Flug auf Knopfdruck

Entwicklung einer autonomen Lieferdrohne

Jugend forscht

Bundeswettbewerb 2022

Fachgebiet Mathematik/Informatik

Verfasser

Alexander Lowa, 17 Jahre

Lubolzerstraße 6, 15910 Schönwalde

alexander.lowa.04@gmail.com

Jan Berndt, 17 Jahre

Münzstraße 10, 03044 Cottbus

janberndt@gmx.net

Betreuender Lehrer

Dr. Torsten Skorubski

Max-Steenbeck-Gymnasium

Universitätsstraße 18, 03046 Cottbus

Cottbus, 04.04.2022

Kurzfassung

In dieser Arbeit haben wir ein Liefersystem auf Basis eines autonomen Quadrocopters prototypisch entwickelt. Besonderen Wert legten wir dabei auf eine intuitive Nutzeroberfläche und eine verlässliche Landung mithilfe von künstlicher Bilderkennung. Unsere Arbeit erklärt die technische Implementation, beleuchtet die verwendete Hardware und schafft ein tiefgehendes Verständnis von künstlichen neuronalen Netzwerken.

Eine Drohne und ein Server bilden das Liefersystem. Der Server stellt eine Webseite bereit, über welche die Nutzer*innen vorgegebene Fluganfragen an die Drohne stellen können. Ein Computer an Bord der Drohne leitet die Anweisungen des Servers an den Mikrocontroller weiter, welcher die eigentliche Steuerung des Luftfahrzeugs übernimmt. Befindet sich die Drohne über ihrem Zielstandort, hält sie mit einer Kamera nach einer spezifischen Landeplattform Ausschau. Durch eine Kombination aus Objekterkennung, Objekttracking und Tiefenschätzung wird die Plattform anvisiert und die Landung durchgeführt. Diese Bildverarbeitungssysteme sind mithilfe von Deep Learning so performant wie möglich umgesetzt. Im Rahmen der Arbeit haben wir eine Landeplattform und ein dazugehöriges Verifikationssystem entwickelt, welches zur Sicherheit der Drohne und ihrer Fracht beiträgt.

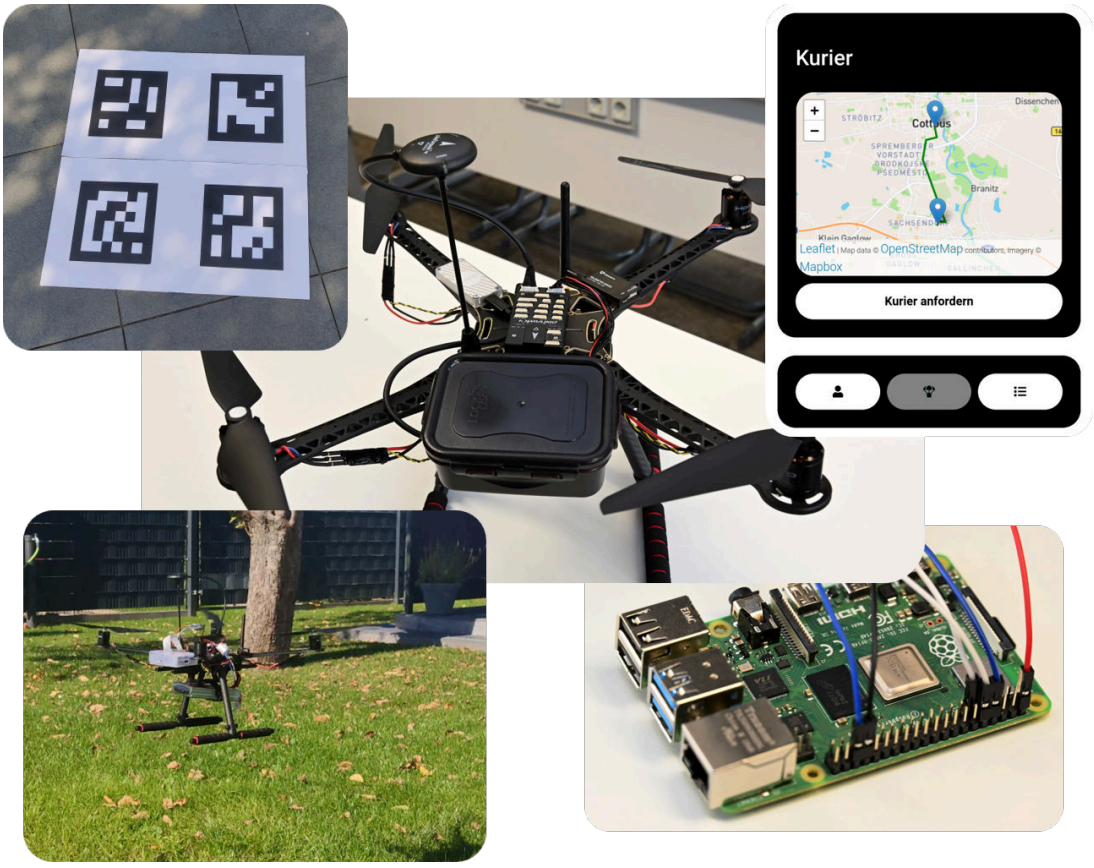

Abbildung 1: Das Projekt in fünf Bildern.

Jan entwickelte den Server mit Website sowie die Zustandslogik der Drohne. Er ist hauptsächlicher Verfasser des 2. und 3. Kapitels der Arbeit. Alexander widmet sich in Kapitel 4 dem eigens entwickelten Landesystem.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ArUco Augmented Reality University of Cordoba

BTU Brandenburgische Technische Universität Cottbus-Senftenberg

CNN Convolutional Neural Network

DBMS Database Management System

GPS Global Positioning System

HAT Hardware Attached on Top

HTTPS HyperText Transfer Protocol Secure

KCF Kernelized Correlation Filters

LBA Luftfahrt-Bundesamt

LIDAR light detection and ranging

MAVLink Micro Air Vehicle Link

MAVSDK Micro Air Vehicle Software Development Kit

mTLS mutual Transport Layer Security

OIDC Open-Identification Connect

OpenCV Open Computer Vision

PDB Power Distribution Board

PX4 Professional Autopilot 4

TLS Transport Layer Security

UPS United Parcel Service

VPS Virtual Private Server

YOLO You Only Look Once

1 Einleitung

Innerhalb des letzten Jahrzehnts haben sich unbemannte Luftfahrzeuge, sogenannte Drohnen, von einer experimentellen Militärtechnologie zum Massenphänomen entwickelt. Ob als Spielzeug für Kinder, Modell für Enthusiasten oder Arbeitsgerät für professionelle Fotografen sind Micro Air Vehicles (MAV) so frei erhältlich und weit verbreitet wie nie zuvor. Doch sowohl Start-Ups als auch Logistik-Giganten zeigen Interesse an Drohnen als Glied in den Lieferketten der Zukunft.

In den USA erhielt 2019 der *United Parcel Service (UPS)* als erste Firma die Zulassung zur uneingeschränkten Nutzung kommerzieller Drohnen [1]. Als Pilotprojekt entwickelte das Logistikunternehmen in Zusammenarbeit mit dem Startup *Matternet* ein Liefersystem für Blutproben. In Raleigh, North Carolina [2] überbringen Quadrocopter Proben über eine vordefinierte Route zur Auswertung in ein nahe gelegenes Labor. Dabei müssen die Drohnen von ausgebildetem Flugpersonal überwacht, be- und entladen werden.

