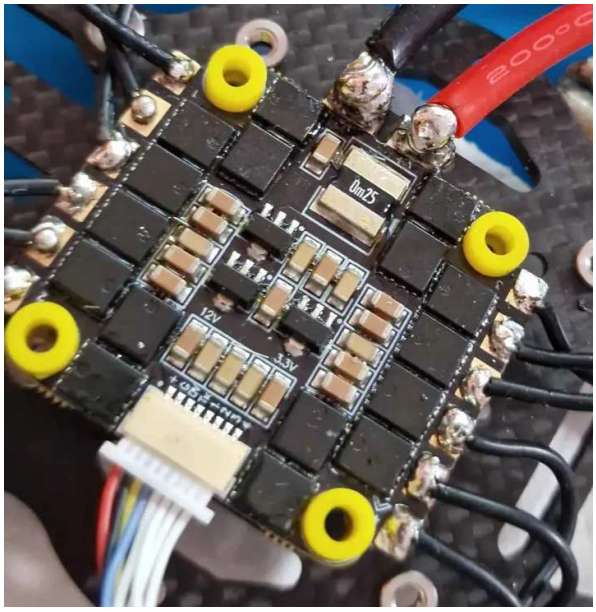


Abbildung 6.3: Dokumentation der initialen Lötverbindungen am T-Motor Velox V50A ESC. Sichtbar sind kalte Lötstellen, ungleichmäßiger Zinnauftrag und eine mangelhafte Benetzung der Pads, die im weiteren Projektverlauf nachgebessert wurden.

Erkenntnis: Elektrische Funktions- und Kontinuitätstests der ESC- und Flight-Controller-Verbindungen sollten *vor* dem mechanischen Zusammenbau des Gesamtsystems erfolgen. Dies ermöglicht im Fehlerfall einen deutlich einfacheren Zugang zu den relevanten Kontaktstellen.

6.2.2.4 Flight-Controller-Konfiguration

Die Inbetriebnahme des Flight Controllers war von kaskadierenden Konfigurationsfehlern begleitet.

GPS-Konfiguration. Die fehlerhafte Anzeige `No GPS` resultierte aus einem Ressourcenkonflikt: `SERIAL2` und `SERIAL3` waren simultan auf GPS konfiguriert. Die Fehlersuche führte zunächst zu unnötigem Umlöten und der Vermutung magnetischer Störungen, bevor die doppelte Protokollzuweisung als Ursache identifiziert wurde.

Motorenrichtung und ESC-Protokoll. Drei von vier Motoren drehten zunächst falsch. Die Korrektur über *BLHeli Suite* scheiterte, da der ESC das `AM32`-Protokoll verwendet. Erst der *AM32 Configurator* ermöglichte die Richtungsumkehr.

ESC-Kalibrierung und Testlimitierung. Die Synchronisation des Throttle-Bereichs zwischen FC und ESC konnte nicht zuverlässig abgeschlossen werden. Die Motoren zeigten unvorhersehbares Drehzahlverhalten. Da die Presssitz-Propeller nur unter Beschädigungsrisiko demontierbar sind, war kein sicherer Motortest möglich. Eine erneute Kalibrierung überschritt das verbleibende Zeitbudget, sodass alle Flugtests entfielen.

Aussagekraft der bodengebundenen Tests. Die zentrale Fragestellung betrifft die Machbarkeit einer hybriden Edge-Offline-Vision-Pipeline, nicht die Flugregelung. Inferenzgeschwindigkeit (46,2 FPS), Pipeline-Latenz (21,5 ms) und Erkennungsqualität sind hardwaregebundene Kenngrößen, die sich im Flug nicht verändern. Alle 14 Mock-Tests bestätigten korrekte Kommandos und Sicherheitsreaktionen. Die Datenakquise-Pipeline erzeugte im Außeneinsatz synchronisierte Aufnahmen mit korrekten GPS-Koordinaten, die erfolgreich durch die DA3-Pipeline verarbeitet wurden. Es fehlt die Validierung der physischen Umsetzung durch Motoren und deren Auswirkung auf die Flugbahn.

Weitere Konfigurationshürden. Zu Beginn wurde der FC ohne SD-Karte betrieben, was die Parameterspeicherung verhinderte. Die Umrüstung der SD-Karte vom RPi zum FC erforderte eine Neuinstallation des RPi-Betriebssystems. Die Akkuspannungsanzeige zeigte fehlerhafte Werte, korrigiert durch Kalibrierung von `BATT_VOLT_MULT`:

$$\text{BATT\_VOLT\_MULT}_{\text{neu}} = \frac{V_{\text{gemessen}}}{V_{\text{angezeigt}}} \times \text{BATT\_VOLT\_MULT}_{\text{alt}} \tag{6.1}$$

Erkenntnis: Eine schrittweise Inbetriebnahme mit Einzelvalidierung jeder Subsystemkomponente und systematischer Protokollierung der Parameter hätte die Fehlersuche beschleunigt.

6.2.3 Grenzen der Offline-Analyse

Die GTX 1650 Ti (4 GB VRAM) erzwingt kleinere Batches (8 statt 56 Frames) und reduzierte Auflösung (336 px statt 504 px). Daraus ergeben sich zwei Einbußen:

1. **Batch-Fragmentierung:** Jeder Batch erzeugt eine eigenständige 3D-Punktwolke; übergreifende räumliche Korrespondenzen fehlen.
2. **Auflösungsverlust:** Feine Strukturen und Objektkanten leiden unter der reduzierten Pixelzahl.

