

Bericht: Drohnenverfolgung mit PTZ-Kamera

Merlin Thierbach,
Vincent Kautz,
Pablo Rauh Corro

Rheinische Friedrich Wilhelm Universität Bonn
Projektgruppe Sensordatenfusion
Sommersemester 2023

Zusammenfassung. Die steigende Verbreitung von Drohnen für den Privatgebrauch führt zu einem erhöhten Bedarf an Systemen zur Detektion und Verfolgung von Drohnen. In dieser Arbeit stellen wir ein Drohnenverfolgungsprogramm vor, welches die automatische Verfolgung einer fliegenden Drohne durch eine PTZ-Netzwerk-Kamera ermöglicht. Das Programm verfügt über Scan- und Tracking- Modi und prozessiert iterativ aufgenommene Frames in einem Ablauf aus Detektion und Klassifizierung mit YOLO [1] (You Only Look Once) oder dem „Real-Time Detection Transformer“ [2] (RT-DETR), Tracking mit dem Kalman-Filter und Kamerasteuerung. Die Funktion des Drohnenverfolgungsprogramms wurde in mehreren Testszenarien erfolgreich demonstriert. Der Programmcode ist öffentlich verfügbar.

1 Einleitung

Mit der steigenden Verbreitung von zivilen Drohnen steigen auch die Möglichkeiten der schädlichen Verwendung. Ein Beispiel für eine solche schädliche Verwendung ist der Gatwick Airport Vorfall 2018[3], bei dem die Sichtung von Drohnen auf dem Flughafengelände zu einer temporären Einstellung des Flugverkehrs führte. Zivile Drohnen bergen jedoch nicht nur ein wirtschaftliches Gefährdungspotential. Auch in der Kriegsführung können zivile Drohnenmodelle genutzt werden, zur Aufklärung oder zum Abwurf von Sprengladungen[4]. Folglich gibt es ein hohes Interesse an wirksamen Systemen zur Erkennung von Drohnen, um gegebenenfalls nach einer Erkennung Abwehrmaßnahmen einleiten zu können.

Eine mögliche Methode zur Erkennung von Drohnen ist die optische Detektion mithilfe von Kameras und echtzeitfähigen Deep-Learning-Detektionsmodellen. Im Gegensatz zur Radardetektion ist diese Methode passiv. Es muss also keine Strahlung emittiert werden, was insbesondere im militärischen Kontext von Vorteil ist, um unentdeckt zu bleiben.

Die Detektionsmodelle müssen schnell prädizieren können, um eine Kamerasteuerung in Echtzeit zu ermöglichen, und sie müssen zuverlässig und präzise Drohnen detektieren können. Diese Anforderungen sollen durch state-of-the-art-Detektionsmodelle wie YOLO oder den RT-DETR erfüllt werden.

Da die Ergebnisse der Detektionsmodelle fehlerbehaftet sein können, ist für die Verfolgung der Flugbahn der Drohnen die Verwendung von Trackingalgorithmen von Vorteil. Ein besonders verbreiteter und fundamentaler Trackingalgorithmus ist der Kalman-Filter.

Im Laufe dieses Berichts werden zuerst die theoretischen Grundlagen der verwendeten Bilddetektionssysteme und des Trackingalgorithmus erläutert. Anschließend wird die Zusammenstellung unseres Datensatzes und der Programmstruktur unserer Implementation erläutert. Dann werden unsere Testszenarien und deren Bedeutung dargelegt. Abschließend wird ein Ausblick über mögliche zukünftige Verbesserungen des Programms gegeben.



Abb. 1. Detektion einer Drohne auf einem Flughafengelände [5]

2 Theorie

Im folgenden Kapitel werden die theoretischen Grundlagen für die verwendeten Bilddetektoren und des Trackingalgorithmus Kalmanfilter erklärt.

2.1 YOLOv8

YOLO ist ein Objektdetektionsmodell. Bei diesen handelt es sich um Konvolutionale Neuronale Netze (CNN), die durch mehrere konvolutionale Schichten Bildmerkmale extrahieren und mittels Backpropagation lernen, Objekte im Bild zu detektieren. Die beiden am weitesten verbreiteten Versionen von Objektdetektoren sind One-Stage-Detektoren und Two-Stage-Detektoren. Charakteristisch für Two-Stage-Detektoren, wie z.B. Faster-R-CNN [12], ist eine Vorverarbeitungsphase, in der potenzielle Regionen mit Objekten identifiziert werden. In einer zweiten Phase werden dann auf diesen Regionen Objektdetektionen durchgeführt. Zwei-Stufen-Detektoren sind oft sehr präzise, erfordern jedoch auch einen hohen Berechnungsaufwand.[13] YOLO hingegen ist ein One-Stage-Detektor. Im Gegensatz zu den Two-Stage-Detektoren wird die Objektdetektion

hier in einem einzigen Vorwärtsdurchlauf durchgeführt und das Netz verzichtet auf eine Vorstufe mit Regionsvorschlägen.