Jans Mutter brachte uns auf eine ähnliche Idee für unsere Projektarbeit. Sie arbeitet in einer Kinderarztpraxis und merkte an, wie oft gerade für Covid-19-Tests ein Kurier gerufen werden müsse, um die Proben ins Labor zu fahren. Ähnlich wie beim UPS-Dienst könnten Drohnen hier effektiv eingesetzt werden. Die Fracht ist klein und leicht genug, um von einem handlichen MAV transportiert zu werden. Auch die zu fliegenden Strecken sind kurz und vorhersehbar, so dass Landeplätze im Voraus eingerichtet werden können.

Wir wollen die UPS-Idee weiter entwickeln, so dass ein zentrales Labor autonome Probenlieferungen aus mehreren Einrichtungen empfangen kann. Mitarbeitende einer Arztpraxis sollten ohne weitere Schulung über eine nutzerfreundliche Website den Luftkurier anfordern können. Schon allein aufgrund rechtlicher Beschränkungen werden wir kein vollständiges System mit, weit entfernten Einrichtungen testen können. Deshalb werden wir den Fokus unserer Arbeit auch weniger auf die konkrete Fracht legen. Stattdessen möchten wir herausfinden, ob schon mit der heutigen Technologie voll autonome Drohnenflüge sicher und verlässlich ausführbar sind.

2 Liefersystem

2.1 Nutzung

2.1.1 Anleitung

Ein Labor kann sich eine feste Landeplattform montieren lassen und die Drohne mieten. Sogenannte Nebeneinrichtungen, welche Lieferungen an das Labor senden möchten, benötigen ebenfalls eigene Plattformen. Die jeweiligen Flugrouten zum Labor planen wir manuell. Jeder Einrichtung erhält daraufhin einen individuellen Anmeldecode, mit dem sich ein*e Mitarbeiter*in auf unserer Website `https://droneso.me/` registrieren kann. Dafür kann ein bestehender Account bei Google oder Apple genutzt werden. Ist das Konto erstellt, verfällt der Code, das Konto aber bleibt permanent mit der Einrichtung verknüpft.

Auf der Website können im *Mitarbeiter*-Fenster (☰) neue Anmeldecodes generiert und bestehende Mitarbeiterkonten verwaltet werden. Im *Ich*-Fenster (👤) kann der eigene Name geändert, die Sitzung beendet oder das Konto permanent gelöscht werden.

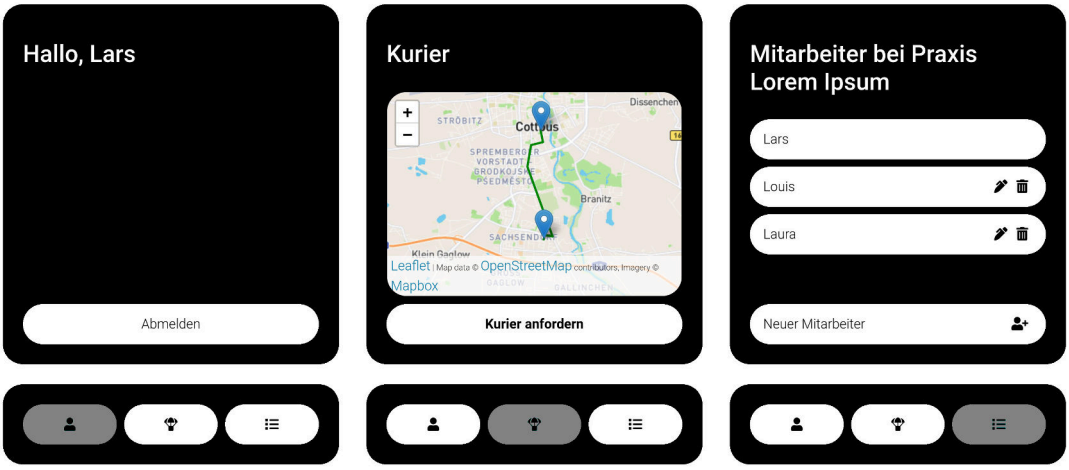

Abbildung 2: Fenster der Website, v. l. n. r.: *Konto*, *Kurier*, *Mitarbeiter*.

Das *Kurier*-Fenster (🪂) zeigt auf einer Karte die einzelnen Einrichtungen und die möglichen Routen der Drohne an. Hier können Mitarbeiter*innen einer Nebeneinrichtung die Drohne anfordern. Diese begibt sich vom Labor aus auf den Weg, sobald dort die Starterlaubnis erteilt wird. Während des Flugs können Position, Akkuladung und Zustand der Drohne ständig von der Start- und Zieleinrichtung aus überwacht werden. Außerdem kann eine Notrückkehr oder Notlandung eingeleitet werden.

Bei einer Notrückkehr versucht die Drohne, wieder am Startpunkt zu landen. Kann sie dort keine Landeplattform mehr erkennen, leitet sie selbstständig eine Notlandung ein. Das heißt, sie nutzt keine intelligenten Systeme, um die Plattform zu treffen oder Hindernissen auszuweichen.

Erreicht die Drohne ihre Zielkoordinaten ohne Komplikationen, führt sie eine planmäßige Landung aus. Mithilfe der Kamera positioniert sie sich genau über der Landeplattform. Sollte sie am Landeplatz Hindernisse erkennen, beginnt sie wieder eine Notrückkehr. Nach der Landung kann die Drohne gefahrlos mit Proben beladen werden. Erteilt ein*e Mitarbeiter*in die Starterlaubnis, fliegt die Drohne zum Labor zurück und steht anschließend wieder für Aufträge bereit.

2.1.2 Mehrwert

Zurzeit werden Probenlieferungen von Kurierdiensten verrichtet, die meist für jede Zustellung einzeln angefordert werden müssen. Unser System hingegen kommt ohne zusätzliche Arbeitskräfte aus und verkürzt Kommunikationswege auf einen Knopfdruck. Ein Quadrocopter kann Wege insbesondere in der Stadt unabhängig vom Verkehr und ohne Pausenzeiten zurücklegen, so dass zeitkritische Proben schnellstmöglich ihr Ziel erreichen.

2.2 Technische Umsetzung

2.2.1 Serverstruktur

Sendet ein Gerät eine *HyperText Transfer Protocol Secure (HTTPS)*-Anfrage an `https://droneso.me/`, wird diese an unseren eigenen *Virtual Private Server (VPS)*, also eine Rechnerinstanz bei einem Server-Anbieter, weitergeleitet. Dort übergibt ein *Nginx*-Proxy die Anfrage an das Tool *Gunicorn*. Dieses verwaltet die Prozesse

unserer eigens mithilfe der Bibliothek *Flask 2.0.1.* programmierten Python-Applikation (GitHub-Link). Sie dient als Schnittstelle, um eine Kommunikation zwischen den Nutzer*innen und der Drohne zu ermöglichen. Das Database Management System (DBMS) *MongoDB* speichert dabei die Daten der Nutzer*innen und Einrichtungen in abruf- und veränderbarer Form.