Auf einer GPU mit ≥12 GB VRAM (RTX 3090, A100) könnten alle Frames in einem Durchlauf bei voller Auflösung verarbeitet werden.

6.2.4 Rechtliche Rahmenbedingungen und Ethik

Der Prototyp (815 g) fällt unter die EU-VO 2019/947 [13] in die offene Kategorie (A1/A3, unter 2 kg). Ohne CE-Kennzeichnung darf er *nicht* über unbeteiligten Personen fliegen. Der Fernpilot benötigt mindestens den EU-Kompetenznachweis (A1/A3) [27]. Die Kamera erfasst kontinuierlich Video; für diese Arbeit wurden nur kontrollierte Szenarien mit eingewilligten Personen verwendet. Bei produktivem Einsatz wären gemäß DSGVO [14] Gesichtsverpixelung und Speicherfristen erforderlich.

Anforderungen für Hochschul-Flugbetrieb. Für Testflüge auf dem Campus wären UAS-Betreiberregistrierung, Haftpflichtversicherung (§ 37 LuftVG) und Genehmigung der Hochschulleitung erforderlich. Da der Campus die A3-Mindestabstände (150 m zu Wohngebieten) nicht erfüllt, wäre eine Betriebsgenehmigung des RP Kassel mit SORA-Risikobewertung nötig [13].

6.3 Ausblick

Auf Basis der gewonnenen Erkenntnisse lassen sich Weiterentwicklungen in drei Zeithorizonten identifizieren.

Kurzfristig (ESC-Kalibrierung und Flugtests). Priorität: Throttle-Bereich via AM32 Configurator synchronisieren, Propeller vor Motortest mit Abzieher demontieren, alle vier Motoren einzeln validieren. Ein kontrollierter Schwebeflug (niedrige Höhe, Sicherheitsleine) würde zeigen, ob das 1,8:1 Thrust-to-Weight-Ratio ausreicht. Erst danach Follow-Me-Tests mit Glass-to-Action-Latenz und PID-Tuning unter Flugdynamik.

Mittelfristig (Sensorik und Modularität). Eine Stereo-Kamera oder ein ToF-Sensor (z. B. VL53L5CX) würde die BBox-basierte Distanzschätzung durch metrische Tiefenwerte ersetzen. Eine Migration auf ROS 2 würde einzelne Komponenten als unabhängige Nodes mit standardisierten Topics entkoppeln.

Langfristig (KI-Erweiterungen). Mit zunehmender NPU-Unterstützung für größere Modelle könnte die Tiefenschätzung direkt auf der Drohne erfolgen, wenn leichtgewichtige Depth-Modelle auf dem Hailo-8L quantisierbar werden. Darüber hinaus bietet der Einsatz von Reinforcement Learning zur Optimierung der Flugtrajektorien anstelle des aktuellen PID-Reglers das Potenzial, komplexere Verfolgungsszenarien (z. B. Zickzack-Bewegungen, Gruppenszenen) robuster zu bewältigen.

7 Zusammenfassung

7.1 Kurzzusammenfassung der Ergebnisse

Diese Arbeit konzipierte und implementierte einen autonomen Drohnen-Prototyp, der eine Zielperson mittels kamerabasierter Objekterkennung verfolgen kann. Das System kombiniert eine echtzeitfähige Edge-Pipeline auf dem Raspberry Pi 5 mit Hailo-8L NPU (YOLO26n, 46,2 FPS, 21,5 ms Latenz) mit einer nachgelagerten Offline-Tiefenanalyse auf Basis von Depth Anything 3. Sämtliche softwareseitigen Performance-Ziele wurden erreicht oder übertroffen.
Die ESC-Kalibrierung konnte im Projektverlauf nicht abgeschlossen werden, sodass keine Flugtests stattfanden. Die Evaluation beschränkt sich daher auf die bodengebundene Validierung der Software-Pipeline, bei der alle 14 Navigations- und Sicherheitstests bestanden wurden. Die tatsächliche Flugperformance, insbesondere die Regelungsgenauigkeit des Follow-Me-Modus und die Wirksamkeit der Hindernisvermeidung bei realer Geschwindigkeit, bleibt unvalidiert.

7.2 Haupterkenntnisse der Arbeit

Die Untersuchung zeigt, dass der Hailo-8L NPU die YOLO-Inferenz um den Faktor 3,7 gegenüber der CPU beschleunigt und stabile 46 FPS bei einer minimalen Leistungsaufnahme von 1,5 W ermöglicht. Damit ist belegt, dass sich dieser Hardware-Stack hervorragend für batteriegetriebene Echtzeitanwendungen eignet.
Der hybride Architekturansatz, Echtzeit-Detektion on-board und metrische Rekonstruktion offline, erwies sich als die richtige Designentscheidung. Er löst den Zielkonflikt zwischen Ressourcenbeschränkung und Analysetiefe auf und liefert wissenschaftlich verwertbare 3D-Daten ohne die Flugsicherheit durch zu hohe Rechenlast zu gefährden.
Eine wesentliche Erkenntnis betrifft das Projektmanagement: Der überwiegende Teil der Laufzeit floss in die physische Systemintegration (3D-Druck, Löttechnik, FC-Konfiguration). Für zukünftige Projekte mit vergleichbarem Scope darf dieser mechanische und elektrotechnische Aufwand keinesfalls unterschätzt werden.

