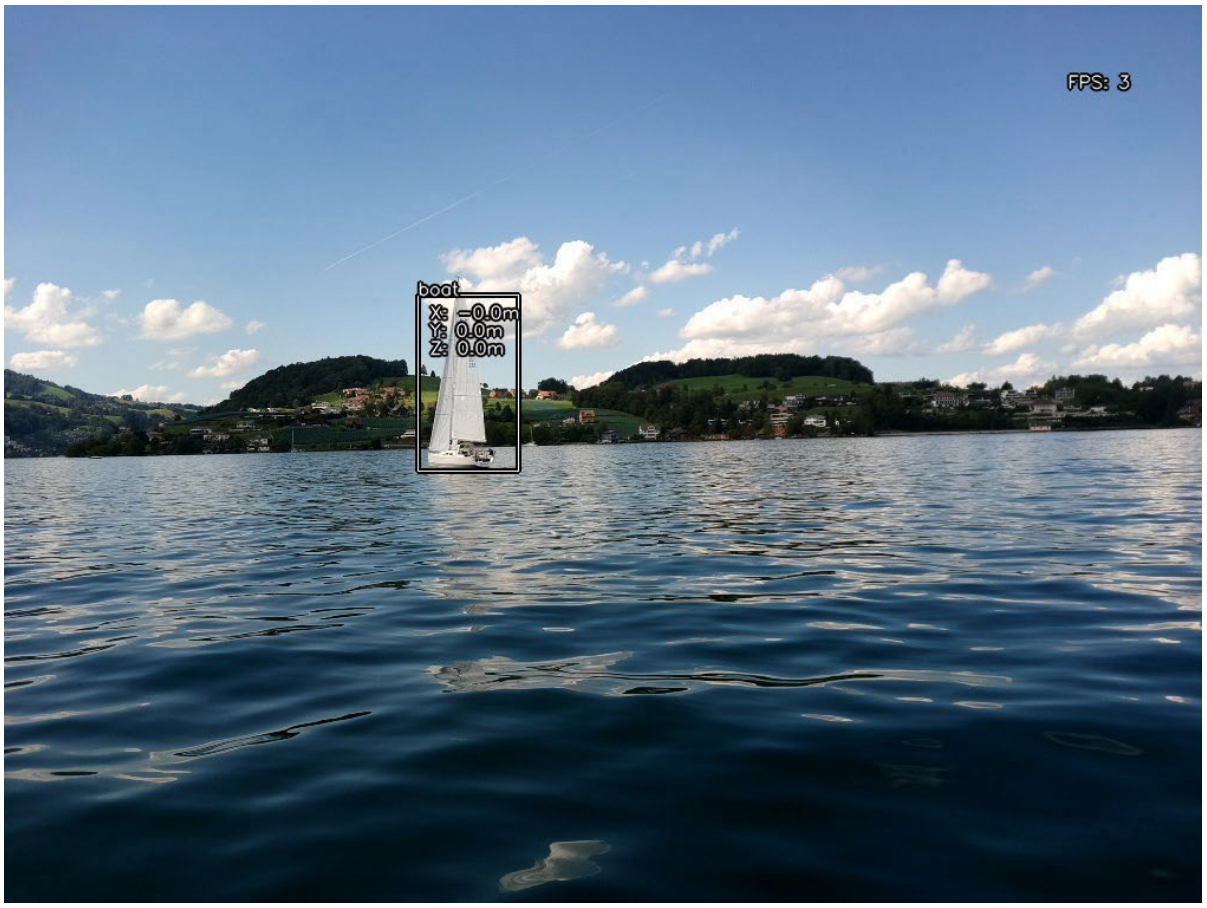
KANTONSSCHULE ZOFINGEN



ENTWICKLUNG EINES KOLLISIONSVERMEIDUNGSSYSTEMS FÜR WASSERFAHRZEUGE

Mit Hilfe von Objekterkennung und Stereo Vision

JOHANNES ZIMMERMANN
JOHANNES.ZIMMERMANN@GMX.CH
WETTBEWERBSARBEIT, APRIL 2024

Inhaltsverzeichnis

Abbildungsverzeichnis 1

Tabellenverzeichnis 2

1 Vorwort 4

2 Abstract 4

3 Einleitung 5

3.1 Ziel der Arbeit 5

3.2 Gründe für ein Kollisions-Vermeidungs-System für Boote 5

3.3 Wissenschaftliche Methode 7

3.4 Themeneingrenzung 7

3.5 Aufbau der Arbeit 7

3.6 Ähnliche Arbeiten 8

3.7 Hypothesen 8

4 Theorieteil 1 – OAK-D Stereokamera und Stereo Vision 9

4.1 Berechnung des Abstands mit Stereo Vision 9

4.2 OAK-D Stereokamera 11

4.3 Weitere Technologien 12

5 Theorieteil 2 – Yolov5-Algorithmus 14

6 Vorgehen und Produktentwicklung 16

6.1 Vorwissen, Grundvoraussetzungen und Anmerkungen 16

6.2 Programm Flow Chart und Logischer Aufbau 16

6.3 Vorbereitung der Entwicklungsumgebung 19

6.3.1 Verwendete Software 19

6.3.2 CuDNN und OpenCV mit GPU-Support 19

6.4 Implementierung und Programmstruktur 21

6.4.1 Programmstruktur 21

6.4.2 OAK-D Stereokamera Pipeline 21

6.4.3 Deployment von Yolov5-Algorithmus und Programm Loop 22

6.4.4 Berechnung weiterer Abstands-Koordinaten 24

6.4.5 Verwendung eines Pathfinding-Algorithmus auf einer Karte 25

6.4.6 Implementierung von Multiprocessing 25

6.4.7 Speicherung der generierten Daten – Logging Modul 26

6.5 Training des eigenen Yolov5-Algorithmus 27

6.5.1 Datenbeschaffung 27

6.5.2 Datenlabeling 28

6.5.3 Training 29
7 Versuchsaufbau der Distanzmessungen 31
7.1 Abstandgenauigkeit von verschiedenen Objekten auf 1-10m – Versuch 1 31
7.2 Abstandsberechnungen auf 8-25m – Versuch 2 32
7.3 Abstandberechnungsgenauigkeit mit unterschiedlichem Material – Versuch 3 33
7.4 Auf offenem Gewässer – Versuch 4 34
8 Ergebnisse 35
8.1 Versuch 1 35
8.2 Versuch 2 38
8.3 Versuch 3 40
8.4 Versuch 4 41
8.5 Yolov5 Trainingsperformance 42
8.6 Datenset Statistik 45
9 Diskussion 47
9.1 Interpretation der Messungsversuche 47
9.2 Genauigkeit der Abstandsmessungen 48
9.3 Interpretation der Yolov5-Ergebnistabellen 49
9.4 Genauigkeit und Performance des selbsttrainierten Yolov5-Algorithmus 49
9.5 Datenset 50
9.6 Skalierbarkeit der Software 50
9.7 Weitere Entwicklung der Software 51
9.7.1 Reichweite 51
9.7.2 Sichtfeld 51
9.7.3 Anwendung in dunkler Umgebung 52
9.7.4 Objektsegmentation 53
10 Reflexion 54
11 Beantwortung der Thesen 55
12 Januar 2024 – Verbesserung der Depth Estimation 57
12.1 Depth Queueing 57
12.2 Implementierung von Depth Validation 58
12.3 Depth Validation Formel 58
12.4 Proportionen eines Segelbootes – weitere nützliche Formeln 58
13 März 2024 – Überblick und Fortschritt 57
13.1 Ziel der neuen Ausrichtung 57
13.2 Trennung von Front End und Back End – Flask und ONNX Runtime 58
13.3 Yolov8, Sparsification und Segmentation 59

14 Referenzliste 60
15 Bestätigung 63
Anhang I 64
Anhang II 65
Programmierparadigmen 65
Instanz- und Statische Methoden 66
Vergleich von Klassen- und Objekt Methoden – Ein Beispiel 67

Abbildungsverzeichnis

Titelblatt (eigene Aufnahme)
Abbildung 1 Datenströme verschiedener Technologien werden kombiniert, um sich gegenseitig zu ergänzen (Mutschler, 2020) 6
Abbildung 2 Die Stereokamera liefert die zu analysierenden Daten, auf andere Quellen wird verzichtet (Staff & Schweiger, 2019) 6
Abbildung 3 Positionen desselben Punktes in zwei Stereobildern (Washington, 2009) 9
Abbildung 4 Darstellung ähnlicher Dreiecke bei der Abstandsberechnung (Washington, 2009) 9
Abbildung 5 Formel zur Berechnung der Depth (Washington, 2009) 10
Abbildung 6 die Epipolare-Ebene und Epipolare-Linie bei einer Stereokamera (Washington, 2009) 10
Abbildung 7 OAK-D Stereokamera Color Camera (innerhalb), Stereo Pair (ausserhalb) (Luxonis, 2022) 11
Abbildung 8 Technische Daten der OAK-D Stereokamera, nicht alle sind für dieses Projekt relevant (Luxonis, 2022) 11
Abbildung 9 Lidar Point Cloud von Strasse (Desk, 2017) 12
Abbildung 10 Depth Map aus Versuch 2 (eigene Aufnahme) 12
Abbildung 11 Übertragung der Positionsdaten zwischen Schiffen, Häfen und Satelliten (Marcom, 2022) 13
Abbildung 12 Auflistung von One-Stage- und Two-Stage-Detektoren (Bandyopadhyay, 2022) 14
Abbildung 13 Unterteilung des Bildes wird dargestellt (links), die vielen Erkennungen werden fusioniert (rechts) (Bandyopadhyay, 2022) 15
Abbildung 14 OAK-D Stereokamera Spezifikationen (Martinez-Carranza, 2021) 16
Abbildung 15 Vereinfachter Programmfluss in Form eines Flussdiagramms (eigene Darstellung) 18
Abbildung 16 Performance Unterschied zwischen CPU und GPU (eigene Darstellung) 19
Abbildung 17 Darstellung der Node-Struktur der OAK-D Pipeline (eigene Darstellung) 22
Abbildung 18 Programmloop und Yolov5 (eigene Darstellung) 23
Abbildung 19 Zentrumsmethode (blau), Gesamtmethode (rot) (eigene Aufnahme) 24
Abbildung 20 Darstellung des zweiten Python-Prozesses (eigene Darstellung) 25
Abbildung 21 Objekterkennung mit selbsttrainiertem yolov5l auf fremder Fotographie (imatrix, 2018) 27
Abbildung 22 Boote mit einer roten Box markiert und der Klasse Boot zugeordnet (eigene Aufnahme) 28
Abbildung 23 Faktoren mit denen veränderte Bilder generiert werden (Roboflow, 2022) 28
Abbildung 24 Das Modelmotorboot (links) und das Modellsegelboot (rechts) (eigene Aufnahme) 31
Abbildung 25 Versuchsaufbau für Versuch 3, während Messung ohne Mensch (eigene Aufnahme). 32
Abbildung 26 Die zu messende Fläche (rot) (eigene Aufnahme) 33
Abbildung 27 Stereokamera (links), Videokamera (rechts) (eigene Aufnahme) 34
Abbildung 28 Abstandsmessungen einer rechteckigen Box (eigene Darstellung) 35
Abbildung 29 Abstandsmessungen eines Segelbootmodells (eigene Darstellung) 36
Abbildung 30 Abstandsmessungen eines Motorbootmodell (eigene Darstellung) 36
Abbildung 31 Darstellung aller Messungen aus Versuch 1 (eigene Darstellung) 37
Abbildung 32 Messungen aus Versuch 2 (eigene Darstellung) 38
Abbildung 33 Mittelwerte mit Ergebnisbereich der Zentrumsmessung (eigene Darstellung) 38
Abbildung 34 Mittelwerte mit Ergebnisbereich der Gesamtmessung (eigene Darstellung) 39
Abbildung 35 Messungen eines Objekts mit Struktur und Textur (eigene Darstellung) 40
Abbildung 36 Beispielbild aus den gesammelten Bildern (eigene Aufnahme) 41
Abbildung 37 Es werden die vielversprechendsten Modelle (in rot) verglichen 44
Abbildung 38 Datenset Statistik (eigen Darstellung) 45

Abbildung 39 Die rote Linie stellt die Wasserlinie dar (eigene Darstellung) 46
Abbildung 40 Datenset Statistik (eigene Darstellung) 46
Abbildung 41 Zentrumsmessung ist gleich gross (blau), Gesamtmessung (rot) (eigene Aufnahme) 47
Abbildung 42 Luxonis Kamera mit einer Reichweite mit mehreren hundert Metern (luxonis, 2022). 51
Abbildung 43 Aufhellungseffekt von Zero-DCE (Guo, 2020) 52
Abbildung 44 Objekt-Segmentation mit Yolact (Bolya, 2018) 53
Abbildung 45 Intersection over Union (Tomar, 2023) 57
Abbildung 46 Depth Validation Formel (eigene Darstellung) 59
Abbildung 47 Breite und Gesamtlänge (eigene Darstellung) 60
Abbildung 48 Proportionen und Formeln 1 (eigene Darstellung) 61
Abbildung 49 Proportionen und Formeln 2 (eigene Darstellung) 62
Abbildung 50 Programmarchitektur CAS Nachfolgerversion (eigene Darstellung) 64
Abbildung 51 Yolov8 Modelle im Vergleich zu ihren Vorgängern (Jocher et al., 2023) 65
Abbildung 52 Sparsification und DeepSparse am Beispiel von Yolov5 (Ultralytics, 2021) 65

Tabellenverzeichnis

Tabelle 1 43
Tabelle 2 43

Abkürzungsverzeichnis

AIS	Automatic Identification System
CAS	Collision Avoidance System
CUDA	Compute Unified Device Architecture
CuDNN	NVIDIA CUDA® Deep Neural Network library
CPU	Central Processing Unit
GPS	Global Positioning System
GPU	Graphics Processing Unit
ISP	Image Signal Processor
IoU	Intersection over Union
LiDAR	Laser Imaging, Detection, And Ranging
MP	Megapixel
Radar	Radio Detection And Ranging
RAM	Random Access Memory
VRAM	Video Random Access Memory
ROI	Region of Interest
UI	User Interface
OOP	Object Oriented Programming
OOL	Object Oriented Language
ONNX	Open Neural Network Exchange
RGB	rot, grün, blau

1 Vorwort

Seit meinem ersten Jahr an der Kantonsschule Zofingen begeistere ich mich für Softwareentwicklung und Programmierung. Als ein Freund meiner Familie erzählte, dass er sich als Ein-Hand-Segler auf eine Weltreise machen wird, sprachen wir darüber, wie es ist, wenn man lange Strecken allein segelt und keine Wachablösung hat. So kam mir die Idee eine Software für Wasserfahrzeuge zu entwickeln, die Segler auf ihren Fahrten unterstützt. Ich war überrascht, dass es hierzu noch keine Überwachungssysteme gab, und fragte mich, wie man solch ein System entwickeln könnte. Bei der Umsetzung meiner Arbeit hat mich mein Betreuer Herr Franck unterstützt, wofür ich ihm sehr danken möchte. Auch möchte ich mich bei meinem ehemaligen Informatiklehrer Herr Scheppler bedanken, der mich im Herbst 2021 beraten hat, als meine Arbeit noch in den Kinderschuhen steckte.

2 Abstract

In dieser Arbeit wird ein *Collision Avoidance System*[1] (CAS) für Wasserfahrzeuge entwickelt. Dieses CAS soll in der Lage sein, mit Hilfe des selbsttrainierten Yolov5-Algorithmus, Objekte zu erkennen und die Distanz zu den Objekten mit der OAK-D Stereokamera zu berechnen. Ein *Pathfinding*- Algorithmus (A*-Algorithmus) wird in die Software implementiert und empfiehlt eine Umfahrungsroute. Diese Route garantiert keine unfallfreie Fahrt. In verschiedenen Versuchen soll ermittelt werden, mit welcher Genauigkeit Yolov5 Boote auf dem Vierwaldstättersee erkennt und wie genau die Abstandsmessungen der OAK-D Stereokamera bei verschiedenen Distanzen und Einflüssen sind.

Der Test des selbsttrainierten Yolov5-Algorithmus auf dem Vierwaldstättersee zeigt, dass Boote mit einem Abstand von bis zu 30m mit einer Genauigkeit von >90% erkannt werden. Die Evaluation mit dem Evaluationsdatenset liefert eine AP@0.5[2] von 0.809. Die Messungen der Stereokamera werden durch geringe Struktur und sich wiederholende Texturen negativ beeinflusst. Ab einem Messabstand von über 10m (bzw. 15 m) beträgt die Ungenauigkeit der Messung bis zu einem (bzw. mehrere) Meter. Es wird gezeigt, dass die entwickelte CAS-Software in der Lage ist, Objekt zu erkennen und zu lokalisieren. Somit ist es denkbar, dass Kollisionen vermieden werden können, indem der Fahrzeugführer frühzeitig alarmiert wird. Für noch höhere Sicherheit wird jedoch eine Stereokamera in einem höheren Preissegment[3] empfohlen.

[1] Kollisionsvermeidungssystem
[2] *Average Precision*
[3] 800-1000 Schweizer Franken

3 Einleitung

In diesem Kapitel wird die Ausrichtung dieser Arbeit umrissen, die Arbeitsmethodik wird besprochen und Ziele werden festgelegt.

3.1 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, ein *Collision Avoidance System*[4] (CAS) für private Wasserfahrzeuge zu programmieren, das mit Hilfe einer Stereokamera und Objekterkennung, prädestinierte Objekte erkennen, lokalisieren und eine mögliche Route, um diese Objekte herum, vorschlagen kann. Die berechneten Positionen werden dem Nutzer mitgeteilt, sodass dieser auf die Änderung der Situation reagieren kann und den Kurs anpasst. So könnte ein Unfall verhindert werden. Die zu entwickelnde Software ist für die Erkennung von Segel- und Motorbooten ausgelegt und ist daher für ein Szenario auf dem Wasser konzipiert. Der Fokus wird auf die relative Positionserkennung zwischen Booten und Kamera, und auf die Objekterkennung gesetzt. Dafür wird ein neuronales Netzwerk mit einem selbst zusammengestellten Datenset trainiert und daraufhin getestet. Es wird ein *Pathfinding*-Algorithmus mit Hilfe einer Bibliothek zu Demozwecken implementiert. Dessen vorgeschlagene Route ist lediglich eine Routenempfehlung anhand der erhaltenen Objektpositionen. Die Route garantiert keine unfallfreie Fahrt. Die Entwicklung eines Autopiloten, der in der Lage ist, auf komplexe Szenarien zu reagieren, dementsprechend autonom zu handeln und eine möglichst optimale Route um Objekte zu finden, ist nicht Teil dieser Arbeit. Es werden isoliert *sensorische Komponenten* eines Autopiloten entwickelt.

3.2 Gründe für ein Kollisions-Vermeidungs-System für Boote

In der kommerziellen Frachtschifffahrt finden Objekterkennungs- und Lokalisierungssoftware basierend auf verschiedensten Technologien im grossen Stile Anwendung. Im Sektor der privaten Schifffahrt ist diese Anwendung noch nicht sehr verbreitet.

In der privaten (Segel-)Schifffahrt besteht das Bedürfnis, kurzfristig mit geringer Latenz vor Hindernissen gewarnt zu werden, um Unfälle zu vermeiden. Dies ist eher für die Privatschifffahrt von Bedeutung, da diese Fahrzeuge häufig keinen konsistenten und langfristigen Kurs verfolgen. Die Routen sind deshalb schwieriger vorauszusagen und es kann daher eher zu unerwarteten Kollisionen kommen. Daher richtet sich das CAS eher an die Erkennung und Lokalisierung von Objekten in einem näheren Umfeld, die eine direkte Gefahr darstellen.

Ein wichtiger Faktor sind auch die höheren Kosten in der Berufsschifffahrt, die bei einem Unfall auftreten, wie zum Beispiel hoher Personenschaden bei Kreuzfahrtschiffen oder Warenverlust und Umweltbeeinträchtigung durch Containerschiffe. Daher sind grosse Unternehmen gewillt deutlich höhere Summen in die Minimierung des Crashrisikos und in die entsprechenden Sicherheitstechnologien zu investieren. Um die Sicherheit weiter zu erhöhen, werden Technologien kombiniert (siehe Abbildung 1). Privatpersonen können meistens viel weniger für die erwähnten Sicherheitstechnologien ausgeben, geschweige denn auf eine Kombination von mehreren Technologien zurückgreifen. Deshalb muss für Privatpersonen ein Kollisionsvermeidungssystem budgetorientiert und erschwinglich sein. Es ist auch wichtig zu berücksichtigen, dass die benötigte Rechenleistung ansteigt, je mehr Technologien kombiniert werden, weil mehr Daten verarbeitet werden müssen. Deshalb kommt für dieses Projekt nur eine Abstandsmessungstechnologie in Frage und keine Kombination.

[4] Kollisionsvermeidungssystem

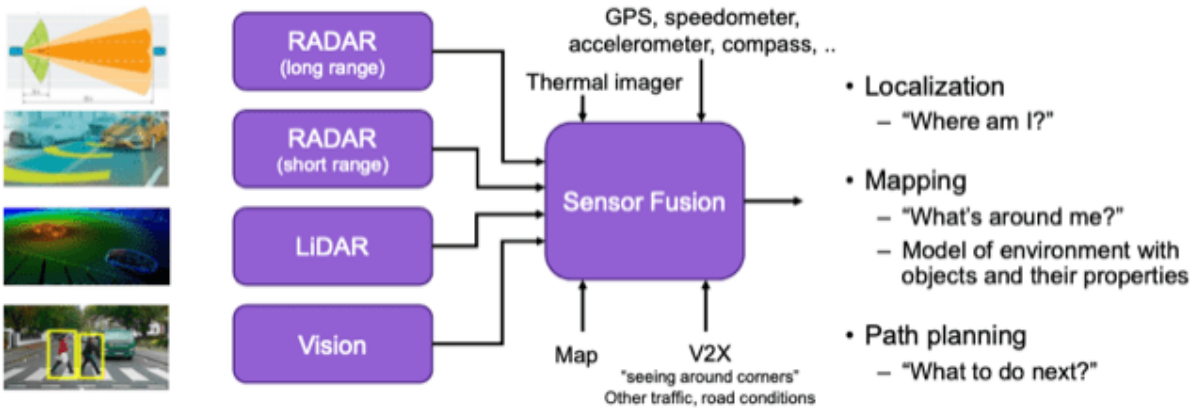


Abbildung 1 Datenströme verschiedener Technologien werden kombiniert, um sich gegenseitig zu ergänzen (Mutschler, 2020)

Durch Stereo Vision können aus den erhaltenen Bildern einer Stereokamera zwei Informationen entnommen werden, nämlich Farbbild und Abstand eines Objekts. Stereokameras sind im Vergleich zu beispielsweise Lidarsensoren, deutlich billiger und sind deshalb für dieses Projekt die optimale Technologie. In dieser Arbeit hat man sich deshalb dazu entschieden ein Produkt zu entwickeln, das nur auf kamerabasierter Datengewinnung aufbaut, um zu erforschen, inwiefern sich die entwickelten Komponenten eines CAS für die praktische Anwendung eignen könnten und wie genau die Positionserkennung von Segelbooten mit der gewählten Stereokamera ist.

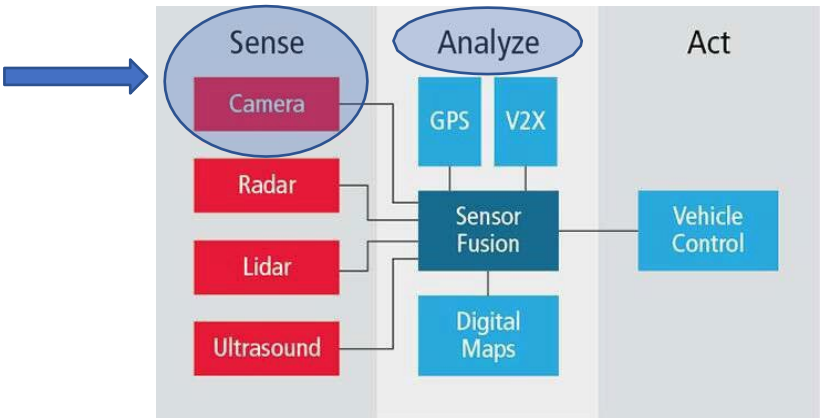


Abbildung 2 Die Stereokamera liefert die zu analysierenden Daten, auf andere Quellen wird verzichtet (Staff & Schweiger, 2019)

3.3 Wissenschaftliche Methode

Diese Arbeit befasst sich mit der Entwicklung eines Kollisionsvermeidungssystems (CAS), mit der Trainingsdatenerhebung und den anschliessenden Versuchen auf dem Vierwaldstättersee zur Evaluation der Performance, die dieses Produkt erbringen kann. Die quantitative empirische Forschungsmethode, die in Versuchen angewendet wird, wird darüber Aufschluss geben, wie genau die Objekte lokalisiert werden können und ob sich die entwickelten Komponenten somit für die Anwendung in der Praxis eignen.

3.4 Themeneingrenzung

Für die Entwicklung eines Kollisions-Vermeidungs-Systems mit Objekterkennung und Objektlokalisierung kommen verschiedenste Technologien in Frage. In diesem Projekt wird nur eine Stereokamera zur Entfernungsberechnung von Objekten eingesetzt. Diese Entscheidung wurde getroffen, da Stereokameras im Vergleich zu anderen Technologien, wie Lidar Sensoren, Radargeräte mit langwelliger Strahlung oder anderen *Time of Flight*-Sensoren, im unteren Leistungssektor ein gutes Preis-Leistungsverhältnis bieten, und daher gut zu den Ansprüchen der angestrebten Zielgruppe passen.

In dieser Arbeit wird ein Programm entwickelt, das verschiedene Komponenten verbindet. Spezielles Userinterfacedesign, *Packaging* und *Distribution*[5] des Programms sind nicht der Fokus dieser Arbeit und werden daher vernachlässigt. Das Userinterface (UI) muss in der Lage sein, die wichtigsten Informationen darzustellen, wie zum Beispiel, die Positionen der Objekte, Achsenabschnitte der Karte und die Videoübertragungen mit den markierten Objekten.

Ein grosser Teil dieser Arbeit ist neben dem Programmieren, das Training des Yolov5-Algorithmus und die Erstellung eines Datensets. Es wird grosser Wert auf Versuche zur Überprüfung des Produkts gelegt. Es ist beispielsweise nicht das Ziel, das grösste und beste Datenset zusammenzustellen. Das Ziel ist es, die vielen verschiedenen Komponenten zusammenzubringen und zu einer funktionstüchtigen Software zu verbinden.

3.5 Aufbau der Arbeit

Die Logik des Programms wird erklärt und die Entwicklung des Programms wird geschildert. Es wird auf das Training des Algorithmus eingegangen und die Erstellung des Datensets. Darauffolgend wird mit einer Versuchsreihe und der späteren Interpretation der Daten die Produktperformance verifiziert und die Leitfragen beantwortet. In der Reflexion wird der Arbeitsprozess kritisch hinterfragt und beurteilt. Englische Begriffe werden teilweise deutschen Äquivalenten vorgezogen, da diese in der Praxis gebräuchlicher sind. Alle Abkürzungen sind im Abkürzungsverzeichnis einsehbar.

Im Anhang I (digital) sind Dateien des Projekts gespeichert. Darunter sind der Code, das Datenset, die trainierten Algorithmen, Produktdemo, Präsentation und Bilddateien, die bei den Versuchen entstanden sind, abgespeichert. Durch Scannen des QR-Codes auf dem Titelblatt wird der digitale Anhang I aufgerufen.

Eine optionale Erklärung von Objekt-Orientierter-Programmierung (OOP) finden Sie in Anhang II.