YOLOv8 unterscheidet sich architektonisch nur geringfügig von YOLOv5 und bringt im Wesentlichen Vereinfachungen in der Nutzbarkeit für Entwickler und Anwender mit sich. YOLOv5 wiederum ist auch eine architektonische Weiterentwicklung von YOLO[6].

Die Merkmalsextraktion wird in CNNs durch das Backbone durchgeführt. YOLOv8 verwendet das CSPDarknet53-Backbone. Dieses erweitert das ursprüngliche Darknet-Backbone um die Cross-Stage-Partial-Technik. Diese partitioniert Feature Maps und führt die Partitionen in einer späteren Schicht wieder zusammen.

Der sogenannte Head bildet Feature Maps, die es vom sogenannten Neck erhält, auf die finalen Prädiktionen ab. Der Neck dient als Verbindungsglied zwischen Head und Backbone. YOLOv5 verwendet als Neck ein Spatial-Pyramid-Pooling-Netzwerk (SPP). SPP wendet mehrere Pooling-Operationen mit verschiedenen Rastergrößen auf eine Feature-Map an und kombiniert die Resultate zu einer einzelnen Merkmalsrepräsentation. SPP hat einen positiven Effekt auf die Detektion von Objekten unterschiedlicher Größen und ermöglicht das Verwenden von Eingabebildern beliebiger Dimensionen [14].

2.2 RT-DETR

Der von Wenyu Lv et al. entwickelte erste in Echtzeit anwendbare „Detection Transformer“ (DETR) klassifizierte den COCOval2017-Datensatz schneller und präziser als alle YOLO-Konkurrenten. COCO (Common Objects in Context) ist eine umfangreicher Bilddatensatz und wird im Bereich der Computer Vision-Forschung für Präzisionsvergleiche verschiedener Modelle genutzt. Der Vorteil von DETRs ist es, die zeitaufwendige Nachbearbeitung von YOLO ähnlichen CNNs durch einen Encoder zu ersetzen. Da dies allerdings sehr rechenintensiv ist, waren DETRs lange nicht in Echtzeitanwendungen tragbar. Während der Entwicklung des RT-DETRs konnte das Bottleneck der DETRs in Form des Encoders identifiziert werden. Um diesen zu ersetzen wurde der „Efficient Hybrid Encoder“ entwickelt. Durch verschiedene Anpassungen konnte der RT-DETR weiter verbessert werden. Grundsätzlich wird zuerst ein Bild eingelesen, von dem einzelne Features mithilfe des Backbone HGNetv2 erkannt, Low-Level-Features ignoriert und die restlichen durch den eigens entwickelten Hybriden Encoder fusioniert werden. Aus den fusionierten Features wird eine feste Anzahl ausgewählt und mithilfe des Decoders werden deren Klassen ermittelt[2].

2.3 Evaluationsmetrik

Für die Bewertung der Qualität der Prädiktionen eines Detektionsmodells benötigt es einer Evaluationsmetrik. Diese beruht auf den bekannten statistischen

Größen Präzision und Recall, welche sich aus der Anzahl an true positives, false positives und false negatives berechnen.

$$\text{precision} = \frac{tp}{tp + fp} \quad \text{recall} = \frac{tp}{tp + fn} \quad (1)$$

Es muss jedoch erst definiert werden, wann eine Prädiktion überhaupt als true positive oder false positive gilt. Dazu wird der IoU-Wert verwendet. Bei diesem handelt es sich um den Quotienten aus der Schnittmenge und Vereinigung der vorhergesagten und tatsächlichen Bounding Box beziehungsweise Maske.

$$\text{IoU} = \frac{\text{prediction} \cap \text{ground truth}}{\text{prediction} \cup \text{ground truth}}. \quad (2)$$

Nun kann ein Objekt als true positive bzw. false positive eingeteilt werden, wenn der zugehörige IoU-Wert über oder unter einem definierten Schwellwert liegt. Als primäre Metrik wird oft die (irreführend benannte) Average Precision (AP) verwendet. Dabei handelt es sich nicht lediglich um den Mittelwert von Präzisionen nach obiger Formel. Die Präzision alleine genügt nicht als Metrik, da sie von false negatives unbeeinflusst ist. Das genaue Vorgehen zur Berechnung der AP kann sich im Detail unterscheiden. Im Allgemeinen wird jedoch eine Precision-Recall-Kurve bestimmt, welche Präzision und Recall für unterschiedliche Konfidenzwerte der Prädiktionen angibt, und es wird die Fläche unter der Kurve als Maßstab für die Qualität des Modells abgeschätzt[8]. Die primäre COCO-Präzisionsmetrik mAP₅₀₋₉₅ ist der Mittelwert der APs bei Verwendung von IoU-Schwellwerten von 0.5 bis 0.95 in Schritten von 0.05[9].

2.4 Kalman-Filter

Der Kalman-Filter ist ein Algorithmus, um aus einer Zeitreihe von fehlerbehafteten Messungen einen durch eine Wahrscheinlichkeitsdichte beschriebenen Zustand abzuschätzen. Der Kalman-Filter folgt direkt aus den fundamentalen Bayesschen Trackinggleichungen, wenn man die Annahme von normalverteilten Zuständen anlegt [7]. Der Kalman-Filter führt iterativ die Schritte Prädiktion und Filterung durch.