7.3 Wissenschaftlicher und praktischer Beitrag

Die Arbeit stellt ein vollständig dokumentiertes, quelloffenes System bereit, das die Lücke zwischen proprietären Follow-Me-Lösungen und akademischen Prototypen ohne verfügbaren Code schließt. Der Vergleich zwischen Edge-Inferenz (Hailo-8L, INT8) und Offline-Tiefenschätzung (DA3, FP32) auf identischen Videosequenzen liefert quantitative Daten für die Abwägung zwischen Latenz und Analysetiefe auf Consumer-Hardware. Die detaillierte Dokumentation der Hardware-Herausforderungen

bietet zukünftigen Projekten eine realistische Planungsgrundlage.

8 Literaturverzeichnis

[1] ABENDROTH, B. ; BRUDER, R. : Die Leistungsfähigkeit des Menschen für die Fahrzeugführung. Version: 2009. http://dx.doi.org/10.1007/978-3-8348-9977-4_2. In: *Handbuch Fahrerassistenzsysteme*. Vieweg+Teubner, 2009. – DOI 10.1007/978–3–8348–9977–4_2

[2] AI CHIP LINK: *Broadcom BCM2712 Processor – Benchmarks, Tests and Specifications*. https://aichiplink.com/blog/Broadcom-BCM2712-Processor-Benchmarks-Tests-and-Specifications_253. Version: 2024

[3] ARDUPILOT DEVELOPMENT TEAM: *ArduPilot: Open Source Autopilot*. https://ardupilot.org/. Version: 2024

[4] ARDUPILOT DEVELOPMENT TEAM: *EKF Failsafe – Copter Documentation*. https://ardupilot.org/copter/docs/ekf-inav-failsafe.html. Version: 2024

[5] ARDUPILOT DEVELOPMENT TEAM: *Radio Failsafe – Copter Documentation*. https://ardupilot.org/copter/docs/radio-failsafe.html. Version: 2024

[6] BAUN, C. ; BLOCH, T. ; DEEGENER, M. ; HAHM, O. ; KAPPES, M. ; UDDUN SYEED, N. : KI-Drohnen: Ein zweisprachiges Handbuch über Entwurf, Bau und Einsatz von KI-fähigen Drohnen für wissenschaftliche Projekte und Lehre / Frankfurt University of Applied Sciences. Version: 2025. https://www.christianbaun.de/Master_Projekt_WS2526/index.html. – Forschungsbericht

[7] BÜNTE, T. ; BREMBECK, J. ; HO, L. M.: Human Machine Interface Concept for Interactive Motion Control of a Highly Maneuverable Robotic Vehicle. In: *IEEE Intelligent Vehicles Symposium (IV)*, 2014, S. 1–6

[8] CHEN, J. ; RAN, X. : Deep Learning with Edge Computing: A Review. In: *Proceedings of the IEEE* 107 (2019), Nr. 8, S. 1655–1674

[9] CHRISTEN, M. ; GUILLAUME, M. ; JABLONOWSKI, M. ; LENHART, P. ; MOLL, K. : Drohnen als Partner im Luftraum / ZHAW Zentrum für Aviatik. 2018. – Forschungsbericht

[10] CONTI, A. : Fahrassistenzsysteme auf dem Prüfstand: Wie viel Sensorik braucht ein Auto? In: *Journal für Mobilität und Sicherheit* (2023)

[11] DJI TECHNOLOGY CO., LTD.: *DJI ActiveTrack – Intelligent Tracking Technology*. https://www.dji.com/. Version: 2024

[12] DUBINSKY, D. : *YOLO26 Object Detection on Raspberry Pi 5 + Hailo-8L*. https://github.com/DanielDubinsky/yolo26_hailo. Version: 2026

[13] EUROPÄISCHE KOMMISSION: *Durchführungsverordnung (EU) 2019/947 über die Vorschriften und Verfahren für den Betrieb unbemannter Luftfahrzeugsysteme*. https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32019R0947. Version: 2019

[14] EUROPÄISCHES PARLAMENT UND RAT: *Verordnung (EU) 2016/679 (Datenschutz-Grundverordnung)*. https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32016R0679. Version: 2016

[15] GOOGLE LLC: *Google Coral Edge TPU*. https://coral.ai/. Version: 2024

[16] GRÄSSLER, I. u. a.: Integration unbemannter Flugsysteme in cyber-physische Produktionssysteme. In: *Tagungsband Design*. 2020, S. 1–10

[17] HAILO TECHNOLOGIES LTD.: *Hailo-8 AI Accelerator Datasheet*. https://hailo.ai/de/products/ai-accelerators/hailo-8-ai-accelerator/. Version: 2024

[18] HENKE, C. ; TICHY, M. ; SCHNEIDER, S. : Ein modularer Ansatz zur Simulation autonomer Systeme. In: *Tagungsband ASIM*, 2020, S. 1–8

[19] JACOB, B. ; KLINKER, S. ; CHEN, D. ; ZHU, M. ; TANG, M. ; HOWARD, A. ; ADAM, H. ; KALENICHENKO, D. : Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, S. 2704–2713

