



Entwicklung eines portablen Abschreksystems gegen unliebsame Kleintiere

Studienarbeit

des Studiengang Technische Informatik- IT-Automotive
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Levin Müller

9. Juni 2023

Bearbeitungszeitraum
Matrikelnummer, Kurs
Ausbildungsfirma
Betreuer

zeitraum
7994341, TINF20-ITA
softwareinmotion GmbH, Schorndorf
Prof. Dr. Janko Dietzsch
Janko.Dietzsch@dhbw-stuttgart.de

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *Entwicklung eines portablen Abschrecksystems gegen unliebsame Kleintiere* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Welzheim, 9. Juni 2023

Levin Müller

Zusammenfassung

The thesis focuses on the development of a portable, efficient, and self-sufficient deterrent system against unwanted small animals. The work encompasses three main aspects: the construction of the self-sufficient system, the housing of the system, and the detection of the target animals.

The primary focus of the thesis lies in the real-time detection and localization of the animals. Object detection and stereo vision have been combined to achieve this goal. Various approaches have been tested for object detection, resulting in successful detection of unwanted small animals. The integration of object recognition and stereo vision also enables the recognition of animals in three-dimensional space. Based on this evaluation, a designed „water cannon“ targeting system can effectively deter animals. With the shock of getting confronted with deterrent System the animals should not come again.

However, there are limitations in achieving real-time detection due to the hardware constraints of the system.

Inhaltsverzeichnis

Abkürzungsverzeichnis	I
Abbildungsverzeichnis	II
Tabellenverzeichnis	III
1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung	2
1.3. Aufgabenstellung	2
2. Grundlagen	3
2.1. Stand der Technik in der Tiervertreibung	3
2.2. Computer Vision	6
2.3. Google Colab	8
2.4. Komponenten	9
3. Umsetzung	13
3.1. Hardwarerealisierung	13
3.2. Dreidimensionales Zielsystem	20
3.3. Ansteuerung	22
3.4. Objekterkennung und -Verarbeitung	26
3.5. Zusätzliche Softwarekomponenten	46
3.6. Kostenaufstellung	53
4. Reflexion und Ausblick	56
Literatur	58
Anhang	64
A. Arduino UNO Testcode	64

Abkürzungsverzeichnis

BB	Bounding-Box
CSI	Camera Serial Interface
CNN	Convolutional Neural Network
CM	Compute Module
DNN	Deep Neural Network
eMMC	Embedded Multi Media Card
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
IOU	Intersection over Union
mAP	mean Average Precision
ML	Machine Learning
NPU	Neural Processing Unit
PAP	Programmablaufplan
PWM	Pulsweitenmodulation
R-CNN	Region-based Convolutional Neural Network
RPi	Raspberry Pi
RPN	Region-Proposal-Network
SATA	Serial AT Attachment
SVM	Support-Vector-Machine
YOLO	You-Only-Look-Once

Abbildungsverzeichnis

1.1.	Verwüsteter Garten	1
2.1.	Ausgelöster Wassertierschreck. [28]	4
2.2.	Beispiele für Objekt Erkennung und Bounding-Box. Quelle: [46]	6
2.3.	(a) Epipolare Linie, die einen Lichtstrahl entspricht und (b die entsprechende epipolare Linien auf der Epipolarebene. Quelle: [46])	9
3.1.	Schaltbild der Abschreckleuchten mit Ansteuerung	14
3.2.	Eingebaute Wasserpumpe mit Wasserversorgung und Spritzschutz	16
3.3.	Diagramm der Fördermenge gegenüber des Drucks. [42]	22
3.4.	Schaltplan der Hardware	25
3.5.	Ordner-struktur des Projektes	27
3.6.	Einzeichnen einer Bounding Box mit dem Programm Label Studio	30
3.7.	Vergleich eines augmentierten und nicht augmentierten Bildes aus den Trainingsdaten (links [51]) und ein Bild das vom Abschrecksystem aufgenommen wurde	33
3.8.	PAP TensorFlow Nachtraining	35
3.9.	mean Average Precision (mAP) des MobileNet-Modells nach 50.000 Trainingsschritten	45
3.10.	mAP des YOLOv8-Modells nach 30 Epochen	45
3.11.	Hintergrundsubtraktion auf eine gegebene Eingabe. Quelle: [12]	47
3.12.	Anwendung zum herausfiltern der Bounding Boxen	49
3.13.	Makeraufnahme von IMX-219 (links) und OV5647 mit Infrarotfilter(rechts)	50
3.14.	Vergleich einer Szene, die ohne (links) und mit (rechts) GStreamer-Argumenten aufgenommen ist	51
3.15.	Aufbau aller Hardwarekomponenten innerhalb eines 3D-Modells	55

Tabellenverzeichnis

2.1. Vergleich dreier Elektromotortypen für die Verwendung im Zielsystem. Quelle: [24]	11
3.1. Vergleich verschiedener Objekterkennungsmodelle auf Basis ihres Formates und Inferenzzeit in Millisekunden	42
3.2. Vergleich des mAP-Wertes verschiedener YOLOv8-Quantifizierung und Modellformate	46
3.3. Kostenübersicht und Erklärung der Funktion der verschiedenen angeschafften Hardware	55

1. Einleitung

In diesem Kapitel sollen die Beweggründe, sowie die geplante Vorgehensweise zur Entwicklung eines portablen Abschrecksystems gegen unliebsame Kleintiere erläutert werden.

1.1. Motivation

Der Anstieg der Waschbärenpopulation in vielen Teilen Deutschlands hat zu Schäden in Gärten, Parks und Friedhöfen geführt. Auch der eigene Garten ist davon leider nicht unversehrt geblieben, was in Abbildung 1.1 zu sehen ist. Die üblichen Mittel wie Ultraschall- und Blitzlicht- Abschrecksystemen scheinen nur von sehr kurzer Dauer einen Erfolg zu bieten. Angesichts dessen ist die Entwicklung eines effektiven Abwehrsystems gegen Waschbären von entscheidender Bedeutung, um weitere Schäden zu minimieren.