3 Datensatz

Es wurde deshalb ein neuer Datensatz erstellt, indem aus den öffentlichen Datensätzen eine vielfältige Auswahl von Aufnahmen von Drohnen in verschiedenen Entfernung, verschiedenen Umgebungen und Posen entnommen wurde. Weiterhin fügten wir eigene im Büro aufgenommene Bilder hinzu, da Innenaufnahmen in dem Datensatz stark unterrepräsentiert waren. Der Leitgedanke bei der Auswahl der Bilder war es, durch die Vielfalt der Bilder eine möglichst robuste Generalisierung des Detektionsmodells zu ermöglichen, sodass dieses auch auf neuen, unbekannten Daten zuverlässig detektiert. Der Datensatz wurde nach einigen Tests weiter verbessert. Dabei wurden größtenteils Objekte fotografiert,

die oft falsch als Drohne klassifiziert wurden, wie zum Beispiel Türklinken und Kabelgewirr. Dies resultierte letztendlich in einen Datensatz aus circa 1700 Bildern, welcher nun auch öffentlich (inklusive Annotationen) zur Verfügung gestellt ist (siehe Anhang). Der Datensatz wurde mit dem Annotationstool *Anylabeling* [11] annotiert. Die Bilder wurden nach Standardpraxis im Verhältnis 80:10:10 in Training-, Validierung- und Testdatensatz unterteilt [16].

4 Methodik

Im folgenden Kapitel wird auf die Detektionsmodelle, das Tracking sowie den Programmablauf und die Kamerasteuerung eingegangen. Abschließend werden noch einige Schwierigkeiten beschrieben, die im Laufe der Entwicklung auftraten.

4.1 Detektionsmodelle

Es wurde die Bibliothek Ultralytics verwendet, um YOLOv8 und RT-DETR zu trainieren und zu nutzen. Für beide Netze wurde ein Transfer Learning durchgeführt: Es wurde mit öffentlich verfügbaren Gewichten der auf COCO vortrainierten Netze gestartet und diese dann neu auf unserem Dronendatensatz trainiert. Die Standardwerte der Hyperparameter wurden nicht verändert. Das Training dauerte 270 Epochen bei YOLO und 297 Epochen beim RT-DETR. In beiden Fällen wurde das Training abgebrochen, weil keine Verbesserung in den letzten 50 Epochen erkennbar war. Für das Drohnenverfolgungsprogramm wurden die Gewichte jener Epoche gewählt, bei denen YOLO beziehungsweise RT-DETR die höchste Präzision auf dem Validierungsdatensatz zeigten.

Abbildungen 2 und 3 zeigen den Verlauf der Präzision auf dem Validierungsdatensatz und der Lossfunktion während des Trainings. Erwartungsgemäß steigt die Präzision zunächst deutlich und ändert sich im späten Verlauf des Trainings nur noch geringfügig. Der Verlauf der Lossfunktion entspricht ebenso dem erwarteten Verlauf, zunächst stark zu fallen und dann abzuflachen.

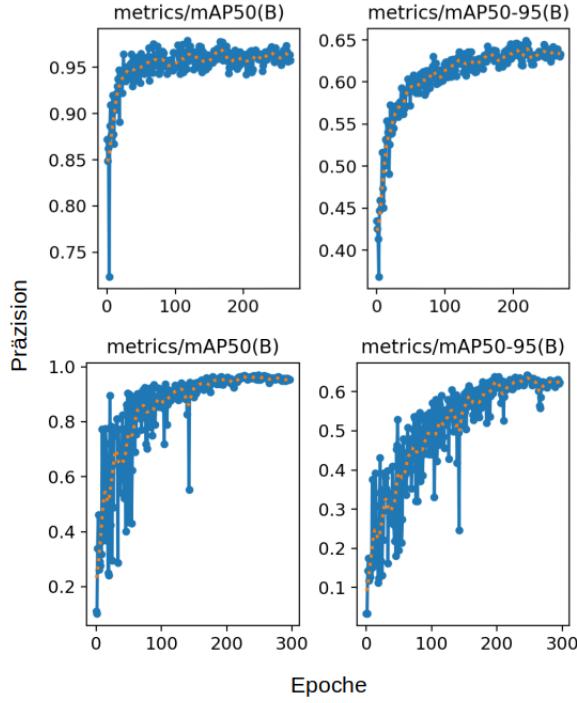


Abb. 2. Präzisionsmetriken mAP_{50} und mAP_{50-95} von YOLO (oben) und RT-DETR (unten) auf dem Validierungsdatensatz in Abhängigkeit von der Epochenzahl. Das Training wurde automatisch genau dann abgebrochen, wenn nach 50 Epochen keine Verbesserung mehr zu beobachten ist.