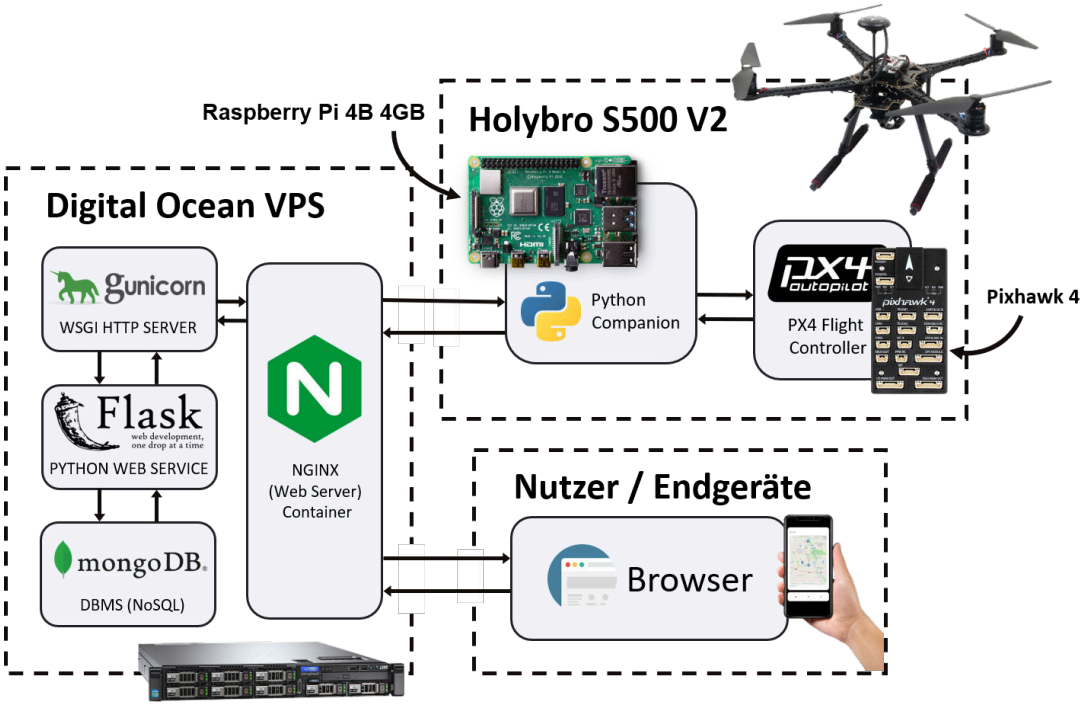

Abbildung 3: Struktur von Server, Drohne und Endgerät (abgerundete Felder sind Soft-, Rechtecke Hardware).

Durch das Authentifizierungsverfahren Open-Identification Connect (OIDC) bestätigen Nutzer*innen ihre Identität über einen Online-Drittanbieter. Im Vergleich zu traditionellen Verfahren ist OIDC komfortabler, weil Nutzer*innen nicht erst ein neues Konto anlegen müssen, und sicherer, weil die Anmeldung von Unternehmen mit höchsten Sicherheitsstandards bereitgestellt wird.

2.2.2 Verwaltung der Drohne

Die Flask-Applikation speichert den letzten sowie den gewünschten Landeplatz der Drohne. Ändert sich der gewünschte Landeplatz auf einen Wert, welcher von der letzten Position abweicht, generiert und verschickt der Server eine neue Mission für die Drohne.

Wir sind mit der brandenburgischen Netzabdeckung [3] vertraut, deswegen geht unser System mit Verbindungsabbrüchen ohne Probleme um. Dafür steht es erstens der Drohne frei, Anweisungen des Servers stillschweigend abzulehnen, sollten diese zu spät kommen oder auf veralteten Informationen basieren. Zweitens verändert der Server seinen Zustand wann immer möglich auf Meldungen der Drohne hin, statt sich darauf zu verlassen, dass diese immer den Anweisungen des Servers folgt.

2.2.3 Verwaltung der Standorte

Jede Einrichtung besitzt einen von fünf inneren Zuständen (siehe Abbildung 4). Zusätzlich zu den angeführten Interaktionsmöglichkeiten haben Nutzer einer Nebeneinrichtung die Möglichkeit, die Drohne für eine Lieferung anzufordern. Anfragen werden mit der jeweiligen Uhrzeit vermerkt und der Reihenfolge nach abgearbeitet, sobald die Drohne im Labor bereitsteht.

Zustand	*Verhältnis der Drohne zur jeweiligen Einrichtung*	*Interaktionsmöglichkeiten*
`IDLE`	Gelandet und für Anfragen bereit	-
`AWAITING_TAKEOFF`	Gelandet und angefragt	Starterlaubnis erteilen
`EN_ROUTE`	Fliegt in Richtung Ziel	Notrückkehr, Notlandung
`EMERGENCY_RETURNING`	Notrückkehr, entfernt sich vom Ziel	Notlandung
`EMERGENCY_LANDING`	Notlandung	-

Abbildung 4: Mögliche Zustände eines Standorts.

Im Verlauf einer Lieferung löst die Drohne bei der Ausgangs- und der Zieleinrichtung Veränderungen des inneren Zustands aus. Der planmäßige Verlauf ist nummeriert. Nicht eingezeichnet ist der `EMERGENCY_LANDING`-Zustand, welcher prinzipiell zu jeder Phase einer Lieferung angenommen werden kann.

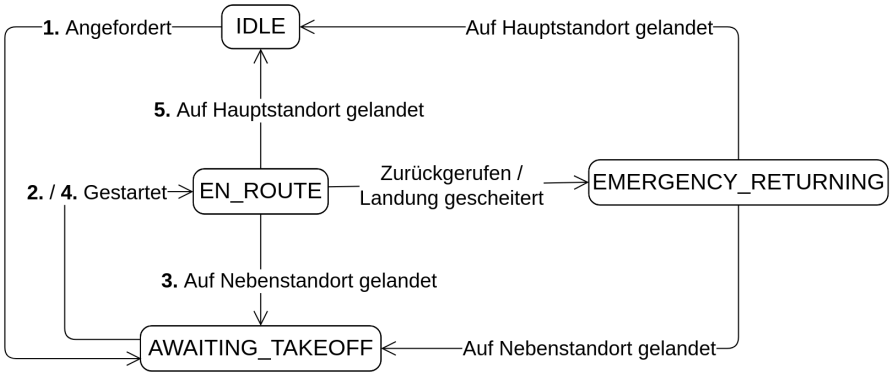

Abbildung 5: Von der Drohne ausgelöste Zustandsübergänge eines Standorts mit vorgesehener Reihenfolge.

3 Drohne

3.1 Hardware

Handelsübliche Freizeitdrohnen sind nicht auf Modifikation ausgelegt. Um tiefer in die Steuerung der Drohne eingreifen zu können, verwenden wir offen zugängliche Einzelteile. Die hier aufgeführte Hardware wurde vollständig vom *Förderverein des Max-Steenbeck-Gymnasiums e.V.* finanziert und geht nach Abschluss der Arbeit in den Besitz der Schule über, um für nachfolgende Projekte genutzt werden zu können.

Das *Holybro S500 V2 Kit* mit einem Karbonfaserrahmen sowie vier Motoren mit zugehörigen Drehzahlreglern und Propellern bildet die Grundlage unseres Quadrocopters. Ein Power Distribution Board (PDB) verbindet die beiden Bordcomputer mit dem Akku am unteren Ende des Rahmens und gewährleistet eine konstante Betriebsspannung von 5V. Die Drohne wird kontrolliert vom *Pixhawk 4 Flight Controller*, welcher

mittels der Firmware Professional Autopilot 4 (PX4) Steuereingaben und Sensordaten verarbeitet. Zur Positionsbestimmung ist ein Global Positioning System (GPS)-Modul angebracht.