Abbildung 1.1.: Ein durch Waschbären verwüsteter Garten

1.2. Zielsetzung

Das Ziel der Arbeit ist es, ein Abschreckssystem zur Fernhaltung von Mardern und Waschbären von privaten Grundstücken zu entwickeln. Im System soll eine Kombination von üblichen Abschreckungsmitteln aus den Baumarkt, einschließlich eines kleinen Wasserwerfers eingebaut sein. Darüber hinaus soll es aus Gründen der einfachen Anwendbarkeit portable an jeglichen Stellen im heimischen Garten aufgebaut werden können. Um zu verhindern das das Abschreckssystem unschuldige Passanten unter Beschuss nimmt, muss das System zwischen Mensch und Tier unterscheiden können.

1.3. Aufgabenstellung

Um die geplanten Ziele zu erfüllen, sind drei Kernelemente herauszuarbeiten. Zum einen soll das System portable sein, das heißt es muss sich selbst mit Energie für einen längeren Zeitraum versorgen können. Dazu bedeutet es die Wasserwerfer-Komponente auch mit Energie und Wasser versorgen zu können. Deshalb muss zusätzlich zur Energieversorgung ein Konzept für die Wasserversorgung erarbeitet werden.

Zum anderen darf es nicht zu sperrig werden, damit es problemlos überall im Garten platziert werden kann. Hier könnten Abstriche hingenommen werden, falls die restlichen Funktionalitäten wie die eigenständige Energieversorgung oder den Einbau eines „Wasserwerfers“ sonst nicht eingesetzt werden können.

Der Hauptteil der Arbeit liegt aber bei der Unterscheidung zwischen einem tierischen Eindringling und einem menschlichen Passanten. Da unschuldige Passanten nicht nass gespritzt werden sollen, soll das Abschreckssystem nur auf Kleintiere auslösen. Um dies zu realisieren, sollen die Kleintiere mittels Objekterkennung und einer Kamera erkannt werden. Da dies in Echtzeit geschehen soll, müssen der Erkennungs- und Zielerfassungsprozess so schnell wie möglich abgeschlossen sein. Für eine präzise Zielausrichtung des „Wasserwerfers“ wird eine Entfernungsmessung vom Ziel zum Zielsystem benötigt, wofür eine zweite Kamera zur Stereo-Vision verwendet wird. Durch die Verwendung der zweiten Kamera können Tiefeninformationen aus den 2D-Bildern der einzelnen Kameras gewonnen werden.

Basierend darauf soll ein Prototype entwickelt werden, der möglichst all die Voraussetzungen erfüllen kann. Dieser Prototyp soll dazu dienen die Funktionsweise des Abschreckssystems im konzeptionellen Anwendungsfall zu demonstrieren und zu veranschaulichen.

2. Grundlagen

2.1. Stand der Technik in der Tiervertreibung

Um unliebsame Besucher aus dem Garten, Haus oder Auto zu vertreiben gibt es viele Geräte auf dem Markt. Zu diesen gehört ein großes Sortiment von Ultraschall-Tierschreck-Systemen, Sprinkleranlagen und verschiedenen Varianten von Weidezäunen. Um einen bestmöglichen Erfolg der Vertreibung zu bieten, sollen die Geräte an den Umschlagsorten der Tiere platziert werden.

Die einzelnen Systeme und deren Vor- und Nachteile werden in den nachfolgenden Unterkapiteln beschrieben.

2.1.1. Ultraschall-Tierschreck

Eine gängige Variante des Ultraschall-Tierschrecks ist der Marderschreck. Der Marderschreck wird in dem Motorraum eines Fahrzeugs platziert und soll verhindern, dass der Marder Schläuche und Kabel durchbeißt. Er verspricht das Fernhalten und Vertreiben der Tiere durch aussenden eines Hochfrequenztons. Der Ton hat dabei eine Frequenz von 17 bis 45 kHz. Für die Tiere ist dieser Frequenzbereich besonders unangenehm. Erwachsene Menschen nehmen diese Töne aber kaum bis gar nicht wahr. [14]

Daher werden auch für den heimischen Garten diese Geräte gerne eingesetzt. Da sie aber nicht länger über die Autobatterie und der Lichtmaschine mit Energie versorgt werden, sind sie häufig an eine kleine Batterie und einem Solarpanel angeschlossen. Die gängigen Varianten eines Ultraschall-Tierschrecks für die Gartenverwendung haben zudem ein eingebautes Blitzlicht. Bei Nacht wird das Tier durch kurze Lichtimpulse zusätzlich beim Durchqueren des Gartens gestört und der Erfolg zur Vertreibung von nachtaktiven Tieren erhöht sich.