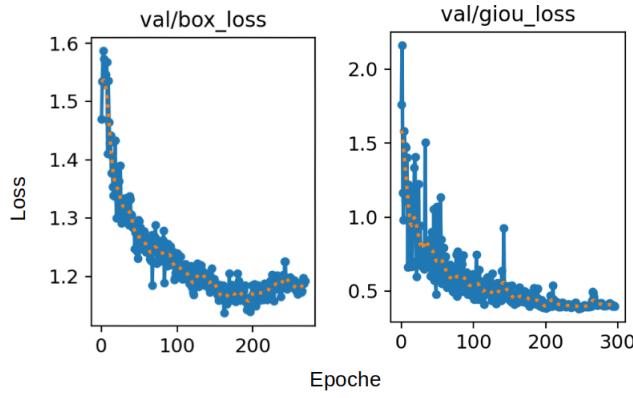


Abb. 3. Wert der Lossfunktion von YOLO (links) und RT-DETR (rechts) in Abhängigkeit von der Epochenzahl. Beide Lossfunktionen behandeln den Box-Loss.

Auf dem Testdatensatz wurde mit YOLOv8 bzw. RT-DETR eine Präzision mAP_{50–95} von 0.63 bzw 0.65 erzielt. Dabei wurden die Gewichte der Modelle wie oben geschildert gewählt.

4.2 Tracking

Für das Tracking wurde der einfache Kalman Filter mit den Schritten Prädiktion und Filterung implementiert. Hierbei wird nur das Detektionsergebnis mit dem höchsten Konfidenzwert berücksichtigt. Das Tracking erhält die Bildkoordinaten des Mittelpunkts der Detektionsbox als Eingabe.

Die Retrodiktion wurde in diesem Programm nicht umgesetzt, da sie im Sinne der Echtzeitverfolgung der Drohne keine Verbesserung bewirkt. Da die Flugbahnen nicht aufgezeichnet werden, gibt es auch keine Daten, die angepasst werden könnten.

Das Tracking ist derzeit nur für eine Kamera mit einem Objekt ausgelegt. Außerdem wird davon ausgegangen, dass jede Detektion zum gleichen Track gehört. Sollte ein Objekt während eines vordefinierten Zeitraums nicht detektiert werden, so gilt dieses als verloren und bei der nächsten Detektion wird ein neuer Track angelegt.

Als Zustandsvektor für das Tracking verwenden wir die X,Y -Koordinaten im Bild, sowie die Geschwindigkeiten in beide Richtungen in Pixel pro Millisekunde. Bei der Dynamik nehmen wir an, dass sich die Geschwindigkeit nicht ändert und die Position sich entsprechend der Geschwindigkeit und der Zeitdifferenzen ändert. Die Messfehlermatrix jeder Messung ist mit dem inversen Konfidenzwert der Detektionsbox gewichtet.

4.3 Programmaufbau

Der Quellcode des Programms kann öffentlich heruntergeladen werden, siehe Anhang.

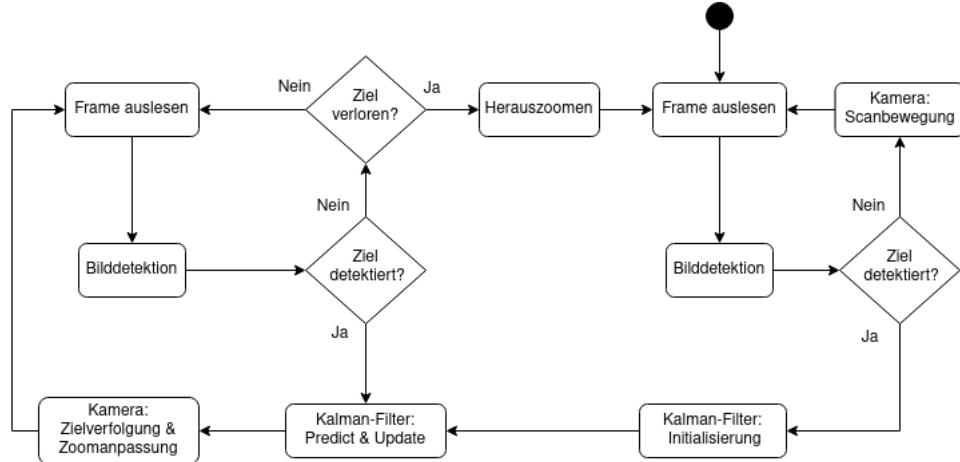


Abb. 4. Aktivitätsdiagramm des Programmablaufs

Ein grober Überblick des Programmablaufs ist durch das Aktivitätsdiagramm in Abbildung 4 dargestellt. Das Programm kann in zwei Modi unterteilt werden; einen Scanmodus und einen Trackingmodus.

Das Programm startet im Scanmodus. In diesem bewegt sich die Kamera bei minimaler Zoomstufe horizontal um sich selber, mit alternierender Aufwärts- und Abwärtsbewegung. Diese Bewegung dient dazu, regelmäßig unterschiedliche Bereiche des Kamerablickfelds zu erfassen. Im Scanmodus wird iterativ ein Kamerabild aufgenommen und für die Prädiktion an das Detektionsmodell YOLO (bzw. RT-DETR, je nach Konfiguration) übergeben. Erst wenn in einem Bild eine Dronendetektion auftritt, wird der Scanmodus verlassen und in den Trackingmodus übergegangen. Dabei wird der Kalman-Filter des Trackingmoduls (re)-initialisiert.