Der Pixhawk allein besitzt nur geringe Kapazitäten zur Bildverarbeitung und Datenübertragung. Als zusätzlicher *Companion Computer* bietet der *Raspberry PI 4B* die nötige Rechenleistung und Erweiterbarkeit. Er ist ausgestattet mit einer Hardware Attached on Top (HAT), welche auch während des Fluges eine Internetverbindung über das 4G-Netzwerk erlaubt.

Abbildung 6: Pixhawk 4, Raspberry Pi Cellular IoT HAT, Raspberry Pi 4B, Coral USB Accelerator, Holybro 2216 KV880 mit Propeller, ZOP Power 14.8V 5500mAh, Raspberry Pi Camera Module V2, PM02 V3, BLHeli S ESC 20A, Holybro M8N GPS (Quellen: Dronecode Foundation, Sixfab, Wikimedia Commons, Coral, Holybro, Banggood, Raspberry Pi Foundation, Premium-Modellbau).

GPS allein ist nicht akkurat genug, um auf unseren rund 80x80cm großen Plattformen zu landen [4]. Deshalb ist an der Drohne die *Raspberry Pi Camera V2* vertikal nach unten gerichtet, so dass die Plattform intelligent erkannt und treffsicher erreicht werden kann. Künstliche neuronale Netze werten die Bilddaten aus, um Landeplattformen oder eventuelle Hindernisse zu erkennen. Der *Google Coral USB Accelerator* an Bord der Drohne ist ein spezieller Prozessor, um die hohe Anzahl von Rechenoperationen der Netze auszuführen.

Abbildung 7: Drohne im Teststand.

3.2 Companion-Programm

Auf dem Raspberry Pi läuft die *Companion*-Applikation (GitHub-Link). Dieses Python-Programm empfängt vom Server Flugrouten zwischen je einer Start- und Zieleinrichtung und setzt diese Missionen dem Zustand der Drohne entsprechend um.

3.2.1 Verbindung zum Pixhawk

Der Raspberry Pi ist mit dem Telemetrie-Anschluss des Pixhawk verlötet. Das Protokoll Micro Air Vehicle Link (MAVLink) definiert Befehle, um z. B. eine Mission mit mehreren Wegpunkten auf den Flight Controller zu übertragen und auszuführen. Das Companion-Programm nutzt diese Spezifikation mithilfe der Codebibliothek Micro Air Vehicle Software Development Kit (MAVSDK).

3.2.2 Verbindung zum Server

Auf dem Raspberry Pi ist ein eigens erstelltes (*self-signed*) Transport Layer Security (TLS)-Zertifikat hinterlegt. In der Kommunikation authentifiziert sich also nicht nur der Server mit seinem Zertifikat gegenüber der Drohne, sondern auch die Drohne gegenüber dem Server. Dieses mutual Transport Layer Security (mTLS)-Verfahren gewährleistet, dass kein Angreifer die Drohne imitieren kann [5].

In regelmäßigen Abständen schickt der Companion eine Meldung vom Typ `HEARTBEAT` an den Server. Sie beinhaltet die momentanen Koordinaten und geschätzte Batterieladung. Außerdem wird jeder neue Zustand des Companions zusammen mit der momentanen Start- und Zieleinrichtung in einem `STATUS_UPDATE` übertragen. Schlägt eine Übertragung fehl, werden die Nachrichten zwischengespeichert und erst wieder versandt, sobald die Verbindung wiederhergestellt ist.

Antwortet der Server mit `NONE` auf eine Nachricht, setzt die Drohne ihr Verhalten fort. Es kann

allerdings auch eine UPDATE-Anweisung mit einer neuen Mission empfangen werden. Außerdem kann die EMERGENCY_RETURN-Nachricht eine Notrückkehr, die EMERGENCY_LAND-Nachricht eine Notlandung veranlassen.

3.2.3 Zustandslogik

Der innere Zustand des Companions (siehe Abbildung 8) bestimmt die Aktionen der Drohne. Wie in Abbildung 9 beschrieben ändert der Companion seinen Zustand basierend auf eigenen Sensordaten oder auf Anweisungen des Servers hin.

Zustand	*Aktion*
IDLE	Rotoren deaktivieren und auf Anweisungen warten
UPDATING	Prüfen, ob neue Mission mit momentaner Batterieladung und Position ausführbar ist
EN_ROUTE	Neue Mission an Pixhawk übertragen
LANDING	Planmäßige Landung gemäß Kapitel 4, Wechsel zu EMERGENCY_RETURNING bei Scheitern
EMERGENCY_RETURNING	Neue Mission in umgekehrter Reihenfolge rückabwickeln
RETURN_LANDING	Wie LANDING, nur wird bei Scheitern ein EMERGENCY_LANDING durchgeführt
EMERGENCY_LANDING	Notlandung
CRASHED	Notgelandete Drohne muss geborgen und neu gestartet werden

Abbildung 8: Mögliche Zustände des Companions.

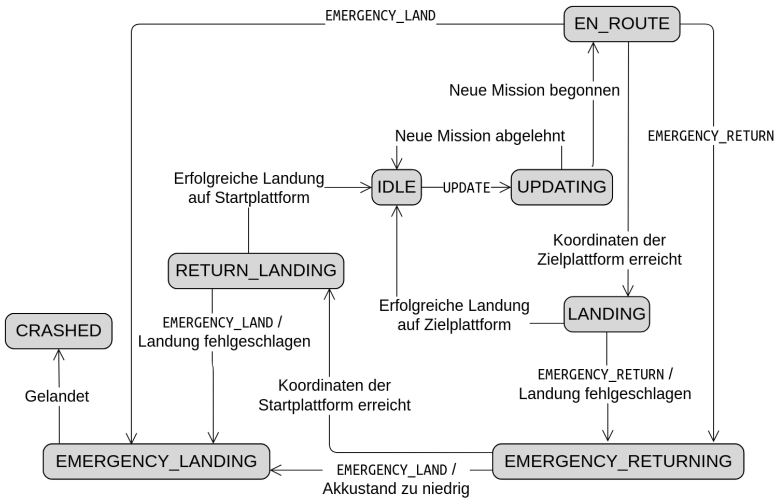

Abbildung 9: Zustandsdiagramm des Companions.