Die Tiere, insbesondere Waschbären, gewöhnen sich allerdings an das Licht und dem Hochfrequenzton. Daher hält der Erfolg der Vergrämung oft nur wenige Wochen an. Ultraschallgeräte können aber auch Probleme bereiten. Die eigenen Haustiere und auch Kleinkinder nehmen den Hochfrequenzton ebenso wahr. Da die Geräte auf jegliche Bewegung reagieren, kann es sein das der Ultraschall-Tierschreck vor Betreten des Gartens

deaktiviert werden muss, damit die Haustiere und Kinder sich im Garten aufhalten können.[41]

2.1.2. Automatische Sprinkleranlage

Eine andere Variante von Abschrecksystem ist der Einsatz von Sprinkleranlagen. Durch das Beschießen mit Wasser werden die Tiere besonders gut gestört. Der automatische Sprinkler wird über einen Bewegungsmelder ausgelöst und versprüht großflächig Wasser im Zielbereich, wie in Abbildung 2.1.2 zu sehen ist. Nach eigener Erfahrung hat eine automatische Sprinkleranlage eine höhere Erfolgsquote, um ungewollte Besucher aus dem heimischen Garten zu vertreiben, aber sie kann durch den Bewegungsmelder auch unbeabsichtigt von einem selbst ausgelöst werden.[41]



Abbildung 2.1.: Ausgelöster Wassertierschreck. [28]

Der Nachteil bei diesem System ist, dass der Sprinkler direkt mit einem Gartenschlauch verbunden werden muss. Dadurch treten deutliche Einschränkungen in der Positionierung des Abschrecksystems auf, da ein Wasseranschluss mit ausreichend Druck und Volumenstrom an ihm befestigt sein muss. Zusätzlich fällt der Druck und der Volumenstrom mit

zunehmender Länge des Gartenschlauches ab, was in den Artikel aus [29] getestet worden ist. Der Author hat in diesem Artikel zwei verschiedenen Gartenschläuche und deren Druck- und Volumenstromverlust gemessen. Bei dem Test des Gartenschlauches mit 1/2 Zoll Durchmesser kam es gegenüber dem mit 3/4 Zoll Durchmesser nach zwei Metern bereits eine Reduzierung des Volumenstroms auf 62%.

Angenommen ein eingesetzter Sprinkler hätte eine Düse mit einer Öffnung von 1,5 Millimeter in einer Höhe von einem Meter befestigt, so hätte die Reduzierung den Zielbereich von knappen 10 Metern auf 6 Meter reduziert. Ein Sprinkler hätte somit mit zunehmender Entfernung proportional zum Volumenstrom an Reichweite verloren.

Ein anderer Punkt, der bei diesen Anlagen häufig vernachlässigt wird, ist die Verschwendungen von Trinkwasser. Vor allem in Zeiten des Energie- und Wassersparens versucht man Verschwendungen zu minimieren. Einige Landkreise sind in den letzten Jahren in den Dürreperioden sogar so weit gegangen, dass das Rasensprengen aus eigener Quelle von 12 bis 18 Uhr verboten worden ist. Durch diese Maßnahmen erhofft man sich besser durch Dürreperioden zu kommen. Ein automatischer Rasensprinkler, der Trinkwasser verwendet sollte daher vermieden werden, um Dürreperioden nicht noch schlimmer zu machen.[30]

2.1.3. Weidezaun

Der Weidezaun ist häufig das letzte Mittel um die Tiere aus dem Garten zu bekommen. Der Zaun wird um den Garten herum aufgebaut und sämtliche Durchgänge, an denen die Tiere in den Garten eindringen können, sollen ebenfalls mit dem Zaun blockiert werden. Wenn das Getier nun versucht durch diese Zugänge in den Garten einzudringen, wird der Eindringling von dem Zaun einen schwachen elektrischen Schlag abbekommen.

Ein Waschbär wird durch diesen Schlag in absehbarer Zeit es nicht noch einmal versuchen denselben Zugang zu verwenden. Häufig suchen die Tiere stattdessen einen anderen Zugang in den Garten. Wenn die Tiere keinen anderen Zugang zum Garten finden, wenden sie sich vom Grundstück ab. In unregelmäßigen Abständen überprüfen die Tiere allerdings ob die Blockade immer noch besteht. Der Weidezaun muss daher ständig eingeschaltet und gewartet werden.[41]

Der Weidezaun hat von den genannten Systemen die höchste Effektivität, wenn es um die Vertreibung der Tiere geht. Durch die Blockierung sämtlicher Zugänge besteht allerdings ein sehr hoher Installationsaufwand. Zusätzlich müssen weitere Elemente, wie Gartentore installiert werden, da sonst auch die Zugänge für den Menschen blockiert sind. Das gesamte System wird somit schnell teuer, weshalb der Weidezaun auch als letztes Mittel betrachtet wird.

2.2. Computer Vision

Computer Vision beschäftigt sich damit Computern das „Sehen“ zu ermöglichen. Schon jahrzehntelang versuchen dies Wissenschaftler zu erreichen. Heutzutage sind sie schon sehr weit gekommen. Computer Vision findet in der Logistik, dem autonomen Fahren, Gesichtserkennung und bei noch vielen anderen Bereichen große Zuwendung. Ziel davon ist es Analysen, Verarbeitung und Interpretationen von Bildern und Videos durch einen Computer möglich zu machen, damit eine Maschinen visuelle Informationen so verstehen kann wie wir Menschen es tun. [46]

2.2.1. Object Detection

Object Detection ist eine Anwendung der Computer Vision, die es ermöglicht, Objekte in einem Bild oder Video zu erkennen und zu lokalisieren. Object Detection-Systeme verwenden Algorithmen, die auf Machine Learning (ML)- und Deep Neural Network (DNN)-Techniken basieren, um Objekte in Bildern oder Videos zu erkennen und zu klassifizieren. Das erkannte Objekt wird mittels einer Bounding-Box (BB) aus dem Bild extrahiert. Beispiele für Objekterkennungen sind in Abbildung 2.2.1 abgebildet.