Auch im Trackingmodus wird iterativ ein Bild aufgenommen und auf diesem mit dem Detektionsmodell prädiziert. Die Detektionsergebnisse werden dann jedoch vom Kalman-Filter verwendet, um die tatsächliche Position der Drohne in Bildkoordinaten abzuschätzen. Anschließend werden die Tracking-Ergebnisse verwendet, um die Drohne zu steuern. Die Kameraausrichtung wird verändert, sodass sich ihr Bildmittelpunkt in Richtung des Trackingergebnisses bewegt. Der Trackingmodus wird verlassen, wenn das Ziel verloren ist. Das Ziel gilt als verloren, wenn in einem konfigurierbaren Zeitraum keine Detektion mehr erfolgte. Als zusätzliches Feature wurde implementiert, dass die Kamera abhängig von der Boxgröße der Detektion ihre Zoomstufe vergrößert oder verkleinert.

Ungewöhnlich mag erscheinen, dass im Fall einer Nicht-Detektion ohne Zielverlust auf das Verwenden des Trackingalgoritmus und anschließendes Adjus-

tieren der Kamerabewegung verzichtet wird. Der Grund dafür ist, dass die Kamerasteuerung über einen continuous-move-Befehl erfolgt (siehe Kapitel Kamerasteuerung). Im Tracking-Modus bewegt sich die Kamera bei Nicht-Detektion ohne Zielverlust also weiterhin in die gleiche Richtung wie in der letzten Iteration mit Detektion. Letztendlich stellt dies also keinen Unterschied dar zur alternativen Vorgehensweise, bei Nichtdetektion das Ergebnis der Kalman-Prädiktion für die Kamerasteuerung zu verwenden.

4.4 Kamerasteuerung

Die verwendete Kamera ist eine Axis-M5525-E-Netzwerkkamera. Hierbei handelt es sich um eine gewöhnliche PTZ-Kamera (Pan-Tilt-Zoom).



Abb. 5. Die verwendete Kamera, eine Axis M5525-E.

Im Allgemeinen erfolgt die Steuerung der Kamera und das Erhalten von Informationen von der Kamera durch das Senden von HTTP-Requests an die IP-Adresse der Kamera. Die genauen URLs der HTTP-Requests wurden der VAPIX-Network-Video-API entnommen[10].

Die Steuerung der Kameraausrichtung erfolgt in unserem Programm nicht durch explizite Angabe von absoluten Pan- und Tilt-Werten, sondern durch die Verwendung eines Continuous-Move-Requests. Dabei handelt es sich um einen Geschwindigkeitsvektor der Pan- und Tilt-Bewegung, der an die Kamera übergeben

wird. Der Geschwindigkeitsvektor wird so gewählt, dass seine Richtung zur Bildkoordinate des Trackingergebnisses zeigt. Die Größe des Geschwindigkeitsvektors wird proportional zur Differenz der Koordinaten von Bildmittelpunkt und Trackingergebnis gewählt.

Da bei steigendem Zoom das Sichtfeld kleiner wird, muss auch der Geschwindigkeitsvektor der Kamerabewegung abhängig von der Zoomstufe festgelegt werden. Ansonsten würde sich die Kamera bei hohem Zoom mit zu hoher Winkelgeschwindigkeit bewegen. Es wurde festgestellt, dass in diesem Kameramodell, entgegen der Erwartung, ein nichtlinearer Zusammenhang zwischen Zoomwert und Sichtfeldgröße besteht. Deshalb wurde der Zusammenhang mithilfe einer Leinwand und Maßband experimentell in einer Wertetabelle festgehalten. Diese Wertetabelle wird im Programm genutzt, um eine Proportionalität zwischen realer Sichtfeldgröße und Geschwindigkeitsvektor herzustellen.

4.5 Problematiken

Das Kameramodell ist nicht für den Anwendungszweck des Trackings optimiert. Ein Problem bestand darin, dass die Kamera, die sich horizontal frei drehen kann, am höchsten vertikalen Punkt einen blinden Fleck hat. Um diesen zu umgehen dreht sich die Kamera einmal um 180°, falls der Befehl zum nach oben drehen weiterhin gesendet wird. Dann werden intern innerhalb der Kamera die vertikalen Koordinaten mit -1 multipliziert. Dadurch drehte sich die Kamera, falls sie ein Objekt detektierte, in genau die entgegengesetzte der gewünschten Richtung. Obwohl dies in der Anleitung und den Spezifikationen der Kamera nicht erwähnt wurde, konnten wir es erkennen und das Problem beheben.