[5] Unter *Packaging* und *Distribution* wird das Verpacken und die Verteilung der Software an den Endnutzer verstanden

3.6 Ähnliche Arbeiten

Da in den letzten Jahren autonomes Fahren sowohl von Wasservehikeln als auch landbasierten Fahrzeugen ein großes Thema war, und weiterhin sein wird, existieren verschiedenste wissenschaftliche Arbeiten von diversen Universitäten. Im letzten Jahrzehnt kam es zu bedeutenden Entwicklungen in der Sphäre der autonomen Wasserfahrzeuge. Diese behandeln jedoch meist nur Untergebiete und widmen sich Teilgebieten, die auch Teil dieser Arbeit sind. Jedoch muss gesagt werden, dass die Implementierung einer autopilotähnlichen Software im Zusammenspiel mit dem Yolo-Algorithmus und einer Stereokamera, auch nach gründlicher Suche, vorerst einzigartig erscheint.

An der Kantonsschule Zofingen wurden bereits Arbeiten zum Thema *Machine Learning* durchgeführt, jedoch nicht im Zusammenhang mit Computervision und der Entwicklung einer CAS-Software. In den letzten Jahren ist jedoch ein positiver Trend im Thema Machine Learning zu erkennen.

3.7 Hypothesen

Die Beantwortung folgender Hypothesen wird im Laufe der Arbeit angestrebt, diese werden verifiziert oder widerlegt.

Es ist möglich mit der entwickelten Software die Mehrheit alle prädestinierten Objekte (Motorboote und Segelboote) mit Hilfe von Objekterkennung zu erkennen.

Die Objekterkennung und Messungsungenauigkeit der Stereokamera zu einem Objekt lassen zu, dass ein Crash verhindert werden kann.

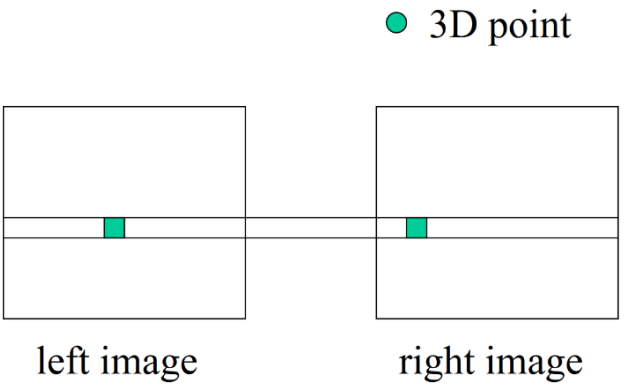
Die Performance (in Bezug auf Bildwiederholungsrate und die Erkennungsrate) der Software ist hoch genug, um für die Verwendung in der Praxis angemessen zu sein.

4 Theorieteil 1 – OAK-D Stereokamera und Stereo Vision

Hier wird vorgestellt, wie die Berechnung eines Abstandes mit einer Stereokamera erfolgt und andere Technologien zur Abstandsmessung werden kurz vorgestellt.

4.1 Berechnung des Abstands mit Stereo Vision

Bevor man den Abstand eines Objekts mit Hilfe von Stereo Vision berechnen kann, muss man zuerst die *Pixel Disparity* (Pixel-Ungleichheit) eines 3D-Punktes im Raum berechnen. Der Abstand zu einem Objekt wird auch als *Depth* bezeichnet. Die *Pixel Disparity*[6] (*d*) ist die Pixeldifferenz der Bildpositionen eines 3D-Punktes in einem Stereobildpaar. Der 3D-Punkt wird einem Stereobildpaar aus zwei Perspektiven abgebildet und hat deshalb eine unterschiedliche Position.



d = X-Wert des linken Bildes – X-Wert des rechten Bildes

Abbildung 3 Positionen desselben Punktes in zwei Stereobildern (Washington, 2009)

In Abbildung 4 wird die Ähnlichkeit verschiedener Dreiecke gezeigt. Das Verhältnis von Depth (z) und Brennweite (f), und von x und xl[7] ist das gleiche. Dasselbe gilt auch für die Verhältnisse der y-Werte. Die Differenz zwischen *xl* und *xr* ist die *Pixel Disparity.* In Abbildung 4 ist die Baseline als *b* bezeichnet, die Baseline ist der Abstand zwischen den Kameras. Eine Vergrösserung der Baseline steigert die Reichweite einer Stereokamera. Die *Image Planes* sind die Bildflächen.

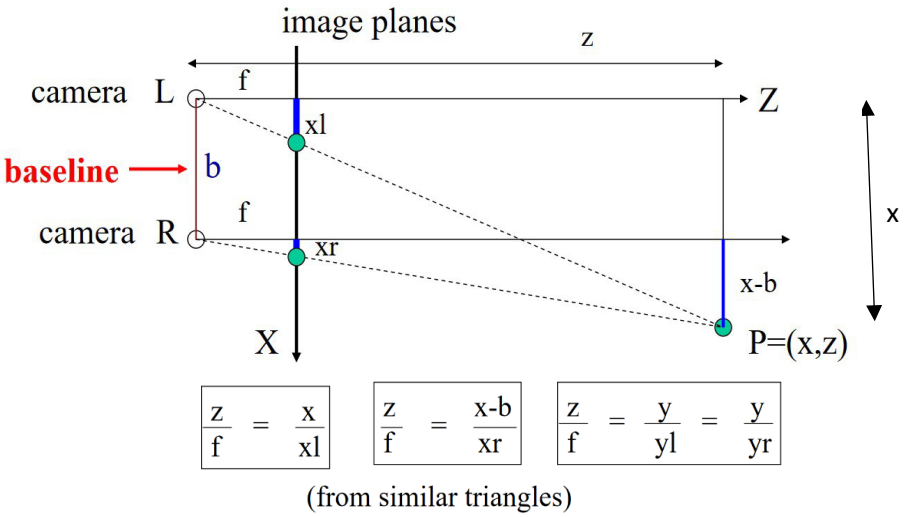


Abbildung 4 Darstellung ähnlicher Dreiecke bei der Abstandsberechnung (Washington, 2009)

[6] Die *Pixel Disparity* wird in Pixeln angegeben

[7] Steht für den x-Wert des Pixels im linken Bild.

Mit diesen Informationen aus Abbildung 4 kann nun die *Depth* (z) berechnet werden. Dafür wird die Brennweite (f), die Baseline (b) und die *Pixel Disparity* (d) benötigt (siehe Abbildung 5). Sobald *z* bestimmt ist lassen sich auch die weiteren Werte *x* und *y* berechnen. Dies wird in Kapitel 6.4.4 implementiert.

$$\text{Depth} \quad z = f*b / (xl - xr) = f*b/d$$

$$x = xl*z/f \quad \text{or} \quad b + xr*z/f$$

$$y = yl*z/f \quad \text{or} \quad yr*z/f$$

Abbildung 5 Formel zur Berechnung der Depth (Washington, 2009)

Damit man die *Pixel Disparity* berechnen kann, müssen erst einmal die zueinander passenden Pixel gefunden werden. Dafür muss nicht das gesamte Bild durchsucht werden. Weil die Bilder auf einer Epipolaren-Ebene sind, muss nur die Epipolare-Linie im Bild durchsucht werden. Die Epipolare-Ebene ist die Fläche, die entsteht, wenn die Punkte *C1, C2* und *P* verbunden werden (Washington, 2009). Wenn die Kameras auf der gleichen Achse angeordnet sind und gleich ausgerichtet sind befindet sich der entsprechende Pixel zwangsläufig auf der Epipolaren-Linie. Die Texte dieses Kapitels bauen auf der folgenden Quelle auf (Washington, 2009). Weitere Informationen zum *Stereo Correspondance Problem* werden unter dieser Quelle gefunden.

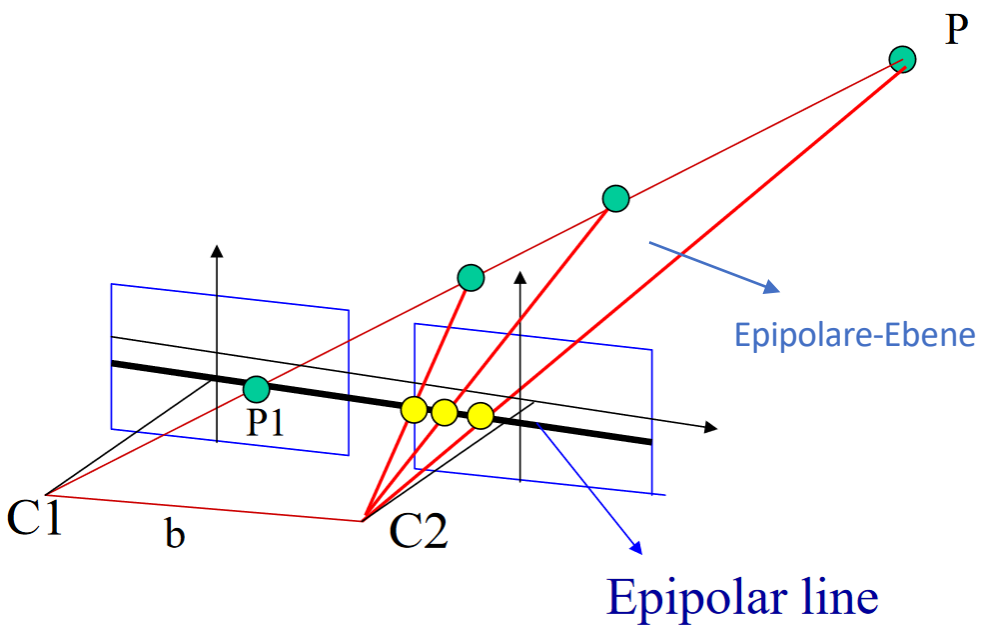


Abbildung 6 die Epipolare-Ebene und Epipolare-Linie bei einer Stereokamera (Washington, 2009)

4.2 OAK-D Stereokamera

Die OAK-D Stereokamera wurde von *Luxonis* entwickelt (Luxonis, 2022). Diese Kamera zeichnet sich dadurch aus, dass sie für einen Kaufpreis von 200 USD[8] eine erstaunliche Leistung hat und reich an Features ist. Mit einem horizontalen Sichtfeld (HFOV) von 69°, ist die Farbkamera[9] für diese Arbeit geeignet. Die die beiden Monochrom-Kameras[10] verfügen über eine Baseline von 7.5cm. Die Kamera könnte als Stereokamera (äussere Sensoren) mit einem zusätzlichen Farbkamera (mittig) bezeichnet werden. Das *Stereo Pair* hat eine maximale Bildwiederholungsrate von 120 Bildern pro Sekunde. Dies bewirkt, dass immer die aktuelle *Depth Map* an den Computer geschickt wird. Für die Farbkamera wird in diesem Projekt eine Bildrate von 30 Bildern pro Sekunde verwendet, da diese Option eine höhere Bildqualität zulässt. Die Kamera verfügt auch noch über integrierte Recheneinheiten, die bestimmte Aufgaben übernehmen können. In diesem Projekt werden diese Recheneinheiten verwendet, um die *Depth Map* zu berechnen. Andere Berechnungen werden jedoch vom *Host*[11] übernommen.

Abbildung 7 OAK-D Stereokamera Color Camera (innerhalb), Stereo Pair (ausserhalb) (Luxonis, 2022)

Camera Specs	Color camera	Stereo pair
Sensor	IMX378	OV9282
DFOV / HFOV / VFOV	81° / 69° / 55°	82° / 72° / 50°
Resolution	12MP (4032x3040)	1MP (1280x800)
Focus	Auto-Focus: 8cm - ∞	Fixed-Focus: 19.6cm - ∞
Max Framerate	60 FPS	120 FPS
F-number	2.0	2.2
Lens size	1/2.3 inch	1/4 inch
Pixel size	1.55µm x 1.55µm	3µm x 3µm

Abbildung 8 Technische Daten der OAK-D Stereokamera, nicht alle sind für dieses Projekt relevant (Luxonis, 2022)

[8] Der eigentliche Kaufpreis in Deutschlag lag im Frühling 2022 bei ca. 320 EUR
[9] In Abbildung 8 als *Color Camera* bezeichnet
[10] Werden zusammen als *Stereo Pair* bezeichnet
[11] Der verbundene Computer

4.3 Weitere Technologien

Bei der kommerziellen Frachtschifffahrt sollen Software und installierte Hardware, wie unter anderem Radar, Lidarsensoren, Kamerabasierte Technologie und das *Automatic Identification System*[12] (AIS), bereits lange im Voraus vor anderen Verkehrsteilnehmern und Hindernissen warnen. Radar[13] arbeitet mit langwelliger Strahlung, während Lidarsensoren kurzwelligere Strahlung verwenden. Der Lidarsensor sendet kurze Pulse[14] von Strahlung in einem steuerbaren und eng fokussierten Strahl aus. So ist er in der Lage mehrere Millionen von Punkten zu produzieren, deren Genauigkeit zwischen wenigen Millimetern bis zu wenigen Zentimetern variiert (*Lidar Mapping*, 2022). Die erhaltenen Punkte werden in sogenannten *Point Clouds* dargestellt.

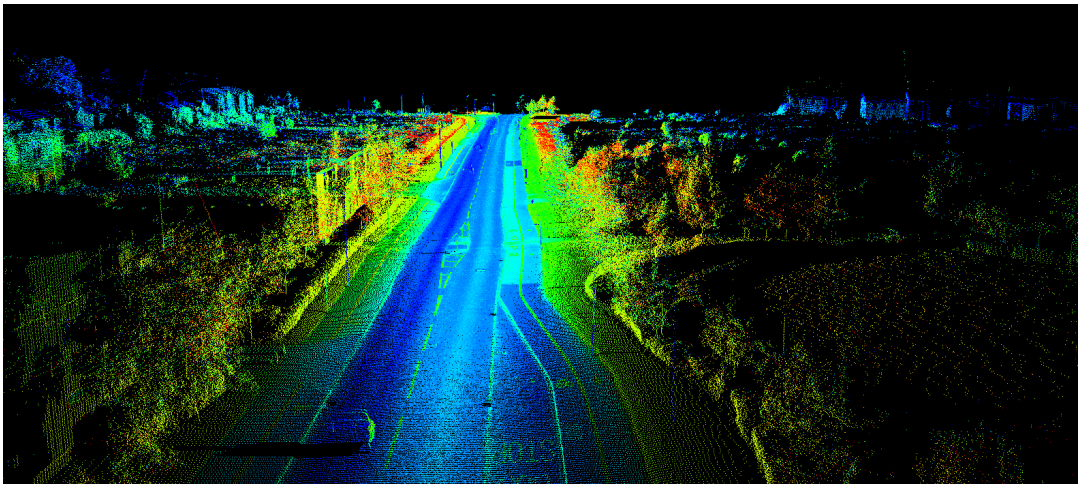


Abbildung 9 Lidar Point Cloud von Strasse (Desk, 2017)

Mit Hilfe einer Stereokamera, einer Kamera mit zwei Bildsensoren, werden zwei Bilder aus zwei unterschiedlichen Perspektiven geschossen, mit Hilfe von Stereo Vision wird daraufhin für jeden Pixel ein Tiefenwert berechnet (siehe Abbildung 10). Das Farbbild kann auch weitere Verwendungszwecke haben, wie zum Beispiel Objekterkennung.

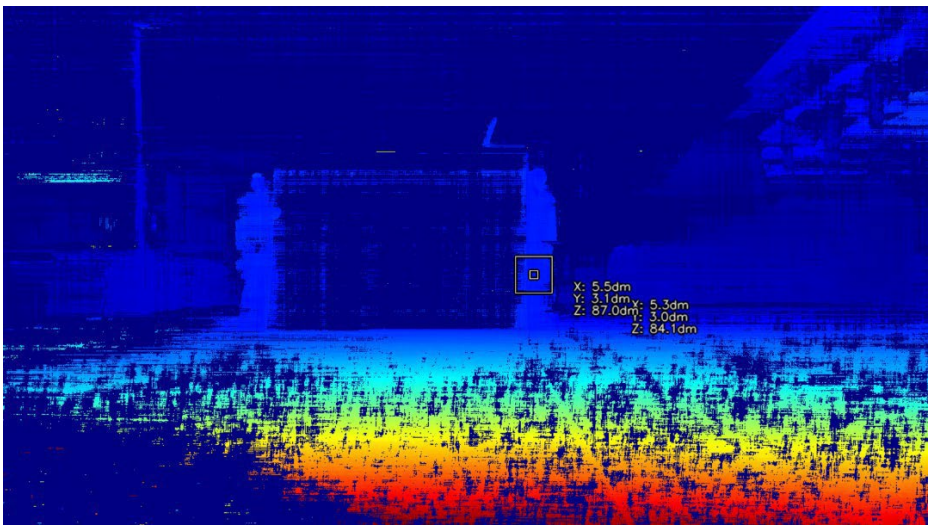


Abbildung 10 Depth Map aus Versuch 2 (eigene Aufnahme)

[12] Automatischen Identifikationssystem
[13] Häufig wird in Radar mit langer und kurzer Reichweite unterteilt
[14] Es existieren auch andere Lidarsensortypen, die kontinuierlich Strahlung aussenden

Das AIS ist dafür verantwortlich, dass die geographische Position in einem Radius von 20 bis zu maximal 350 nautischen Meilen an andere Schiffe und Häfen gesendet werden, zusätzlich können die Positionsdaten auch an Satelliten gesendet werden (siehe Abbildung 11).

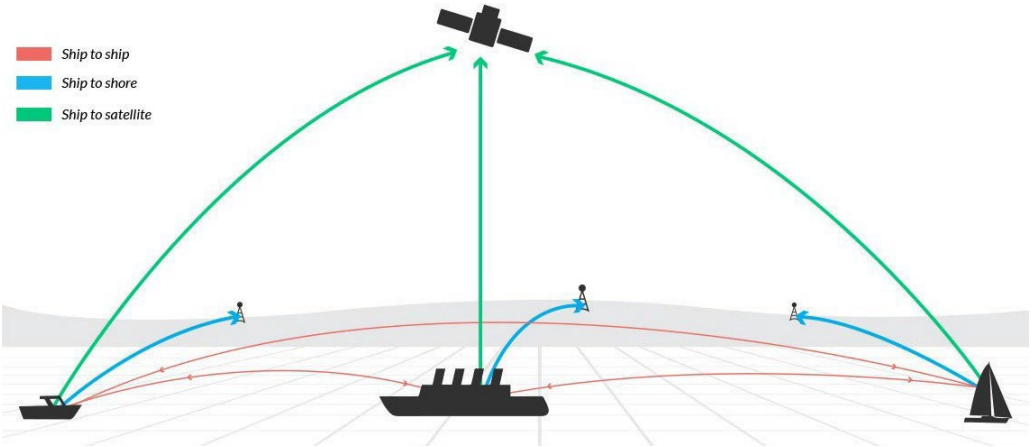


Abbildung 11 Übertragung der Positionsdaten zwischen Schiffen, Häfen und Satelliten (Marcom, 2022)

5 Theorieteil 2 – Yolov5-Algorithmus

Yolov5[15] steht für *You Only Look Once,* und ist ein Artificial Neural Network (ANN)[16] (ultralytics, 2022b). Ein ANN ist eine Methode im Feld *Machine Learning*, um Daten, ähnlich wie das menschliche Gehirn, zu verarbeiten (Anand, 2022). Das Prinzip unseres Gehirns Daten zu verarbeiten, wird mit künstlichen neuronalen Netzwerken und Verbindungen zwischen künstlichen Neuronen von einem Computer simuliert. Ein neuronales Netzwerk kann auf verschiedene Aufgaben trainiert werden. Das neuronale Netzwerk macht im Trainingsprozess Voraussagen. Es versucht zum Beispiel, die Position eines Bootes in einem Bild einzuzeichnen. Diese Voraussage wird dann mit der Lösung überprüft. Das Neuronale Netzwerk passt daraufhin die Verbindungen zwischen den Neuronen an. Genau wie in unserem Gehirn werden Verbindungen zwischen Neuronen stärker ausgeprägt, wenn sie häufig benutzt werden, bzw. zur richtigen Lösung eines Problems[17] führen. Zurückgebildet werden sie jedoch, wenn die Verbindung nicht benutzt wird, bzw. falsche Ergebnisse liefert[18]. Der beschriebene Trainingsprozess wird wiederholt, bis die Voraussagen des ANN möglichst genau sind.

Der obere Abschnitt gibt einen vereinfachten Überblick. In Wirklichkeit sind die Prozesse, die innerhalb eines neuronalen Netzwerksablaufen deutlich komplizierter. Diese zu erklären würde den Rahmen dieser Arbeit sprengen. Das Verständnis der genauen Abläufe ist nicht essenziell, um den Inhalt dieser Arbeit zu verstehen.

Yolov5[19] wird zur Objekterkennung und Objektlokalisierung in Bildern verwendet. Yolov5 zeichnet sich dadurch aus, dass er als einzelnes neuronales Netzwerk in der Lage ist zwei Aufgaben zu erledigen. Er erkennt die Klasse eines Objekts und wo sich das Objekt befindet. Deshalb wird er auch als *One-Stage-Detector* bezeichnet (Bandyopadhyay, 2022). *Two-Stage-Detektoren* haben zwei Teile. Ein Teil stellt fest, wo im Bild sich ein Objekt befindet, der andere Teil um was für ein Objekt es sich handelt. Durch die *One-Stage*-Architektur ist Yolov5 deutlich schneller als *Two-Stage-Detektoren*.

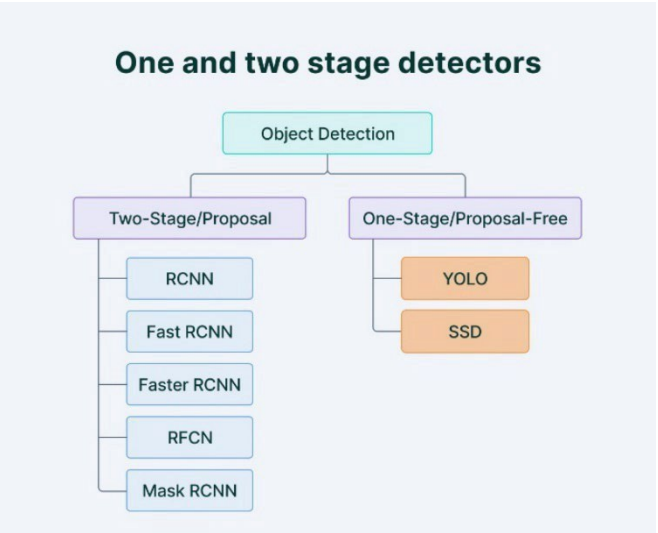


Abbildung 12 Auflistung von One-Stage- und Two-Stage-Detektoren (Bandyopadhyay, 2022)

[15] Yolo Version 5
[16] Yolov5 gehört zur Unterklasse der *Convolutional Neural Networks*, genaueres ist für dieses Projekt nicht relevant
[17] Auf ANN bezogen
[18] Auf ANN bezogen
[19] Yolo steht für *You Only Look Once*

Yolov5 ist durch seine kurze Inferenzzeit für die Anwendung in Echtzeit geeignet und deshalb eine gute Wahl für dieses Projekt. Der Yolov5-Algorithmus unterteilt ein Bild in viele kleine Unterabschnitte. In jedem dieser Abschnitte wird dann nach Objekten gesucht (Abbildung 13). Dadurch entstehen viele *Bounding Boxes*. Als *Bounding Box* wird die vorhergesagte Markierung um ein Objekt bezeichnet. Dadurch entstehen Regionen im Bild, die wie in Abbildung 13, mit hoher Wahrscheinlichkeit ein Objekt der Klasse Hund zuordnen. Die verschiedenen *Bounding Boxes* werden dann fusioniert. So erhält man die Klasse und Position eines Objekts im Bild.

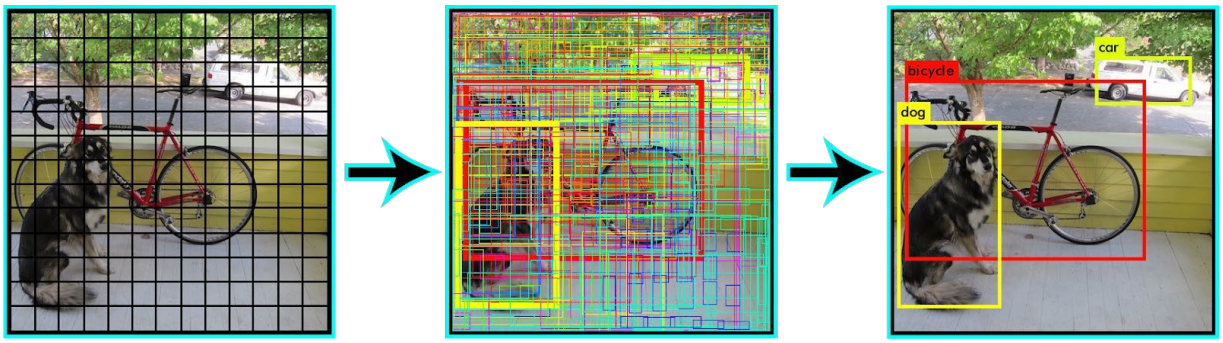


Abbildung 13 Unterteilung des Bildes wird dargestellt (links), die vielen Erkennungen werden fusioniert (rechts) (Bandyopadhyay, 2022)

6 Vorgehen und Produktentwicklung

Hier wird die Produktentwicklung vorgestellt. Dies umfasst die Programmierung und Logik des Programms. Des Weiteren wird ein Datenset zusammengestellt und ein Objekterkennungs-Algorithmus trainiert.

6.1 Vorwissen, Grundvoraussetzungen und Anmerkungen

Da im Folgenden die Programmstruktur genauer beschrieben wird, ist es daher von Vorteil, über ein Grundwissen von Python und anderen Programmierkonzepten zu verfügen, um die Beschreibungen und den Code genau nachvollziehen zu können. Eine Erklärung des Yolov5-Algorithmus und der verwendeten Stereokamera finden Sie in Kapitel 4 und 5. Es wird empfohlen den digitalen Anhang I während dem Lesen zu konsultieren (siehe Anhang I).

6.2 Programm Flow Chart und Logischer Aufbau

Anhand der vereinfachten Abbildung 15 am Ende dieses Kapitels, die den Datenfluss und die Verarbeitung der Software veranschaulicht, lässt sich das Grundgerüst der Software erklären. Wie in Abbildung 15 zu erkennen ist, werden Farbbilder, hier als *RGB Frame*[20] bezeichnet, des mittleren Sensors zur Verarbeitung an den Computer geschickt. Der mittlere Kamerasensor ist für das Erfassen farbiger Bilder verantwortlich und hat eine Auflösung von 12 Megapixel (MP). Die äusseren Sensoren sind monochrom und bilden zusammen die Stereokamera. Eine Monochrom-Kamera nimmt keine Farben, sondern nur die Intensität des Lichts wahr (Schwarzweissbild). Dadurch kann eine Monochrom-Kamera mehr Licht aufnehmen und hat eine höhere Sensorempfindlichkeit (Kumar, 2022). Mit Hilfe der beiden Bilder der Monochrom-Kameras wird ein Abstandswert für jeden Pixel berechnet (siehe Kapitel 4). Die Sammlung aller Abstandsdaten nennt man *Depth Map*.

Abbildung 14 OAK-D Stereokamera Spezifikationen (Martinez-Carranza, 2021)

[20] In *OpenCV* wird eine Bild Quelle auch als Frame bezeichnet, deshalb die Bezeichnung *RGB Frame*

Der *Depth Frame*[21] und *RGB Frame* werden an den Computer weitergeleitet (siehe Abbildung 15 nächste Seite). Der laufende Prozess auf dem Computer sucht daraufhin fortlaufend nach ihn erreichenden *RGB Frames* und *Depth Frames*. Der *RGB Frame* wird dem Yolo-Algorithmus als Input gegeben, als Output erhält man einen *RGB Frame*, in dem alle erkannten Objekte markiert sind. Die Position eines jeden Objekts wird nun skaliert und an die Auflösung des *Depth Frame* angepasst. Der nun markierte Bereich des Depth Frame, der aus Abstandsdaten für jeden Pixel besteht, wird zur Kalkulation der Distanz zum Objekt verwendet. Es wird der Durchschnittliche Abstandswert Aller Pixel innerhalb der Markierten Fläche berechnet. Dafür gibt es zwei vorgeschlagene Methoden (siehe Kapitel 6.4.4). Nun hat man den Abstandswert eines Objekts. Zu diesem eindimensionalen Abstandswert werden nun zwei weitere Abstandswerte auf zwei weiteren Achsen errechnet. Nun ist man in der Lage die Position des Objekts relativ zur Mitte des Kamerasensors anzuzeigen.