Um diese Detektion zu erhalten gibt es verschiedene Basen und Architekturen von Object Detection Systemen, die im folgenden beschrieben sind. [46]

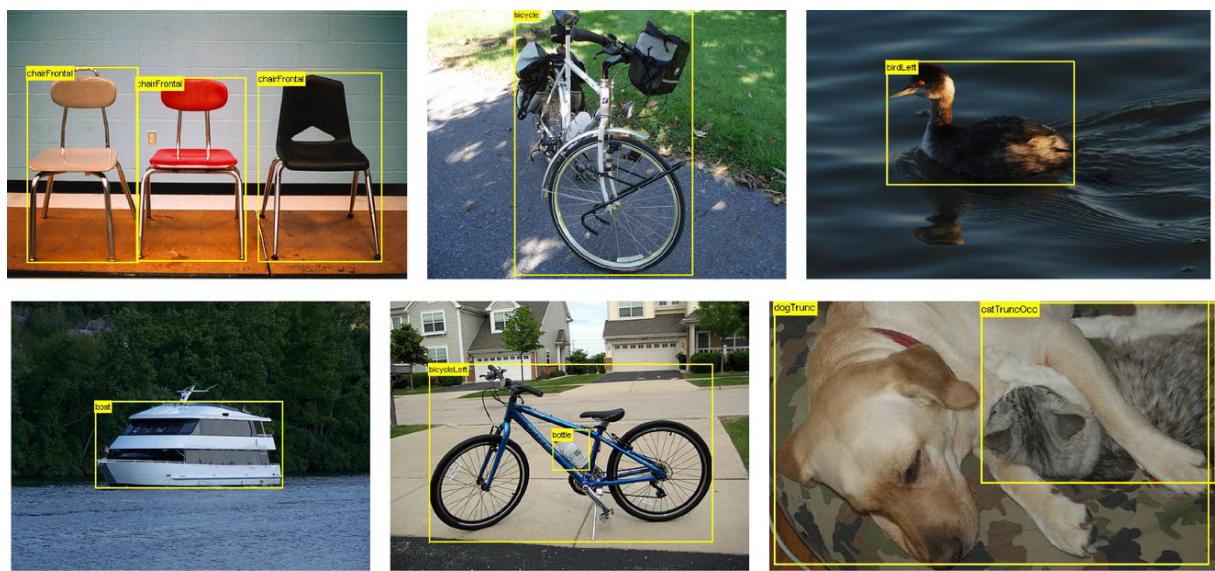


Abbildung 2.2.: Beispiele für Objekt Erkennung und Bounding-Box. Quelle: [46]

2.2.2. Two-Stage-Detectors

Eins der frühen Modelle die diese Technik verwendet hat ist das Region-based Convolutional Neural Network (R-CNN). Es werden hirfür zwei große Schritte angewendet. Weshalb das R-CNN und ähnliche Modelle als *two-stage-Detectors* bezeichnet werden.

Der erste Schritt bei diesen Modellen ist es mittels eines Region-Proposal-Network (RPN) rechteckige Regionen in einen Bild zu bestimmen. Das RPN schlägt hierbei eine Unterteilung des Eingabebildes in verschiedene Regionen vor. Das R-CNN-Netzwerk lässt sich dabei 2000 Regionen vorschlagen. Aus diesen Regionen werden anschließend im zweiten Schritt die Features eines Eingabebildes durch ein Convolutional Neural Network (CNN) extrahieren. Im Falle von R-CNN werden die Features mittels einer Softmax-Schicht und einer Support-Vector-Machine (SVM) ausgewertet und Objekte detektiert.

Dabei sollte die Anzahl der Regionen vernünftig klein sein, damit das nachfolgende extrahieren und klassifizieren in einer absehbarer Zeit erfolgt. Dieser zusätzliche Zeitaufwand, der für das RPN aufgewandt wird, ist zudem auch ein Nachteil gegenüber anderen Ansätzen. Die Two-Stage-Detektoren erhalten dadurch zwar eine höhere Genauigkeit in der Lokalisierung und Klassifizierung, benötigen aber auch mehr Zeit.[11]

2.2.3. One-Stage-Detectors

One-Stage Detektoren sind einer der neueren Ansätze Object-Detection zu realisieren. Sie basieren auf die menschliche Natur und gehen von einer Single-Shot Erkennung aus. Dafür wenden sie entgegen den Two-Stage-Detektoren keinen Region-basierten Algorithmus an. Stattdessen unterteilen sie ein Eingabebild in ein $S \times S$ großes Gitternetz. Auf den Gitter-Boxen wird daraufhin eine Klassifizierung und Lokalisierung vorgenommen.

Dies war auch der erste Ansatz in der You-Only-Look-Once (YOLO)-Architektur. Durch diesen Ansatz ist das YOLO-V1 Modell entgegen den Two-Stage-Detektor Faster R-CNN laut den Autoren in [2] neun mal schneller bei der Detektierung. YOLO-V1 ist aber deutlich schlechter, da deutlich weniger Objekte richtig detektiert worden sind. In Version 2 wird daher statt einem Gitternetz die Anchor-Boxen verwendet. Anchor-Boxen haben keine einheitliche Größe. Sie sind rechteckige Boxen, welche in verschiedenen Größen und Seitenverhältnissen angewendet werden. Die Anchor-Boxen werden auf einer Reihe von vordefinierten Positionen über ein Eingabebild verteilt.

Bei diesem Ansatz wird die Detektierung durch die Anchor-Boxen ermöglicht, ohne einen Verlust der Geschwindigkeit zu verursachen. Die Version 2 von YOLO enthält noch andere Anpassungen, die die Anzahl an richtigen Detektionen weiter erhöht hat. [2]

Es gibt noch weitere nicht Anchor oder Gitter betriebene Architekturen, wie die *CenterNet* oder *CornerNet*-Architektur. Diese ermitteln den Mittelpunkt eines Objektes oder deren

Kanten. Sie erhöhen die Präzession eines Detektors, benötigen aber wiederum mehr Ausführungszeit.[49]

2.2.4. Tiefenberechnung

Für das Zielsystem ist es nötig den Abstand vom Objekt zur Zielvorrichtung zu ermitteln. Heutzutage gibt es ML-basierte Techniken, die die Abstandsermittlung mit jeglicher Kamera durchzuführen können, aber es gibt einfachere und ältere Methode die Abstand zu einem Objekt zu bestimmen.