[20] JOCHER, G. : *YOLOv5 by Ultralytics*. https://github.com/ultralytics/yolov5. Version: 2020

[21] KOUBAA, A. ; QURESHI, B. ; SRITI, M. ; ALLOUCH, A. ; JAVED, Y. ; AL-SULTAN, M. : Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey. In: *IEEE Access* 10 (2022), S. 87632–87654. http://dx.doi.org/10.1109/ACCESS.2022.3198084. – DOI 10.1109/ACCESS.2022.3198084

[22] LACKNER, F. u. a.: Performance-Analyse privater 5G-Campusnetze für industrielle Anwendungen. In: *Tagungsband Kommunikation in verteilten Systemen (KiVS)*, 2024, S. 1–8

[23] LI, K. ; ZHANG, H. ; WANG, Y. ; LEI, J. : Optimized IDW Algorithm for Accurate GPS-IMU Time Synchronization Using Acceleration and Temporal Factors. In: *IEEE Sensors Journal* (2025)

[24] LI, Y. ; FAN, Q. ; HUANG, H. ; HAN, Z. ; GU, Q. : A Modified YOLOv8 Detection Network for UAV Aerial Image Recognition. In: *Drones* 7 (2023), Nr. 5. https://www.mdpi.com/2504-446X/7/5/304

[25] LIN, H. ; CHEN, S. ; LIEW, J. H. ; CHEN, D. Y. ; LI, Z. ; SHI, G. ; FENG, J. ; KANG, B. : Depth Anything 3: Recovering the Visual Space from Any Views. In: *arXiv preprint arXiv:2511.10647* (2025)

[26] LO, L.-Y. ; YIU, C. H. ; TANG, Y. ; YANG, A.-S. ; LI, B. ; WEN, C.-Y. : Dynamic Object Tracking on Autonomous UAV System for Surveillance Applications. In: *Sensors* 21 (2021), Nr. 23, S. 7888. http://dx.doi.org/10.3390/s21237888. – DOI 10.3390/s21237888

[27] LUFTFAHRT-BUNDESAMT: *EU-Kompetenznachweis und EU-Fernpiloten-Zeugnis für unbemannte Luftfahrtsysteme*. https://www.lba.de/DE/Drohnen/Fernpiloten/Kompetenznachweis.html. Version: 2024

[28] MATEK SYSTEMS: *H743-WLITE Flight Controller – Board Layout and Pinout*. https://www.mateksys.com/?portfolio=h743-wlite. Version: 2022

[29] MAVLINK PROJECT: *MAVLink Developer Guide: Serialization and Message Format*. https://mavlink.io/en/guide/serialization.html. Version: 2024

[30] MCENROE, P. ; WANG, S. ; LIYANAGE, M. : A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges. In: *IEEE Internet of Things Journal* 9 (2022), Nr. 17, S. 15435–15459. http://dx.doi.org/10.1109/JIOT.2022.3176400. – DOI 10.1109/JIOT.2022.3176400

[31] MEREI, A. ; MCHEICK, H. ; GHADDAR, A. ; REBAINE, D. : A Survey on Obstacle Detection and Avoidance Methods for UAVs. In: *Drones* 9 (2025), Nr. 3, 203. https://www.mdpi.com/2504-446X/9/3/203

[32] NAGEL, M. ; AMJAD, R. A. ; VAN BAALEN, M. ; LOUIZOS, C. ; BLANKEVOORT, T. : A White Paper on Neural Network Quantization. In: *arXiv preprint arXiv:2106.08295* (2021)

[33] ONNX PROJECT: *ONNX: Open Neural Network Exchange*. https://onnx.ai/. Version: 2024

[34] PEREIRA, G. P. ; CHAARI, M. Z.: A Look into the Performance of the Raspberry Pi AI HAT+ for Computer Vision Applications. In: *2025 11th International Conference on Communication and Signal Processing (ICCSP)*, 2025

[35] PESTANA, J. ; SANCHEZ-LOPEZ, J. L. ; PUENTE, P. de l. ; CARRIO, A. ; CAMPOY, P. : A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles. In: *Journal of Intelligent & Robotic Systems* 73 (2014), S. 387–400. http://dx.doi.org/10.1007/s10846-013-9953-0. – DOI 10.1007/s10846–013–9953–0

[36] PULLELA, A. ; WINGS, E. : 3D-Objekterkennung mit Jetson Nano und Integration mit KUKA KR6-Roboter für autonomes Pick-and-Place. In: *Tagungsband des 4. Symposiums für Autonome Systeme*, 2022

[37] RANFTL, R. ; BOCHKOVSKIY, A. ; KOLTUN, V. : Vision Transformers for Dense Prediction. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, S. 12179–12188

[38] RANFTL, R. ; LASINGER, K. ; HAFNER, D. ; SCHINDLER, K. ; KOLTUN, V. : Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44 (2022), Nr. 3, S. 1623–1637

[39] RASPBERRY PI FOUNDATION: *Raspberry Pi 5 Product Brief*. https://www.raspberrypi.com/products/raspberry-pi-5/. Version: 2023

[40] RASPBERRY PI FOUNDATION: *Picamera2: Python Library for Raspberry Pi Cameras*. https://github.com/raspberrypi/picamera2. Version: 2024