Die vom Kamerasensor relative Position wird auf einer Karte abgebildet. Dieser Prozess wird mehrfach pro Sekunde wiederholt, sodass als Ergebnis eine Karte mit Echtzeit Objektpositionen entsteht. Auf diese Karte wird nun ein *Pathfinding*-Algorithmus[22] gesetzt, dessen Aufgabe es ist, eine Route um die markierten Objekte zu finden. Weil die Berechnung einer Route mit einem *Pathfinding*-Algorithmus sehr ressourcenintensiv ist, wird ein zusätzlicher Python-Prozess[23] gestartet. Ein zusätzlicher Prozess erlaubt es, Code gleichzeitig auszuführen und verhindert, dass das Programm stark verlangsamt. Es wird also nicht zuerst der Yolo-Algorithmus und dann der *Pathfinding*-Algorithmus ausgeführt. Sondern der *Pathfinding*-Algorithmus wird ausgeführt, während der Yolo-Algorithmus bereits läuft. Das Programm aktualisiert nach einer Iteration des oben erklärten gesamten Programmablaufs das *User Interface*[24] (UI). Wenn zu diesem Zeitpunkt ein Tastendruck der Taste *Q* registriert wird, erfolgt der Abbruch des Programms. Es ist möglich weitere Tastenkombinationen einfach hinzuzufügen, um verschiedene Programmparameter anzupassen.

In den folgenden Kapiteln werden die oben zusammengefassten Prozesse genauer evaluiert.

[21] Die Begriffe *Depth Frame* und *Depth Map* werden in dieser Arbeit als Synonyme verwendet
[22] Ein Algorithmus, der nach Bestimmten Regeln eine Route um ein Hindernis herumfindet
[23] Ein Python-Prozess ist eine weitere Instanz des *Python Interpreters*, die Code ausführt.
[24] Benutzeroberfläche

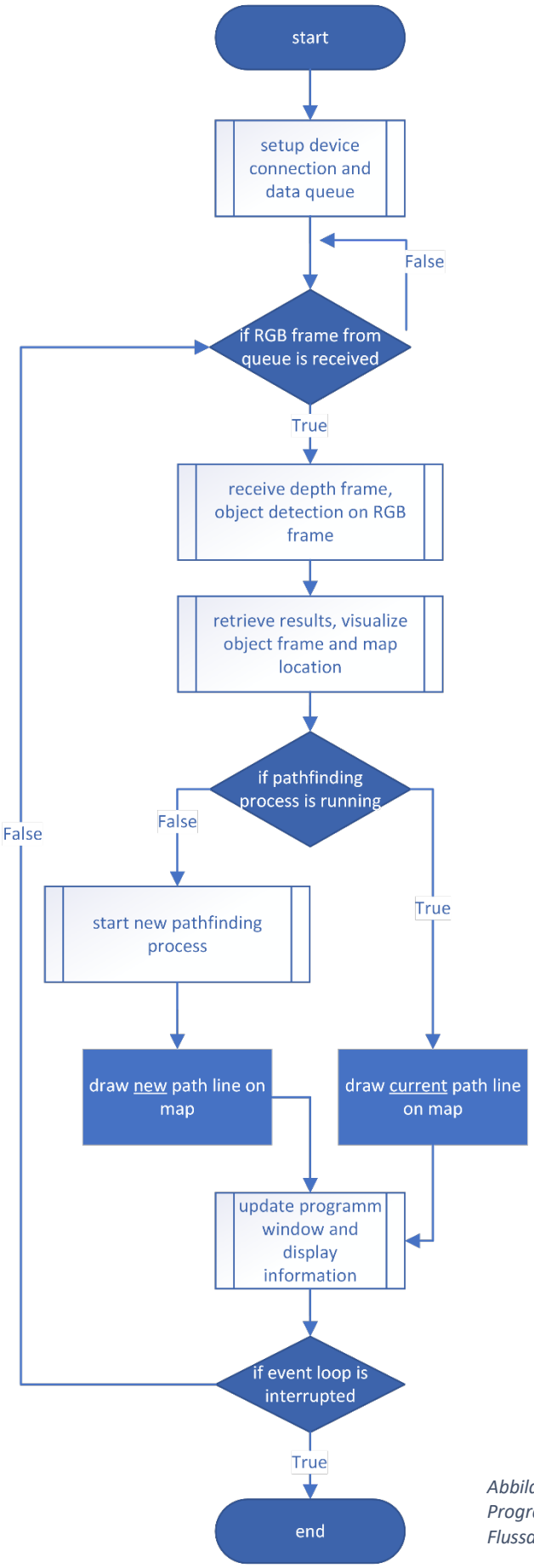


Abbildung 15 Vereinfachter Programmfluss in Form eines Flussdiagramms (eigene Darstellung)

6.3 Vorbereitung der Entwicklungsumgebung

Dieses Kapitel beschreibt die verwendeten Bibliotheken und Unterschiede zwischen CPU und GPU.

6.3.1 Verwendete Software

In diesem Projekt wird ausschliesslich die Programmiersprache Python verwendet. Python ist auf dem Gebiet *Machine Learning* sehr bewährt und hat viele Bibliotheken, die Programmentwicklung unterstützen. *OpenCV* ist eine *Computer Vision*[25] Bibliothek, die die Handhabung von Bilddaten erleichtert. Jedoch ist die Geschwindigkeit mit der Python Code ausführt im Vergleich zu anderen Programmiersprachen eher langsam. Für eine genauere Erläuterung der Struktur von Python siehe Anhang II. *OpenCV* ermöglich die Bearbeitung und Darstellung von Bildern und liefert das Fundament der Software. Der Yolov5-Algorithmus wird mit Hilfe von *PyTorc*h trainiert und ausgeführt. Die Bibliothek des Stereokameraherstellers *Luxonis* genannt *Depthai,* Ermöglich die Kommunikation mit der Stereokamera. Des Weiteren wird *Numpy* für schnelles Rechnen verwendet.

OpenCV als Bibliothek bietet einige Limitationen, es ist daher nur schwer möglich ein *User Interface* mit *OpenCV* zuerstellen. Es ist aber auch nicht das Ziel dieser Arbeit eine Software mit einem umwerfenden UI zu programmieren. Es wird sich mit einer rudimentären Darstellung von Informationen zufriedengegeben.

6.3.2 CuDNN und OpenCV mit GPU-Support

CuDNN sorgt dafür, dass mit Hilfe von der Compute Unified Device Architecture (CUDA) die CUDA-Recheneinheiten einer Grafikkarte (GPU) verwendet werden können. CUDA-Kerne sind für dafür optimiert eine grosse Anzahl an Kalkulationen derselben Art mit hoher Geschwindigkeit durchzuführen. Genau dies wird beim Training und ausführen von *Artificial Neural Networks* (ANN) benötigt. Eine GPU mit CUDA-Kernen ist also in der Lage schneller Voraussagen mit einem ANN zu machen als eine Central Processing Unit (CPU), weil eine CPU nicht für eine solche Fülle an einfachen Kalkulationen der gleichen Art ausgelegt ist.

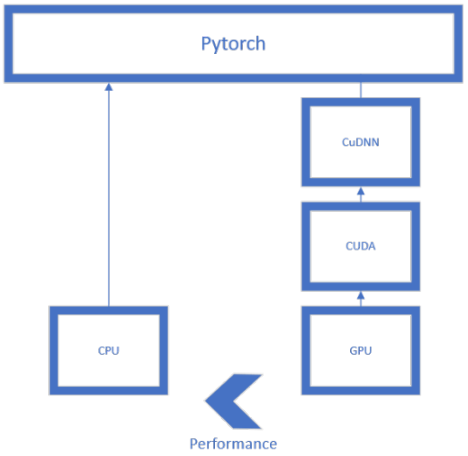


Abbildung 16 Performance Unterschied zwischen CPU und GPU (eigene Darstellung)

[25]Computerbasiertes Sehen

Der grundsätzliche Unterschied zwischen GPU und CPU wird so beschrieben:

> „Die Architektur einer CPU ist auf generelle Programmanforderungen ausgelegt und muss eine Vielzahl Befehle und Datentypen unterstützen. Neben einer geringen Anzahl von Kernen (Cores) nehmen Cache und Controller größere Teile der Architektur der CPU in Anspruch. Eine GPU hingegen ist auf Grafikberechnungen spezialisiert und arbeitet mit relativ wenigen Befehlen und vielen Cores. Dies ist darauf zurückzuführen, dass GPUs hauptsächlich für Pixelberechnungen konzipiert wurden. Die Architektur erlaubt es, viele einfache Befehle gleichzeitig an die Cores zu übertragen und parallel abzuarbeiten. Cache und Controller sind wesentlich einfacher aufgebaut und kleiner.“ (Luber, 2022)

6.4 Implementierung und Programmstruktur

Die Implementierung und die verwendete Programmstruktur werden in diesem Kapitel genauer beschrieben.

6.4.1 Programmstruktur

Um das angestrebte Programm zu entwickeln, wird auf eine Objekt-Orientierte-Struktur zurückgegriffen. Dies ist von grossem Nutzen, da das Programm in konstanter Kommunikation mit anderen Hardware- und Software Komponenten steht. Python als Programmiersprache bietet Support für viele verschiedene Programmier-Paradigmen unter anderem Funktionelle Programmierung und Objekt Orientierte Programmierung (OOP). OOP bietet sich deshalb an, da erfahrungsgemäss das Entwerfen eines ganzen Softwaresystems mit Funktionaler Programmierung nicht umsetzbar ist.

Wenn man den Code (Anhang I) genauer unter die Lupe nimmt, fällt auf, dass das Programm aus einer Hauptklasse besteht, von der zu Beginn der Ausführung eine Instanz[26] erstellt wird. Des Weiteren werden verschiedenste Bibliotheken importiert. Aus dieser Ansammlung an Klassen wird wiederum eine weitere Klasse importiert. Eine Klasse enthält verschiedene Funktionen, diese Funktionen werden, nebst selbst Programmierten Funktionen, im eigenen Code angewendet. Eine Funktion ist eine Sammlung von Anweisungen oder Verfahren zur Durchführung einer bestimmten Aufgabe, eine Methode ist im Gegensatz dazu ein Satz von Anweisungen, die mit einem Objekt verknüpft sind, also eine klassenspezifische Funktion (OOP). Weitere Informationen befinden sich in Anhang II.

6.4.2 OAK-D Stereokamera Pipeline

Die Abbildung 17 stellt den Aufbau der verwendeten Pipeline dar, diese Pipeline besteht aus einzelnen Nodes die wie folgt verknüpft werden.

Das Bild der Farbkamera (*ColorCamera*), die am mittleren Sensor lokalisiert ist, wird mit einer *Image-Manipulation-Node* (*ImageManipRGB*) auf die richtige Fenstergrösse skaliert und angepasst. Als Format wird *ISP verwendet.* Der *ISP*-Output zeichnet sich dadurch aus, dass im Vergleich zum RAW[27]-*Bild, Post-Processing*[28] stattfindet. Das Ergebnis ist ein schärferes und rauschfreieres Bild. Der verwendete *Denoiser*[29] wird für jede spezifische Anwendung selbst angepasst. Es findet auch kein *Cropping*[30] des Videos statt. Das *Video*-Format leitet sich vom *ISP*-Format ab und wird von 12 MP auf mindestens 4K[31] zugeschnitten. Es ist die Aufgabe der *XLinkOut*-Node (XLinkOut) den empfangenen Datenfluss an den *Host*[32] zu übertragen, dieser nimmt diesen dann entgegen. Wie in Abbildung 17 zu erkennen ist empfängt der Computer also vier verschiedene *Image Frames:* Das Farbbild (*ColorCamera*), das skalierte Farbbild (*ImageManipRGB*), die skalierten Abstandsdaten (*ImageManipDepth*) und das Monochrombild der Linken Monochrom-Kamera (*ImageManipLeft*).

[26] Wenn eine Klasse instanziiert wird, wird ein Objekte mit den in Ihr definierten Eigenschaften erstellt.
[27] RAW ist ein Bildformat, bei dem Bilder nicht komprimiert oder nachbearbeitet werden (Rohdaten)
[28] Damit ist eine Nachbearbeitung des Bildes gemeint
[29] Rauschunterdrückungsssoftware
[30] Zuschneiden
[31] Auflösung von 3840 × 2160 Pixel
[32] In diesem Fall der verbundene Computer

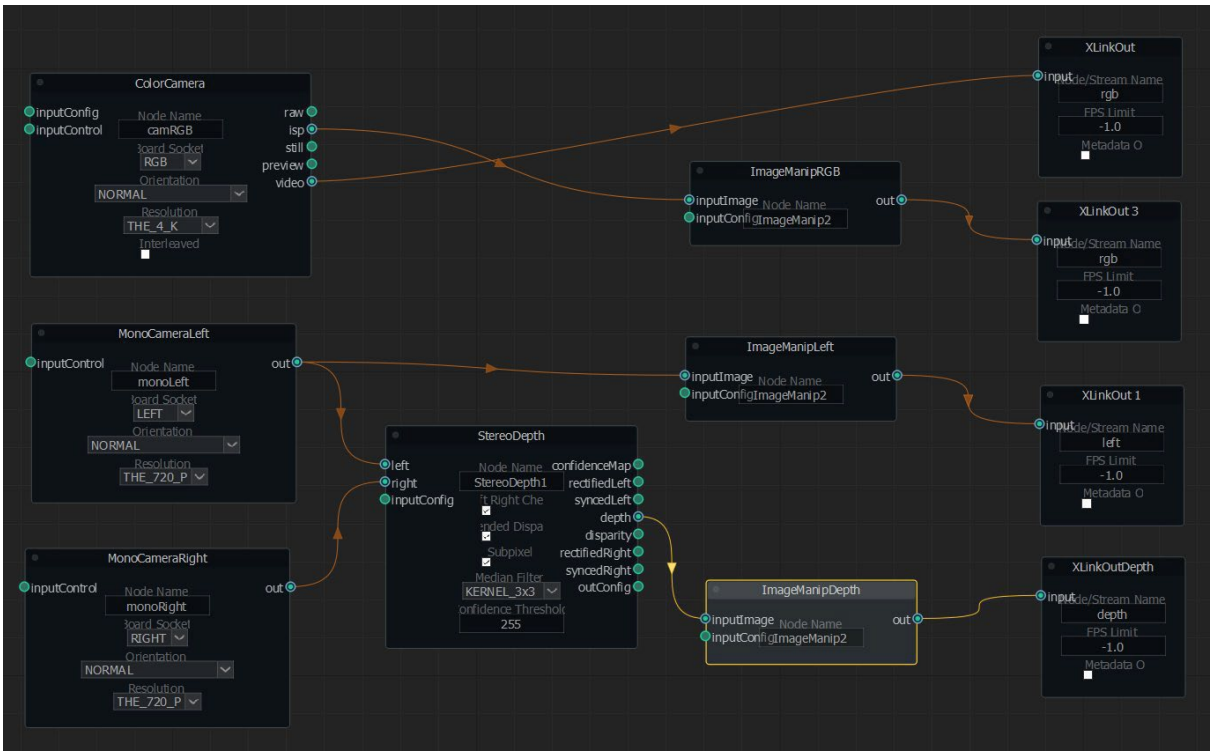


Abbildung 17 Darstellung der Node-Struktur der OAK-D Pipeline (eigene Darstellung)

6.4.3 Deployment von Yolov5-Algorithmus und Programm Loop

Das Programm erhält von der multisensorischen Kamera (OAK-D von Luxonis) einen RGB-, Depth- und Mono-Frame. Als Input für den Yolov5-Algorithmus dient das erhaltene Farbbild des mittleren Farbsensors. Dieses Bild wird auf die passende Inputauflösung skaliert, diese variiert je nach selbsttrainierter Modellvariante (siehe Kapitel 6.5), eine häufig verwendete Auflösung ist 416 x 416 Pixel.

Das selbsttrainierte Modell wird mit Hilfe der *Load*-Methode der Klasse *Hub* aus der Bibliothek *PyTorch* ausgeführt (ultralytics, 2022b). Es besteht jedoch auch die Möglichkeit bereits vortrainierte (*Pretrained*) Modelle zu laden und zu verwenden. Das entwickelte Programm ist mit jeglichen Yolov5-Modellen kompatibel.

Als Output gibt Yolov5 einen mehrdimensionalen Array, aus dem darauffolgend, die wichtigsten Daten entnommen werden. Die wichtigsten Output-Werte sind: Die *Bounding Box*, die *Confidence* und die Klasse des Objekts. Die *Bounding Box* legt fest, wo sich ein oder mehrere Objekte auf dem Bild befinden. Die Klasse beschreibt welcher Klasse ein Objekt angehört[33] und die *Confidence* beschreibt, wie sicher sich Yolov5 der vorausgesagten Information ist. Die Output-Werte werden mit einem *for loop* [34] entpackt. Wenn die *Confidence* der *Prediction* (*Voraussage*) des Modells grösser ist als eine gesetzte Untergrenze wird die *Bounding Box* auf dem Bildschirm angezeigt. Erfahrungsgemäss ist ein Wert von ungefähr 70% mehr als ausreichend. Der oben beschriebene Ablauf wird mit einem *while-loop* wiederholt, bis das Programm durch einen spezifizierten Tastendruck unterbrochen wird.

[33] Zum Beispiel die Klasse Boot

[34] Eine Methode, mit der bestimmte Befehle wiederholt werden können

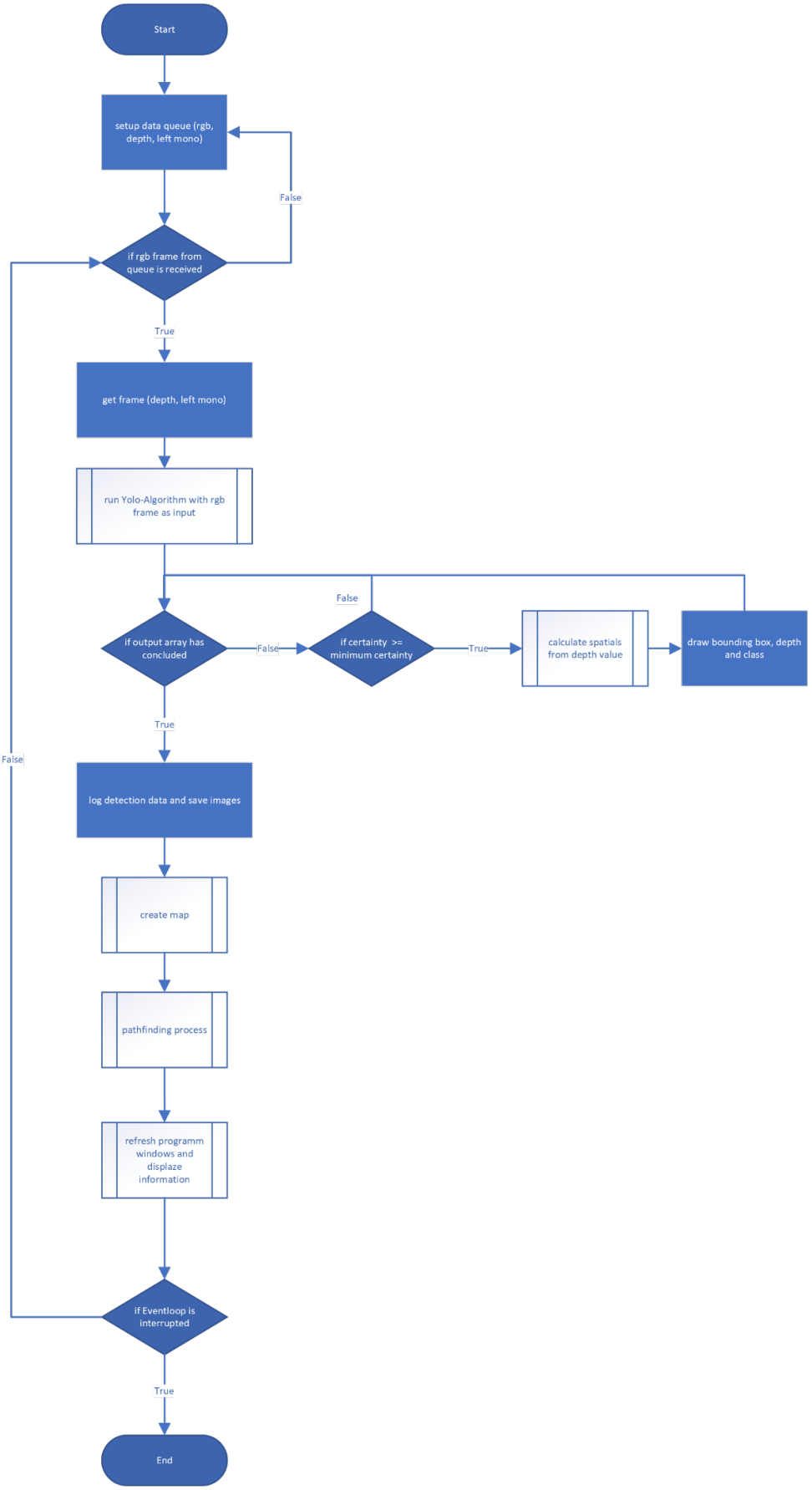


Abbildung 18 Programmloop und Yolov5 (eigene Darstellung)

6.4.4 Berechnung weiterer Abstands-Koordinaten

Das Zentrum der vom Modell erhaltenen, *Bounding Box* wird berechnet. Von dort kann die *Region of Interest* (ROI) gesetzt werden. Die ROI legt fest, wie gross die Fläche im Zentrum sein soll von der, der durchschnittliche Abstand zwischen Objekt und Kamera berechnet wird.

Hier hat man sich für die *Zentrumsmessung* entschieden, da das Zentrum der *Bounding Box* eines Objekts, in diesem Fall von Motor- und Segelbooten, meist vom Objekt eingenommen wird. Um jedoch grosse Abweichungen zu vermeiden, wird auch der Durchschnittswert des Abstandes der gesamten *Bounding Box* berechnet. Diese beiden Resultate werden dann verglichen, und der geringere Abstand wird als Hauptwert angezeigt. Der genaue Einfluss dieser Methode auf die akkurate Abstandsbestimmung wird in der Versuchsreihe festgestellt. Eine alternative Methode wäre Bildsegmentierung zu betreiben. Dabei würde die bereits erkannte *Bounding Box* in Vorder- und Hintergrund segmentiert. Anhand dieser Segmentierung könnte dann der Tiefenwert aller Pixel in der exakten Form des Objekts zur Berechnung des durchschnittlichen Abstands verwendet werden. Diese Methode wird jedoch nicht verwendet, da sie die Implementierung eines weiteren Algorithmus bedingen würde, was noch mehr Ressourcen verbrauchen würde. Auch würde diese Lösung die Komplexität deutlich erhöhen, um jedoch trotzdem ein ähnliches Ergebnis zu liefern. Daher macht der erhöhte Aufwand, dies zu einer eher unattraktiveren Lösung. Es muss jedoch festgehalten werden, dass die momentane Lösung über bestimmte Schwachstellen verfügt und daher tendenziell fehleranfälliger ist (siehe Kapitel 9.1).

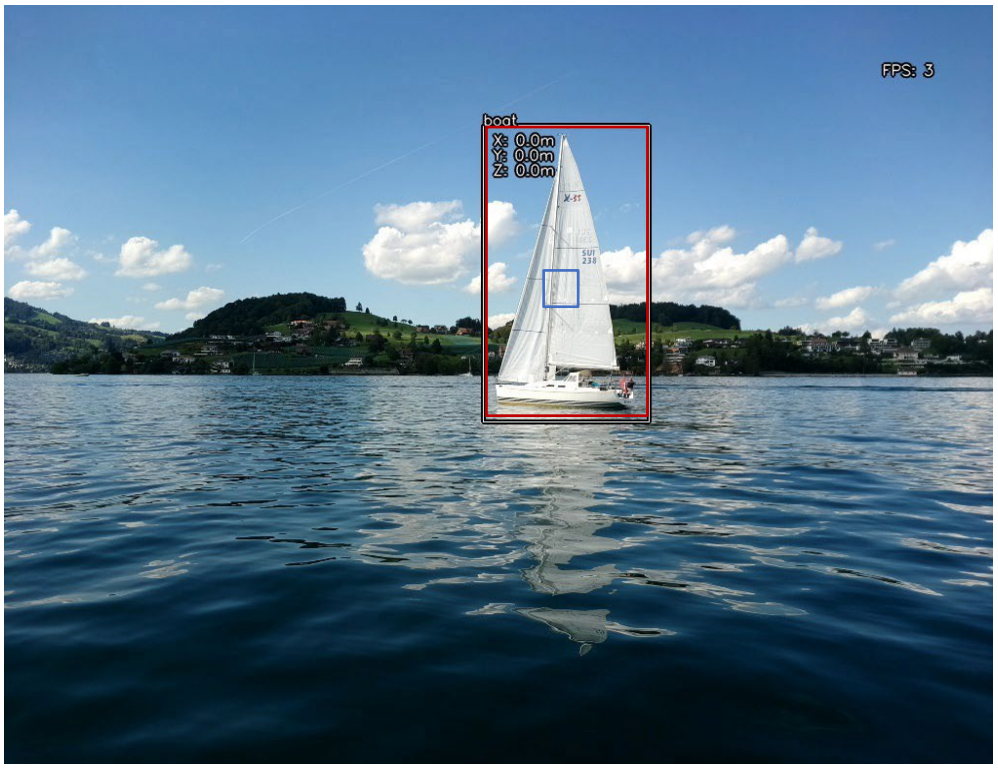


Abbildung 19 Zentrumsmethode (blau), Gesamtmethode (rot) (eigene Aufnahme)

Nachdem man den durchschnittlichen Abstand auf der Z-Achse erhält, werden mit dessen Hilfe die Abstandswerte der X- und Y-Achse relativ vom Mittelpunkt des Farbkamera Sensors berechnet. Die Z-Achse ist hier die Achse, die waagrecht von der Kamera weg zeigt. Die OAK-D Stereokamera besitzt die Fähigkeit, die weiteren Achsen auf der Kamera internen Hardware zu berechnen. Um aber für bessere Performance zu sorgen werden die X- und Y-Achse auf dem *Host* berechnet (Kapitel 4.1). Dafür wird die Klasse *HostSpatialsCalc* vom *Depthai* Github-Repository modifiziert und implementiert (luxonis, 2022b). Das Ergebnis der Berechnung wird auf einer Karte, relativ von der Kamera, dargestellt.

6.4.5 Verwendung eines Pathfinding-Algorithmus auf einer Karte

OpenCVs Funktionen zum Zeichnen von geometrischen Formen werden verwendet, um eine Karte in einem Fenster mit dem *Field of view* (FOV), das der Kamera entspricht zu erstellen. Die ausgerechneten Positionen der Objekte werden nun auf der Karte eingezeichnet. Diese Karte hat standardmässig einen Auflösung 1000 x 1000 Pixel. Um jedes Objekt befindet sich ein Quadrat, das Objekt liegt im Zentrum dieses Quadrats. Dieser Abstand ist als Puffer gedacht, dieser Puffer verhindert, dass der Pfad des *Pathfinding*-Algorithmus die Position des Objekts schneidet und es so zu einer Kollision kommt. Dieser Abstand kann beliebig gewählt werden, jedoch scheint ein Wert von mindestens 10m vom Zentrum in der Praxis angebracht.

Zur Implementierung des *Pathfinding*-Algorithmus wurde die Bibliothek *Pathfinding* verwendet (Andreas, 2022). Es wurde der A*-Algorithmus gewählt, da er im Vergleich zu anderen verfügbaren Algorithmen effizienter arbeitet und verlässliche Routen liefert. Der A* kombiniert die Funktionalität des Best-First-Search-Algorithmus und Dijkstras Algorithmus, sodass er für die meisten Anwendungen die beste Wahl ist (Red Blob Games, 2022). A* agiert auf einem zweidimensionalen Array mit einer selbstgewählten Auflösung von 50 x 50 Flächen, auf diese wird die Karte (1000 x 1000 Pixel) skaliert. Dieser Schritt ist wichtig, weil eine Auflösung von 1000 x 1000 Flächen viel zu hoch für den *Pathfinding*-Algorithmus wäre. Folgen einer solchen Auflösung sind eine schlechte Performance und eine überdetaillierte Strecke. Eine Fläche von 50 x 50 ergibt eine Wiederholungsrate von ca. zwei bis drei Mal pro Sekunde, die Strecke passt sich also zwei bis drei Mal pro Sekunde an neue Objekte an.