Die meisten Lebewesen und wir Menschen haben eine einfache Möglichkeit dreidimensionale Strukturen zu erfassen und wahrzunehmen. Dabei setzen wir Menschen auf unsere zwei Augen. *Stereo Vision* umfasst sich mit dem Bereich das binokularen Sehen von uns Menschen, um auch Computern es zu ermöglichen dreidimensional zu sehen.

Dabei werden zwei Kameras verwendet, die gleichzeitig ein *linkes* und ein *rechtes* Bild aufnehmen. Die Kameras stehen dabei um eine kleine Abstand auch genannt *Baseline* voneinander entfernt.

Der Prozess zur Bestimmung der Tiefe erfolgt dann durch die Verwendung der epipolaren Geometry und Triangulation. Dabei betrachtet man einen Punkt im *linken* Bild und den entsprechenden Punkt im *rechten* Bild. Da die Kameras auf einer Höhe zueinander stehen, muss der Punkt nur auf der x-Achse, entlang der epipolaren Linie der beiden Kameras gesucht werden. Das ermitteln der passenden Punkte wird dabei als *stereo matching* bezeichnet. Aus der *Baseline* und den Differenzpunkten zwischen den Koordinaten der einzelnen Bildern kann anschließend die Tiefeninformationen gewonnen werden. Das beschreiben Konzept ist in Abbildung 2.2.4 zu sehen.

Bei der Methode wird das Triangulieren zwischen den Punkten verwendet. [31]

2.3. Google Colab

Google Colab ist eine webbasierte Plattform von Google, die es ermöglicht, Python-Code in der Cloud auszuführen. Mit Colab können Jupyter-Notebooks erstellt, bearbeitet und ausgeführt werden.

Ein großer Vorteil der Nutzung von Google Colab ist, dass eine *Tesla T4* Grafikkarte kostenlos zur Verfügung steht. Dadurch können auch ML-Aufgaben effizient durchgeführt werden. Ein weiterer Vorteil von Google Colab ist, dass keine aufwendige Konfiguration erforderlich ist. Gängige Bibliotheken wie z.B. *TensorFlow* können ohne großen Aufwand installiert werden. Zudem werden einige Bibliotheken im Cache gespeichert, was bedeutet, dass eine erneute Installation nur wenige Minuten dauert. Dies macht Google Colab

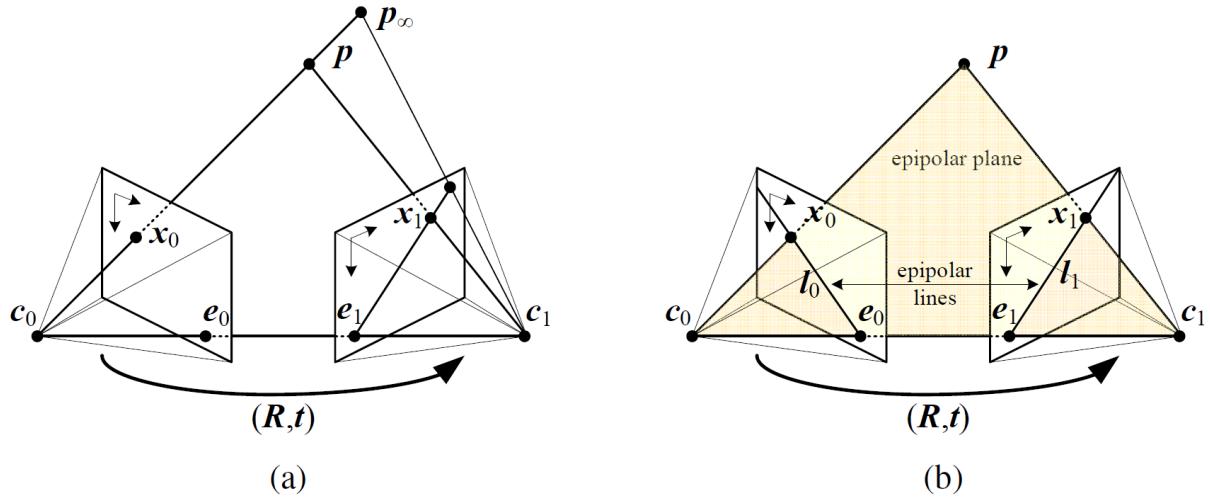


Abbildung 2.3.: (a) Epipolare Linie, die einen Lichtstrahl entspricht und (b) die entsprechende epipolare Linien auf der Epipolarebene. Quelle: [46])

besonders attraktiv.

Ein weiterer Vorteil von Google Colab ist die Möglichkeit, Notebooks über einen Link mit anderen Nutzern zu teilen. Dadurch wird ein schneller und einfacher Einstieg ermöglicht. [15]

2.4. Komponenten

Für die Realisierung eines portablen Abschrecksystems werden Aktoren und Komponenten benötigt, welche in diesem Kapitel beschrieben werden.

2.4.1. Aktoren

Das Abschrecksystem umfasst verschiedene Aktoren, die zur Vertreibung unliebsamer Kleintiere eingesetzt werden. Im Folgenden sind die Funktionen und Anforderungen der Aktoren sowie gegebenenfalls benötigte Zusatzelemente aufgelistet.

Tonwiedergabe

Waschbären, Marder und andere Tiere empfinden Hochfrequenztöne als äußerst unangenehm. Laut Westfalia [55] können Frequenzen im Bereich von 20 bis 40 kHz und ein Schalldruck von mindestens 100 dB bereits erfolgreich dazu beitragen, Waschbären zu vertreiben.