[41] RASPBERRY PI LTD: *Raspberry Pi AI HAT+ Documentation*. https://www.raspberrypi.com/documentation/accessories/ai-hat-plus.html. Version: 2024

[42] RCDRONE: *FPV Compute Thrust to Weight – A Comprehensive Analysis of Calculating and Utilizing Thrust-to-Weight Ratio for FPV Drones*. https://rcdrone.top/blogs/articles/fpv-compute-thrust-to-weight. Version: 2024

[43] REDMON, J. ; DIVVALA, S. ; GIRSHICK, R. ; FARHADI, A. : You Only Look Once: Unified, Real-Time Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, S. 779–788

[44] SEITZ, M. u. a.: Remote-Engineering und bildgestützte Inbetriebnahme mit einer Web-SPS. In: *Tagungsband Automation*, 2022, S. 7–9

[45] SKYDIO INC.: *Skydio Autonomy Engine – AI-Powered Obstacle Avoidance*. https://www.skydio.com/autonomy. Version: 2024

[46] STILLER, C. ; KAMMEL, S. ; LULCHEVA, I. ; ZIEGLER, J. : Probabilistische Methoden in der Umfeldwahrnehmung Kognitiver Automobile. In: *at - Automatisierungstechnik* 56 (2008), Nr. 11, S. 560–568. http://dx.doi.org/10.1524/auto.2008.0738. – DOI 10.1524/auto.2008.0738

[47] ULTRALYTICS INC.: *YOLOv8: State-of-the-Art Object Detection*. https://docs.ultralytics.com/models/yolov8/. Version: 2023

[48] ULTRALYTICS INC.: *YOLO11: Next Generation Object Detection*. https://docs.ultralytics.com/models/yolo11/. Version: 2024

[49] ULTRALYTICS INC.: *Model Comparisons: Choose the Best Object Detection Model*. https://docs.ultralytics.com/compare/. Version: 2026

[50] ULTRALYTICS INC.: *Raspberry Pi Deployment Guide for YOLO*. https://docs.ultralytics.com/guides/raspberry-pi/. Version: 2026

[51] ULTRALYTICS INC.: *Ultralytics YOLO26: Real-Time Object Detection*. https://docs.ultralytics.com/models/yolo26. Version: 2026

[52] WACHENFELD, W. ; WINNER, H. : Lernen autonome Fahrzeuge? In: *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Vieweg, 2015

[53] WANG, Y. ; LIU, T. u. a.: Hardware-Based Time Synchronization for a Multi-Sensor System. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024

[54] WANNER, D. ; HASHIM, H. A. ; SRIVASTAVA, S. ; STEINHAUER, A. : UAV avionics safety, certification, accidents, redundancy, integrity, and reliability: a comprehensive review and future trends. In: *Drone Systems and Applications* (2024). http://dx.doi.org/10.1139/dsa-2023-0091. – DOI 10.1139/dsa–2023–0091

[55] WEYER, J. : Autonomie und Kontrolle: Arbeit in hybriden Systemen am Beispiel der Luftfahrt. In: *Mensch – Technik – Organisation: Vernetztes Denken in der Produktionstechnik*. Springer, 2007, S. 1–15

[56] WIEGMANN, E. ; PRELL, R. : *KI-Technologie zur Unterstützung des Drohneneinsatzes bei Feuerwehren*. Springer Vieweg, 2025. http://dx.doi.org/10.1007/978-3-658-48639-6_12. http://dx.doi.org/10.1007/978-3-658-48639-6_12

[57] WOFK, D. ; MA, F. ; YANG, T.-J. ; KARAMAN, S. ; SZE, V. : FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In: *International Conference on Robotics and Automation (ICRA)*, 2019, S. 6101–6108

[58] WU, X. ; SUN, H. ; WU, R. ; FANG, Z. : EverySync: An Open Hardware Time Synchronization Sensor Suite for Common Sensors in SLAM. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024

[59] YANG, L. ; KANG, B. ; HUANG, Z. ; XU, X. ; FENG, J. ; ZHAO, H. : Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. In: *arXiv preprint arXiv:2401.10891* (2024)

[60] YANG, L. ; KANG, B. ; HUANG, Z. ; ZHAO, Z. ; XU, X. ; FENG, J. ; ZHAO, H. : Depth Anything V2. In: *arXiv preprint arXiv:2406.09414* (2024)

A Hardware-Spezifikationen, Stückliste und Kostenaufstellung

A.1 Hardware-Komponenten und Kostenaufstellung

In der folgenden Tabelle sind alle für den Prototyp verwendeten Komponenten sowie deren Anschaffungskosten aufgeführt.

Funktion	**Komponente**	**Preis [€]**
Flight Controller	Matek H743-WLITE (STM32H743)	79,90
ESC (4-in-1)	T-Motor Velox V50A	52,90
Motoren (4×)	Emax Eco II 2004 3000KV	67,60
Frame	SpeedyBee Bee35 3,5” CineWhoop	39,90
GPS/Kompass	HGLRC M100-5883	19,90
Propeller (4×)	HQProp DT90MM	3,19
Companion Computer	Raspberry Pi 5 (8 GB RAM)	92,50
KI-Beschleuniger	Hailo-8L AI HAT+ (13 TOPS)	105,00
Vision-Kamera	Raspberry Pi Camera 3 Wide NoIR	29,90
RC-Sender	Radiomaster Boxer ELRS	169,00
RC-Empfänger	Radiomaster RP1 ELRS	17,90
FPV-Brille	Skyzone Cobra X V4	249,00
FPV-Kamera	Caddx Ratel 2 Pro	36,90
Video-Transmitter	SpeedyBee TX800	19,90
VTX-Antenne	Foxeer Lollipop 4	19,90
Spannungsregler (BEC)	Matek BEC125-PRO (5,25 V)	16,90
Akku	Racepow 4000 mAh 4S Li-Ion	39,90
Ladegerät	SkyRC B6neo	34,90
Zubehör	Smoke Stopper, Schrauben, Löt Utensilien, Stand Offs, ...	20,90
Gesamt		**1.115,19**

Tabelle A.1: Stückliste und Kostenaufstellung des UAV-Prototyps.