6.4.6 Implementierung von Multiprocessing

Das Python Multiprocessing Modul ist von grosser Bedeutung, da es für eine grosse Performancesteigerung verantwortlich ist. Multiprocessing erstellt einen weiteren Python Prozess, dies sorgt dafür, dass zwei Python Prozesse gleichzeitig laufen und Code ausführen. So ist es möglich, dass mehrere Aufgaben gleichzeitig gelöst werden.

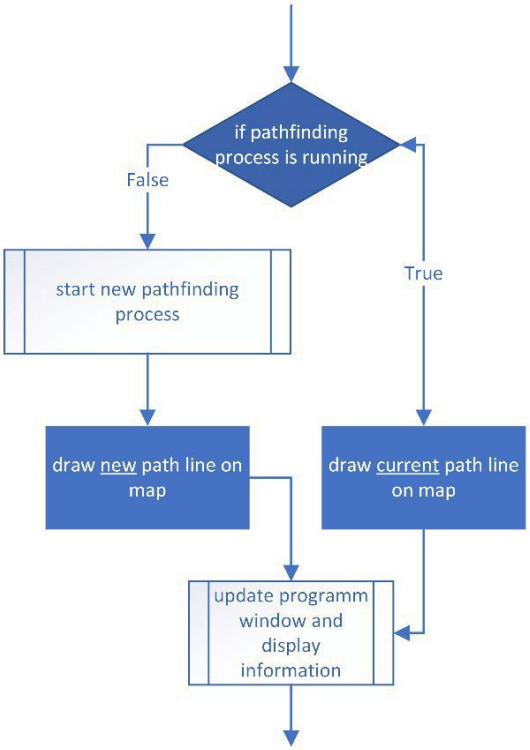


Abbildung 20 Darstellung des zweiten Python-Prozesses (eigene Darstellung)

Während jeder Iteration des Programm-Loops wird geprüft, ob ein Pathfinding-Prozess läuft. Falls dies nicht der Fall ist, startet Python einen weiteren Prozess und informiert diesen über die aktuelle Position der Objekte. Der Prozess verwendet A* und gibt eine mögliche Umfahrungsroute der Hindernisse als Output. Die veraltete Route wird dann entfernt und die neue Route wird auf die Grösse der Karte skaliert und auf diese gezeichnet. Wenn jedoch bereits ein *Pathfinding*-Prozess ausgeführt wird, wird die vorherige Route noch einmal auf die Karte gezeichnet und Python erstellt keinen neuen Prozess.

Dieses Vorgehen sorgt für eine Verzögerung von ca. einer halben Sekunde in der Anpassung an neue Objekte. Multiprocessing macht jedoch die gleichzeitige Darstellung von Objekten mit einer hohen Bildwiederholungsrate und die Darstellung einer Route mit einer geringeren Bildwiederholungsrate möglich. Sonst wäre nämlich der Pathfinding-Algorithmus der limitierende Faktor und das Programm müsste mit der nächsten Iteration[35] warten, bis der Prozess abgeschlossen ist. Das Ergebnis wäre ein Programm mit einer insgesamten Bildwiederholungsrate, die der des Pathfinding-Algorithmus entspricht (gering).

6.4.7 Speicherung der generierten Daten – Logging Modul

Zur Speicherung der generierten Daten wird das *Logging* Modul verwendet. Dieses speichert wichtige daten in einer .log Datei. Wichtige Daten sind unter anderem folgende:

- Exakte Uhrzeit und Datum
- Wichtigkeit des Logs
- Klasse des erkannten Objekts
- Abstände (x,y,z)
- ROI
- Name des gespeicherten Bildes
- *Confidence*-Wert

[35] In diesem Fall der einmalige Ablauf des Erkennungs- und Messprozesses für ein Bild

6.5 Training des eigenen Yolov5-Algorithmus

Die Erstellung eines Datensets und das darauffolgende Training ist in verschiedene Schritte unterteilt, auf die nun eingegangen wird.

6.5.1 Datenbeschaffung

Im Sommer 2022 wurden 689 Bilder von Wasserfahrzeugen auf dem Vierwaldstättersee gesammelt. Als Wasserfahrzeuge werden Motor- und Segelboote (mit gehisstem Segel) definiert. Es wurden unterschiedliche Bilder gemacht, bei denen folgende Faktoren variieren:

- Anzahl Objekte
- Grösse der Objekte
- Position der Objekte (links, rechts, oben, unten)
- Winkel der Segelboote zur Horizontalen Wasserlinien (leichte Rotation)

Weitere Faktoren die zu beachten sind:

- Segelboote im Datenset haben ausschliesslich gehisste Segel
- Orientierung des Bildes (normal)
- Die Aufnahmen wurden tagsüber gemacht, es war sonnig
- Manche Bilder weisen einen Blaustich auf
- Der Hintergrund bestand häufig aus bewaldeten Flächen und mit Häuseransammlungen
- Menschen sind im Datenset zu finden, sind jedoch nicht identifizierbar

Die Fähren des Vierwaldstättersees wurden nicht als Boot markiert, da sie nicht dem durchschnittlichen Boot des Vierwaldstättersees entsprechen, was jedoch nicht bedeutet, dass der Algorithmus sie nicht erkennen kann.

Abbildung 21 Objekterkennung mit selbsttrainiertem yolov5l auf fremder Fotographie (imatrix, 2018)

6.5.2 Datenlabeling

Die ausgewählten Bilder wurden auf die Online Plattform *Roboflow* hochgeladen und dort gelabelt (*Roboflow*, 2022). Unter Datenlabeling versteht man, dass auf den Bildern die zu erkennenden Objekte markiert werden und deren Klasse vermerkt wird. Es wurde darauf geachtet, dass die umschliessende Box so eng wie möglich um das Objekt liegt, jedoch ohne das Objekt zu berühren. Kaum zu erkennende Objekte in weiter Entfernung, die sich kaum von Hintergrund abheben, mit einer Grösse von wenigen Pixel wurden nicht markiert. Dies wurde getan, um den Labeling-Prozess effizienter zu gestalten und ein klar fokussiertes Datenset herzustellen. Das bedeutet, dass ein Boot klar, als solches erkennbar ist und nicht die Aufmerksamkeit des Modells ausgedünnt wird. Beispielsweise würde ein kleines Boot, welches mit einem Haus im Hintergrund verschwimmt, die Daten verfälschen. Ausserdem würden Häuser tendenziell häufiger als Boote erkannt werden.

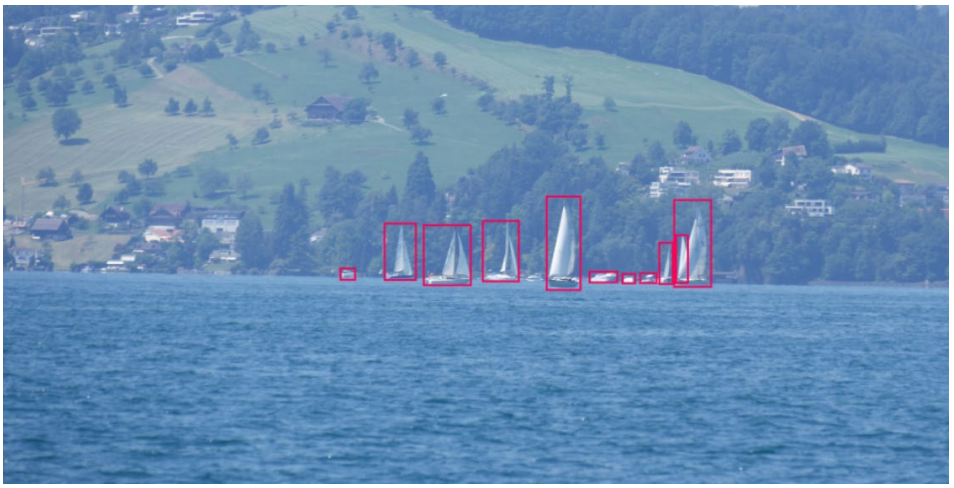


Abbildung 22 Boote mit einer roten Box markiert und der Klasse Boot zugeordnet (eigene Aufnahme)

Roboflow besitzt die Fähigkeit ein Datenset zu vergrössern, indem es Bilder aus dem Datenset wiederverwendet und aus ihnen neue generiert. Mit *Roboflow* wurden aus 689 Bildern 1653 Bilder mit den Veränderungen in Abbildung 23 generiert.

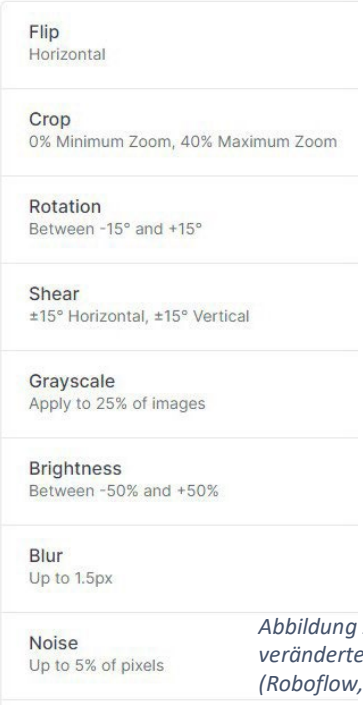


Abbildung 23 Faktoren mit denen veränderte Bilder generiert werden (Roboflow, 2022)

Die generierten Bilder werden schliesslich wie folgt aufgeteilt:

- 87% Trainingsbilder (1446)
- 8% Validierungsbilder (137)
- 4% Testbilder (70)

Trainingsbilder werden verwendet um das Machine-Learning-Modell (Yolov5) zu trainieren. Die Validierungsbilder finden Anwendung bei der Optimierung der Hyperparameter. Unter Hyperparametern versteht man Parameter, die beeinflussen, wie und wann das Modell im Trainingsprozess lernt. Sie beeinflussen nicht die finale Performance des Modells, sondern wie man zu diesem Ergebnis kommt. Ein Beispiel für einen Hyperparameter ist zum Beispiel die Lernrate, sie legt fest, in welchem Ausmass alte Information mit neuer Information überschrieben wird. Das Validierungsdatenset und Testdatenset beinhalten beide Daten, die dem trainierten Modell unbekannt sind. Diese Daten werden als *Unseen Data* bezeichnet. Deshalb liefern diese Datensets unvoreingenommene Resultate. Davon ausgehend, dass das Testdatenset selbst *unbiased*, also unvoreingenommen ist, kann trotzdem eine ungewollte Ungleichverteilung der Bilder das Endergebnis verzerren. Testdatensets[36] werden häufig zum Vergleichen verschiedener Modelle angewendet.

6.5.3 Training

Zum Trainieren des Datensets wird das Trainingsskript vom Yolov5 Github-Repository verwendet (ultralytics, 2022c). Um verschiedene Versionen des Modells zu vergleichen, werden beim Training die Auflösung des Datensets und die Batchsize variiert. Eine höhere Auflösung sorgt dafür, dass kleinere Objekte mit einem Durchmesser von wenigen Pixeln genauer erkannt werden können. Die erhöhte Auflösung verbraucht aber beim Training mehr Ressourcen. Die Yolov5l-Modelle wurden mit einer Nvidia A100 GPU mit 40GB VRAM[37] trainiert. Getestet und validiert wurden die Modelle mit einer GTX 1080 GPU. Da der grosse VRAM der A100 GPU nicht ausreicht, um das Yolov5l6-Modell mit einer Datenset Auflösung von 1280x1280[38] Pixeln zu trainieren, muss die Auflösung auf 896 x 896 reduziert werden. Alle anderen Modelle mit einer höheren Auflösung wurde mit einer Auflösung von 928x928 Pixeln trainiert. Für manche Modelle gibt es eine geringe Abweichung von diesen Zahlen, weil das Trainingsprogramm eine solche Auflösung nicht bei allen Algorithmen zulässt. In einem solchen Fall wurde die nächstmögliche Auflösung gewählt. Weitere Modelle werden neben der höheren Auflösung auch mit einer geringeren Auflösung von 416x416 Pixeln trainiert. Ein Modell mit einer geringeren Auflösung benötigt weniger Rechenleistung, um Voraussagen zu machen und bietet daher bessere zeitliche Performance.

Die Batchsize ist ein Hyperparameter, der die Anzahl Samples, die aus dem Datenset genommen werden, definiert. Für jedes Sample macht das Modell eine *Prediction*, diese Voraussage wird dann mit dem *Label*, dem gewünschten Output, verglichen. Diese Form des Lernprozesses wird als *Supervised Learning* bezeichnet. Durch Batches von Samples arbeitet sich der Algorithmus nun durch. Wenn dieser Prozess abgeschlossen ist, wird die Präzision geprüft und die modelinternen Parameter werden angepasst. Wie häufig dieser Prozess abläuft, legt die Anzahl Epochen fest (Brownlee, 2022). Alle Modelle werden für 200 Epochen trainiert. Diese Zahl ist adäquat für die Grösse des Datensets und spart Trainingszeit. Der am besten performende Zwischenstand während der 200 Epochen wird abgespeichert.

[36] Bzw. Validierungsdatensets
[37] Video-RAM
[38] Wie das Yolov5l-Model der offiziellen Quelle

Es gibt vier verschiedene Yolov5-Modell-Typen (L, M, S, N). Die Architektur der Modelle ist ungefähr gleich, wobei aber die Komplexität der Modelle variiert. Yolov5l hat die grösste Komplexität und dadurch die höchste Genauigkeit, aber auch die längste Inferenzzeit (Inference time). Yolov5n ist am wenigsten komplex und hat die geringste Genauigkeit aber jedoch die schnellste Inferenzzeit (ultralytics, 2022a). Die Inferenzzeit ist die Zeit, die vom Modell beansprucht wird, um eine Voraussage zumachen. Zu jeder Modellversion gibt auch noch eine Variante, die für grössere Auflösung von 1280x1280 optimiert ist. Diese Modellversion wird durch eine *6* am Ende des Namens gekennzeichnet. Modelle ohne diese Kennzeichnung sind für eine Auflösung von 640x640 optimiert.

Format: Yolov5 Version, Auflösung des Datensets, Version des Datensets, Anzahl Epochen, Batchsize

- Yolov5l, 416x416, Version 8, 200 Epochen, Batchsize 32
- Yolov5l, 416x416, Version 8, 200 Epochen, Batchsize 128
- Yolov5l, 928x928, Version 15, 200 Epochen, Batchsize 32
- Yolov5l6 896x896, Version 19, 200 Epochen, Batchsize 32
- Yolov5s, 416x416, Version 8, 200 Epochen, Batchsize 32
- Yolov5s, 416x416, Version 8, 200 Epochen, Batchsize 128
- Yolov5n, 416x416, Version 8, 200 Epochen, Batchsize 32
- Yolov5n, 416x416, Version 8, 200 Epochen, Batchsize 128

7 Versuchsaufbau der Distanzmessungen

Im ersten Versuch werden die Distanzmessungen von verschiedenen Objekten auf eine Distanz von einem bis zu 15m getestet. Im zweiten Versuch wird die Präzision der Abstandmessungen der Stereokamera auf eine Distanz von bis zu 25m gemessen.

7.1 Abstandgenauigkeit von verschiedenen Objekten auf 1-10m – Versuch 1

Es wird an vier Objekten der Messwerte der Stereokamera (OAK-D) mit dem realen Abstand zwischen Kamera und Objekt verglichen und so die Genauigkeit festgestellt.

- Modellsegelboot (62x13x94) 20 cm über dem Boden
- Modellmotorboot (57x19x16) 6 cm über dem Boden
- Kartonbox (40x20x10) direkt auf dem Boden

Abbildung 24 Das Modelmotorboot (links) und das Modellsegelboot (rechts) (eigene Aufnahme)

Der Versuch wird in einem abgeschlossenen Raum vor einer Fensterscheibe durchgeführt. Ein Massband wird rechtwinklig zur Scheibe angeordnet. Die Fensterscheibe ist durchlässig für die Distanzberechnung der Stereokamera, und hat daher nur einen geringen Einfluss auf die Ergebnisse. Die Kamera wird 12m entfernt platziert und befindet sich 20cm über dem Boden. Die verschiedenen Objekte werden in Abständen von einem bis 10m auf dem Massband platziert. Der Objektmittelpunkt befindet sich auf der entsprechenden Metermarkierung. Zur Berechnung der Abstände wird eine programmierte Software, die auf teilweise auf Code von *Luxonis* basiert, verwendet (Luxonis, 2022a). Das Objekt wird durch den Tester mit Hilfe der Software anvisiert, sodass die *Bounding Box* das Objekt umschliesst. Daraufhin werden der durchschnittliche Abstandwert der gesamten *Bounding Box* und der einer weiteren Zone im Zentrum der Box berechnet. Diese Zone wird als Quadrat im Mittelpunkt der *Bounding Box* mit einem Durchmesser 10 Pixeln definiert. Beide Durchschnittswerte werden dem Nutzer angezeigt. Wenn die Grösse der *Bounding Box* angepasst wird, bleibt die Grösse der Box im Zentrum jedoch gleich.

Die erhaltenen Abstandwerte werden so lange *aufgerundet,* bis keine Veränderung des Abstands durch das Rauschen der Ergebnisse zu erkennen ist. Der Versuch wird bei guter Beleuchtung und Tageslicht durchgeführt.

7.2 Abstandsberechnungen auf 8-25m – Versuch 2

In diesem Versuch wird die Variationsbreite der berechneten Abstandswerte zu einer Fläche in einem Bereich von 8 bis 25m festgestellt. Eine bräunliche Sperrholzplatte (6m x 2.5m) wird auf dem Asphalt platziert und aufrecht stabilisiert. Ein Massband wird rechtwinklig zur Sperrholzplatte platziert und die Kamera wird in ein Meter-Abständen (8m-25m) auf das Massband gestellt. Die Kamera befindet sich 50cm über dem Boden. Es wird die gleiche Software aus Kapitel 7.1 verwendet. Das Minimum und Maximum des durch Rauschen variierenden gesamten- und zentralen Durchschnitts werden festgehalten. Aus ihnen wird der Durchschnitt berechnet, das Minimum und Maximum geben den Variationsbereich und dienen als Fehlerindikatoren in der Ergebnisgrafik.

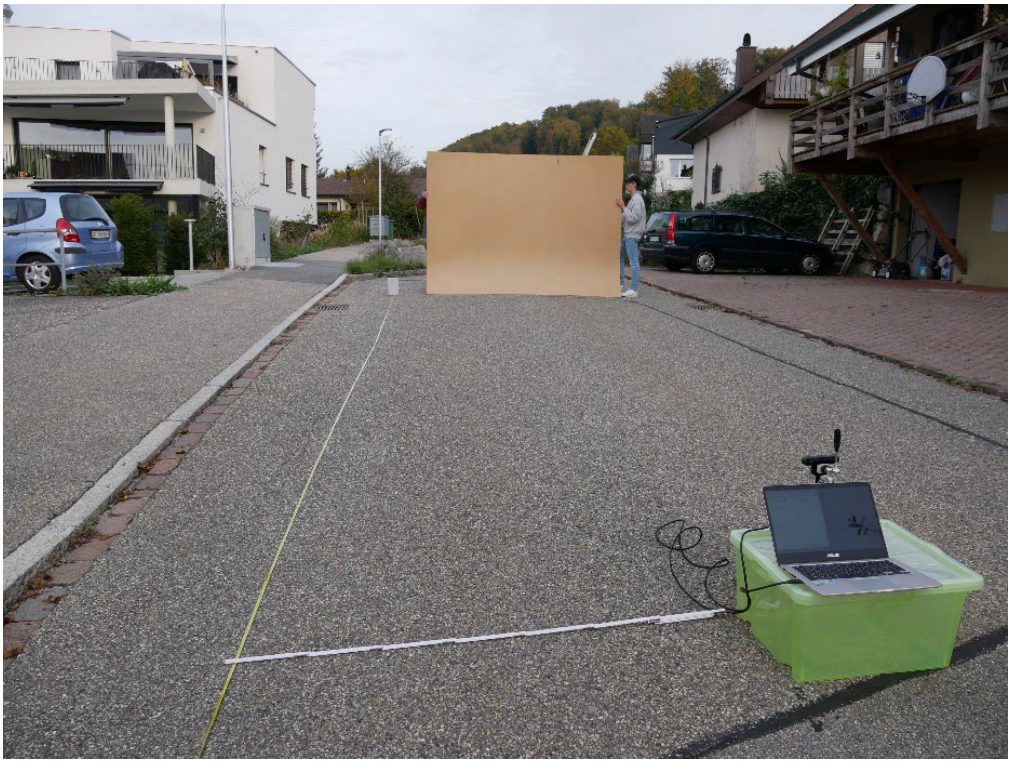


Abbildung 25 Versuchsaufbau für Versuch 3, während Messung ohne Mensch (eigene Aufnahme)

7.3 Abstandberechnungsgenauigkeit mit unterschiedlichem Material – Versuch 3

Ziel ist es ist es das Materiale der zu bestimmenden Objekte mit Hilfe dieses Versuchs zu diversifizieren. In diesem Versuch wird die Distanz zu bewachsenen Felsen in einem Abstand von 5 bis 15m berechnet. Die Textur der Felsen ist im Vergleich zur verwendeten Sperrholzplatte in Versuch 2 deutlich erhöht. Jeder erhaltene Wert wird auf ganze Dezimeter aufgerundet. Bei diesem Versuch werden dreidimensionale Objekte gemessen, der Unterschied in Abstand wird hier aber vernachlässigt.

Abbildung 26 Die zu messende Fläche (rot) (eigene Aufnahme)

7.4 Auf offenem Gewässer – Versuch 4

Das Programm wird mit einem Laptop auf einem Segelboot getestet. Mit diesem Versuch soll bestimmt werden, ober der selbsttrainierte Yolov5-Algorithmus auch unter realen Bedingungen Motor- und Segelboote des Vierwaldstättersees erkennt.

Neben der Stereokamera wurde noch eine zweite Kamera installiert, um Videoaufnahmen zu machen. Es wird bestimmt wie viele Boote vom Algorithmus erkannt werden können. Die Software wird gestartet, wenn sich eine Verkehrssituation ergibt. Die Bilder der Objekte werden gespeichert (siehe Anhang I). Da die Software auf dem Laptop eine geringe Bildwiederholungsrate hat, gibt es einen zeitlichen Abstand zwischen den Bildern.

Abbildung 27 Stereokamera (links), Videokamera (rechts) (eigene Aufnahme)

8 Ergebnisse

In diesem Kapitel werden die Ergebnisse der durchgeführten Versuche graphisch dargestellt und näher beschrieben. Es wird auch auf die Trainingsperformance der verschiedenen Yolov5-Modelle eingegangen.

8.1 Versuch 1

In Versuch 1 wurden die Abstandsmessungen verschiedener Objekte mit der Zentrumsmethode und dem Durchschnittsabstandswert der gesamten Box gemacht. Die Daten in Abbildung 28 zeigen geringen bis garkeinen Einfluss der Berechnungsmethode auf den berechneten Abstand einer Pappkiste. Die Resultate bei einem Abstand von 4m und 5m weichen voneinander ab, da durch eine minimal ungenaue Platzierung der *Bounding Box* wenige Pixel, die nicht zum Objekt gehören (Hintergrund) mit einberechnet werden. Es ist auch erkennbar, dass die Diskrepanz zwischen den berechneten Durchschnittswerten und den wirklichen Messwerten immer grösser wird.

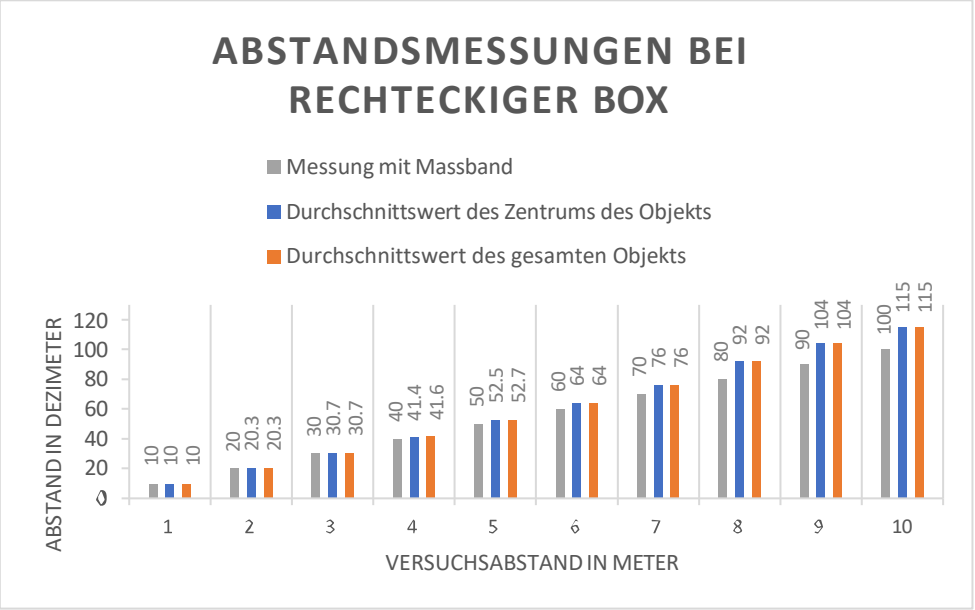


Abbildung 28 Abstandsmessungen einer rechteckigen Box (eigene Darstellung)

In wird gezeigt, dass die zentrale Messmethode in einem Bereich von 1seb bis 10m genauere Resultate liefert. Die Diskrepanz zwischen der Messung mit Massband und den beiden Durschnittwerten wächst kontinuierlich.

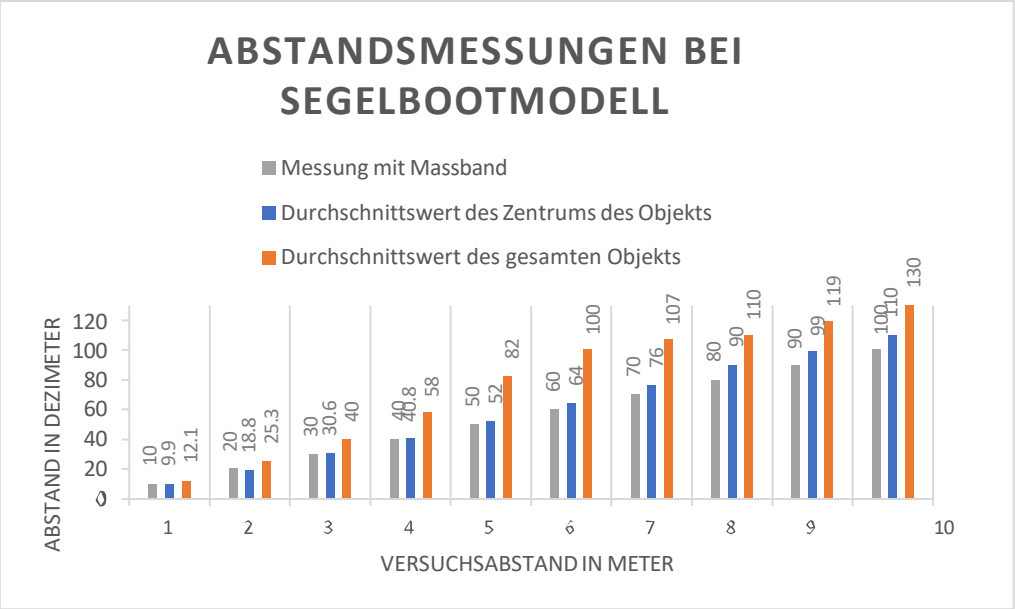


Abbildung 29 Abstandsmessungen eines Segelbootmodells (eigene Darstellung)

In Abbildung 30 sind die Zentrumsberechnungen genauer als die Messungen der gesamten *Bounding Box*. Die Ergebnisse der beiden Messmethode sind aber näher beieinander als in Abbildung 29.