Gemäß den Informationen aus [22] entspricht ein Watt leistungsaufnahme bei einem Meter Abstand vom Lautsprecher in etwa einem Schalldruck von 90 dB. Da die meisten Mikrocontroller nicht genug Leistung liefern können, um einen Lautsprecher mit höheren Leistungsanforderungen anzusteuern, wird ein Verstärker benötigt. Zusätzlich ist es erforderlich, dass das Abschrecksystem einen möglichst großen Radius abdecken kann, wodurch ein höheres Leistungsvermögen als die angegebenen 100 dB erforderlich wird.

Lichtimpulse

Zusätzlich ist es sinnvoll Blinklichter zur visuellen Abschreckung einzusetzen. Da die meisten unliebsamen Tiere nachtaktiv sind, kann durch den stroboskopischen Effekt eine Vertreibung des Tieres erreicht werden. [41]

Auch hier können Mikrocontroller nicht das nötige Leistungsvermögen erbringen. Daher wird eine externe Energieversorgung und Schaltlogik benötigt.

Wasserversorgung

Wie in Kapitel 2.1.2 beschrieben, haben sich Sprinkleranlagen als wirksam bei der Vertreibung von Tieren erwiesen. Da das System jedoch tragbar sein soll, ist es nicht möglich, einen Wasserschlauch anzuschließen. Zusätzlich muss der Wasserdruck ausreichend sein, um die Tiere in einem Radius von 10 Metern zu erfassen.

Daher ist es erforderlich, eine Pumpe in das System einzubauen, die sowohl ausreichenden Druck als auch ausreichendes Pumpvermögen bietet. Die Pumpe muss ebenfalls von einer externen Stromversorgung und Schaltlogik gesteuert werden, da ein gezielter und leistungsstarker Einsatz erforderlich ist.

Zielaktorik

Für die Zielaktorik ist ein Zwei-Aktoren Zielsystem notwendig, welches horizontales und vertikales ansteuern von Zielwinkel ermöglicht. Elektromotoren eignen sich sehr gut für diese feinfühlige Ansteuerung. Allerdings gibt es große Unterschiede zwischen den verschiedenen Motortypen. Einige davon sind in folgender Tabelle 2.1 beschrieben.

Hinzu kommt das die Motoren in ein dreidimensionales Zielsystem eingebaut werden. Daher sollte die Form und Größe ebenfalls betrachtet werden, da der Platz gering ist.

Motortyp	Vorteile	Nachteile
Schrittmotor	<p>Einfache und präzise Ansteuerung möglich. Sie werden daher auch bei CNC-Fräsen und 3D-Druckern verwendet.</p> <p>Besitzen ein hohes Haltemoment.</p> <p>Berechnung der Position auch in der Open-Loop Steuerung möglich, da sie eine exakte Strecke pro Ansteuerungsschritt ermöglichen.</p>	<p>Da sie einen inkrementelle Positionierung über die Schrittweite haben, müssen sie eventuell mit einem Getriebe unterstellt werden. Zum Beispiel würde bei einer Schrittweite von $1,8^\circ$ beim Radius von 10 Metern pro Schritt 35 Zentimeter zurückgelegt werden.</p> <p>Des Weiteren kann es bei Zielvorgängen zu kurzzeitigen hohen Drehzahlen kommen. Der Schrittmotor verliert bei hohen Drehzahlen an Drehmoment und kann einzelne Schritte auslassen oder sogar stoppen.</p>
DC-Motor	<p>Einfache Ansteuerung über Pulsweitenmodulation (PWM).</p> <p>Hohes Drehmoment bei niedriger Drehzahl.</p>	<p>H-Brücke nötig, da die Energie nicht über einen herkömmlichen Mikrocontroller geliefert werden kann.</p> <p>Motorsteuerung und Regelung notwendig.</p> <p>Sensoren werden benötigt, da eine Open-Loop Ansteuerung nur sehr geringe Genauigkeit bietet.</p>
Servomotor	<p>Keine direkte Regelung nötig, da sie eine integrierte Closed-Loop Regelung haben.</p> <p>Sie können dynamisch auf hohe Belastungen reagieren.</p> <p>Durch die Closed-Loop Regelung haben sie eine hohe Genauigkeit und es sind keine zusätzlichen Bauelemente oder Treiberbauteile nötig.</p>	<p>Häufig haben die erhältlichen Servomotoren nur einen geringen Positionsradius von 180°.</p> <p>Da sie konstant versuchen, eine Sollposition zu erreichen, kann es zu Jitter im System kommen.</p>

Tabelle 2.1.: Vergleich dreier Elektromotortypen für die Verwendung im Zielsystem. Quelle: [24]

2.4.2. Sensorik

Der Aufgabenstellung entsprechend benötigt das System lediglich zwei optischen Sensoren, die in der Lage sind Bilder auch in der Nacht aufzunehmen.

Je nach Wahl der Aktoren können weitere Sensoren, wie Winkel- oder Stromsensoren im System verbaut sein. Da ein leistungsfähiger Mikrocontroller für die Object Detection verwendet werden soll, ist ein Temperatursensor für die Überwachung und zum Schutz vor Überhitzung zu empfehlen.

2.4.3. 3D-Druck

Eine Realisierung eines vollständigen Prototyps des Abschrecksystems benötigt viel Zeit und Teile, die auf dem Markt nicht immer erhältlich, nicht auf den Anwendungsgebiet zugeschnitten oder sehr teuer sind.

Daher werden für das Abschrecksystem Einzelteile mittels CAD-Modellen und dem 3D-Drucker der DHBW hergestellt. Das ermöglicht eine schnelle auf dem Anwendungsgebiet zugeschnittene Herstellung von Einzelteilen zu günstigen Kosten.