B Software-Dokumentation: Repository-Struktur und Setup-Guide

Repository-Struktur

```
MAIN/
|-- run.py                        # Hauptprogramm (Entry Point)
|-- config.py                     # Zentrale Konfiguration
|-- data_logger.py                # Video + Telemetrie Aufnahme (RPi)
|-- offline_processor.py          # Post-Flug Analyse (PC)
|-- requirements.txt              # Python Dependencies
|
|-- vision/
|   +-- vision_system.py          # YOLO + ByteTrack + BBox-Proxy
|
|-- flight_control/
|   +-- mavlink_controller.py     # MAVLink Interface
|
|-- navigation/
|   +-- autonomous_navigator.py # Follow-Me + Exploration
|
+-- safety/
    +-- emergency_system.py     # Multi-Level Safety
```

Listing B.1: Projektstruktur

Zentrale Konfiguration

```
# HARDWARE
CAMERA_WIDTH  = 1920
CAMERA_HEIGHT = 1080
CAMERA_FPS    = 30
CAMERA_FOV    = 75

# Flight Controller (UART)
FC_CONNECTION_STRING = "/dev/ttyAMA0"
FC_BAUD_RATE = 57600

# YOLO
YOLO_MODEL          = "yolo26n"
YOLO_FALLBACK_MODEL = "yolov8n"
YOLO_CONFIDENCE     = 0.5
```

```
YOLO_IOU_THRESHOLD  = 0.45

# BBox-Distanz-Proxy
BBOX_PROXY_PERSON_HEIGHT_PX = 400   # px
BBOX_PROXY_REF_DISTANCE     = 2.5   # m

# Follow-Me
FOLLOW_DISTANCE = 3.5   # m
FOLLOW_SPEED    = 0.3   # m/s
TARGET_LOST_LOITER_FRAMES = 30

# Hindernisvermeidung
OBSTACLE_MIN_DISTANCE     = 2.0  # m
OBSTACLE_WARNING_DISTANCE = 4.0  # m

# Geschwindigkeitsgrenzen
MAX_HORIZONTAL_SPEED = 0.5   # m/s
MAX_VERTICAL_SPEED   = 0.3   # m/s
MAX_YAW_RATE         = 15    # deg/s

# Sicherheit
BATTERY_CRITICAL_VOLTAGE    = 13.2  # V
COLLISION_DISTANCE_THRESHOLD = 1.5  # m
EMERGENCY_TIMEOUT = 10              # s
RC_OVERRIDE_PRIORITY = True

# Logging
TARGET_VISION_FPS = 20
```

Listing B.2: config.py: Parameter des Gesamtsystems (gekürzt)

Hauptschleife

```
def start(self, mode=FlightMode.FOLLOW_ME, live_view=False):
    self.is_running = True
    self.data_logger.start_recording()
    self._start_pipeline_csv()

    while self.is_running:
        loop_start = time.time()

        use_tracking = (mode == FlightMode.FOLLOW_ME)
        vision_data = self.vision.process_frame(
            use_tracking=use_tracking
        )
        if vision_data is None:
            continue

        if self.data_logger.recording:
            self.data_logger.log_frame(vision_data.get('frame'))

        safety_ok = self.emergency.check_safety(vision_data)
```

```
        if not safety_ok:
            self.emergency.handle_emergency()
            self._log_pipeline_row(vision_data, loop_start, safety_ok)
            continue

        if mode == FlightMode.FOLLOW_ME:
            self.navigator.follow_me(vision_data)
        elif mode == FlightMode.EXPLORATION:
            self.navigator.explore_area(vision_data)
        elif mode == FlightMode.LOITER:
            self.navigator.hold_position()

        self._log_pipeline_row(vision_data, loop_start, safety_ok)
        if self.show_live_view:
            self._render_live_view(vision_data)

        time.sleep(1.0 / TARGET_VISION_FPS)
```

Listing B.3: run.py: Hauptschleife des DroneSystem (Auszug)