4 Landung

4.1 Ablauf

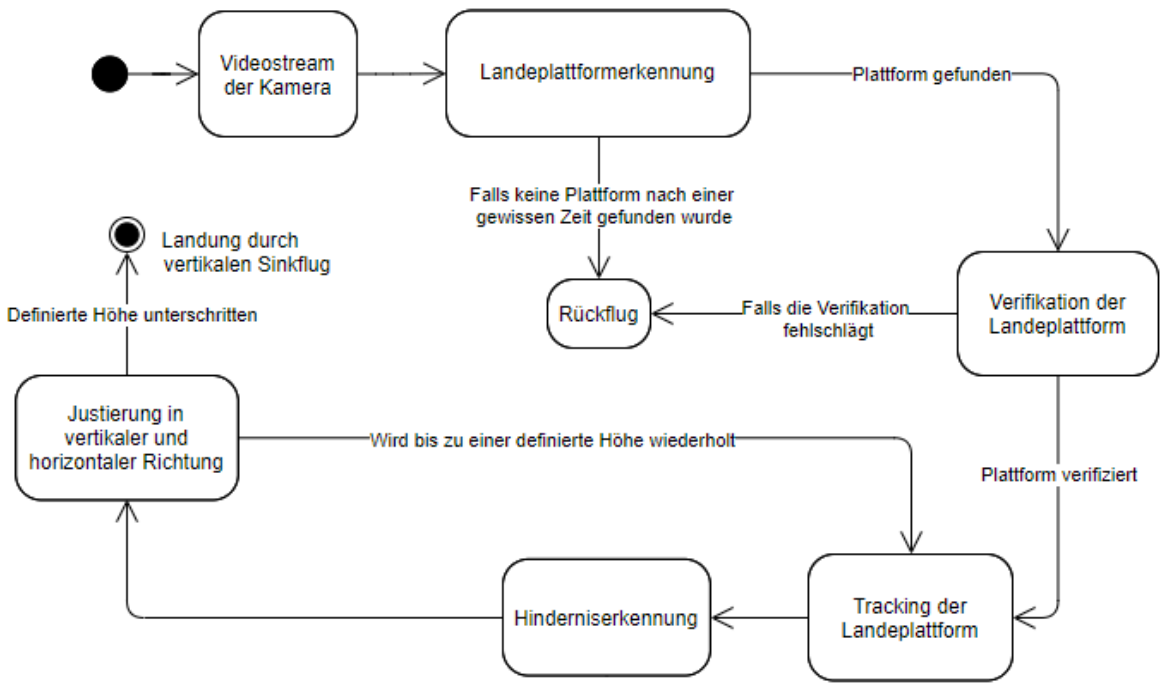

Abbildung 10: Ablauf der Landung.

Im `LANDING`- bzw. `RETURN_LANDING`-Zustand sucht die Drohne zunächst durch das Objekterkennungssystem (siehe Abschnitt 4.1) im Videostream der Kamera nach einer Landeplattform. Die beiden Zustände unterscheiden sich dabei nur dadurch, was geschieht, wenn die Landung fehlschlägt. Wird nach einer bestimmten Zeit keine Plattform erkannt, kehrt der Companion im `LANDING`-Zustand wieder zu seinem Startpunkt zurück. Ist er jedoch bereits zurückgekehrt und findet auch im `RETURN_LANDING`-Zustand keine Plattform, führt er als letzte Maßnahme ein `EMERGENCY_LANDING` durch.

Im Normalfall jedoch wird die erkannte Landeplattform vom Verifikationssystem (siehe Abschnitt 4.2) auf ein gültiges Muster geprüft. Ist die Plattform nicht die richtige, wird die Landung ebenfalls abgebrochen.

Ansonsten verfolgt das Trackingsystem (siehe Abschnitt 4.3) die Position der Plattform relativ zur Drohne. So kann sie sich während des Sinkflugs exakt über der Plattform positionieren. Dabei wird die Landung neu versucht, sollte ein Hindernis über der Plattform (siehe Abschnitt 5.5) erkannt werden.

4.2 Erkennung

Um eine Landeplattform aus einem Bild herauszufiltern, haben wir ein eigenes Objekterkennungssystem auf Basis der *You Only Look Once (YOLO) v5*-Technik entwickelt. Bei etwas geringerer Genauigkeit hat unse-

re Implementation dabei eine schnellere Inferenzzeit als YOLO selbst. Wir haben das Modell anhand eines selbst erstellten Datensatzes von etwa 16400 Bildern auf unsere Landeplattformen trainiert. Über eine modifizierte Schnittstelle der Python-Bibliothek Open Computer Vision (OpenCV) kann der Companion nun den Videostream der Kamera in das Modell leiten, um die rechteckige `Bounding Box` einer etwaigen Plattform zu erkennen.

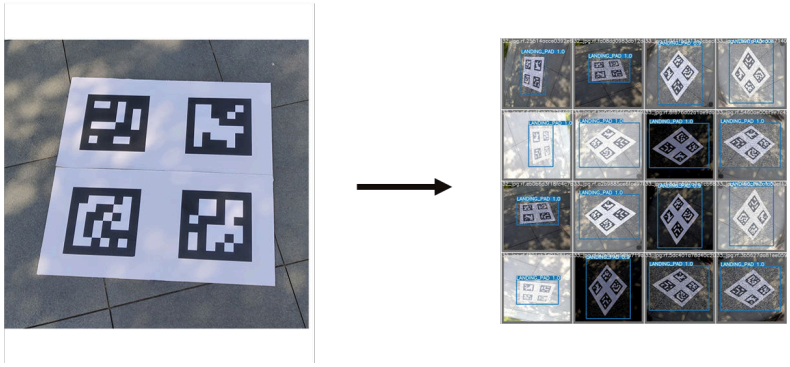

Abbildung 11: Mögliche Ausgabe unseres Objekterkennungssystems bei einer Landeplattform als Eingabe.

4.3 Verifikation

Auf jeder Landeplattform ist eine einmalige Kombination aus vier *Augmented Reality University of Cordoba (ArUco)-Markern* abgedruckt. Diese Muster helfen dabei, die Plattformen zu erkennen und auseinanderzuhalten. Über die GPS-Position und die ArUco-Muster kann eine Plattform also doppelt überprüft werden, bevor die Drohne zur Landung ansetzt.

Zuerst wird der vom Objekterkennungssystem ermittelte Bereich aus dem Bild extrahiert. Spezielle Filter reduzieren anschließend Störungen wie z. B. Rauschen im Bild. In dem so aufbereiteten Bildausschnitt werden Kanten gesucht. Dazu nutzen wir einen *Canny-Kantendetektionsprozess*, bestehend aus Glättung des Eingangsbildes, Sobel-Kantendetektion und Filterung der Kanten mit einer Schwellwertoperation. Anstelle der streng rechtwinkligen Begrenzungsbox aus der Objekterkennung kann nun ein exaktes Viereck entlang der Kanten der Plattform gezeichnet werden. Das ermittelte Viereck wird perspektivisch in ein Quadrat transformiert, um die einzelnen Marker optimal dekodieren zu können:

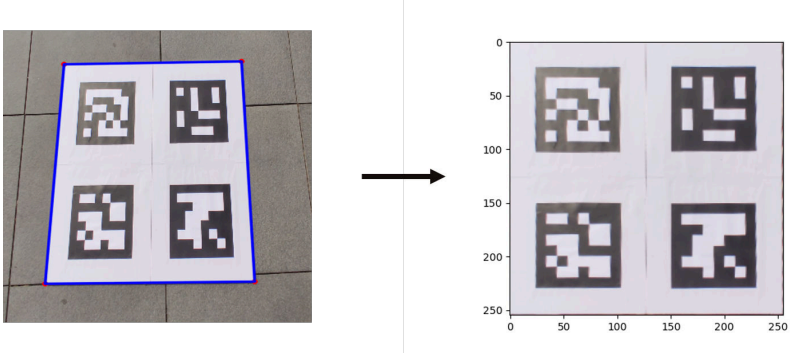

Abbildung 12: Detektierte Kanten einer Plattform und perspektivische Transformation.

Der ArUco-Detektor aus der OpenCV-Bibliothek kann nun die einzelnen Marker in ihre numerische Repräsentation umwandeln. Ist die Zahlenkombination wie erwartet, wird die Landung fortgesetzt.