Zudem hat die Firma softwareinmotion einen eigenen 3D-Drucker, der in den Praxisphasen für die Studienarbeit verwendet werden kann. Der Weg zur DHBW für das Abholen der 3D-Drucke wäre in den Praxisphasen nur schwer möglich gewesen.

Die CAD-Modelle werden mit der Autodesk Software *Fusion 360* erstellt.

3. Umsetzung

3.1. Hardwarerealisierung

Für einen optimalen Vertreibungseffekt werden verschiedene Aktoren im System verbaut. Die einzelnen Geräte werden alle über eine externe Energieversorgung, beziehungsweise Batterie betrieben. Bei der Wahl der Batterie, ist ein wichtiges Kriterium, wie lange das System ohne Energiezufuhr funktionsfähig bleibt.

Dabei wurde berücksichtigt, wie viel das System im Normalverbrauch ohne Einschalten der Aktoren verbraucht. Eine grobe Schätzung ergab, dass das System ungefähr 10 ± 2 Watt pro Stunde verbrauchen würde. Aus diesem Grund fiel die Wahl auf eine 50 Ah Autobatterie von *BlackMax* [3] gefallen. Diese Batterie bietet ausreichend Kapazität, um das System über zwei Tage mit Energie zu versorgen.

Ein weiterer Entscheidungsgrund für die Autobatterie ist der Strombedarf der Geräte. Wie nachfolgend beschrieben, kann die Wasserpumpe im laufenden Zustand bis zu 16 Ampere Strom beziehen. Viele leichte und kleine Batterien könnten langfristig durch diese Belastung Schaden nehmen. Da Autobatterien eine hohe Belastungsgrenze haben und hohe Ströme ermöglichen, sind sie daher für den Prototypen besser geeignet.

Für die Unterbringung aller Teile wird eine 40x55x30 Compute Module (CM) große Aluminiumkiste verwendet. Die Aluminiumkiste ist witterungsbeständig, und die Wärmeentwicklung im Inneren kann weitestgehend vernachlässigt werden. Aluminium hat nämlich eine gute Temperaturleitfähigkeit und kann somit einen Wärmestau verhindern. Dabei ist auch der Empfehlung aus einem *Autodesk Instructables*-Projekt folge geleistet worden. In diesem Projekt wurde eine bewegungsausgelöste „Wasserpistole“ aufgebaut. Allerdings war bei dem *Instructables*-Projekt eine Vertreibung von Rehen vorgesehen, die sonst die Rosen im Garten aufgegessen hätten. Das Ziel dieser Arbeit ist es aber Waschbären und andere Kleintiere zu vertreiben, daher sind hier noch weitere Aktoren eingebaut. Zudem ist das *Instructables*-Projekt nicht autark und portable, da es mit Strom und Wasser mittels Hausanschluss versorgt worden ist. [16]

Die eingebauten Bauteile werden nachfolgend beschrieben.

3.1.1. Blitzlicht

Für den Einsatz eines Abschrecklichtes mit Blitzlichtfunktion werden zwei LED-Scheinwerfer der Marke *NAIZY* verwendet. Die Scheinwerfer sind für den Einsatz als Erweiterungsleuchten für Geländefahrzeuge gedacht. Sie sind wasserdicht und für den Außeneinsatz geeignet. Mit 1600 Lumen und grellweißem Farbton erzielen sie einen hohen stroboskopischen Effekt, mit dem die Tiere abgeschreckt werden. Aufgrund dieser Charakteristiken und des geringen Energieverbrauchs von 18 Watt sind sie für das Abschrecksystem sehr gut geeignet. [8]

Die beiden LED-Scheinwerfer sind an den beiden oberen Ecken der Box befestigt. Sie werden mit Transistoren durch den Mikrocontroller gesteuert. Um einen Gewöhnungseffekt zu vermeiden, sind die Scheinwerfer mit 0,5 bis 10 Hz beschaltet. Die Frequenz und Schaltung der Scheinwerfer wird dynamisch angepasst. Im Schaltbild aus Abbildung 3.1.1 wird die Verwendung eines MOSFET-Treiberbausteins erkennbar. Der Treiber ermöglicht das Beschalten der Scheinwerfer mit höherer Leistung, als es mit dem Arduino möglich wäre. Der Treiber verfügt über eine zusätzliche Masse-Leitung, die mit dem Arduino verbunden ist. Dadurch ist es möglich den Strom zu schalten, auch wenn der Signalgeber und *DC-Out* nicht die selbe Masse haben. [6]

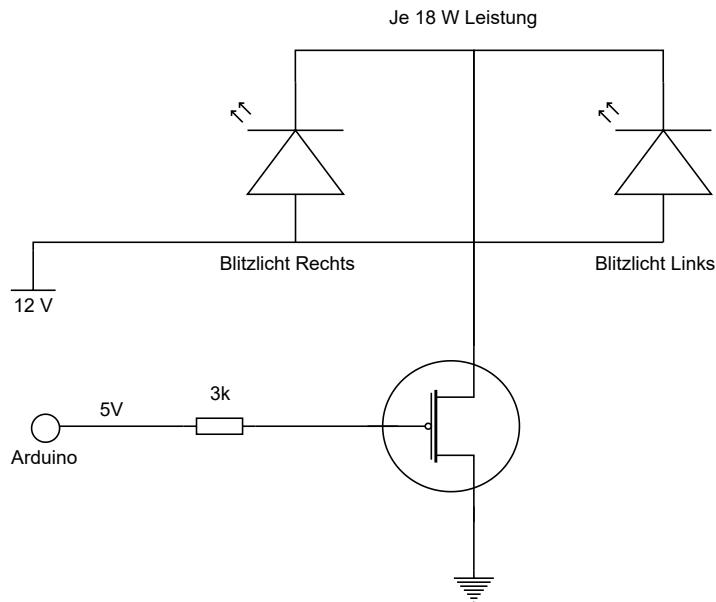


Abbildung 3.1.: Schaltbild der Abschreckleuchten mit Ansteuerung

3.1.2. Wasserversorgung und Pumpeneinheit

Für den Einsatz eines *Abschusssystems* mit Wasser benötigt das Abschrecksystem weitere Komponenten, welche folgend beschrieben werden.