Vision-Pipeline

```
def process_frame(self, use_tracking=False):
    frame = self._capture_frame()
    if frame is None:
        return None
    detections = self._detect_objects(frame, use_tracking)
    obstacles  = self._identify_obstacles(detections)
    self._update_fps()
    return {
        'frame': frame,
        'detections': detections,
        'obstacles': obstacles,
        'fps': self.current_fps,
        'timestamp': time.time()
    }

def _detect_objects(self, frame, use_tracking=False):
    detections = []
    if self.hailo_engine is not None:
        # Hailo-8L
        padded, scale, pad_w, pad_h = self._letterbox_frame(
            cv2.cvtColor(frame, cv2.COLOR_BGR2RGB), 640
        )
        raw_dets, _ = self.hailo_engine.infer(
            np.expand_dims(padded, 0).astype(np.uint8),
            conf_threshold=self.confidence_threshold
        )
        for det in raw_dets:
            ...
        if use_tracking and detections:
            detections = self.byte_tracker.update(detections)
```

```
    else:
        # Ultralytics YOLO Fallback
        if use_tracking:
            results = self.yolo_model.track(
                frame, tracker="bytetrack.yaml", persist=True
            )[0]
        else:
            results = self.yolo_model(frame)[0]
        for box in results.boxes:
            ...
    return detections
```

Listing B.4: vision_system.py: YOLO-Inference mit zwei Backends (Auszug)

```
def _get_distance_from_bbox(self, detection):
    """Distanz  ber  BBox-H he."""
    x1, y1, x2, y2 = detection['bbox']
    bbox_height = y2 - y1
    if bbox_height <= 0:
        return None
    return (BBOX_PROXY_PERSON_HEIGHT_PX * BBOX_PROXY_REF_DISTANCE) \
           / bbox_height

def _identify_obstacles(self, detections):
    """Filtert Hindernisse unterhalb WARNING_DISTANCE."""
    obstacles = []
    for det in detections:
        distance = det.get('distance')
        if distance is not None and distance < OBSTACLE_WARNING_DISTANCE:
            obstacles.append({
                'class': det['class'],
                'distance': distance,
                'position': det['center'],
                'critical': distance < OBSTACLE_MIN_DISTANCE,
            })
    obstacles.sort(key=lambda x: x['distance'])
    return obstacles
```

Listing B.5: vision_system.py: BBox-Distanz-Proxy und Hinderniserkennung

Follow-Me Navigation

```
def follow_me(self, vision_data):
    if self.vision.tracked_target_id is None:
        new_id = self.vision.select_target(vision_data['detections'])
        if new_id is None:
            self.fc.set_mode('LOITER')
            return

    target = self.vision.get_target_detection(
        vision_data['detections']
    )
```

```
    if target is not None:
        self.frames_without_target = 0
        if self._check_path_clear(vision_data['obstacles']):
            velocity = self._calculate_follow_velocity(
                target['center'], target.get('distance'),
                target_distance=FOLLOW_DISTANCE
            )
            velocity = self._limit_velocity(velocity, FOLLOW_SPEED)
            self.fc.send_velocity_command(
                vx=velocity['forward'], vy=velocity['right'],
                vz=velocity['down'], yaw_rate=velocity['yaw_rate']
            )
        else:
            self.fc.stop_movement()
    else:
        # Target verloren
        self.frames_without_target += 1
        if self.frames_without_target > TARGET_LOST_LOITER_FRAMES:
            self.vision.clear_target()
            self.fc.set_mode('LOITER')
        else:
            self.fc.stop_movement()

def _calculate_follow_velocity(self, person_pos, person_distance,
                               target_distance):
    cx, cy = person_pos
    offset_x = (cx - CAMERA_WIDTH/2) / (CAMERA_WIDTH/2)   # -1..+1
    offset_y = (cy - CAMERA_HEIGHT/2) / (CAMERA_HEIGHT/2)

    # PID auf Distanzfehler
    if person_distance:
        distance_error = person_distance - target_distance
        forward = self._pid_control(distance_error, kp=0.5)
    else:
        forward = 0.2 if offset_y < 0 else -0.2

    return {
        'forward': forward,
        'right':    offset_x * 0.3,
        'down':     0.0,
        'yaw_rate': offset_x * 0.5
    }

def _pid_control(self, error, kp=1.0, ki=0.05, kd=0.2):
    dt = time.time() - self._pid_last_time
    self._pid_integral += error * dt
    self._pid_integral = max(-2.0, min(2.0, self._pid_integral))
    derivative = (error - self._pid_last_error) / dt
    self._pid_last_error = error
    self._pid_last_time = time.time()
    return max(-1.0, min(1.0,
        kp * error + ki * self._pid_integral + kd * derivative))
```

Listing B.6: autonomous_navigator.py: Follow-Me mit PID-Regelung (Auszug)

Sicherheitssystem

```python
def check_safety(self, vision_data):
    """Pr ft alle Sicherheitsbedingungen."""
    if not self._check_battery():   return False  # < 13.2V
    if not self._check_gps():       return False  # GPS-Verlust
    if not self._check_vision(vision_data): return False  # Timeout
    if not self._check_collision(vision_data): return False
    if not self._check_geofence():  return False  # H he/Distanz
    if self._check_rc_override():   return True   # Pilot   bernimmt
    return True

def _check_collision(self, vision_data):
    if vision_data and 'obstacles' in vision_data:
        for obs in vision_data['obstacles']:
            if obs['distance'] < COLLISION_DISTANCE_THRESHOLD:
                self.collision_count += 1
                self.fc.stop_movement()
                if self.collision_count > 5:
                    self.emergency_landing()
                    return False
                return False
        self.collision_count = max(0, self.collision_count - 1)
    return True

def handle_emergency(self):
    self.fc.stop_movement()
    voltage = self.fc.get_battery_voltage()
    if voltage < BATTERY_CRITICAL_VOLTAGE:
        self.battery_failsafe()
    elif not self.fc.has_gps_fix():
        self.emergency_landing()
    else:
        self.fc.return_to_launch()
```