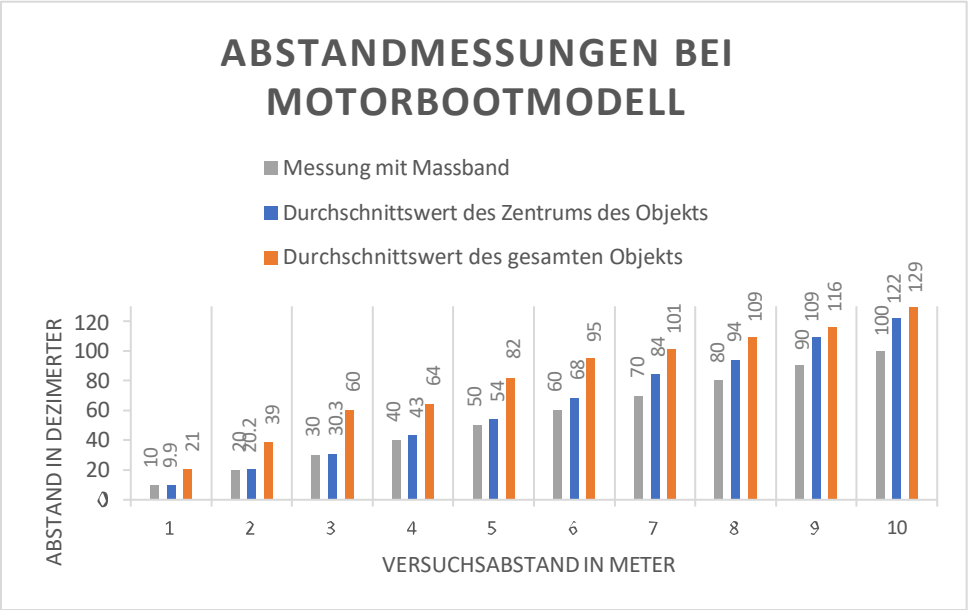


Abbildung 30 Abstandsmessungen eines Motorbootmodell (eigene Darstellung)

Bei den Messungen in Versuch 1 wird der Abstand mit Hilfe des Zentrums des Objekts berechnet. Diese Methode liefert unter den gewählten Umständen in diesem Versuchsaufbau die besten Ergebnisse. Sie ist somit die genaueste Methode.

Laut den Ergebnissen in Abbildung 31 handelt es sich bei der Verschlechterung der Genauigkeit im Vergleich zum realen Abstand um einen annährend linearen Zusammenhang. Dies gilt nur für den betrachteten Bereich. Die Messungen des Zentrums zeigen einen lineareren Verlauf als die Messungen der gesamten Fläche. Die Abstandsmessungen der gesamten Fläche sind durchschnittlich grösser und weichen mehr vom realen Abstand ab. Die Zentralen Messungen sind näher beieinander, sodass in Abbildung 31 mehrere Geraden übereinander liegen.

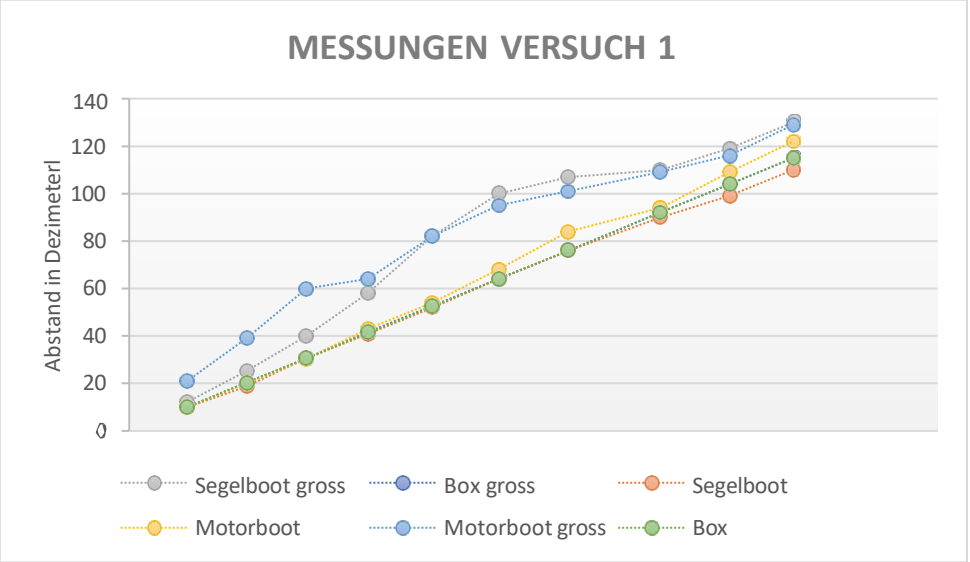


Abbildung 31 Darstellung aller Messungen aus Versuch 1 (eigene Darstellung)

8.2 Versuch 2

Wie in Abbildung 32 ersichtlich wächst die Diskrepanz zwischen dem realen Wert und den Gemessenen Werten, während der Unterschied zwischen der Zentrumsmessung und Gesamtmessung über die 8-25m im Verhältnis ähnlich bleibt.

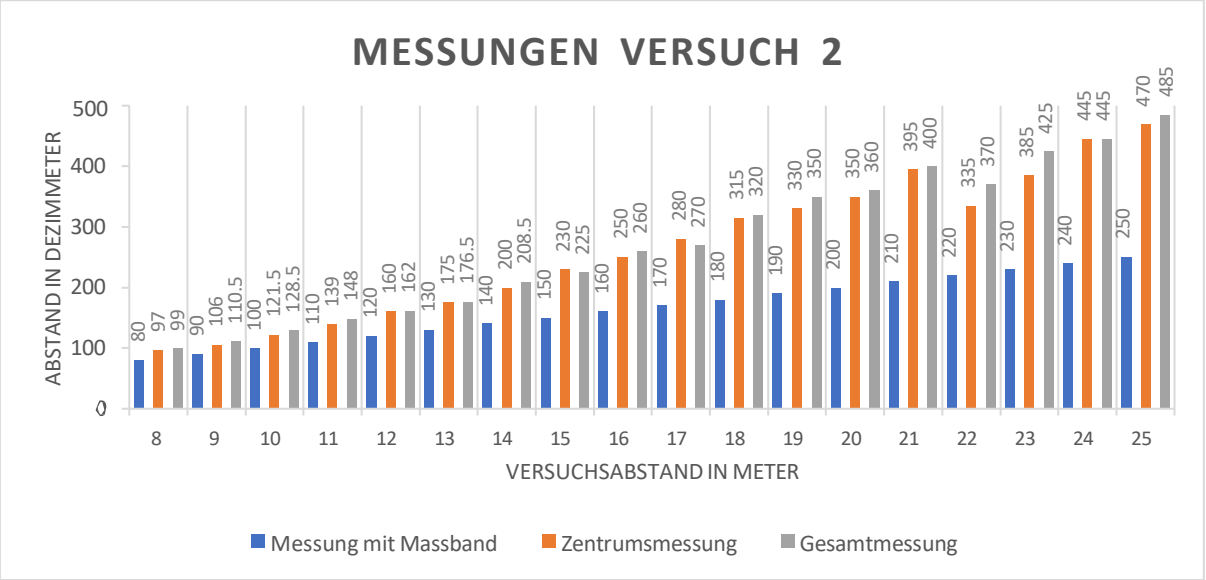


Abbildung 32 Messungen aus Versuch 2 (eigene Darstellung)

Aus Abbildung 33 und Abbildung 34 ist ersichtlich, dass die Differenz zwischen dem wirklichen Abstand und Messung kontinuierlich wächst. Die Zentrumsmessungen in Abbildung 33 hat eine etwas geringere Abweichung zum realen Abstand als die Messungen der gesamten *Bounding Box* in Abbildung 34, jedoch ist der Ergebnisbereich grösser.

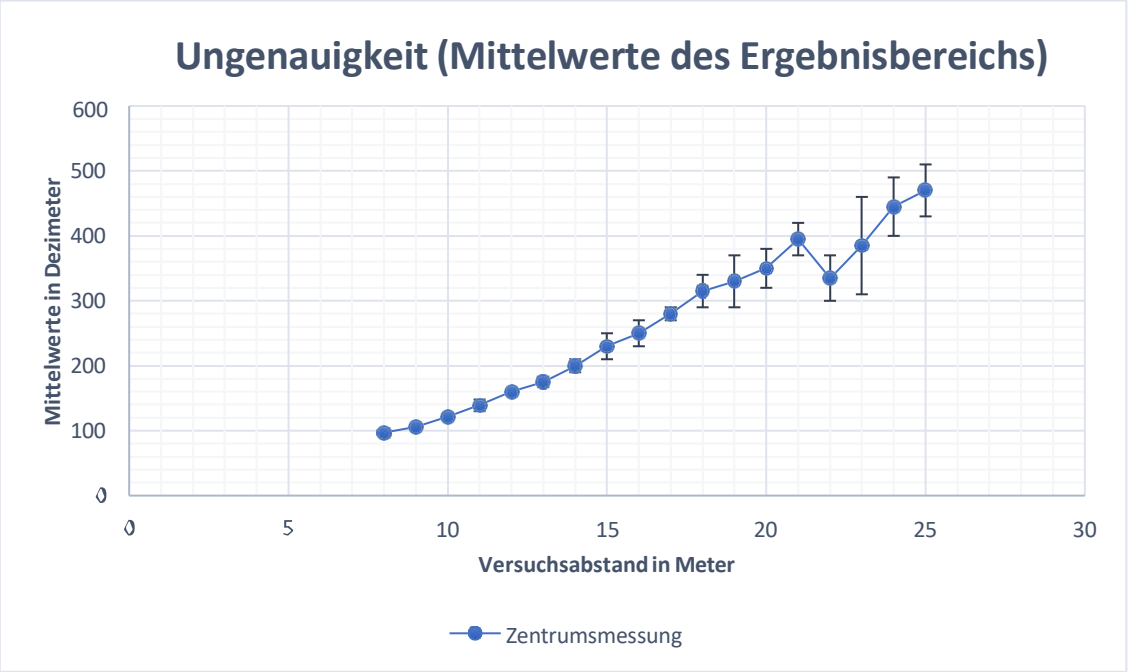


Abbildung 33 Mittelwerte mit Ergebnisbereich der Zentrumsmessung (eigene Darstellung)

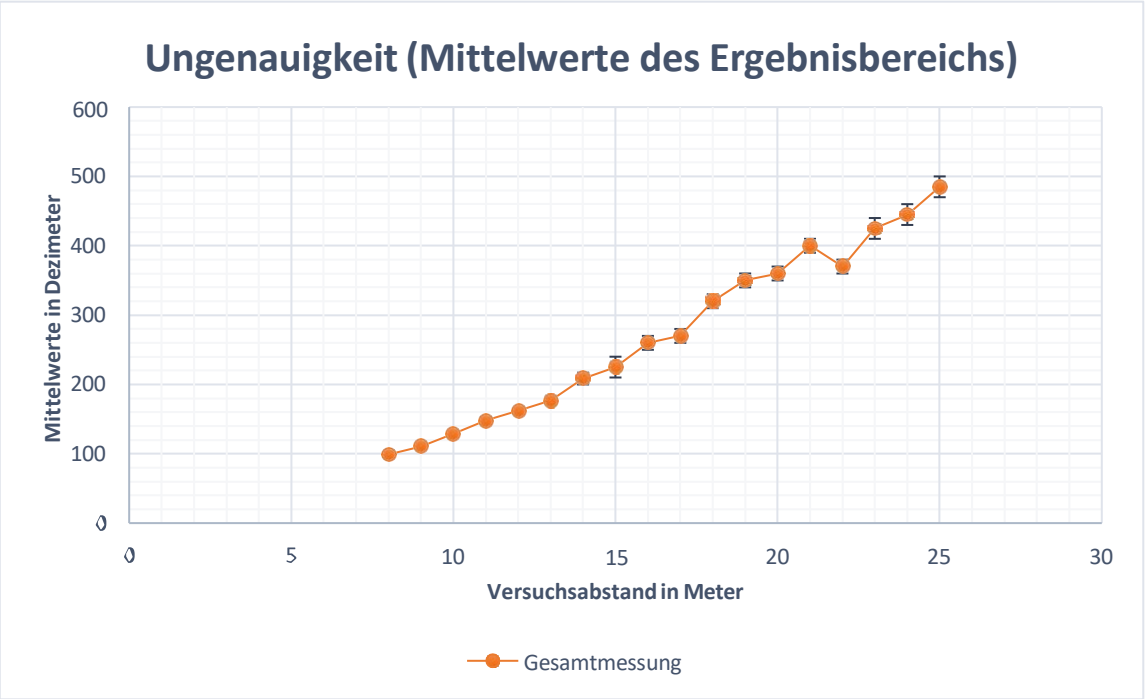


Abbildung 34 Mittelwerte mit Ergebnisbereich der Gesamtmessung (eigene Darstellung)

8.3 Versuch 3

In Abbildung 35 ist der Durchschnittsabstandswert des Zentrums geringer als der, der gesamten *Bounding Box*. Die Abstandwerte sind im Vergleich zu Abbildung 32 in Kapitel 8.2 geringer.

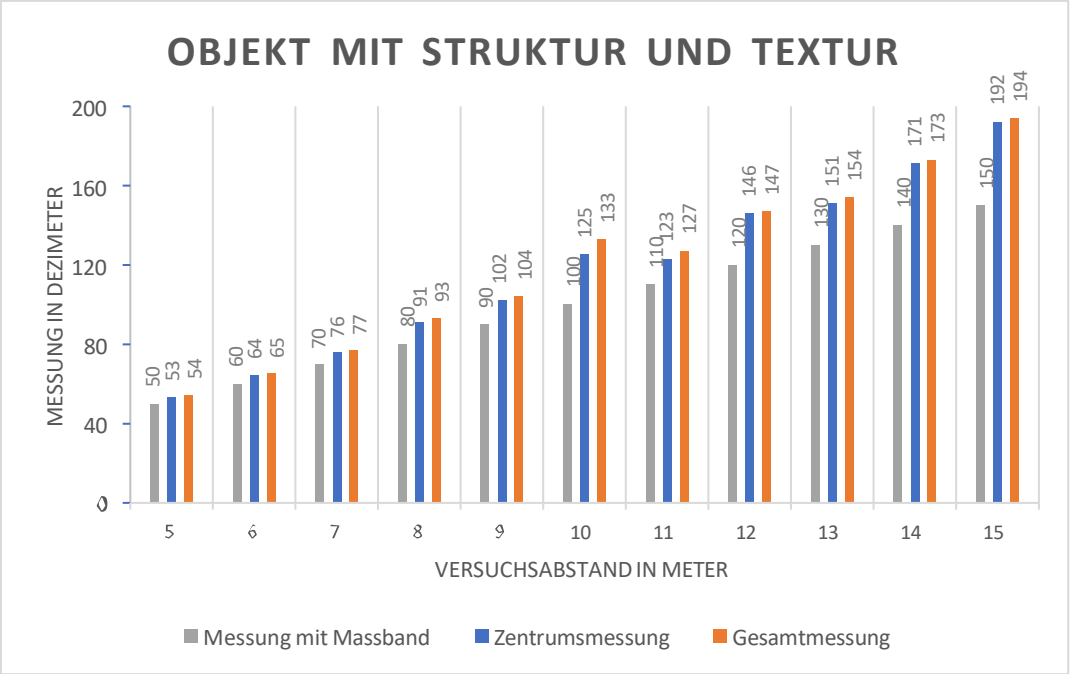


Abbildung 35 Messungen eines Objekts mit Struktur und Textur (eigene Darstellung)

8.4 Versuch 4

Auf 1306 gesammelten Bildern war auf einen Abstand von bis zu ungefähr 30m jeweils ein grosses Boot im Vordergrund. 1272-mal wurde das Boot erkannt, 34-mal wurde es nicht erkannt. Daraus resultiert eine Präzision von 97.3%. Von insgesamt 1616 Objekten in 1306 Bildern wurden 276 Objekte nicht erkannt, wenn sie sich im Hintergrund (bis zu mehreren hundert Metern entfernt) befanden. Wenn diese Boote auch mit einkalkuliert werden, ist die Präzision auf alle Objekte 80.8%

Abbildung 36 Beispielbild aus den gesammelten Bildern (eigene Aufnahme)

8.5 Yolov5 Trainingsperformance

Tabelle 1 zeigt die Evaluation der *Pretrained* Modelle, während Tabelle 2 die selbsttrainierten Modelle zeigt. Die vortrainierten Modelle wurden von *Ultralytics* mit dem *COCO*-Datenset[39] trainiert. Das *COCO*-Datenset verfügt über insgesamt 123‘287 Bilder mit insgesamt 886‘284 Instanzen. Das Datenset hat 80 Klassen darunter auch die Klasse Boot, die 3146 Bilder enthält (ca. 10000 Instanzen). Es wird nur geprüft, wie genau die vortrainierten Modelle Boote erkennen können. Das Training mit einem grösseren Datenset hat keinen Einfluss auf die Inferenzzeit.

Alle Modelle wurden mit dem Validierungsdatenset getestet (siehe Kapitel 6.5.2). Der *Recall*-Wert bezeichnet die Trefferquote der Voraussagen des Models. Die *Average Precision* (AP) beschreibt die Fläche unter der Funktion des *Precision-Recall*-Diagramms. Dadurch kann ein einzelner Wert Auskunft über die Performance des Models geben. Die *Intersection Over Union* (IoU) beschreibt die prozentuale Überlappung zwischen der vorausgesagten *Bounding Box* und der eigentlichen *Bounding Box*. Alle Voraussagen mit einem IoU-Wert über einem bestimmten Mindestwert werden als korrekt klassifiziert. Für den *AP@0.5 muss die IoU* also mindestens 0.5 oder grösser sein. Für den AP@0.5:0.95 werden alle APs zwischen 0.5 und 0.95 mit einem Zwischenschritt von 0.05 berechnet. Je grösser die IoU wird, desto kleiner wird auch die Genauigkeit, da die Anzahl der Voraussagen, die eine IoU von 0.95 oder grösser haben, kleiner ist. Der Durchschnitt aller APs in einem Bereich von 0.5 bis 0.95 wird als AP@0.5:0.95 bezeichnet. Für eine genaue Erläuterung der Definitionen der Begriffe und Formeln siehe (Liu, 2020) und (Yohanandan, 2020).

Das Model mit der höchsten AP@0.5 von 0.73 ist *yolov5l6 1280,* das mit einer Bildgrösse von 960 getestet wurde. Das Model ist aber im Vergleich zu *yolov5l* mit einer Bildgrösse von 640 mehr als vier Mal so langsam. Da das *yolov5l6 1280* Model eine Architektur hat, die für eine Auflösung von 1280 Pixel optimiert ist, performt es beim Test mit einer geringeren Auflösung von 448 Pixel schlechter. AP@0.5 sinkt auf 0.633. Zu beachten ist jedoch, dass die Präzision von 0.862 die höchste der vortrainierten Modelle ist. Das yolov5l 640 Model mit einer etwas geringeren AP@0.5, kombiniert hohe Genauigkeit mit deutlicher erhöhter Geschwindigkeit.

Am unteren Ende des Leistungsspektrum liefert yolov5m 640, getestet mit einer Bildgrösse von 416 Pixeln, relativ gesehen die beste Performance. Yolov5m 640 wurde ursprünglich mit einer Bildgrösse von 640 trainiert. Eine Reduktion beim Test auf eine Bildgrösse von 416 sorgt für einen geringe AP@0.5 Einbüsse. Der grosse Vorteil an dieser Reduktion an Auflösung ist jedoch, dass die Inferenzzeit ungefähr halbiert wird.

Die selbsttrainierten Yolov5 Modelle (L6, L, S, N) zeigen durchschnittlich eine bessere Genauigkeit als die vortrainierten Modelle vom Yolov5 Github-Repository (ultralytics, 2022a). Das yolov5l 416 Model (Batchsize 128), dass mit einer Bildgrösse von 416 getestet wurde, hat die höchste AP@0.5:0.95 und die zweithöchste Präzision der selbsttrainierten Modelle. Yolov5l6 896 mit Testgrösse von 448 hat eine um 0.001 höhere Präzision, aber eine 11.7 % schlechtere AP@0.5. Das Model yolov5l 416 mit einer Testgrösse von 928 (zweites von oben), hat eine zu hohe Inferenzzeit (566.8 ms), um für das Produkt verwendet zu werden. Weil Modelle mit einer geringeren Batchsize Vorteile haben können (siehe Kapitel 9.3), ist das yolov5 416 Model (Batchsize 16), getestet mit Testbildern der Grösse 416, ein vielversprechendes Model. Es hat einen 2.4 % geringeren AP@50, aber jedoch den höchsten Recall (0.746) aller selbsttrainierten Modelle und ist gleich schnell.

[39] *Common Objects In Context*

Für interaktive Grafiken der Trainingsperformance und des Trainingsprozesse finden sie im digitalen Anhang I.

Tabelle 1

Trainingsperformance der vortrainierten Yolov5-Modelle

Model	Precision	Recall	AP@0.5	AP@0.5:0.95	Image Size	ms
yolov5l6 1280	0.814	0.661	0.73	0.402	960 (928)	113.4
yolov5l 640	0.825	0.637	0.704	0.394	640	27.1
yolov5l 640	0.816	0.586	0.687	0.373	928	728.1
yolov5l 640	0.775	0.585	0.65	0.363	416	12.2
yolov5m 640	0.81	0.576	0.649	0.355	640	16.2
yolov5m 640	0.804	572	0.634	0.347	416	7.6
yolov5l6 1280	0.862	0.548	0.633	0.351	448 (416)	16.6
yolov5m 640	0.713	0.528	0.612	0.32	928	32.8
yolov5s 640	0.683	0.459	0.536	0.257	640	8
yolov5s 640	0.699	0.471	0.525	0.258	416	4.1
yolov5s 640	0.598	0.456	0.475	0.206	928	16
yolov5n 640	0.658	0.359	0.403	0.185	416	3.2
yolov5n 640	0.535	0.415	0.403	0.173	640	3.9
yolov5n 640	0.497	0.395	0.381	0.15	928	7.8

Tabelle 2

Trainingsperformance der selbsttrainierten Yolov5-Modelle

model	Precision	Recall	AP@0.5	AP@0.5:0.95	Image Size	ms	Batch Size
yolov5l 416	0.883	0.74	0.809	0.416	416	9.7	128
yolov5l 416	0.862	0.702	0.796	0.37	928	566.8	16
yolov5l 928	0.862	0.702	0.796	0.37	928	46.6	16
yolov5l 416	0.868	0.72	0.792	0.367	928	62.7	128
yolov5l 416	0.835	0.746	0.79	0.394	416	9.6	16
yolov5s 416	0.885	0.695	0.763	0.375	416	2.5	128
yolov5l6 896	0.825	0.668	0.751	0.354	960	64.4	16
yolov5n 416	0.827	0.663	0.733	0.316	928	4.8	16
yolov5s 416	0.811	0.678	0.731	0.3	928	10.3	128
yolov5l6 896	0.884	0.663	0.724	0.322	448	12.6	16
yolov5n416	0.835	0.64	0.716	0.299	928	4.9	128
yolov5n 416	0.812	0.639	0.702	0.339	416	1.4	16
yolov5n416	0.85	0.622	0.698	0.338	416	1.7	128
yolov5l 928	0.814	0.589	0.68	0.295	416	11.8	16

Die selbsttrainierten Yolov5 Modelle zeigen einen AP@0.5 Zuwachs von über 10 %. Die AP@0.5:0.95 ist beim selbsttrainierten Model über 5 % grösser. Die selbsttrainierten Modelversionen treffen oder übertreffen die Genauigkeit des vortrainierten Models von (ultralytics, 2022a) mit fast dreifacher Geschwindigkeit (2.83x).

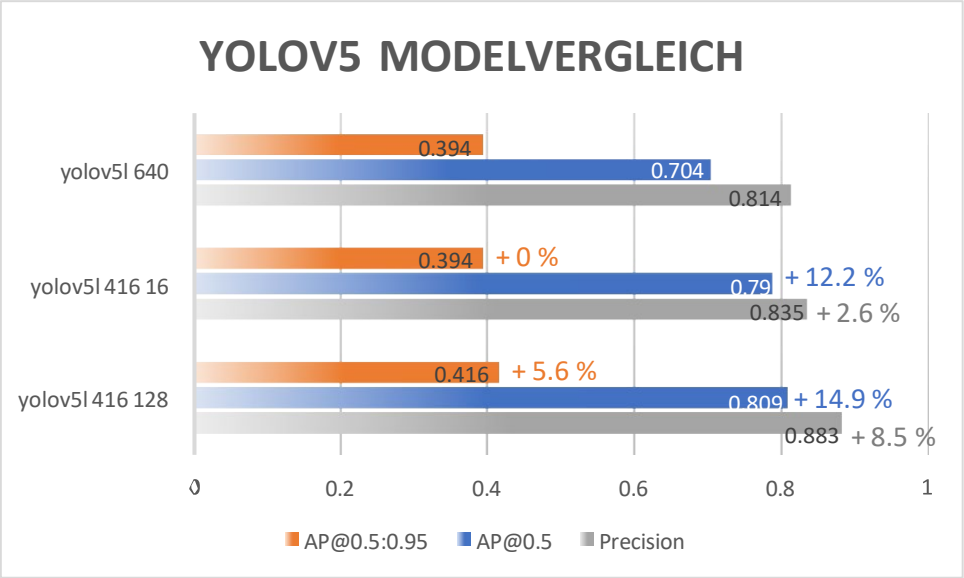


Abbildung 37 Es werden die vielversprechendsten Modelle (in rot) verglichen

8.6 Datenset Statistik

Links oben aus Abbildung 38 lässt sich entnehmen, dass es im gesamten Datenset 4000 Instanzen gibt, also insgesamt 4000 Objekte im Datenset vorhanden sind. Die Achsen der Abbildung 38 und Abbildung 40 sind zwischen 0 und 1 beschriftet. Dieses Intervall stellt die Position im Bild dar, wobei die Koordinate (0,0) die untere Kante links und (1/1) die obere rechte Kante eines beliebigen Bildes darstellt. Oben rechts in Abbildung 38 werden die Positionen aller Boxen aufgezeigt, mit denen die Objekte markiert werden. Der zentrale Bereich lässt darauf schliessen, dass zu einem grossen Teil eher kleinere Boxen um Objekte gezeichnet wurden. Kleinere Boxen bedeuten eine grosse Entfernung zur Kamera. Links unten in Abbildung 38 werden die Positionen aller Instanzen im Bild beschrieben. Je dunkler ein Punkt ist, desto mehr Instanzen liegen an einem Ort. Es ist ersichtlich, dass die meisten Objekte oberhalb der Wasserlinie sind (siehe Abbildung 39). Unten rechts in Abbildung 38 kristallisieren sich zwei Gruppen heraus: Segelboote und Motorboote. Denn Segelboote haben eine geringe Länge und eine grosse Höhe, während Motorboote verhältnismässig lang für ihre Höhe sind.