Des Weiteren konnte gelegentlich beobachtet werden, dass vom Programm Bilder verarbeitet wurden, die mehrere Sekunden von der Echtzeit zurückliegen. Diese Latenz übt sich stark negativ auf das Ziel des Echtzeittrackings aus, da die Kamerasteuerung dann mit Trackingergebnissen aus der Vergangenheit erfolgt. Die Latenz tritt insbesondere dann auf, wenn sich die Kamera mit hoher Winkelgeschwindigkeit dreht oder stark zoomt. Ein Benchmarking der Ausführungszeit des Programms offenbarte, dass die über TCP laufende HTTP-Kommunikation mit der Kamera manchmal deutlich länger dauert als im Normalfall. Das Bechmarking wurde durchgeführt, indem Zeitstempel vor Senden eines Requests und nach Vollenden des Requests erfasst werden. Im Normalfall hatte das Senden der Requests eine Dauer von ca 20ms. Es gibt jedoch auch Zeitintervalle, in denen diese Dauer temporär auf über 100ms ansteigt. Da pro Frame mehrere Requests gesendet werden müssen, und mit FPS-Raten von mindestens 10 gearbeitet wird, resultiert dies in einer erheblichen Verzögerung. Dies ist wahrscheinlich ursächlich für die beschriebene Latenz. Es wurden zwar Maßnahmen in Betracht gezogen, wie man diese Verzögerungen durch die Netzwerkkommunikation verhindern könnte. Ein Wechsel auf UDP ist nicht möglich, da die Kamera kein HTTP über UDP unterstützt. Auch ein Kappen der Verbindung bei

Verzögerungen wurde als nicht sinnvoll erachtet. Es handelt sich also letztendlich um eine Limitation durch die verwendete Hardware.

5 Tests und Untersuchungen

Als Testumgebung verwendeten wir entweder das Büro oder den Flur des Büros. Die meisten Tests fanden im Büro statt, in welchem der Abstand zwischen Kamera und Drohne maximal 4 Meter betrug. Alle Tests fanden in Innenräumen statt. Dies führte dazu, dass das Bild der Drohne zwar durchweg hochauflöst war, die Kamera sie aber auch mit relativ hohen Winkelgeschwindigkeiten verfolgen musste. Für die Bilddetektion wurde ein Computer mit einem Intel i9 sowie einer 2080 Ti und 1050 Ti verwendet. Die Bilddetektion wurde auf der 2080 Ti berechnet. Als Drohne wurde eine Tello EDU verwendet. Die Verfolgung der Drohne war während der Testszenarien für mehrere Minuten ohne Verlust der Drohne möglich.



Abb. 6. Verschiedene Detektionsergebnisse, falsch positive Detektion im rechts oberem Bild