4.4 Tracking

Die gegebene Hardware ist nicht schnell genug, um die Erkennung der Landeplattform bei jeder Bewegung der Drohne erneut auszuführen. Dieses Problem lösen wir mit *Tracking-Filtern*, welche die Position der einmal erkannten Plattform in Echtzeit bis zur Landung weiter verfolgen können. Wir nutzen die *Kernelized Correlation Filters (KCF)*-Trackingmethode, welche in unseren Tests am besten abschnitt.

4.5 Tiefenschätzung

Um die Entfernung eines Sensors zu einem Hindernis zu ermitteln, werden in der Regel Tiefensensoren wie z. B. light detection and ranging (LIDAR) genutzt. Für unser Projekt sind diese jedoch zu kostspielig, zu schwer und besitzen einen zu hohen Stromverbrauch. Stattdessen können neuronale Netzwerke allein aus den Bildern der Kamera die Bildtiefe abschätzen. Das ist nicht nur bei der Positionsbestimmung der Landeplattform hilfreich, sondern kann sogar genutzt werden, um während der Landung etwaigen Hindernissen auszuweichen.

Bisherige Forschungsarbeiten in diesem Bereich haben sich eher auf die Genauigkeit der Ergebnisse als auf die Geschwindigkeit der Berechnungen konzentriert. Wir haben für unseren Anwendungsbereich jedoch ein Tiefenschätzungsnetzwerk entwickelt, welches bei vergleichbarer Genauigkeit eine vielfach niedrigere Inferenzzeit bietet.

4.5.1 Netzwerkarchitektur

Wir verwenden zur Tiefenschätzung ein Convolutional Neural Network (CNN) mit Encoder-Decoder-Architektur. Der Encoder extrahiert charakteristische Merkmale aus dem eingegebenen Bild, aus welchen der Decoder eine Tiefenschätzung generiert.

Ein Encoder nach dem hochmodernen *MobileNet*-Schema extrahiert charakteristische Merkmale aus dem eingegebenen Bild. Er ist durch seine unkonventionelle Architektur aus n $m \times m$ Depthwise-Convolutional-Ebenen und einer 1×1 Pointwise-Convolutional-Ebene besonders effizient.

Das Decoder-Netzwerk generiert aus den extrahierten Merkmalen die tatsächliche Tiefenschätzung. Es ist aus fünf kaskadierten *Upsample-Ebenen* und einer einzelnen *Pointwise-Convolutional-Ebene* zusammengesetzt. Um die Faltungen der Ebenen zu vereinfachen, nutzen wir das *Depthwise-Decomposition*-Verfahren.

Bei dem Abbau eines Bilds durch den Encoder und dem darauf folgenden Wiederaufbau durch den Decoder geht im Allgemeinen Bildqualität verloren. Um mehr Informationen zu erhalten, haben wir drei sog. *Skip-Verbindungen* zwischen den High-Level Featuremaps des Encoders und den Inputs der einzelnen Upsample-Layer des Decoders gezogen.

4.5.2 Trainingsverfahren

Unser Netzwerk wird nach der *Supervised Learning*-Methode trainiert. Dabei werden während des Trainings die optimalen Parameter für das Netz ermittelt, um den Fehler der Ausgabe zu minimieren. Mathematisch lässt sich dieser Umstand so beschreiben: Man definiere f als Netzwerk und θ als dessen Parameter. Ist ein Bild I_k

gegeben, wird die Tiefenschätzung durch $D = f(I_k, \theta)$ beschrieben. Generell gilt für die optimalen Parameter in Bezug auf die allgemeine Kostenfunktion l_{all}, die Menge I der Trainingsbilder und die dazugehörige Menge D der *Ground-Truth-Tiefenschätzungen*:

$$\theta^* = arg\,\min_w \sum_k l_{all}(f(I_k;\,\theta), D_k)$$

Für die Kostenfunktion haben wir uns an dem von [6] vorgestellten Design orientiert. Hierbei wird eine Menge L aus 78 verschiedenen *Mean-Removed-*, *Gradient-* und *Normal-Loss*-Kostenfunktionen l_i mit jeweiligen Gewichten w_i zu einer *allgemeinen Kostenfunktion* l_{all} kombiniert:

$$l_{all} = \sum_i w_i\, l_i$$

Die Gewichte aller Kostenfunktionen werden auf $\frac{1}{|L|}$ initialisiert. Im ersten Trainingsdurchgang werden sie wie folgt anhand der durchschnittlichen Ausgabe $\bar{l}_i$ der jeweiligen Kostenfunktion angepasst:

$$w_i^{(0)} = \frac{\bar{l}_{all}}{\bar{l}_i} \qquad for\ each\ l_i \in L$$

Über das weitere Training hinweg werden die Gewichte der Kostenfunktionen periodisch angepasst, um das Netzwerk optimal umzuformen. $B_i^{(t)}$ drückt dabei die Summe aller Kosten der Funktion l_i aus, welche über eine Periode t aus n Trainingsbildern hinweg berechnet werden. Zudem definiert $B_i^{(t-1)}$ das Gewicht von $B_i^{(t)}$, welches nach der vorherigen Periode $t-1$ ermittelt wurde und für die momentane Periode t verwendet wird. $B_{all}^{(t)}$ kann demnach folgendermaßen ausgedrückt werden:

$$B_{all}^{(t)} = \sum_i w_i^{(t-1)} \cdot B_i^{(t)}$$

Ist eine Periode abgeschlossen, aktualisieren wir das Gewicht $w_i^{(t)}$, indem wir $B_i^{(t)}$ mit $B_i^{(t-1)}$ vergleichen. Dabei entspricht $R_i^{(t)} = \frac{B_i^{(t)}}{B_{all}^{(t)}}$ dem Verhältnis der i-ten Kosten zu den Gesamtkosten. $R_i^{(t)} - R_i^{(t-1)}$ ist damit die Veränderung von R. Daraus ergibt sich für das neue Gewicht $w_i^{(t)}$:

$$w_i^{(t)} = w_i^{(t-1)} \cdot (1 - \lambda \cdot \frac{\Delta R_i^{(t)}}{R_i^{(t)}})$$

Der Faktor λ gibt dabei an, welche Kostenfunktionen mit welcher Stärke in die Gewichtungsberechnung einbezogen werden sollen. Ein ständig von 3 zu -3 dekrementierendes λ lieferte im direkten Vergleich die besten Resultate.

Ähnlich wie ein Trainer dem Sportler erst die Grundzüge, und dann die Feinheiten eines Bewegungsablaufs vermittelt, passt sich unsere Kostenfunktion l_{all} laufend dem Wissensstand des Netzwerks an. Dieser Prozess nimmt etwa 30% mehr Trainingszeit in Anspruch als das Training mit herkömmlichen Kostenfunktionen. Dafür macht die Tiefenschätzung am Ende bei gleicher Geschwindigkeit im Schnitt 18% weniger Fehler - ein voller Erfolg!

4.6 Positionsermittlung

Vom Tracker erhalten wir kontinuierlich vier Bildpunkte, welche die Position der vier Ecken unserer erkannten und verifizierten Landeplattform angeben. Um zur Landung ansetzen zu können, muss aus diesen Punkten zunächst die Position der Plattform relativ zur Drohne ermittelt werden.

Dafür wird zunächst anhand der tatsächlichen Breite b und der auf den Sensor projizierten Breite s_b der Landeplattform die vertikale Distanz d_z zur Plattform ermittelt. Gegeben sind dabei die Brennweite f der Kamera und die Breite s ihres Sensors.