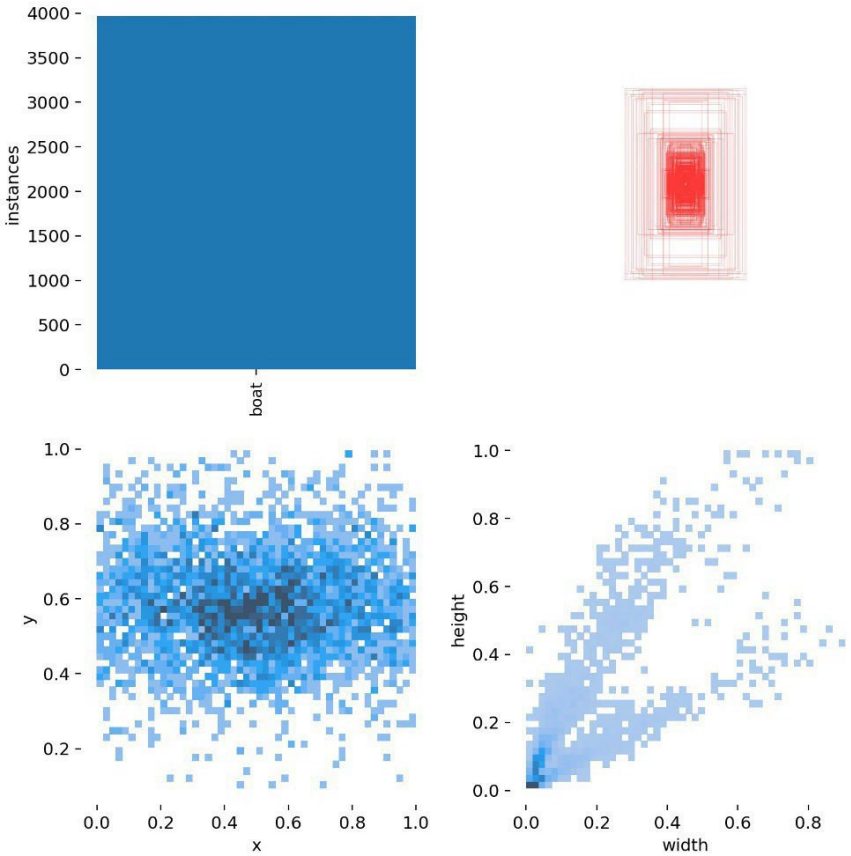


Abbildung 38 Datenset Statistik (eigen Darstellung)

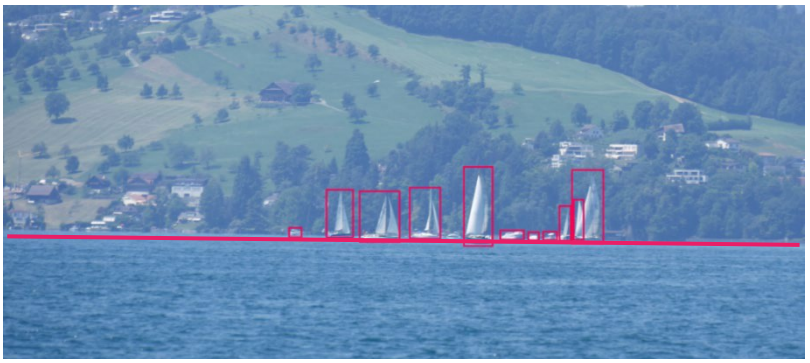


Abbildung 39 Die rote Linie stellt die Wasserlinie dar (eigene Darstellung)

Abschnitte 1 und 3 in Abbildung 40 zeigen, dass bei einer geringen Breite des Objekts die Instanzen am besten auf der X-Achse verteilt sind. Während Objekte mit einer geringen Breite sich hauptsächlich über der Wasserlinie befinden. Dasselbe gilt für die Höhe in den Abschnitten 2 und 4. Die meisten Objekte haben eine Höhe und Breit von unter 0.5 (siehe Abschnitte 5 und 6).

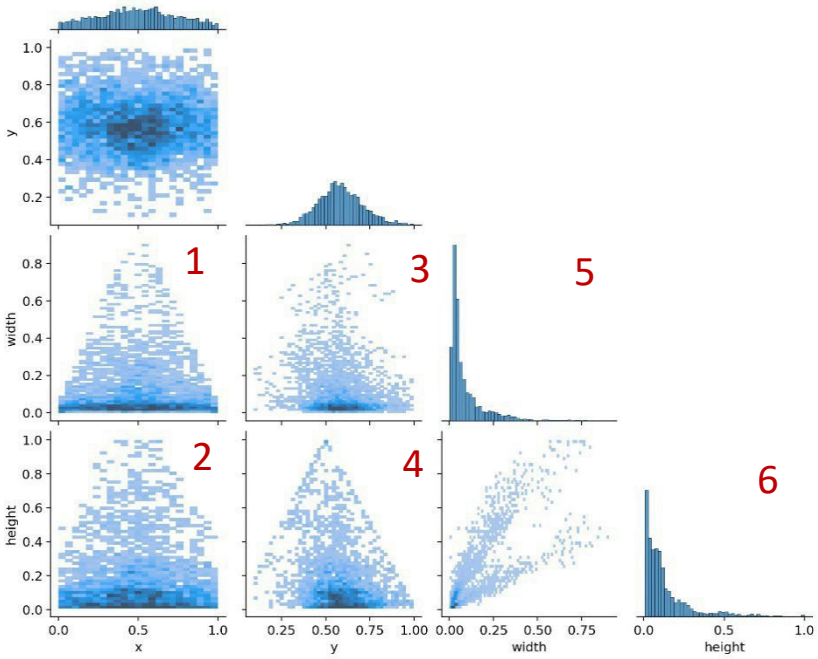


Abbildung 40 Datenset Statistik (eigene Darstellung)

9 Diskussion

In diesem Kapitel werden die erhalten Ergebnisse der Versuche eingeordnet und analysiert. Anschliessend werden Methode zur Verbesserung und Optimierung der Software vorgeschlagen.

9.1 Interpretation der Messungsversuche

Wie in Kapitel 8.1 zu erkennen ist, wächst die Ungenauigkeit der Messungen je nach Versuch ab einem Abstand von 10m auf mehrere Meter. Verschiedenste Faktoren sind hierfür verantwortlich.

Neben dem natürlichen Genauigkeitsverlust der Stereokameradaten (Kapitel 4) konnten durch Versuche einige einflussreiche Faktoren isoliert werden. Unter anderem beeinflussen die Platzierung der zu messenden Fläche durch den Yolov5-Algorithmus, die Messmethode (Zentrum oder gesamte *Bounding Box*) und das Material des zu messenden Objekts die Genauigkeit der Abstandmessung. In den Versuchen wird die Platzierung der *Bounding Box* als korrekt angenommen. Durch Versuch 1 lässt sich feststellen, dass die Zentrumsmessmethode genauere Ergebnisse liefert als die Gesamtmessung. Dies gilt vor allem für Objekte, die eine nicht viereckige Form haben, bzw. eine Form haben, die nicht genau durch eine rechteckige Box abgedeckt werden kann. Daher die folgende Erkenntnis: Je grösser der Anteil von Hintergrund-Pixeln im Vergleich zu Pixeln, die zum Objekt gehören, desto grösser ist der Einfluss des Hintergrunds auf den Messwert. Da der Hintergrund hinter dem Objekt gelegen ist, wird der Messwert verfälsch und somit grösser. Objekte, die im Vordergrund sind, könnten einem umgekehrten Effekt haben, jedoch würde mit grosser Wahrscheinlichkeit ein halbverdecktes Objekt nur halb vom Yolov5-Algorithmus erkannt werden. Somit wird der Vordergrund eliminiert. Versuch 1 liefert ausreichende Daten und zeigt, dass die Zentrumsmethode am genausten ist.

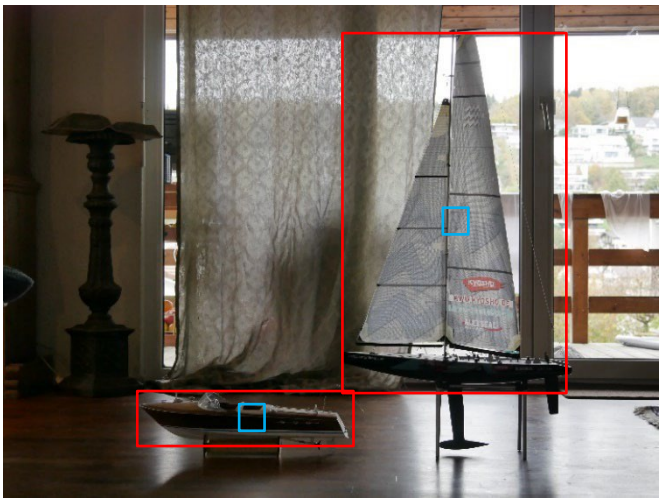


Abbildung 41 Zentrumsmessung ist gleich gross (blau), Gesamtmessung (rot) (eigene Aufnahme)

Das Verhältnis der gesamten *Bounding Box* und dem Zentrumsbereich spielt auch eine Rolle. Aus den Daten in Versuch 1 lässt sich erschliessen, dass die Messergebnisse der beiden Methoden sich immer weiter einander annähern, je weiter entfernt ein Objekt ist. Dies verhält sich dementsprechend, da der Zentrumsbereich verhältnismässig immer grösser wird, da er nicht während der Verkleinerung der gesamten Box mit schrumpft. Der Zentrumsbereich soll seine grösse behalten, da er sonst bei einer grossen Verkleinerung der gesamten Box selbst bis auf den Bruchteil eines Pixels skaliert werden würde. Auch wenn der Zentrumsbereich noch eine Fläche von wenigen Pixeln aufweist, ist das Ergebnis durch die Reduktion von Pixeln mehr der Variabilität eines einzelnen Pixels ausgesetzt. Dies wird zum Problem, wenn die *Depth Map* der Stereokamera durch Rauschen beeinflusst wird. Eine mögliche Gegenhandlung ist die Senkung des *Confidence Threshold* von 255 auf einen geringeren Wert. Dies würde jedoch jeden Pixel mit einem Wert unter dem *Confidence Threshold* aus der *Depth Map* entfernen. Das Ergebnis wäre eine *Depth Map* mit wenig Messpunkten.

Es lässt sich also sagen: Je kleiner die Anzahl Pixel der *Bounding Box* ist, desto mehr gleicht sich der Zentrumsbereich an den Gesamtbereich an und verliert somit seinen Vorteil.

Mit den Daten aus Versuch 2 wird klar, dass die Verwendung der Zentrumsmessungen bei Distanzen von *über 10m* bei Objekten mit *low texture* Materialien nicht ratsam ist. Obwohl die der Mittelwerte Zentrumsmethode geringer sind, ist hier die Zentrumsmethode nicht die bessere Lösung. Denn der Ergebnisbereich der Gesamtmethode ist deutlich geringer. Die grössere Pixelzahl der Gesamtmethode führt im Vergleich zur Zentrumsmethode bei einer grossen Abstandsvariabilität zu einem geringeren Ergebnisbereich. Aus diesem Grund sind die Messungen der Gesamtmethode in bei der Verwendung von *low texture* Objekten verlässlicher.

Die Ergebnisse aus Versuch 3 zeigen, dass die Genauigkeit durch das Material des zu erkennenden Objekts beeinflusst wird. Dies ist eine bekannte Schwäche von Stereokameras in diesem Preissegment[40] (luxonis, 2022). Die OAK-D Stereokamera hat Schwierigkeiten, die Depth Map für sich wiederholende Strukturen, wie zum Beispiel für eine weisse Wand, zu berechnen. Dies liegt am grundsätzlichen Prinzip von Stereo Vision, denn Stereobilder werden auf korrespondierende bzw. ähnliche Pixel untersucht. So kann mit Hilfe des Abstandes der beiden, auf einer Achse parallelen, Sensoren, die die Stereobilder aufgenommen haben, schlussendlich die Depth Map berechnet werden (siehe Kapitel 4). Dieser Prozess wird durch sich ähnelnde Pixel deutlich erschwert. Pixel ähneln sich häufig an Oberflächen mit wenig Textur und Struktur. Die Messungen in Versuch 2 können aus diesem Grund als Worst-Case-Szenario betrachtet werden. Vielversprechend sind Daher die Ergebnisse von Versuch 3. Ein grosser Vorteil der Daten aus Versuch 3 gegenüber den Daten aus Versuch 2 ist minimale Variabilität der Ergebnisse, anstatt den Ergebnisbereich und seinen Mittelwert anzugeben reicht es hier aus, den erhaltenen Wert auf ganze Dezimeter aufzurunden. Da es die Aufgabe der Software ist Boote zu erkennen und Boote über eine Reihe an Strukturen und Texturen verfügen werden die Daten aus Versuch 3 als repräsentativ angesehen, denn sie stellen ein Szenario auf dem Wasser besser dar als die Daten aus Versuch 2.

9.2 Genauigkeit der Abstandsmessungen

Die Abstandmessungen der OAK-D Stereokamera werden ab einem Abstand von 10m bis zu mehreren Metern ungenau. Es werden verschiedenen Messmethoden präsentiert, um diese Ungenauigkeit zu reduzieren. Sie sind in der Lage in einem Abstand von bis zu 10m die Ungenauigkeit je nach Objekt deutlich zu senken. Ab einem Abstand von 10m ist die Verminderung der Ungenauigkeit nur noch minimal im Vergleich zur eigentlichen Ungenauigkeit. Wenn man die Ergebnisse mit Daten von *Luxonis* vergleicht, wird klar, dass bei besten Bedingungen die Kamera bereits ab 8m eine Ungenauigkeit von 6% (0.4m) aufweist (Luxonis, 2022). Diese weitet sich ab 13m auf 8.1% (1.1m) aus. Obwohl die Kamera eine theoretische Reichweite von 35m hat, ist die Ungenauigkeit selbst unter den besten Umständen (gute Beleuchtung, Objekt mit Textur usw.) sehr gross. Die Stereokamera sollte daher nur auf einem Abstand von höchstens 15m eingesetzt werden. Es wird davon ausgegangen, dass Boote anhand ihrer Materialien sich nicht zu sehr von den getesteten Objekten unterscheiden. Deshalb gilt dieselbe Distanzempfehlung auch für das Erkennen von Booten auf dem Wasser. Da die Kamera Schwierigkeiten hat sich wiederholende Strukturen mit wenig Textur zu erkennen, ist das Messen von Punkten auf der Wasser Oberfläche nicht möglich. Es ist also nur die Distanzmessung von Objekten auf dem Wasser und nicht vom Wasser selbst möglich.

Trotz der oben genannten Punkte wäre der Betrieb eines CAS mit der OAK-D Stereokamera möglich, jedoch nicht sehr effektiv, da man genaue Messungen nur in einem sehr eingegrenzten Gebiet erhält. Das gute Preisleistungsverhältnis der Stereokamera steht jedoch ausser Frage, da sie mit vielen

[40] Mehrere hundert Schweizer Franken

verschiedenen Features ausgestattet ist und führend in ihrer Preisklasse ist. Da sie nicht für weite Strecken ausgelegt ist, sollte man das Produktbudget erweitern und auf eine Stereokamera mit besserer Genauigkeit und Reichweite zurückgreifen. Eine Strategie wie die Reichweite erweitert werden kann, wird in Kapitel 9.7.1 besprochen.

9.3 Interpretation der Yolov5-Ergebnistabellen

Ein Yolov5-Algorithmus, der mit einem Bild getestet wird, das eine grosse Auflösung hat, hat durchschnittlich eine längere Inferenzzeit. Bilder mit einer kleineren Auflösung sorgen für eine kürzere Inferenzzeit, weil weniger Pixel analysiert werden müssen. Ein Algorithmus, der mit einer eher geringeren Auflösung trainiert wird, kann seine Präzision erhöhen, wenn er mit Bildern mit einer grösseren Auflösung getestet wird. Diese grössere Auflösung sorgt aber dafür, dass der Algorithmus eine überproportionale Inferenzzeit hat. Will man das verhindern, wird die Inferenzzeit reduziert, indem der ursprüngliche Algorithmus mit Bildern trainiert wird, die eine höhere Auflösung haben. Die resultierende AP ist genau gleich (siehe Tabelle 1 Zeile 2,3).

Die Genauigkeit des Algorithmus sinkt, wenn er mit einer grossen Auflösung trainiert wird und daraufhin mit Bildern getestet wird, die eine kleine Auflösung haben. Dieses Vorgehen hat katastrophale Auswirkungen. Der Yolov5 hat in diesem Fall eine geringere AP als der yolov5n (Tabelle 2 Zeile 14).

Man sieht, dass die AP des Algorithmus steigt, wenn die Batchsize erhöht wird (siehe Tabelle 2. Das muss aber nicht unbedingt ein positives Zeichen sein. Dies kann ein Anzeichen *von Overfitting* sein. *Overfitting* findet statt, wenn ein Algorithmus zu sehr auf das Datenset, mit dem er trainiert wird, angepasst ist. Dadurch schwindet die Fähigkeit des Algorithmus zu Generalisieren. Beispielsweise wäre es für den Algorithmus schwieriger Boote zu erkennen, die nicht Teil des Datensets sind oder etwas andere Merkmale haben. Keskar et al. (2017) argumentiert mit klarer Evidenz, dass eine kleinere Batchsize die Fähigkeit eines Models verbessert zu generalisieren. Es muss jedoch darauf geachtet werden, dass die Batchsize immer noch verhältnismässig bleibt. In dieser Arbeit kämen Batchgrössen von 16 bis 256 *Samples*[41] in Frage. Der limitierende Faktor ist hier die bereitstehende Rechenleistung. Je nach Model eine ist eine Batchsize grösser als 200 nicht zu lässig. Die optimale Batchsize konnte in dieser Arbeit nicht ermittelt werden, es wurden Modelle mit Batchgrössen von 16 und 128 verglichen. Ein ungefährer Richtwert ist daher klar.

9.4 Genauigkeit und Performance des selbsttrainierten Yolov5-Algorithmus

Damit das CAS funktioniert, muss der Yolov5-Algorithmus genug Zeit haben, um die Bilder, die er von der Stereokamera erhält, zu verarbeiten. Die Stereokamera sendet Videomaterial mit einer Frequenz von 30 Bilder pro Sekunde an den Computer. Der Algorithmus muss also 30 Bilder pro Sekunde verarbeiten können. Das bedeutet, dass die Interferenzzeit des Algorithmus unter 33.3ms sein muss. Dieser Wert wurde unter isolierten Bedingungen vom selbsttrainierten Yolov5-Model erreicht. In Kombination mit der gesamten Software ist jedoch eine geringere Bildrate zu beobachten, weil noch anderer Code ausgeführt werden muss. Dieses Problem liess sich mit Hilfe weiterer Prozesse, die parallel Code ausführen, eindämmen.

Die Genauigkeit des selbsttrainierten Yolov5-Models wurde mit den Testdaten validiert und unter realen Bedingungen aufgezeigt. Der Test der Objekterkennung auf dem See zeigt, dass eine verlässliche Erkennung von Objekten mit einer Bildrate von mindestens 2 Bilder pro Sekunde möglich ist. Als verlässlich wird die Objekterkennung bezeichnet, weil mehr als 90% aller Objekt in einem Abstand von bis zu ungefähr 30 m erkannt wurden. Die Objekterkennungsrate ist jedoch geringer

[41] In diesem Fall Bilder

(80.1%), wenn alle Objekte, die auf den aufgenommenen Bildern abgebildet sind, in die Berechnung einfliessen. Darunter sind auch Objekte, die bis zu mehreren hundert Metern entfernt sind.

Die Software wird sogar noch verlässlicher, je höher die Bildwiederholungsrate ist. Denn die Wahrscheinlichkeit Boote genau zu erkennen, steigt weiter an, je grösser die Bildrate ist. Je häufiger ein aktuelles Bild der Sachlage dem Algorithmus präsentiert werden kann, desto grösser ist also die Chance ein Boot zu erkennen. Wichtig ist es festzuhalten, dass ein genaues Model die Distanzmessungen mit der Stereokamera auch unterstützt, weil die *Bounding Box* korrekt gesetzt wird.

9.5 Datenset

Das Datenset ist mit ca. 4000 Instanzen in 1446 Bildern kleiner als die empfohlene Mindestgrösse von *Ultralytics* (ultralytics, 2022a). Laut den Angaben sollen in einem Datenset mindesten 1500 Bilder mit insgesamt 10000 Instanzen pro Klasse sein. Obwohl nur 40% dieses Wertes erreicht wird, ist die Performance des Algorithmus beim Test mit den Validierungsdaten gleich gut oder besser. Das eigene Datenset ist im Vergleich zum *COCO*-Datenset sehr fokussiert und enthält nur Bilder von Booten des Vierwaldstättersees. Das *COCO*-Datenset enthält verschiedene Arten von Booten. Darunter sind auch Schlauchboote, Kajaks und Gondolas (COCO, 2022). Das eigene Datenset ist also besser an die Umgebung des Vierwaldstättersees angepasst und ist daher trotz der geringeren Datenmenge besser für die Anwendung auf dem Vierwaldstättersee geeignet. Jedoch ist es wahrscheinlich, dass die Fähigkeit des *COCO*-Datensets zu Generalisieren[42] grösser ist.

9.6 Skalierbarkeit der Software

Eine Stärke der Software ist es, dass sie so konzipiert ist, dass einzelne Teile einfach ausgetauscht werden können. Es möglich das Yolov5-Modell durch jedes andere Yolov5-Modell auszutauschen, ohne den Code zu verändern. Es ist daher in Sekunden möglich, sofern ein anderes Modell vorhanden ist, die beiden Modelle auszutauschen. Die Software ist mit jeglicher Hardware von Luxonis kompatibel. Man kann die Software einfach und schnell umfunktionieren, sodass sie zum Beispiel Menschen erkennt und lokalisiert. Zusätzlich kann ein anderer *Pathfinding*-Algorithmus aus einer Sammlung an Algorithmen ausgesucht werden. Die Auflösung der Karte, auf welcher der *Pathfinding*-Algorithmus ausgeführt wird, kann angepasst werden, sodass weniger Rechenleistung verbraucht wird. Es ist auch möglich das Modell mit einem der anderen selbsttrainierten Modelle auszutauschen, welches weniger Rechenleistung benötigt. Die Anpassung der Stereo Depth Auflösung ist auch möglich, dies sorgt für eine flüssigere Bilddarstellung. Alle diese Schritte können ergriffen werden, um die Software an die Leistung der verwendeten Hardware anzupassen. Dabei nimmt man aber eine schlechtere Erkennungsrate der Software in Kauf. So kann die Software auf einem Low-End Laptop oder auch auf einem High-End Computer verwendet werden. Dies hilft dabei den Formfaktor für unterschiedliche Boote anzupassen. Es ist theoretisch sogar möglich, eine angepasste Version der Software auf den internen Rechenressourcen der OAK-D Stereokamera auszuführen. Dies würde für eine handliche Form sorgen. Eine Alternative wäre auch, nur bestimmte Kalkulationen auf die Kamera auszulagern.

[42] Boote mit Ähnlichen Eigenschaften zu erkennen

9.7 Weitere Entwicklung der Software

Neben einer erneuten Datensammlung, die für ein verbessertes Datenset und somit auch einen genaueren Yolov5-Algorithmus verantwortlich ist, gibt es noch weiteres Optimierungspotential.

9.7.1 Reichweite

Um die Reichweite der Stereokamera zu erweitern, müsste man die sogenannte Baseline von 7.5cm auf 15cm erhöhen. Der folgende Vorschlag für ein Produkt von *Luxonis* wirkt sehr vielsprechend.

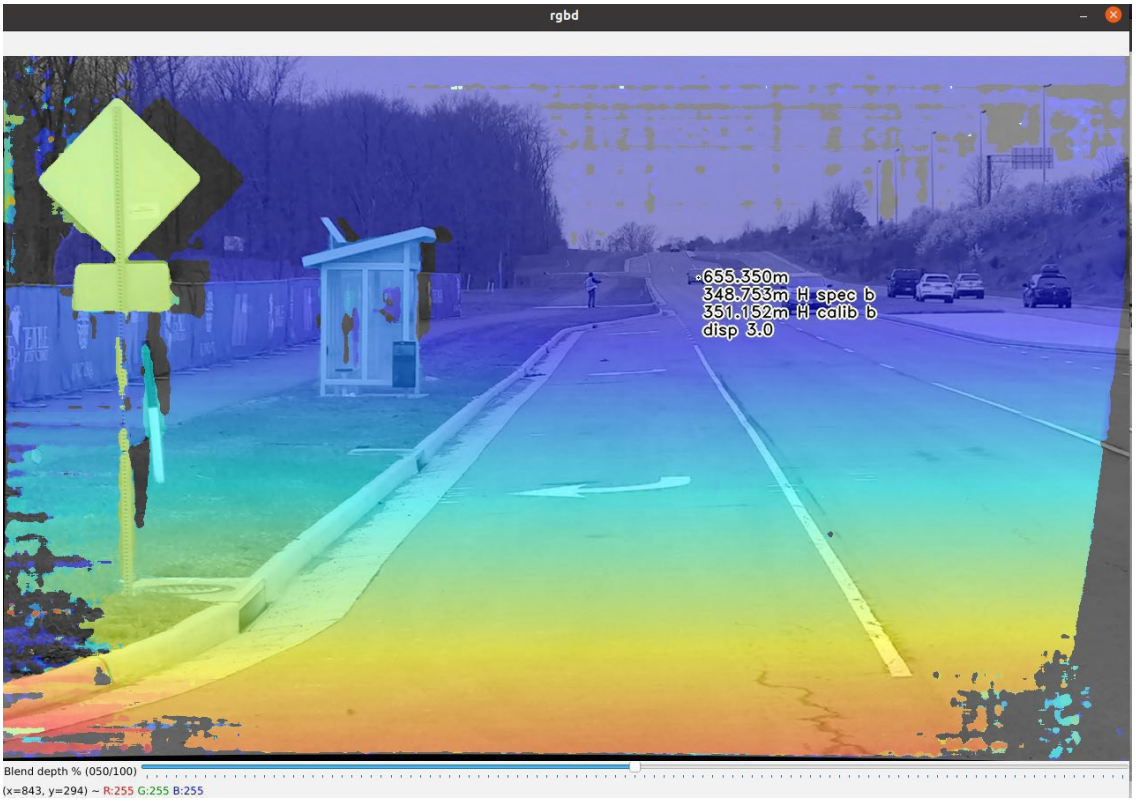


Abbildung 42 Luxonis Kamera mit einer Reichweite mit mehreren hundert Metern (luxonis, 2022)

Zum Zeitpunkt dieser Arbeit wird dieser Vorschlag umgesetzt und könnte Anfang 2023 in die Produktion gehen. Dies würde die Reichweite einer Stereokamera von Luxonis auf mehrere hundert Meter erweitern. Der Preis des fertigen Produkts wird unter 1000 Schweizer Franken sein (Luxonis, 2022). Deshalb wäre eine solche Kamera eine Alternative zur OAK-D Kamera, die für dieses Projekt verwendet wurde, um der Verwendung durch Privatpersonen zu entsprechen.

9.7.2 Sichtfeld

Im momentanen Zustand der Software ist der Fahrzeugführer noch für jede Reaktion selbst verantwortlich, die einen Crash zu verhindern. Wenn man jedoch der Software diese Fähigkeiten geben wollte, musste man zuerst das *Field of View* (FOV) erweitern. Im momentanen Zustand hat das CAS nur ein FOV von 69°. Eine Erweiterung des FOV würde die Software auch weitaus sicherer machen. Um das CAS in alle Richtungen um das Fahrzeug zu erweitern, müsste man also weitere Stereokameras anbringen und diese synchronisieren. Die Synchronisierung mehrerer OAK-D Stereokameras ist möglich.

9.7.3 Anwendung in dunkler Umgebung

Eine grosse Schwäche der OAK-D Stereokamera ist es in einer dunklen Umgebung Messungen durchzuführen. Denn die *Depth Map* wird unregelmässig und ungenau. Dieses Problem könnte man eindämmen, indem ein neuronales Netzwerk verwendet wird, um die Bilder, die von der Kamera kommen, aufzuhellen. Zero-DCE ist ein solches neurales Netzwerk (Guo et al., 2020). Zero-DCE ist vor allem für diese Anwendung geeignet, weil es sehr ressourcensparend ist. Es wäre also möglich die Software mit einer ähnlichen Bildwiederholungsrate zu betreiben.

Abbildung 43 Aufhellungseffekt von Zero-DCE (Guo, 2020)

9.7.4 Objektsegmentation

Anstatt der Verwendung von Objekterkennung könnte Instanz-Segmentation verwendet werden. Instanz-Segmentation markiert im Bild genau, wo sich ein Objekt befindet. Die genaue Angabe einer Fläche im Bild, die zu einem Objekt gehört, würde genauere Messungen ermöglichen. Dieselbe Form müsste nur auf die *Depth Map* übertragen werden. Diese Methode der Messung wäre mit Sicherheit genauer als eine Zentrumsmessung. Objekt-Segmentation wäre beispielsweise mit dem Yolact-Modell möglich (Bolya et al., 2019).