Listing B.7: emergency_system.py: Multi-Level Safety-Checks (Auszug)

Setup-Guide

Raspberry Pi (On-Board)

```
# Python Virtual Environment
python3 -m venv venv && source venv/bin/activate

# Dependencies installieren
pip install picamera2 pymavlink ultralytics loguru

# Data Logger starten (120s Aufnahme)
python run.py --mode log --log-duration 120
```

```
9
10 # Follow-Me starten
11 python run.py --mode follow
```

PC (Offline-Analyse)

```
1 # Dependencies installieren
2 pip install torch torchvision ultralytics opencv-python loguru
3
4 # Offline-Verarbeitung (YOLO + DA3-Tiefensch tzung)
5 python offline_processor.py flight_data/flight_YYYYMMDD_HHMMSS/
```

C Rohdaten-Auszüge: Telemetrie-Logs und Frame-Timestamps

Telemetrie-CSV Format

```
timestamp_s,system_time_ms,lat,lon,alt_m,roll_deg,pitch_deg,
  yaw_deg,vx_ms,vy_ms,vz_ms,battery_v,battery_pct,
  gps_fix,gps_satellites,flight_mode
0.0001,1775534623568,50.5861000,8.6785000,5.00,0.00,1.50,
  0.00,1.00,0.00,0.00,16.00,100,1,14,GUIDED
0.0502,1775534623618,50.5861025,8.6785000,5.01,0.10,1.50,
  0.50,1.00,0.03,0.00,16.00,100,1,14,GUIDED
0.1004,1775534623668,50.5861050,8.6784999,5.01,0.20,1.50,
  1.00,1.00,0.05,0.00,16.00,100,1,12,GUIDED
0.1504,1775534623718,50.5861075,8.6784997,5.02,0.30,1.49,
  1.50,1.00,0.08,0.00,16.00,100,1,9,GUIDED
0.2005,1775534623769,50.5861100,8.6784995,5.02,0.40,1.48,
  2.01,0.99,0.10,0.01,16.00,100,1,13,GUIDED
```

Listing C.1: Auszug telemetry.csv: Bodentest mit simulierter Telemetrie (10 Hz)

Pipeline-Log Format

```
frame,elapsed_s,fps,n_detections,n_persons,target_id,
  velocity_vx,velocity_vy,velocity_vz,safety_ok,mode,
  inference_ms
0,0.583,1.7,1,1,1,-0.298,-0.036,0.000,True,GUIDED,566.7
1,0.860,3.8,1,1,1,0.000,0.000,0.000,True,GUIDED,258.0
2,1.099,4.2,1,1,1,0.000,0.000,0.000,True,GUIDED,236.6
3,1.336,4.2,1,1,1,0.000,0.000,0.000,True,GUIDED,234.7
4,1.575,4.2,1,1,1,0.000,0.000,0.000,True,GUIDED,236.3
5,1.815,4.2,1,1,1,0.000,0.000,0.000,True,GUIDED,237.0
6,2.067,4.0,1,1,1,0.000,0.000,0.000,True,GUIDED,250.5
```

Listing C.2: Auszug pipeline_log.csv: Integrationstest Follow-Me-Pipeline

Offline-Analyse Auszug

```
frame_id,timestamp_s,num_detections,persons_detected,
  obstacles_detected,mean_depth,min_depth,max_depth,
  lat,lon,alt_m,yaw_deg,processing_time_ms
1,0.033,1,0,0,0.638,0.338,1.520,50.5861017,8.6785000,
  5.01,0.33,39851.7
2,0.067,1,0,0,0.560,0.307,1.161,50.5861033,8.6785000,
```

```
  5.01,0.66,36967.3
3,0.100,1,0,0,0.595,0.315,1.281,50.5861050,8.6784999,
  5.01,1.00,40575.0
6,0.200,2,1,0,0.625,0.348,1.672,50.5861100,8.6784995,
  5.02,2.00,37790.8
```

Listing C.3: Auszug frame_analysis.csv: DA3-Tiefenanalyse (289 Frames, CPU)

Session-Ordner Struktur

```
test_flight_20260407_060342/
|-- video.mkv                  # Crash-resistentes Video (H264/MKV)
|-- video.mp4                  # Konvertiertes Video (H264/MP4)
|-- telemetry.csv              # GPS, Attitude, Battery (10 Hz)
|-- frame_timestamps.csv       # Hardware-Zeitstempel pro Frame
+-- results/                   # Offline-Analyse Ergebnisse
    |-- annotated_video.mp4    # Video mit YOLO-Bounding-Boxes
    |-- frame_analysis.csv     # Detektionen + Tiefe pro Frame
    |-- statistics.json        # Aggregierte Statistiken
    |-- geo_map.html           # GPS-Trajektorie (Leaflet)
    |-- depth_000001.png       # Exemplarische Tiefenkarten
    +-- analysis_summary.txt   # Zusammenfassung der Analyse
```

Listing C.4: Aufnahme-Session test_flight_20260407_060342

Eidesstattliche Erklärung

Hiermit erkläre ich, Veronika Herold, dass ich die vorliegende Bachelorarbeit selbstständig und ohne unerlaubte Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Fulda, den

07.04.2026

Datum

Unterschrift