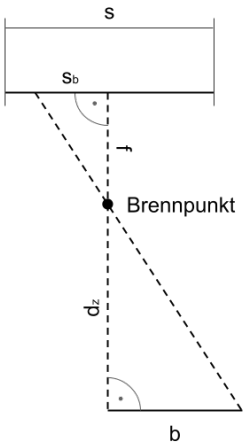

Abbildung 13: Seitenverhältnisse des Kamerabilds von der Landeplattform.

$$\frac{b}{d_z} = \frac{s_b}{f} \Leftrightarrow b = \frac{d_z \cdot s_b}{f}$$

s_b lässt sich dabei mithilfe der horizontalen Auflösung des Sensors p und der Breite der Plattform in Pixeln p_b berechnen:

$$\frac{s_b}{s} = \frac{p_b}{p} \Leftrightarrow s_b = \frac{s \cdot p_b}{p}$$

Zusätzlich können die in Kapitel 4.4 ermittelten Tiefenschätzungen der Landeplattform gemäß der von [7] beschriebenen Methode in Meter denormalisiert werden. Die errechnete Distanz d_z wird zusätzlich mit der Tiefenschätzung abgeglichen, um einen noch genaueren Wert zu ermitteln.

Auf ähnliche Weise wie d_z kann nun die relative Position d_y der Landeplattform auf der Y-Achse (links/rechts) ermittelt werden. Dabei wird nicht mehr die projizierte Breite s_b der Plattform benötigt. Stattdessen nutzen wir nun die Abweichung s_y des projizierten Mittelpunkts der Landeplattform vom Mittelpunkt des Kamerasensors:

$$\frac{d_y}{d_z} = \frac{s_y}{f} \Leftrightarrow d_y = \frac{d_z \cdot s_y}{f}$$

s_y erhalten wir aus der Abweichung p_y des Mittelpunkts der Landeplattform vom Bildmittelpunkt:

$$\frac{s_y}{s} = \frac{p_y}{p} \Leftrightarrow s_y = \frac{s \cdot p_y}{p}$$

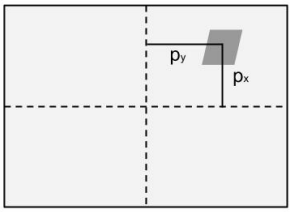

Abbildung 14: Ermittlung von p_x und p_y der Landeplattform im Bild der Kamera.

Die Distanz d_x auf der X-Achse (vorne/hinten) lässt sich mit p_x ebenso berechnen. Dabei muss lediglich für s die Sensorhöhe statt Breite, und für p die vertikale statt der horizontalen Auflösung eingesetzt werden.

4.7 Steuerung der Drohne

Basierend auf der ermittelten relativen Position der Plattform kann die Drohne nun zur Landung ansetzen. Dafür nutzen wir die manuelle Steuerung von MAVLink, welche es uns erlaubt, die Drohne wie mit einem Joystick zu kontrollieren. Hat die Drohne die Plattform erkannt, beginnt sie einen kontinuierlichen Sinkflug. Dabei werden die X- und Y-Abweichungen von der Landeplattform in Meter mit einem bestimmten Gewicht multipliziert als Steuerungseingabe verwendet. So korrigiert die Drohne ständig ihren Kurs, um sich mittig über der Landeplattform zu halten.

Außerdem nutzen wir während des Fluges die Tiefenschätzung, um Hindernissen auszuweichen. Das Bild der Kamera wird dafür in $n \times n$ Sektoren unterteilt. Unterschreitet die Tiefe eines Sektors einen bestimmten Schwellwert, weicht die Drohne in die jeweils andere Richtung aus.

Befindet sich die Drohne nur noch ca. einen Meter über der Plattform, werden Hinderniserkennung und Plattformtracking beendet, so dass die Drohne geradewegs auf dem Boden aufsetzt.

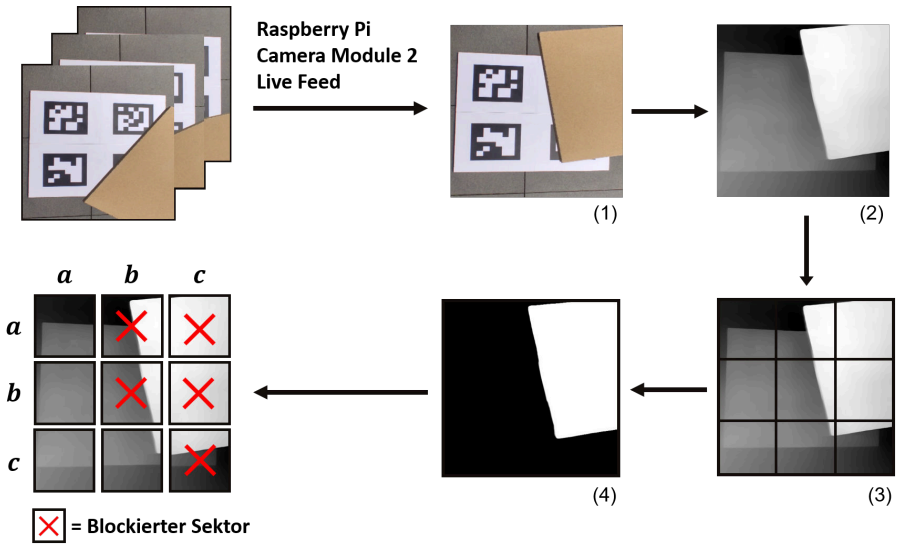

Abbildung 15: Ablauf der Hinderniserkennung: 1) Eingabebild; 2) Die durch unsere Architektur ermittelte Tiefenschätzung; 3) Einteilung in 3×3 Sektoren; 4) Visualisierung der Fläche, die den Tiefen-Schwellwert überschreitet; Resultat: Das MAV muss sich nach Links bewegen

5 Diskussion

5.1 Rechtslage

Der Artikel *Zum Jahreswechsel gelten neue EU-Regelungen für Drohnen* [8] beschreibt die EU-Verordnung 2019/945, welche den Betrieb unbemannter Luftfahrzeuge seit Anfang des Jahres 2021 regelt. Um Drohnen mit über 250g Startmasse zu fliegen, ist ein Online-Kompetenznachweis beim Luftfahrt-Bundesamt (LBA) zu erbringen. Ferner fallen Luftfahrzeuge, welche außerhalb der direkten Sichtlinie geflogen werden, in die registrierungspflichtige *spezielle Katgorie* und benötigen eine Betriebsgenehmigung der zuständigen Luftfahrtbehörde. Flüge in der Nähe fremder Grundstücke oder bei Nacht sind ebenfalls prinzipiell verboten. Eine praktische Anwendung des Systems ist also nicht ohne Weiteres möglich.

5.2 Probentransport

Einen Flug mit Nutzlast haben wir nicht in die Tat umgesetzt. Das Problem, Proben leicht zugänglich, doch geschützt vor Erschütterungen, Temperaturschwankungen und Witterung an der Drohne zu befestigen, ist noch zu lösen.

5.3 Mehrere Drohnen, mehrere Labore

Bisher können Lieferungen nur von einer einzelnen Drohne getätigt werden. Stünden im Labor mehrere Landeplattformen bereit, könnten auch mehrere Drohnen gleichzeitig agieren. Die Routenverteilung, so dass die Drohnen optimal genutzt werden, ohne dass auf Landeplätzen Kollisionen auftreten, ist algorithmisch zu lösen.