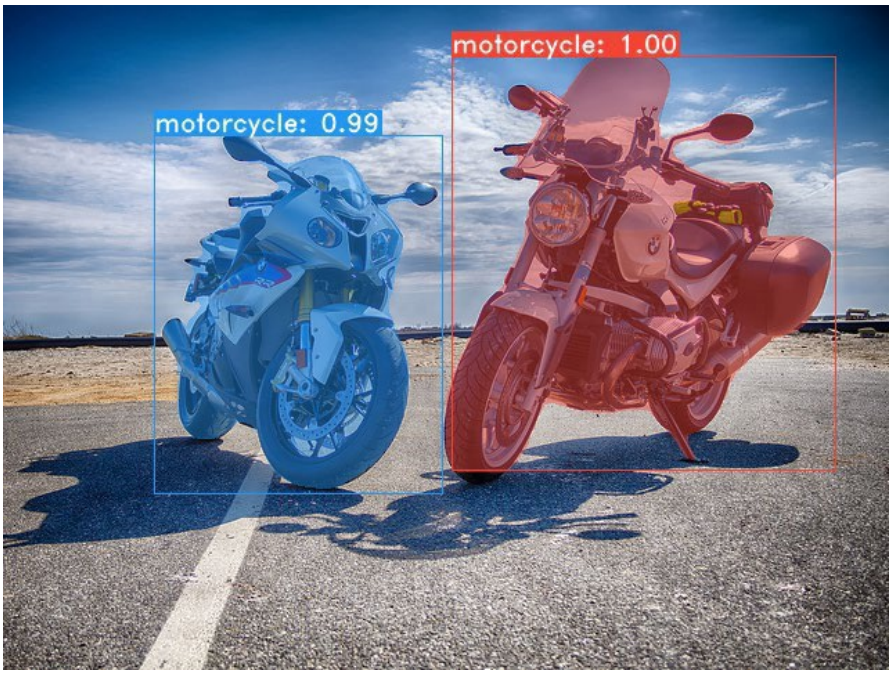


Abbildung 44 Objekt-Segmentation mit Yolact (Bolya, 2018)

10 Reflexion

Da ich mich seit meinem ersten Jahr an der Kantonsschule Zofingen für Informatik und Programmierung begeistere und gerne segle wusste ich schon früh, dass mein Projekt mit diesen Bereichen zu tun haben sollte. Wichtig wurde daher der Planungsprozess der Arbeit, der im Herbst 2021 begann. Während der Planung habe ich gemerkt, dass die Beschränkung auf bestimmte Teilaufgaben eines CAS und das Setzen von Schwerpunkten essenziell für die Machbarkeit des Projekts ist. Denn ursprünglich war es meine Idee einen vollumfassenden Autopiloten für Segelboote zu entwickeln. Die Gespräche mit Fachpersonen und Freunden haben mir dann sehr geholfen, meinen Entwurf zu konkretisieren und das Thema einzugrenzen. Sodass ich mich dafür entschieden habe ein CAS zu entwickeln. Nichtsdestotrotz war der schwierigste Teil der Planung, abzuschätzen, welche Teile der Software in Eigenproduktion entwickelt werden sollten und bei welchen Komponenten es besser war sich auf bereits existierende Produkte zu stützen. Ich habe mich deshalb aktiv dagegen entschieden, eine Stereokamera selbst zu bauen, da dies einem Projekt für sich nahekommt. Die Schwerpunkte dieses Projekts umfassen das Training eines Models mit eigenem Datenset, die Verwendung der Stereokamera und Entwicklung der Software, die diese verschiedenen Teile verbindet und performant genug ist, um in der Praxis angewendet zu werden. Eine grosse Schwierigkeit bei einem solchen Ziel ist es, dass sich das ganze wie mit einer Kette verhält. Reisst ein Glied der Kette, ist die ganze Kette unbrauchbar. Es war deshalb essenziell, dass die einzelnen Funktionsteile der Software erfolgreich entwickelt wurden, um schlussendlich ein funktionierendes Produkt vorweisen zu können.

Ich war während der Programmentwicklung häufig mit Problemen konfrontiert, die ich nicht durch wiederholte Anläufe lösen konnte. Durch die Informatikvorlesungen an der Universität Basel habe ich gelernt, Probleme mehr aus Distanz zu betrachten, und damit flexibler in meinen Lösungswegen zu werden. Im Verlauf der Arbeit habe ich auch verstanden, dass es wichtig ist, sich für das eigene Thema wirklich zu begeistern, das hat mir geholfen, auch schwierige Passagen zu überstehen.

Die Entwicklung der Software, gefolgt von der Evaluation des Produkts und dem eigentlichen Verfassen der Arbeit, stellt jedoch nur einen von drei Schritten dar. Retrospektiv betrachtet habe ich den Aufwand, der mit dem zweiten und dritten Schritt verbunden ist, unterschätzt. Obwohl ich mir mehr Zeit für die Entwicklung der Software eingeplant habe, bestand das Problem darin, dass ein Produkt, welches mit mehr Zeitaufwand entwickelt wird, auch ausführlicheres Testen und ausführlichere Dokumentation benötigt. Ich hätte also besser berücksichtigen können, dass mehr Aufwand in der Entwicklung auch mehr Aufwand in der Evaluation und Dokumentation nach sich zieht.

11 Beantwortung der Thesen

Es ist möglich mit der entwickelten Software die Mehrheit alle prädestinierten Objekte (Motorboote und Segelboote) mit Hilfe von Objekterkennung zu erkennen.

Die durchgeführten Versuche bestätigen, dass die entwickelte Software in der Lage ist Motor- und Segelboote des Vierwaldstättersees mit einer Wahrscheinlichkeit von 80.1%[43] zu erkennen. Dies gilt für alle untersuchten Objekte. In einem Bereich von bis zu ca. 30m beträgt die Präzision mehr als 90%. Ein Objekt galt beim Versuch auf dem Vierwaldstättersee erst als erkannt, wenn sich das Modell der Erkennung zu 70% sicher war.

Die Evaluation des Yolov5-Algorithmus mit Hilfe des Evaluationsdatensets[44] liefert eine Präzision von 88.3% und eine AP[45] von 0.809. Dies ist genug Evidenz, um für die Qualität des selbsttrainierten Yolov5-Modells zu sprechen.

Die Objekterkennung und Messungsungenauigkeit der Stereokamera zu einem Objekt lassen zu, dass ein Crash verhindert werden kann.

Der selbsttrainierte Objekterkennung ist bei Verwendung der OAK-D Stereokamera in der Lage, Objekte zu erkennen, die über die Reichweite der Abstandsmessungen der Stereokamera hinausgehen. Daher ist die Erkennung einer möglichen Gefahr mit der Software möglich, bevor der Abstand des erkannten Objekts berechnet werden kann. Eine alleinige Gefahrenerkennung nur durch die Objekterkennung ist möglich. Jedoch ist das Aufzeigen eines möglichen Ausweichmanövers mit Hilfe des *Pathfinding*-Algorithmus nur möglich, wenn die Position eines erkannten Objekts erfasst werden kann. Aus den durchgeführten Versuchen zur Messungsgenauigkeit lässt sich entnehmen, dass die Stereokamera auf eine Distanz von 10m eine Abweichung von bis zu ca. einem Meter hat. Die Verwendung der Kamera für Messungen, die auf wenige Meter genau sein sollen, wird nur bis zu einer Distanz von 15m empfohlen. Die Abstandsdaten der Kamera werden genauer je näher ein Objekt der Kamera ist. Eine erhöhte Ungenauigkeit auf eine weitere Entfernung (+15m) ist eher verkraftbar, denn für ein Ausweichmanöver sind kürzere Distanzen, die bei der OAK-D am genauesten sind, am wichtigsten. Des Weiteren helfen Methoden wie Depth Validation, das als sekundäre *Depth Estimation* Methode verwendet wird, und Depth Queueing dabei die Ungenauigkeiten der Stereokamera einzugrenzen und die Objektabstandsmessungen verlässlicher zu machen. Eine besondere Stärke besteht auch darin, dass mit Hilfe von Depth Validation, *Depth Estimation* ausserhalb der Reichweite der Stereokamera durchgeführt werden kann.

Das Verhindern eines Crashs ist anhand der erhobenen Daten und durchgeführten Versuche möglich, um jedoch dem Fahrzeugführer mehr Zeit zu geben und die Software sicherer zu machen sollte eine Stereokamera mit erhöhter Genauigkeit und Reichweite verwendet werden. Eine Stereokamera mit bis zu mehreren hundert Metern Reichweite, liegt mit einem Preis von unter 1000 CHF immer noch in der Preisklasse privater Wasserfahrzeughalter. Die OAK-D Stereokamera (300 CHF) ist in Bezug auf die Anwendbarkeit in der Praxis der limitierende Faktor. Es ist wichtig anzumerken, dass dies die Qualität der Software nicht herabstuft, denn die Software ist auch mit hochwertigeren Stereokameras kompatible. Gewonnene Erkenntnisse über die OAK-D Stereokamera sind auch auf andere Stereokameras übertragbar.

[43] Wenn Objekte mit einer weiteren Entfernung von bis zu mehreren Hundert Meter mit einberechnet werden
[44] Auch Boote mit einer Distanz von mehreren 100 Metern enthalten
[45] *Average Precision*

Die Performance (in Bezug auf Bildwiederholungsrate und die Erkennungsrate) der Software ist hoch genug, um für die Verwendung in der Praxis angemessen zu sein.

Aus den durchgeführten Versuchen wird ersichtlich (Versuch 4), dass eine geringe Bildrate der Software ausreicht, um gute Ergebnisse zu erzielen. Die Software ist jedoch im momentanen Zustand eher für leistungsstarke Desktopcomputer ausgelegt, mit solchen erzielt die Software eine Bildwiederholungsrate von bis zu 35 Bilder pro Sekunde. Bei einer geringen Wiederholungsrate von zwei Bildern pro Sekunde müsste ein Objekt in einem Abstand von 30m eine Geschwindigkeit von mehr als 145 km/h haben, um unerkannt an der Kamera vorbeizukommen. Deshalb ist eine Bildwiederholungsrate von mindestens 15[46] Bildern pro Sekunde mehr als genug an Leistung. Trotz der Implementierung eines *Pathfinding*-Algorithmus ist die Software in der Lage diese Leistung zu erbringen, dies zeigt, dass die Software so konzipiert ist, dass sie es schafft, mehrere Aufgaben gleichzeitigt zu lösen[47].

[46] Benötigte Geschwindigkeit von >1992 km/h

[47] *Multiprocessing*

12 Januar 2024 - Verbesserung der Depth Estimation

Um die Objektabstandsdaten, die mit Hilfe von *Stereo Vision* erhalten werden weiter zu verfeinern, werden zwei Methoden angewendet: *Depth Queueing* und *Depth Validation.* Diese dienen dazu, die Ergebnisse der Depth Estimation mittels *Stereo Vision* robuster zu machen und zu validieren.

12.1 Depth Queueing

Als *Depth Queueing* wird hier der Prozess bezeichnet, bei dem aktuelle Abstandswerte mit früheren Abstandswerten verglichen werden und ein durchschnittlicher Abstand aus verschiedenen Depth Frames berechnet wird. Diese Methode hilft dabei Rauschen, kleine Ungenauigkeiten, aus den Objektabstandwerten zu entfernen. Das Resultat ist vertrauenswürdiger, da es auf mehreren Messungen aufbaut und so einzelne Ungenauigkeiten weniger ins Gewicht fallen.

Die Datenstruktur Queue wird verwendet um eine limitierte Anzahl (k=5) von Abstandswerten eines Objekts zu speichern. Ein neu erfasster Abstandwert wird zur Queue eines Objekts hinzugefügt, wenn er sich vom durchschnittlichen Abstandwert der Queue um 15% unterscheidet. Da nun mehr als 5 Abstandswerte vorhanden wären, wird der letzte Wert der Queue entfernt. Der neue Durchschnittswert der Queue wird berechnet.

Um neue und vergangenen Abstandswerte eines Objekts zuordnen zu können, muss ein Objekt zwischen den verschiedenen Erkennungszyklen des Yolo-Algorithmus als ein und dasselbe Objekt erkannt werden (Instanz Erkennung). Dafür wird die aktuelle und die vorherige *Bounding Box* verglichen und die *Intersection over Union* (IoU), also die Teilmenge der beiden Boxen, berechnet. Ist diese grösser als ein vorbestimmter Schwellenwert (0.3), handelt es sich um eine leicht verschobene Box des gleichen Objekts (dieselbe Instanz).

Für jede erkannte Instanz wird dann ein Objekt erstellt (Instanz der Klasse DepthQueueing), welches die Queue mit den Abstandswerten, die dem Objekt zu geordnet wurden, besitzt. Alle Objektinstanzen mit ihrer eigenen Objektabstandswert-Queue, werden in einer Queue aufgelistet (Instant der Klasse QueueManager), sodass immer alle durch den Yolo-Algorithmus erkannten Objekte ihre eigene Objekt-Instanz haben. Dies ist sehr vorteilhaft, da jedem Objekt separate Eigenschaften, wie die Objektgeschwindigkeit, Objekttyp, Richtungsvektor und weitere Informationen zu geordnet werden können. Zusätzlich können auch die in Kapitel 12.4 erwähnten Methoden und Eigenschaften angewendet werden (*Collision-Avoidance-System/depthqueueing.py*, 2024).

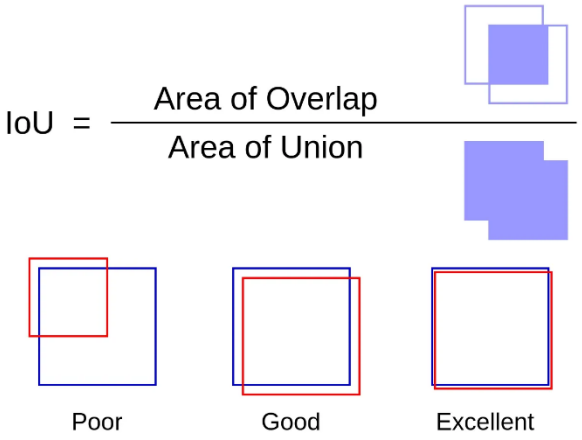


Abbildung 45 Intersection over Union (Tomar, 2023)

12.2 Implementierung von Depth Validation

Für die Durchführung von *Depth Validation* nehmen wir eine durchschnittliche Länge des Wasserfahrzeugs an, für welches der Abstand bestimmt werden muss. So sind wir in der Lage den Objektabstand zu approximieren und diese Approximation mit der Abstandsmessung der Stereokamera zu vergleichen und so ihre Genauigkeit zu prüfen.

Die Anwendung von *Depth Validation* mit Hilfe diese Methode, die nun vorgestellt wird, basiert auf zwei Beobachtungen. Die erste Beobachtung ist, dass der Aufbau von Booten, insbesondere Segelbooten, bestimmten Proportionen unterliegt. So steht zum Beispiel Mastlänge und Fahrzeuglänge in einem bestimmten Verhältnis. Dies gilt auch für das Verhältnis von Breite und Länge des Wasserfahrzeugs (siehe Kapitel 12.4). Die zweite Beobachtung ist, dass sich Wasserfahrzeuge je nach Gewässertyp in ihrer Grösse und Länge unterscheiden. Segelboote, die für küstennahes Segeln oder Segeln auf Binnengewässern konzipiert sind, sind meist kürzer gebaut. Segelboote für die Fahrt auf hoher See sind vergleichsweise länger. Man kann daher die Annahme machen, dass in unserem Beispiel des Vierwaldstättersees die Wahrscheinlichkeit grösser ist, Segelboote eines kleineren Typs anzutreffen. Diese Annahme wird aus eigener Erfahrung bestätigt. Aufgrund der genannten Beobachtungen und Annahme ist es möglich den Segelbooten des Vierwaldstättersees und Segelbooten auf Seen im Allgemeinen einen ungefähren Längenbereich zuzuordnen. Dieser liegt zwischen 9 bis 12 Metern (Life of Sailing, 2024).
Mit Hilfe von Einsicht in regionale Gewässerstatistiken, wie dem Wasserfahrzeugregister des Vierwaldstättersees, kann dieser Bereich verfeinert werden. Hier bei gilt jedoch: Je grösser der akzeptierte Längenbereich ist, desto grösser ist auch das Spektrum der erlaubten Objektabstandswerte und umgekehrt. Daraus folgt, dass wenn der Längenbereich erlaubter Wasserfahrzeuge zu weit gesetzt ist, die Validation ihre Funktion verliert, da Abweichungen in den erkannten Objektsabstandswerten von mehreren Metern nicht als fehlerhaft klassifiziert werden. Es ist also wichtig, den erlaubten Längenbereich der Wasserfahrzeuge möglichst eng zu setzten, damit bereits kleine Ungenauigkeiten der primären *Depth Estimation* Methode (*Stereo Vision*) erkannt werden.

12.3 Depth Validation Formel

Mit Hilfe der folgenden Schritte lässt sich der Abstand eines Objekts zur Kamera berechnen. Dabei machen wir uns trigonometrische Methoden und die bereits bekannten Eigenschaften unserer Stereokamera zu Nutze.

Mit Hilfe der folgenden Schritte lässt sich der Abstand eines Objekts zur Kamera berechnen, dafür benötigen wir diese Informationen:

- Auflösungsbreite der Kamera
- Sichtfeld der Kamera
- Pixellänge des Objekts, die es auf dem Bild der Kamera einnimmt
- Eine vermutete Länge des Objekts in Metern

1. **Berechnung der Winkelauflösung:**

   Die Winkelauflösung repräsentiert den von jedem Pixel am Kameraobjektiv unterbrochenen Winkel. Sie wird als Verhältnis der Auflösungsbreite der Kamera zu ihrem Sichtfeld berechnet. Je breiter das Sichtfeld, desto kleiner ist die Winkelauflösung, und umgekehrt.

$$\text{Winkelauflösung} = \frac{\text{Auflösungsbreite}}{\text{Sichtfeld}}$$

2. **Berechnung der Winkelgrösse in Grad:**

   Die Winkelgrösse des Objekts ist der Winkel, den es relativ zur Kamera einnimmt. Dieser Parameter wird berechnet, indem die Pixel-Länge des Objekts durch die Winkelauflösung dividiert wird. Dies gibt uns die Anzahl der Grad, die das Objekt im Sichtfeld der Kamera umfasst.

$$\text{Winkelgrösse (in Grad)} = \frac{\text{Pixel-Länge}}{\text{Winkelauflösung}}$$

3. **Berechnung der Entfernung zur Kamera:**

   Schliesslich verwenden wir Trigonometrie, um die Entfernung zur Kamera zu berechnen. Wir wissen, dass der Tangens der Hälfte der Winkelgrösse dem Verhältnis der halben Objektlänge zur Entfernung zur Kamera entspricht. Durch Umstellen der Formel können wir die Entfernung zur Kamera lösen:

$$\text{Entfernung zur Kamera} = \frac{\text{Objektlänge}}{2 \times \tan\left(\frac{\text{Winkelgrösse (in Grad)}}{2}\right)}$$

Indem wir diesen Schritten folgen, können wir die Entfernung zur Kamera basierend auf der Länge des Objekts, seiner Pixel-Länge, der Auflösungsbreite der Kamera und ihrem Sichtfeld bestimmen.

Abbildung 46 Depth Validation Formel (eigene Darstellung)

12.4 Proportionen eines Segelbootes – weitere nützliche Formeln

Dank *Stereo Vision* and *Depth Validation* ist man in der Lage, mit Hilfe von weiteren Formeln, weitere Informationen über ein Wasserfahrzeug (in diesem Fall Segelboot) herauszufinden. Dies ist möglich, da die meisten Wasserfahrzeuge, insbesondere Segelboote, genauen Proportionen und Bauverhältnissen unterliegen. So lässt sich bei beispielsweise die Breite des Schiffes anhand der Gesamtlänge berechnen, aber auch die Gesamtlänge mit Hilfe der Breite («Beam (Nautical)», 2023). Dies ist erforderlich, wenn ein Wasserfahrzeug in frontalem Kurs zur Stereokamera steht.

$$\text{Breite} = \text{Gesamtlänge}^{2/3} + 1$$

Abbildung 47 Breite und Gesamtlänge (eigene Darstellung)

Des Weiteren können verschiedenste weitere Längen und Proportionen ausgenutzt werden, um weitere Informationen über das Segelboot zu erhalten, die Kollisionsvermeidung unterstützen. Es kann die Rumpfgeschwindigkeit berechnet werden, also die maximale Geschwindigkeit, mit der ein Rumpf die Wellen durchschneiden kann, ohne über die Wellen zu Gleiten. Diese Geschwindigkeit stellt meist die Höchstgeschwindigkeit dar (Segellexikon, 2024). Die Wasserlänge, ein Teil der Formeln, wird in diesem Fall anhand der Gesamtlänge approximiert, da diese sich kaum von ihr unterscheidet. Die maximal mögliche Geschwindigkeit eines Objekts zu berechnen, stellt in der Kollisionsvermeidung einen grossen Vorteil dar, da nun berechnet werden kann wie viel Zeit verbleibt, um eine Kollision zu vermeiden und Gegenmassnahmen einzuleiten. So können Objekte mit einer höheren Maximalgeschwindigkeit in einer Gefahrensituation mit mehreren Objekten, in ihrer Umfahrung priorisiert werden und die Ausweichroute mit Hilfe des *Pathfinding*-Algorithmus dahingehend angepasst werden, dass ein höherer Mindestabstand bei der Umfahrung einzuhalten ist.

Die Gesamtlänge des Segelboots, die auch in anderen Formeln nebst der Rumpfgeschwindigkeit erscheint, kann ungefähr ermittelt werden. Dafür wird der Objektabstand der Stereokamera und die Umformung der *Depth Validation* Formel verwendet. Als Alternative kann auch eine ungefähre Länge wie bei der *Depth Validation* Methode angenommen werden (siehe Kapitel 12.2). Weitere Formeln beinhalten unter anderem die Masthöhe, Segelfläche und Segeleigenschaften (rustleryachts, 2021).

Durch die auf den folgenden Seiten aufgelisteten Formeln kann man herausfinden, um was für einen Typ von Segelboot es sich handelt und kann daher auf weitere Eigenschaften dieser Klasse von Segelbooten schliessen. Wenn zum Beispiel der Mast im Verhältnis zur Gesamtlänge des Bootes länger ist und das Segelboot daher auch eine grössere Segelfläche besitzt, lässt sich damit aussagen, dass mit einer hohen Wahrscheinlichkeit die Charakteristiken eines auf Geschwindigkeit optimierten Segelbootes zutreffen (Racing). Dieses kann schneller bei geringerem Wind segeln und somit auch eine höhere Geschwindigkeit erreichen (Life of Sailing, 2024). Die folgenden Verhältnisse überschneiden sich zwar teilweise in Ihren Aussagen auf Stabilität und Performance. Wenn jedoch, verschiedene Proportionsverhältnisse auf das Gleiche hindeuten ist die Wahrscheinlichkeit grösser, dass der Schluss auf weitere Eigenschaften zutrifft.

1. **Rumpfgeschwindigkeitsformel:**

$$V = 1.34 \times \sqrt{\text{Wasserlinienlänge}}$$

Beschreibung: Die Rumpfgeschwindigkeitsformel liefert eine Näherung der maximalen Geschwindigkeit, die ein Verdrängungsrumpf erreichen kann, bevor er anfängt zu gleiten. Sie basiert auf der Länge der Wasserlinie des Bootes.
Implikationen: Diese Formel hilft dabei, die theoretische maximale Geschwindigkeit zu verstehen, die ein Boot basierend auf seiner Länge erreichen kann. Sie ist besonders nützlich für Verdrängungsrümpfe, bei denen die Geschwindigkeit durch die Länge des Bootes und den Wellenerzeugungswiderstand begrenzt ist.

2. **Masthöhen und Gesamtlängenverhältnis:**

$$\text{Masthöhen und Gesamtlängenverhältnis} = \frac{\text{Masthöhe}}{\text{Gesamtlänge}}$$

Beschreibung: Diese Verhältnis vergleicht die Höhe des Masts mit der Gesamtlänge des Bootes.
Implikationen: Es liefert Einblicke in die Proportionen der Rigging des Bootes. Ein höheres Verhältnis deutet auf einen im Verhältnis zur Länge des Bootes höheren Mast hin, was sich auf die Segelfläche, die Stabilität und die Leistungseigenschaften auswirken kann.

3. **Breiten und Längenverhältnis:**

$$\text{Breiten und Längenverhältnis} = \frac{\text{Breite}}{\text{Länge}}$$

Beschreibung: Dieses Verhältnis vergleicht die Breite des Bootes mit seiner Länge.
Implikationen: Es zeigt das Verhältnis der Breite des Bootes zu seiner Länge an. Ein höheres Verhältnis deutet auf ein breiteres Boot im Verhältnis zu seiner Länge hin, was sich auf Stabilität, Innenraum und Seetüchtigkeit auswirken kann.

4. **Segelflächen und Wasserlinienlängenverhältnis:**

$$\text{Segelflächen und Wasserlinienlängenverhältnis} = \frac{\text{gesamte Segelfläche}}{\text{Wasserlinienlänge}}$$

Beschreibung: Dieses Verhältnis vergleicht die gesamte segelfläche des Bootes mit seiner Wasserlinienlänge.
Implikationen: Es hilft dabei, die Effizienz des Segelplans im Verhältnis zur Grösse des Bootes zu bewerten. Ein höheres Verhältnis deutet auf mehr Segelfläche pro Längeneinheit hin, was potenziell zu besserer Leistung bei leichten Winden führen kann.

Abbildung 48 Proportionen und Formeln 1 (eigene Darstellung)

5. **Segelseitenverhältnis:**

$$\text{Segelseitenverhältnis} = \frac{\text{Segelspannweite}}{\text{Segelhöhe}}$$

Beschreibung: Das Segelseitenverhältnis vergleicht die Breite des Segels mit seiner Höhe.
Implikationen: Es bietet Informationen über die Form und Effizienz des Segels. Ein höheres Seitenverhältnis deutet auf ein längeres und schmaleres Segel hin, was unter bestimmten Windbedingungen zu besserer Leistung führen kann.

6. **Segelflächen und Schiffsbreitenverhältnis:**

$$\text{Segelflächen und Schiffsbreitenverhältnis} = \frac{\text{gesamte Segelfläche}}{\text{Schiffsbreite}}$$

Beschreibung: Dieses Verhältnis vergleicht die Gesamtsegelfläche des Bootes mit seiner Breite.
Implikationen: Ein höheres Verhältnis deutet auf mehr Segelfläche im Verhältnis zur Breite des Bootes hin, was sich auf Stabilität, Segelhandling und Leistungseigenschaften auswirken kann.

Abbildung 49 Proportionen und Formeln 2 (eigene Darstellung)

13 März 2024 – Überblick und Fortschritt

Über ein Jahr nach der offiziellen Abgabe der Maturaarbeit ist es eine persönliche Ambition das Projekt weiterzuführen. Seit der Ideenfindung im Herbst 2021 und der darauffolgenden Abgabe im Oktober 2022 haben sich die für diese Arbeit relevante Technologien weiterentwickelt. Ziel dieses Kapitel ist es, die neue Ausrichtung des Projekts in Anpassung an diesen Entwicklungen zu beschreiben.

13.1 Ziel der neuen Ausrichtung

Ziel der neuen Ausrichtung ist es die Effizienz der Software weiter zu steigern, sodass sie mit einer höheren Bildwiederholungsrate auf Geräten mit geringerer Rechenleistung eingesetzt werden kann. Diese Geräte verfügen meistens über keine dedizierte GPU, weshalb die Voraussagen des Neuronalen Netzwerks mit Hilfe der CPU gemacht werden müssen. Dies birgt jedoch Performance Nachteile, da eine CPU nicht für solche Kalkulationen ausgelegt ist (siehe Kapitel 6.3.2). Aus diesem Grund ist ein möglichst geringer Ressourcenverbrauch der Applikation wichtig. Die Ausführung des Models und das Model selbst wird dementsprechend optimiert.

Des Weiteren wird das *User Interface* überarbeitet. Es wird das Webframework Flask verwendet um ein browserbasiertes, lokales Dashboard anzuzeigen. Im Vergleich zu *OpenCV* bietet Flask eine robustere Entwicklungsplattform und mehr Flexibilität in Design und Funktionalität.