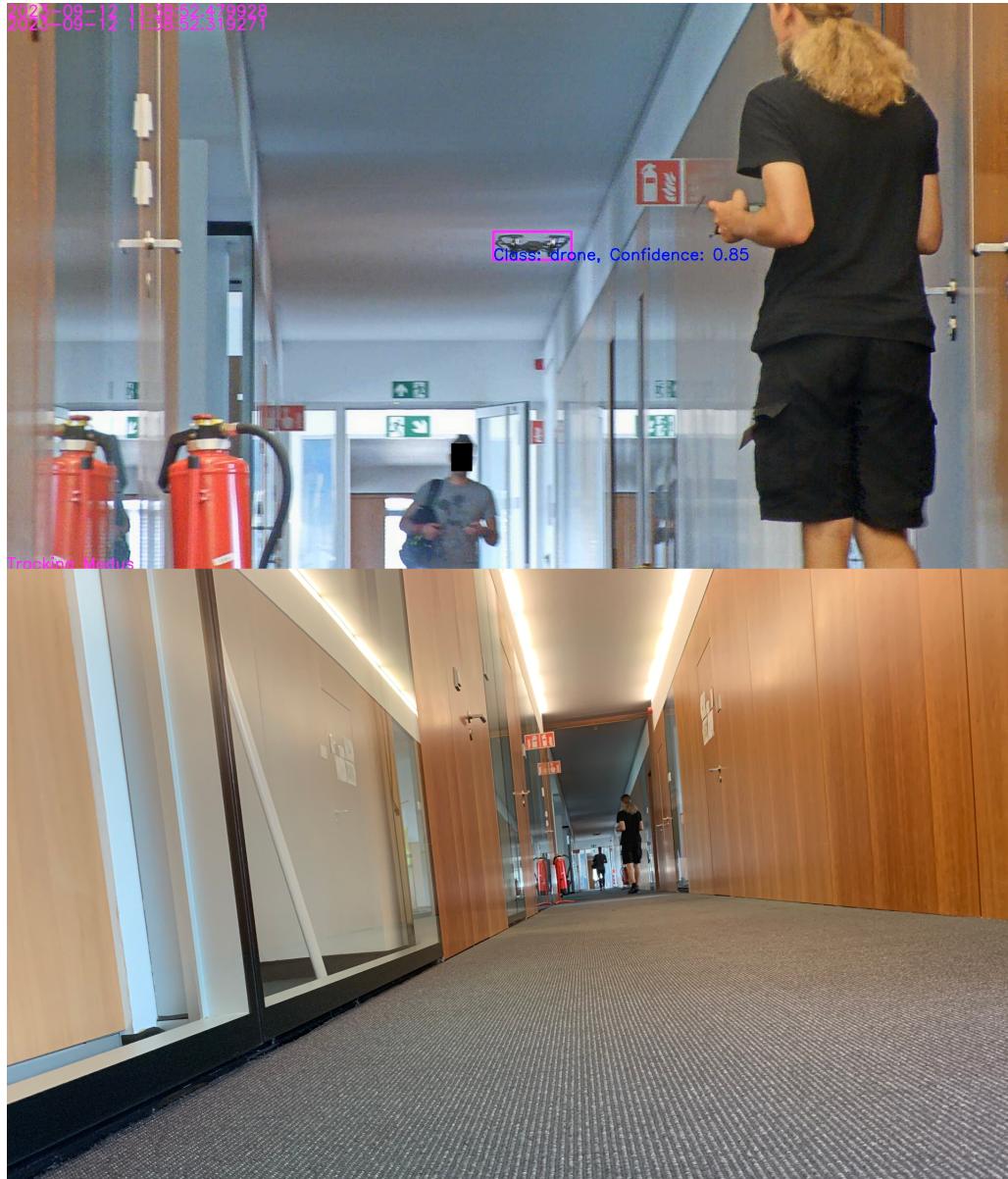


Abb. 7. Im oberen Bild ist das Ergebnis des Programms zu sehen, im unteren Bild die gleiche Szene ohne Zoom als Referenz.

6 Evaluation/Diskussion

Im folgenden Kapitel werden die Ergebnisse der Tests und die Auswirkungen der Nutzung des RT-DETR oder YOLO diskutiert. Dabei wird auf die Stärken und Schwächen des Programms sowie die Szenarien, in denen es anwendbar ist, eingegangen.

6.1 Qualität der Detektion

Die Erzeugung eines qualitativ hochwertigen Datensatzes und das Verwenden von leistungsstarken Detektionsmodellen ermöglichte die zuverlässige Detektion der Drohne in unserer Testumgebung.

Dennoch waren gelegentlich Falschpositivdetektionen zu beobachten, zum Beispiel von Kabelgewirr oder Zeichnungen auf einem Whiteboard. Dies liegt vermutlich an ähnlichen geometrischen Strukturen, die manchmal gut zu einer Drohne passen. Dieses Problem ließe sich beheben, indem man dem Datensatz mehr Bilder mit Drohnen ähnlichen Objekten, die keine Drohnen sind, zufügt.

Die Detektion im Scanmodus funktionierte auf geringen Distanzen von 4m zuverlässig. Bei größeren Distanzen funktionierte die Erstdetektion sehr unzuverlässig. Da als Testumgebung nur Innenräume zur Verfügung standen, konnte die Performanz der Detektion (und des Trackings) von sehr weit entfernten Drohnen nicht überprüft werden.

Leichte Unterschiede durch die Wahl des Detektionsmodells sind beobachtbar: RT-DETR schien in realen Testszenarien zuverlässiger zu detektieren (also mit weniger Falschmeldungen) als YOLO, jedoch auch langsamer als YOLO. YOLO benötigte für seine Prädiktion circa 8-30ms pro Frame, der RT-DETR hingegen ungefähr 30-50ms pro Frame. Die langsamere Prädiktion von RT-DETR gegenüber YOLO wirkt sich negativ auf die Drohnenverfolgung aus, da es dann eine höherere Latenz zwischen Bildaufnahme und Kamerasteuerung gibt. Für die Wahl des Detektionsmodells muss man also zwischen Detektionsqualität und Latenz abwägen.

6.2 Qualität der Drohnenverfolgung

Mit beiden Detektionsverfahren kann unsere Drohne in unserer Testumgebung zuverlässig mithilfe des Scanmodus gefunden und anschließend verfolgt werden. Lediglich bei zu hohen Winkelgeschwindigkeiten oder vor einer starken Lichtquelle wird die Drohne öfters verloren. Sobald die Drohne detektiert wird, zoomt die Kamera bis auf eine vorgegebene Boundingboxgröße in das Bild hinein, sodass eine verlässliche Verfolgung bei größeren Distanzen möglich ist. Eine zuerst auf geringer Distanz detektierte Drohne konnten wir mithilfe des Zooms auch dann noch verfolgen, wenn sie sich bis zu 20 Meter von der Kamera entfernte. Der Zoommodus ermöglichte also das Verfolgen von Drohnen bei Distanzen, in denen keine Erstdetektion mehr möglich ist.

7 Ausblick

In der beschränkten Entwicklungszeit von 3 Arbeitswochen konnten zwar alle notwendigen Eigenschaften des Dronentrackingprogramms implementiert werden. Dennoch gibt es noch Erweiterungen, mit denen sich das Programm verbessern ließe. Diese sollen hier kurz beschrieben werden.