Auch die Verwaltung von mehreren Labore auf einmal würde interessante Wahlmöglichkeiten eröffnen. Falls Labore unterschiedlich spezialisiert sind, könnten die Nutzer*innen so für jede einzelne Lieferung den perfekten Empfänger auswählen. Die Proben könnten aber auch automatisch je nach Belastung der einzelnen Labore verteilt werden.

5.4 Probleme mit der Bilderkennung

Der Drohne fehlt noch die Möglichkeit, nach einer Landeplattform zu suchen, die einmal aus dem Blickfeld verschwunden ist. Außerdem könnten die Landeplattformen selbst mit Kameras ausgestattet werden, um die Positionsermittlung während der Landung zu vereinfachen. Aus Zeit- und Kostengründen haben wir uns gegen diese Variante entschieden.

5.5 Entwicklung eines eigenen Flight Controllers

Zu Beginn unserer Arbeit recherchierten wir umfassend über die Firmwares verschiedener Flight Controller, um die Drohne nicht von Grund auf selbst programmieren zu müssen. Die Arbeitsgruppe an der Brandenburgische Technische Universität Cottbus-Senftenberg (BTU) erklärte uns jedoch, ihren Bordcomputer völlig ohne vorgefertigte Software entwickelt zu haben, um mehr Kontrolle über die Drohne zu erhalten. Wir taten die Idee damals als zu aufwändig ab.

Stattdessen verzögerte der PX4-Flight Controller unsere Arbeit um viele Monate. Mit über 500.000 Zeilen Code [9] handelt es sich um ein hochkomplexes Programm (zum Vergleich: unsere Flask-Applikation

besteht aus knapp 2000 Zeilen). Die Ergründung von Fehlern beim Flug gestaltete sich durch mangelndes Verständnis der Firmware schwierig und wenig Erkenntnisreich. Gerade für Nachfolgeprojekte wäre es deswegen eine physikalisch, technisch und informatisch hochinteressante Aufgabe, einen eigenen Flight Controller für die Drohne zu entwickeln.

5.6 Alternativer Companion Computer

Auch die Kombination aus Raspberry Pi und Coral USB Accelerator erwies sich als unpraktisch. Trotz des Strebens, die benötigten Systeme so Leistungsarm wie möglich zu halten, ist der Pi vollständig ausgelastet. Um Erweiterungen zu ermöglichen, sollte dieser durch eine stärkere Platine ersetzt werden. Ein möglicher Kandidat ist das *Jetson TX2 Module* des amerikanischen Grafikprozessorenherstellers NVIDIA. Auch ist es denkbar, das Videosignal der Kamera während der Landung an einen externen Server zu streamen, statt alle Berechnungen an Bord der Drohne auszuführen.

6 Zusammenfassung

Im Verlauf unserer Arbeit stießen wir auf die Grenzen der heutigen Technologie. Die PX4-Firmware, welche unsere Drohne steuert, gilt als eine der besten weltweit. Dennoch hat sich gerade im Zusammenspiel mit unserem Companion-Programm gezeigt, dass noch technische Entwicklungen nötig sind, um Luftfahrzeuge so einfach steuern zu können wie Spielzeugautos.

Dass Prozessoren, wie sie auf der Drohne angebracht sind, erst 2015 erfunden wurden [10], zeigt, auf welch unerforschtem Gebiet sich unsere Arbeit bewegt. Gerade Alexanders neuronales Netzwerk zur Hinderniserkennung ist eine völlige Neuheit und setzt Standards in der Geschwindigkeit von Tiefenschätzungsalgotithmen.

Auch die Architektur des Servers war immerhin für uns eine bislang unbekannte Erfahrung, welche nicht nur von Erfolgen gezeichnet war. Dennoch ist der Server nun voll funktionsfähig und wurde anhand einer simulierten Version des Companion-Programms ausgiebig getestet.

Erste Teststarts der Drohne sind uns bereits geglückt. Bisher hatten wir jedoch keine Möglichkeit, unseren Prototypen als vollständiges System vom Klick auf der Webseite bis zur Landung auf der Plattform zu testen. Wir arbeiten mit mit Nachdruck daran, beim *Jugend forscht*-Bundeswettbewerb ein finales Produkt präsentieren zu können.

Literatur

[1] „U.S. Transportation Secretary Elaine L. Chao Announces FAA Certification of UPS Flight Forward as an Air Carrier“. In: *Federal Aviation Administration* (2019). URL: https://www.faa.gov/newsroom/us-transportation-secretary-elaine-l-chao-announces-faa-certification-ups-flight-forward.

[2] Megan Dickey. „UPS partners with drone startup Matternet for medical sample deliveries“. In: *TechCrunch* (2019). URL: https://techcrunch.com/2019/03/26/ups-partners-with-drone-startup-matternet-for-medical-sample-deliveries/.

[3] „LTE-Report für Deutschland: Telekom liegt vorn, O2 deckt nur 66 Prozent ab“. In: *Focus* (2018). URL: https://www.focus.de/digital/internet/netzabdeckung-getestet-grosser-lte-report-fuer-deutschland-telekom-ist-spitze-aber-o2-holt-stark-auf_id_8926691.html.

[4] Navigation National Coordination Office for Space-Based Positioning und Timing. *GPS Accuracy*. https://www.gps.gov/systems/gps/performance/accuracy/. Zugriff am 03.03.2022. 2022.

[5] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Techn. Ber. RFC 8446. Mozilla, Aug. 2018.

[6] Jae-Han Lee und Chang-Su Kim. „Multi-Loss Rebalancing Algorithm for Monocular Depth Estimation“. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.

[7] Andrew J. Davison Javier Civera und J. M. Martínez Montiel. „Inverse Depth Parametrization for Monocular SLAM“. In: *IEEE Transacitons on Robotics* 25.5 (2008).

[8] Bundesministerium für Verkehr und digitale Infrastruktur. *Zum Jahreswechsel gelten neue EU-Regelungen für Drohnen*. https://www.bmvi.de/SharedDocs/DE/Artikel/LF/drohnen.html. Zugriff am 29.11.2021. 2021.

[9] Dronecode. *PX4 Drone Autopilot*. https://github.com/PX4/PX4-Autopilot. Zugriff am 04.04.2022.

[10] Joe Osborne. „Google's Tensor Processing Unit explained: this is what the future of computing looks like“. In: *TechRadar* (2016). URL: https://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-this-is-what-the-future-of-computing-looks-like-1326915.

Unterstützungsleistungen

Förderverein des Max-Steenbeck-Gymnasiums e.V.

Der Förderverein des Max-Steenbeck-Gymnasiums e.V. hat die Einzelteile des Quadrocopters vollständig finanziert.

Brandenburgische Technische Universität Cottbus-Senftenberg

Dipl.-Ing. (FH) Thomas Bachmann, Anwendungsprogrammierer am Lehrstuhl Industrielle Informationstechnik, hat uns in das *Drohnen-Labor* der BTU eingeladen und stand und beim Kauf der Einzelteile beratend zur Seite.

Matthias Scholz-Dürrschmied, ehemals Technischer Mitarbeiter am Lehrstuhl Industrielle Informationstechnik, hat uns zu einem Flugtest seiner Projektgruppe an der BTU mitgenommen und uns Einblicke in das dort entwickelte Kartierungssystem mit autonomen Luftfahrzeugen ermöglicht.