Ein wichtiges neues Feature ist der Wechsel von relativen Positionen der erkannten Objekte zu absoluten Positionen, die mit Hilfe eines GPS-Moduls erhoben werden. Die erhaltenen Positionen der Objekte werden im Dashboard auf einer OpenStreetMap-Karte dargestellt (*OpenStreetMap*, 2023).

13.2 Trennung von *Front End* und *Back End* – Flask und ONNX Runtime

Die Nachfolger Version des CAS unterscheidet sich in ihrer Architektur grundlegend vom Vorgänger, das *Front End* und *Back End* werden getrennt. Als *Front End* wird hier der Teil der Software bezeichnet, der primär für das *User Interface*, also die visuelle Darstellung zuständig ist, während das *Back End* für die ressourcenintensiven Berechnungen, wie beispielsweise die Ausführung des Models und des *Pathfinding-Algorithmus* zuständig ist. *Flask,* als *lightweight Python Web Developement Framework,* empfängt die berechneten Werte (*Path*, *Bounding* Box und *Image*) und ist dann für deren Darstellung auf der Nutzeroberfläche verantwortlich (*Flask*, 2023). *Flask* ist streng genommen ein Backend Framework, welches in diesem Fall im weiteren Sinne für das *Front End* verantwortlich ist.

ONNX (Open Neural Network Exchange) ist ein offener Maschine Learning Formatstandard, der das *Deployment* von Modellen auf verschiedensten Plattformen ermöglicht. Das Yolo PyTorch-Model wird zu einem ONNX-Model konvertiert und kann dann auf kompatiblen *Runtime Engines* ausgeführt werden (*ONNX*, 2023). In unserem Fall wird das ONNX-Model mit *DeepSparse* eingesetzt. *DeepSparse* ist eine für CPU optimierte *Runtime Engine*, weshalb sie hier verwendet wird (Magic, 2021). *DeepSparse* bietet die Möglichkeit lokal eingesetzt zu werden und via API-Endpoints mit dem Model zu kommunizieren. Dies ermöglicht es unter anderem auch, das Model mit Hilfe von *DeepSparse* auf einer *AWS-Instanz* zu hosten. Die Folge ist eine Softwarearchitektur, die es ermöglicht das Back End, in die Cloud auszulagern. Dies bringt jedoch Konsequenzen in puncto Latenz mit sich.

Im Vergleich zur vorherigen Programmstruktur (siehe Kapitel 6.4.3), in welcher die Ausführung des Models und die anschliessende Darstellung der Ergebnisse (*Image* und *Bounding Box*) im gleichen Prozess stattfinden, sind diese Aufgaben in separate Prozesse getrennt. Es entfällt die gegenseitige Beeinflussung der Prozesse und das daraus resultierende *Bottleneck*. Man beachte, dass das *Pathfinding* in der neuen Programmstruktur, so wie zuvor, ein separater Prozess bleibt.

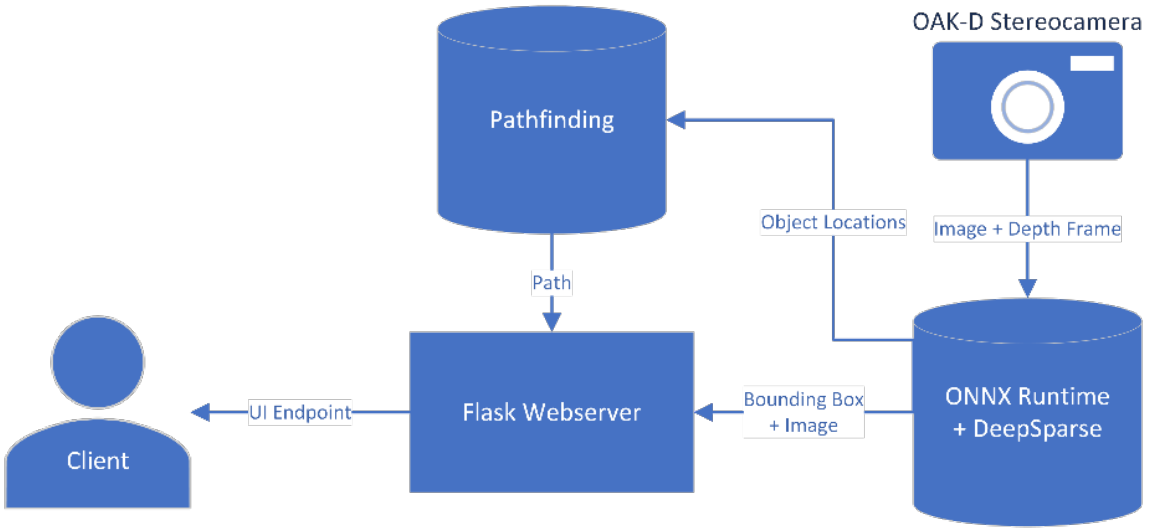


Abbildung 50 Programmarchitektur CAS Nachfolgerversion (eigene Darstellung)

13.3 Yolov8, Sparsification und Segmentation

Im Januar 2023 wurde Yolov8, der Nachfolger von Yolov5, von *Ultralytics* veröffentlicht (Bastian, 2023). Yolov8 weist im Vergleich zu Yolov5 einen deutlichen Sprung in Genauigkeit aber auch Geschwindigkeit auf. Die Versionen Yolov8n (nano) und Yolov8s (small) sind für unseren Anwendungszweck eher geeignet, da weniger Parameter eine geringere Inferenzzeit bedeuten. Wenn man die AP@0.5:0.95 der Nano-Versionen Yolov5n (28) und Yolov8n (37.3) vergleicht ist ein deutlicher Zuwachs zu erkennen (Jocher et al., 2023). Die Nano-Version Yolov8n liefert ähnliche Genauigkeit im Vergleich zur Small-Version Yolov5s, wobei sie jedoch über 3.2 Millionen Parameter verfügt, während Yolov5s über mehr als das doppelte an Parametern besitzt (7.3 Millionen).

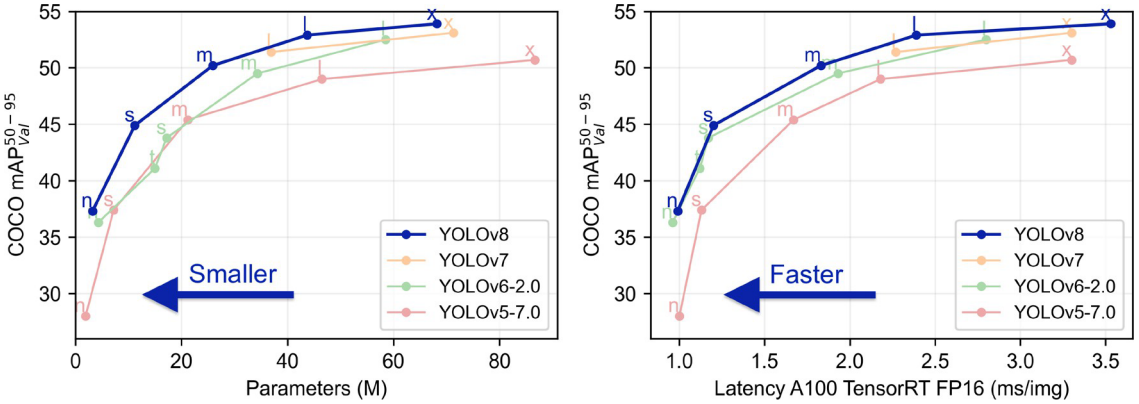


Abbildung 51 Yolov8 Modelle im Vergleich zu ihren Vorgängern (Jocher et al., 2023)

Um Yolov8 auf geringeren Ressourcenverbrauch zu optimieren wird ein Prozess namens *Sparsification* angewendet. *Sparsification* beschreibt das Vereinfachen eines Models, es werden redundante Verbindungen entfernt, wobei die Genauigkeit grösstenteils erhalten bleibt, das Resultat ist ein kleineres und effizienteres Model (*Neural Magic Docs*, 2023). In Kombination mit der CPU-Optimierung von *DeepSparse* kann ein *Sparse*-Model seine Bildwiederholungsrate erheblich steigern (Ultralytics, 2021).

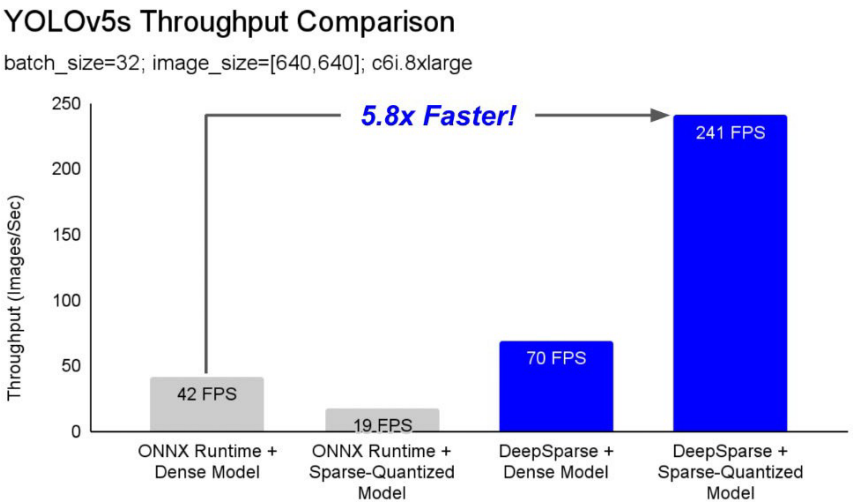


Abbildung 52 Sparsification und DeepSparse am Beispiel von Yolov5 (Ultralytics, 2021)

Wie in Kapitel 9.7.4 beschrieben ist Instanz-Segmentation eine geeignete Methode, um den Abstand eines erkannten Objekts genauer zu bestimmen. Dieses Feature wird mit Hilfe der Segmentations-Version von Yolov8 implementiert (Yolov8-seg). Diese Version verfügt über eine ähnliche Parameteranzahl im Vergleich zur Version ohne Segmentation und beeinflusst die Performance deshalb nicht schwerwiegend.

14 Referenzliste

Anand, M. (2022, Juli 17). What is Neural Network in ML? *Medium*. https://madhukaranand.medium.com/what-is-neural-network-in-ml-4912f8667b34

Andreas. (2022). *Python-pathfinding* [Python]. https://github.com/brean/python-pathfinding

Bandyopadhyay. (2022). *YOLO: Real-Time Object Detection Explained*. https://www.v7labs.com/blog/yolo-object-detection

Bastian, M. (2023, Januar 15). *YOLOv8 shows the enormous possibilities of computer vision*. THE DECODER. https://the-decoder.com/yolov8-shows-the-enormous-possibilities-of-computer-vision/

Bolya, D. (2018). *You Only Look At CoefficienTs* [Python]. https://github.com/dbolya/yolact

Bolya, D., Zhou, C., Xiao, F., & Lee, Y. J. (2019). *YOLACT: Real-time Instance Segmentation* (arXiv:1904.02689). arXiv. https://doi.org/10.48550/arXiv.1904.02689

Brownlee, J. (2022, August 9). Difference Between a Batch and an Epoch in a Neural Network. *Machine Learning Mastery*. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Class method vs Static method in Python. (2017, Mai 11). *GeeksforGeeks*. https://www.geeksforgeeks.org/class-method-vs-static-method-python/

COCO. (2022). *COCO - Common Objects in Context*. https://cocodataset.org/#explore

Desk, N. (2017, September 29). LizardTech awarded US patent for LiDAR point cloud compression. *Geospatial World*. https://www.geospatialworld.net/news/lizardtech-awarded-us-patent-lidar-point-cloud-compression-2/

Flask. (2023). GitHub. https://github.com/pallets/flask/tree/main/src/flask

Functional programming. (2022). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Functional_programming&oldid=1105315300

Guo, C., Li, C., Guo, J., Loy, C. C., Hou, J., Kwong, S., & Cong, R. (2020). *Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement*. https://doi.org/10.48550/arXiv.2001.06826

Jocher, G., Chaurasia, A., & Qiu, J. (2023). *YOLO by Ultralytics* (8.0.0) [Python]. https://github.com/ultralytics/ultralytics

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima* (arXiv:1609.04836). arXiv. https://doi.org/10.48550/arXiv.1609.04836

Kumar, P. (2022, Januar 20). *A short guide to why monochrome cameras have the edge over color cameras – e-con Systems*. https://www.e-consystems.com/blog/camera/technology/a-short-guide-to-why-monochrome-cameras-have-the-edge-over-color-cameras/

Lidar Mapping. (2022). VectorNav. https://www.vectornav.com/applications/lidar-mapping

Liu, Y. (2020, Mai 18). The Confusing Metrics of AP and mAP for Object Detection. *Medium*. https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef

Luber, S. (2022). *Was ist CUDA?* https://www.bigdata-insider.de/was-ist-cuda-a-851005/

luxonis. (2022b). *Depthai-experiments* [Python]. Luxonis. https://github.com/luxonis/depthai-experiments/blob/ca641bdf14c02127ec8a70b2eb4811f450e4f5cf/gen2-calc-spatials-on-host/calc.py

Luxonis. (2022a). *Depthai-experiments* [Python]. Luxonis. https://github.com/luxonis/depthai-experiments/blob/9e7d7c45a0659f90ddff5106bbbea5e84a017566/gen2-calc-spatials-on-host/main.py

luxonis. (2022). *Long-Range Depth Model—OAK-D-LR · Issue #247 · luxonis/depthai-hardware*. GitHub. https://github.com/luxonis/depthai-hardware/issues/247

Luxonis. (2022). *OAK-D — DepthAI Hardware Documentation*. https://docs.luxonis.com/projects/hardware/en/latest/pages/BW1098OAK.html#id3

Luxonis. (2022). *OAK-D-LR (Pre-Order)*. Luxonis. https://shop.luxonis.com/products/oak-d-lr-pre-order

luxonis. (2022). *Stereoscopy in repetitive-pattern environments*. GitHub. https://github.com/luxonis/depthai-python/issues/675

Luxonis. (2022). Google Docs. https://docs.google.com/document/d/1F4Y6S6KtZ4f8RBE4W-o9x6xVXbqsw8UIGWPkML-on1Y/edit?usp=embed_facebook

Magic, N. (2021). *DeepSparse* [Python]. https://github.com/neuralmagic/deepsparse

Martinez-Carranza. (2021). *Figure 1. OAK-D is a camera capable of running neural networks...* ResearchGate. https://www.researchgate.net/figure/OAK-D-is-a-camera-capable-of-running-neural-networks-simultaneously-while-providing-depth_fig1_356100571

Mössenböck, H. (2011). *Sprechen Sie Java? Eine Einführung in das systematische Programmieren* (4. Aufl.). dpunkt.verlag.

Mutschler, A. S. (2020, Oktober 8). *Sensor Fusion Challenges In Cars*. Semiconductor Engineering. https://semiengineering.com/sensor-fusion-challenges-in-cars/

Neural Magic Docs. (2023). Neural Magic Docs. https://docs.neuralmagic.com/user-guides/sparsification/

Objektorientierte Programmierung. (2022). In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Objektorientierte_Programmierung&oldid=225451293

ONNX. (2023). https://onnx.ai/

OpenStreetMap. (2023). OpenStreetMap. https://www.openstreetmap.org/

Programmierparadigma. (2022). In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Programmierparadigma&oldid=225818360#Deklarative_Programmierparadigmen

Programming paradigm. (2022). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Programming_paradigm&oldid=1109164972#Further_paradigms

Red Blob Games. (2022). *Introduction to A**. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Roboflow. (2022). https://roboflow.com/

ultralytics. (2022b). *Ultralytics/yolov5* [Python]. Ultralytics. https://github.com/ultralytics/yolov5

ultralytics. (2022c). *Ultralytics/yolov5* [Python]. Ultralytics. https://github.com/ultralytics/yolov5/blob/999482b45163c1b808a187b02183f324a9c782cb/train.py

ultralytics. (2022a). *Ultralytics/yolov5* [Python]. Ultralytics. https://github.com/ultralytics/yolov5

ultralytics. (2022a). *Ultralytics/yolov5* [Python]. Ultralytics. https://github.com/ultralytics/yolov5

ultralytics. (2022b). *Ultralytics/yolov5* [Python]. Ultralytics. https://github.com/ultralytics/yolov5

Ultralytics. (2021). *Neural Magic's DeepSparse*. https://docs.ultralytics.com/yolov5/tutorials/neural_magic_pruning_quantization

Washington. (2009). *Index of /courses/cse455/09wi/Lects*. https://courses.cs.washington.edu/courses/cse455/09wi/Lects/

What is the meaning of shared state and the side effects in functional programming? (o. J.). Quora. Abgerufen 17. September 2022, von https://www.quora.com/What-is-the-meaning-of-shared-state-and-the-side-effects-in-functional-programming

Yohanandan, S. (2020, Juni 9). *MAP (mean Average Precision) might confuse you!* Medium. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

Beam (nautical). (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Beam_(nautical)&oldid=1157274784

Collision-Avoidance-System/depthqueueing.py. (2024). https://github.com/Jojo942004/Collision-Avoidance-System/blob/main/depthqueueing.py

Life of Sailing. (2024). *Average Sailboat Length | Life of Sailing*. https://www.lifeofsailing.com/post/average-sailboat-length

Life of Sailing. (2024). *Average Sailboat Mast Height | Life of Sailing*. https://www.lifeofsailing.com/post/average-sailboat-mast-height

rustleryachts. (2021, Dezember 20). *How to choose a boat—Yacht measurements explained*. Rustler Yachts. https://www.rustleryachts.com/yacht-measurements-explained/

Segellexikon. (2024). *Segellexikon*. https://www.segeln-lernen.de/segellexikon-rumpfgeschwindigkeit.html

Tomar, N. (2023, Februar 7). What is Intersection over Union (IoU) in Object Detection? *Idiot Developer*. https://idiotdeveloper.com/what-is-intersection-over-union-iou/

15 Bestätigung

Diese Arbeit wurde mit bestem Wissen und Gewissen selbständig durchgeführt, wenn auf fremde Hilfe zurückgegriffen wurde, wurde sie explizit genannt. Ausschnitte aus wissenschaftlichen Quellen wurden korrekt mit Textzitaten ausgewiesen.

Dies bestätigt somit:

Johannes Zimmermann

Anhang I

Für weitere Informationen und Ressource (Präsentation, Sourcecode, Trainingsdaten, Modelle, Datensets, Versuchsdaten) siehe USB-Stick Anhang I oder https://github.com/Jojo942004/Collision-Avoidance-System

Anhang II

Programmierparadigmen

Ein Programmierparadigma beschreibt die Art und Weise, wie ein Code organisiert ist und wie ein Programm logisch aufgebaut ist. Programmierparadigmen können verwendet werden, um Programmiersprachen nach ihren Eigenschaften zu klassifizieren. Einer Programmiersprache können mehrere dieser Prinzipien zugrunde liegen. Diese Prinzipien sollen den Programmierer dazu bringen, Probleme in einer bestimmten Art zu lösen und ihn bei diesem Prozess unterstützen. Man unterscheidet hier in 2 Untergruppen nämlich der imperativen Programmierung und Deklarativen Programmierung. Bei der imperativen Programmierung gibt es Programmierer bestimmte Instruktionen, die beschreiben, in welcher Reihenfolge der Computer bestimmte Instruktionen ausführen muss. Bei der deklarativen Programmierung ist jedoch weniger das ”Wie“ im Vordergrund, sondern der Programmierer teilt mit, welches Ergebnis gewünscht ist, jedoch nicht, wie man zur Lösung kommt. Man könnte dieses Vorgehen, daher als abstraktes Vorgehen bezeichnen, weil nicht konkret definiert wird wie vorgegangen werden soll. («Programmierparadigma», 2022) («Programming Paradigm», 2022).

Objekt Orientierte Programmierung (OOP) und Funktionale Programmierung sind weitere für dieses Projekt relevante Prinzipien. Allan Kay, der als Vater der Objekt Orientierten Programmierung angesehen wird, beschreibt OOP wie folgt:

- Alles ist ein Objekt
- Objekte kommunizieren durch das Senden und Empfangen von Nachrichten (welche aus Objekten bestehen)
- Objekte haben ihren eigenen Speicher (strukturiert als Objekte)
- Jedes Objekt ist die Instanz einer Klasse (welche ein Objekt sein muss)
- Die Klasse beinhaltet das Verhalten aller ihrer Instanzen (in der Form von Objekten in einer Programmliste)
- Um eine Programmliste auszuführen, wird die Ausführungskontrolle dem ersten Objekt gegeben und das Verbleibende als dessen Nachricht behandelt («Objektorientierte Programmierung», 2022)

Ihm zufolge ist alles ein Objekt und hat daher einen ausschlaggebenden Vorteil im Vergleich zu funktionaler Programmierung. U. Funktionale Programmierung hat zwei Haupteigenschaften:

- Eine Funktion wird als mathematische deterministische Funktion behandelt. Wenn eine Funktion aufgerufen wird, liefert sie für die gleichen gegebenen *Arguments* immer das gleiche Ergebnis. Diese Prozedur wird als *pure* bezeichnet («Functional Programming», 2022).
- Es treten keine *Side-Effects* (Nebenwirkungen) auf.

Side-Effects und Shared State werden wie folgt beschrieben:

> In *pure* functional programming no state is modified, instead new values are created. A *side-effec*t is when this rule is violated and the program modifies a state variable or does some

> IO[48]. The idea is that a pure function is deterministic and if repeated with the same inputs will compute the same result. Functional languages vary in their purity. The most pure require that *IO* and *state*[49] are explicit in the function signatures.
>
> *Shared state* is state which is shared between more than one function or more than one data-structure. When the state is *immutable* (can't be changed) this is relatively harmless and is basically a memory saving mechanism. When the shared state is *mutable* (can be changed), it can be hard to reason about. If shared state is mutable and used simultaneously by multiple threads, then the program will need to use *locks* or other mechanisms to operate on the state. This makes for more complexity and less modularity. However, shared state maybe be required for certain applications, as all the alternatives are less performant (*What Is the Meaning of Shared State and the Side Effects in Functional Programming?*, o. J.).

Wie nun festgehalten, wird klar, dass OOP eine wichtige Eigenschaft im Vergleich

zu *Functional Programming* besitzt. Ein Objekt Orientiertes Programm ist in der Lage dynamisch die Eigenschaften eines Objekts anzupassen, da jedes Objekt die Fähigkeit hat Eigenschaften zu speichern. Auf diese Eigenschaft der Datenstruktur kann dann von verschiedenen weiteren Objekten zugegriffen werden. Durch die resultierende Verknüpfung kann eine Änderung dieser Eigenschaft kann unvorhergesehene Folgen haben.

Neben den oben genannten Programmierparadigmen existieren noch viele weitere Programmierparadigmen, die die Entwicklungsstruktur einer Programmiersprache beschreiben. Es ist wichtig festzuhalten, dass, wie das Zitat vom sogenannten Vater der OOP Alan Kay zeigt, die Definition von verschiedensten Paradigmen eher ein Spektrum ist, und sich mit der Zeit entwickeln kann. Diese Paradigmen sind nicht dazu da um, Programmiersprachen haargenau zu klassifizieren, sondern sie sollen dem Entwickler einen ungefähren Anhaltspunkt geben. Daher ist es offensichtlich, das verschiedenste Programmiersprachen auf der detailorientierten Ebene viele verschiedene Paradigmen beinhalten. Dies erschwert eine genauere Definition.

Instanz- und Statische Methoden

Es gibt zwei verschiedene Arten von Methoden in einer Klasse, diese sind folgende:

- *Static Method (und Class Method in Python)* *(statische Komponenten)*
- *Instance Method* *(objektbezogene Komponenten)*

Im Kontext zur zwingend Objekt Orientieren Programmiersprache Java und anderen OOP Sprachen wie C++ definiert (Mössenböck, 2011, S. 152) die beiden Methodentypen wie folgt:

> Um den Unterschied zu verstehen, müssen wir uns zunächst vor Augen führen, dass auch eine Klasse ein Objekt ist. Ein Klassenobjekt ist gewissermaßen eine Schablone, die das Aussehen aller Objekte dieser Klasse festlegt. Sowohl das Klassenobjekt als auch die Objekte, die von dieser Klasse erzeugt werden, können Felder und Methoden haben. (S. 152)

Da Python als Programmiersprache den Entwickler nicht dazu zwingt sein Programm Objekt-orientiert zu strukturieren, so wie beispielsweise Java, hat er die Wahl. Durch die simplere Syntax von Python kommt es dazu, dass Python viele Konzepte syntaktisch vereinfacht und anders als andere OOP-

[48] Input/Output
[49] Zustand

Sprachen. Ohne Kenntnis des eigentlichen Konzepts kann es daher schwierig sein nachzuvollziehen, welche Prozesse wirklich ablaufen.

Vergleich von Klassen- und Objekt Methoden – Ein Beispiel

Um Python Klassen- und Instanz Methoden zu vergleichen nehmen wir eine leicht modifizierte Java Implementierung aus (Mössenböck, 2011, S. 152) und implementieren diese in Python.

```python
class NotReallyRight:
    fractionCounter = 0

    def resetFractionCounter(self):
        Fraction.fractionCounter = 0

    def printFractionCounter(self):
        print(Fraction.fractionCounter)

    def __init__(self, int1, int2):
        self.int1 = int1
        self.int2 = int2
        Fraction.fractionCounter += 1

f = NotReallyRight(1,2)
g = NotReallyRight(1,2)
f.printFractionCounter()
g.printFractionCounter()
f.printFractionCounter()
g.printFractionCounter()
f.printFractionCounter()
```

```java
class Fraction {
    // ----- class field
    static int fractionCounter = 0;

    // ----- object fields
    int n = 1;
    int d = 1;

    // ----- class method
    static void resetFractionCounter () {
        fractionCounter = 0;
    }

    // ----- class method
    static void printFractionCounter () {
        System.out.println(fractionCounter)
    }

    // ----- constructor and object methods
    Fraction (int n, int d) {
        this.n = n; this.d = d;
        fractionCounter++;
    // new object created => increase counter
    }
    ...

}
```

Es fällt auf, dass die Python Klasse *NotReallyRight* deutlich kürzer ist als die Java-Klasse *Fraction. Fraction* besitzt zwei Klassenmethoden, eine setzt den *fractionCounter* zurück, die andere gibt den *fractionCounter* aus. *NotReallyRight* hat jedoch zwei Objektmethoden. Beide Klassen geben den gleichen Output (2 2 0 0), wenn zwei Instanzen mit den Objektattributen 1 und 2 erstellt[50] werden. In *NotReallyRight* werden Objektmethoden verwendet, um auf Klassenattribute zuzugreifen und diese zu verändern. Dies entspricht nicht dem OOP-Paradigma, denn Instanzen dürfen keinen Einfluss auf den State (Zustand) einer Klasse haben. In Python gibt zwei Klassenmethoden-Typen:

- "A static method is also a method that is bound to the class and not the object of the class. This method can't access or modify the class state." («Class Method vs Static Method in Python», 2017)

- "A class method is a method that is bound to the class and not the object of the class. […] They[51] have the access to the state of the class as it takes a class parameter that points to the class and not the object instance. It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that will be applicable to all the instances." («Class Method vs Static Method in Python», 2017)

[50] Siehe Code unterhalb Klasse NotReallyRight
[51] Klassenmethoden

Python lässt in *NotReallyRight* eine falsche Handhabung der Methoden zu, Java jedoch nicht, es zeigt sich ein zwei schneidiges Schwert: Die Flexibilität von Python. Eine korrektere Implementation sieht aus wie folgt.

```
class Fraction:
    #class attribute
    fractionCounter = 0

    @classmethod
    def resetFractionCounter(cls):
        cls.fractionCounter = 0

    @staticmethod
    def returnFractionCounter():
        print(Fraction.fractionCounter)

    def __init__(self, int1, int2):
        #instance attributes
        self.int1 = int1
        self.int2 = int2
        Fraction.fractionCounter += 1
    …

f = Fraction(1,2)
g = Fraction(1,2)
f.returnFractionCounter()
g.returnFractionCounter()
f.resetFractionCounter()
g.returnFractionCounter()
f.returnFractionCounter()
```