Die gelegentlich im Detektionsschritt erzeugten Falschpositivmeldungen wirken sich stark negativ auf das Tracking aus, da sie zu einer falschen Kamerabewegung führen, die im schlimmsten Fall ein Zielverlust zur Folge hat. Prinzipiell können die Detektionsergebnisse durch eine Verbesserung, beziehungsweise Vergrößerung des Trainingsdatensatzes aufgebessert werden. Eine direktere Methode, diesen Falschpositivmeldungen entgegen zu wirken, wäre folgende: Falschpositivdetektion treten zumeist bei stationären Objekten (z.B. Türklinken) auf. Man merke sich bei einer Detektion die momentane Kamerausrichtung. Wenn Detektionen häufig in der exakt gleichen Kamerausrichtung registriert werden, so soll die Konfidenz dieser Detektionen niedriger gewichtet werden.

Für den Trackingalgorithmus wurde die einfachste Version des Kalman-Filters verwendet. Komplexere Ansätzen, wie z.B. das Multi-Hypothesis-Tracking, erzeugen robustere Trackingergebnisse und sind auf Szenarien mit mehreren Drohnen erweiterbar. Dafür könnte man mehrere Kameras verwenden und müsste Algorithmen aus der Multisensordatenfusion verwenden.

Die größte Einschränkung des Drohnenverfolgungsprogramms liegt wohl in der gelegentlich auftretenden Latenz. Der Wechsel auf eine für den Anwendungszweck des Trackings optimierte Kamera würde hier wahrscheinlich Verbesserungen bewirken.

8 Fazit

Es wurde ein Programm entwickelt, welches das Verfolgen einer Drohne mit einer PTZ-Netzwerkamera ermöglicht. Programmfunctionen wie Scan- und Trackingmodi sowie eine Zoomfunktion wurden erfolgreich in das Programm eingebaut und erläutert. Die beiden Detektionsmethoden YOLO und RT-DETR wurden implementiert und verglichen. Der Vergleich zeigte, dass bei der Wahl der Detektionsmethode eine Abwägung zwischen der schnelleren Ausführungsdauer von YOLO und der präziseren Detektion von RT-DETR getroffen werden muss. Eine Verbesserung der Dronendetektion wäre mit geeigneter Hardware oder durch die Verwendung von komplexeren Trackingmethoden möglich.

Literatur

1. Girshick, Ross and Donahue, Jeff and Darrell, Trevor and Malik, Jitendra: *You Only Look Once: Unified, Real-Time Object Detection* - URL <https://arxiv.org/abs/1506.02640>, Abrufdatum: 07.09.2023

2. Wenyu Lv, Yian Zhao, Shangliang Xu, Jinman Wei, Guanzhong Wang, Cheng Cui, Yunling Du, Qingqing Dang, Yi Liu: *DETRs Beat YOLOs on Real-time Object Detection* - URL <https://arxiv.org/abs/2304.08069v2>, Abrufdatum: 14.09.2023
3. The Guardian - URL <https://www.theguardian.com/uk-news/2020/dec/01/the-mystery-of-the-gatwick-drone>, Abrufdatum: 14.09.2023
4. Merkur - URL <https://www.merkur.de/politik/herstellung-entwicklung-beweglich-ziele-gegenoffensive-ukraine-russland-drohnen-92513273.html>, Abrufdatum: 15.09.2023
5. Dedrone - URL <https://www.dedrone.com/industry/airports>, Abrufdatum: 07.09.2023
6. Ultralytics - URL https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure, Abrufdatum: 07.09.2023
7. Wolfgang Koch *Advanced Target Tracking Techniques* - URL <https://www.sto.nato.int/publications/STO%20Educational%20Notes/RTO-EN-SET-086bis/EN-SET-086bis-02.pdf>, Abrufdatum: 14.09.2023
8. Github - URL <https://github.com/rafaelpadilla/Object-Detection-Metrics>, Abrufdatum: 7.09.2023
9. Cocodataset - URL <https://cocodataset.org/#detection-eval>, Abrufdatum: 7.09.2023
10. Axis - URL <https://www.axis.com/vapix-library/subjects/t10175981/section/t10035974/display>, Abrufdatum: 12.09.2023
11. Viet Anh Nguyen: *Effortless data labeling with AI support* - URL <https://github.com/vietanhdev/anylabeling>, Abrufdatum: 14.09.2023
12. Ren, Shaoqing and He, Kaiming and Girshick, Ross and Sun, Jian: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* - URL <https://arxiv.org/abs/1506.01497> Abrufdatum: 16.09.2023
13. Petru Soviany, Radu Tudor Ionescu *Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction* - URL <https://ieeexplore.ieee.org/document/8750729> Abrufdatum: 16.09.2023
14. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition* - URL <https://arxiv.org/abs/1406.4729> Abrufdatum: 16.09.2023
15. COCO - URL <https://cocodataset.org/#home>, Abrufdatum: 7.09.2023
16. Datensatzsplit - URL <https://cs230.stanford.edu/blog/split/>, Abrufdatum: 7.09.2023

Anhang

Der Programmcode kann hier heruntergeladen werden:

https://github.com/ProjektgruppeSDF/drone_tracking

Der für das Training erstellte Datensatz inklusive Annotationen kann hier heruntergeladen werden:

https://www.dropbox.com/sh/4uc76f7wkfismth/AAAF3n8RATF9KYvuUhkS_cufa?dl=0