Pumpe

Auch wenn die Aluminiumkiste witterungsbeständig ist, schützt sie nur mäßig vor kalten Temperaturen. Dabei können viele Wasserpumpen schaden nehmen, wenn sie über den Winter draußen sind. Bei Membranpumpen ist dies ein kleineres Problem. Die zu den Oszillationsverdrängerpumpen gehörende Pumpenart ist außerdem sehr wartungsfreundlich und außerordentlich robust. Sie können daher selbst schwierigen Bedingungen und Temperaturumschwünge leicht standhalten.

Ihre Funktionsweise beruht darauf, den *Schöpfraum* periodisch zu vergrößern und zu verkleinern. Dadurch erzielen sie ihre Pumpwirkung und werden durch ihre Robustheit, Ölfreiheit (keine Verschmutzung der zu fördernde Flüssigkeit oder Gase) und Wartungsfreundlichkeit auch in chemischen Laboranwendungen verwendet. [27]

Für die Auswahl einer geeigneten Pumpe kamen zudem die Anforderungen an Fördermenge und Druck. Sprinkleranlagen werden wie in Kapitel 2.1.2 beschrieben mit einem Garten-schlauch betrieben und können mit dieser Versorgung eine Reichweite von 10 Metern erreichen. Durch einen Hausanschluss fließen üblicherweise 20 Liter Wasser pro Minute mit einem Druck von etwa 4 Bar.

Die eingesetzte Pumpe von *SEAFLO* hat eine maximale Fördermenge von 17 Liter pro Minute und kommt dadurch nahe an diesen Richtwert ran. Dabei nimmt sie bis zu 16 Ampere Strom auf. Auch bei dieser Leistungsaufnahme von über 190 Watt können die MOSFET-Treiber, die zum An- und Ausschalten der LED-Scheinwerfer verwendet werden, genutzt werden. Sie sind angegeben mit einer Dauerbelastung von 15 Ampere und mit Kühlung bis zu 30 Ampere. Da die Treiber nur für kurze Dauer eingeschaltet werden, ist eine zusätzliche Kühlung nicht nötig. [6, 42]

Der Aufbau ist ähnlich zu dem Schaltbild 3.1.1. Allerdings wird zusätzlich eine Schutzdiode benötigt. Da Pumpen durch den internen Betrieb mittels Elektromotor eine induktive Last bilden, kann bei hartem Ein- und Ausschalten der MOSFET-Treiber durch Spannungsstöße Schaden nehmen [45]. Daher wird eine Schottky-Diode parallel zur Last eingebaut. Dies verhindert ein Aufbauen zu hoher Spannung, durch der der Transistor Schaden nehmen könnte. Schottky-Dioden eignen sich besonders gut für den Einsatz, da sie nur wenig Leistung aufnehmen und dadurch energiesparend sind. [5, 45]

Versorgung mit Wasser

Damit die Pumpe mit Wasser versorgt werden kann, muss ein Wassertank in das System integriert sein. Allerdings ergeben sich dadurch verschiedene Probleme.

Das größte Problem, das gelöst werden muss, ist die Unterbringung des Wassertanks. Durch den Einbau der verschiedenen Aktoren sowie der Autobatterie wurde in der Aluminiumkiste

3. Umsetzung

bereits viel Platz verwendet. Ein speziell an den verfügbaren Platz angepasster Tank kann zudem nicht mittels 3D-Drucker hergestellt werden, da die Drucke nicht wasserdicht sind. Ein herkömmlicher und einbaubarer Wassertank könnte daher nur wenige Liter fassen.

Ein weiteres Problem betrifft die Dichtigkeit des Tanks und der Pumpe. Beim Transport der Aluminiumkiste könnte eine große Menge Wasser leicht austreten. Aufgrund der Leitfähigkeit des Wassers könnten die Elektronikkomponenten Schaden nehmen. Dies wäre jedoch nicht so gravierend, da die Elektronik lediglich durchbrennen und der Stromkreislauf unterbrochen würde. Allerdings besteht bei der Aluminiumkiste die Gefahr, dass sie vollständig unter Strom gestellt wird. Dadurch würde sogar eine potenzielle Lebensgefahr entstehen, wenn man die Kiste berührt.

Um die Sicherheit zu erhöhen, wird der Wassertank deshalb außerhalb der Kiste platziert und der Zulauf erfolgt durch eine seitliche Öffnung an der Kiste. Die Pumpe befindet sich jedoch weiterhin innerhalb der Kiste, um sie vor den äußeren Witterungsbedingungen zu schützen. Sie wird durch einen gedruckten Spritzschutz räumlich von den anderen Aktoren und der Spannungsversorgung getrennt. Der 3D-Druck ermöglicht jedoch keinen vollständigen Schutz, da er nur gegen geringe Wassermengen dicht hält. Spritzwasser und kleinere Leckagen durch die Pumpe können jedoch bewältigt werden. Die eingebaute Pumpe mit Zulauf und Spritzschutz ist in Abbildung 3.1.2 zu sehen. Dabei wird vorerst eine Gießkanne als Ersatz für einen Wassertank verwendet.

Vor dem Einbau in die Kiste wurde ein Dichtigkeitstest der Pumpe und des Spritzschutzes durchgeführt. Dabei traten nur bei bestimmten Bedingungen, wie dem seitlichen Legen oder Schütteln der Pumpe, geringe Mengen Wasser aus der Pumpe aus. Daher sollte der gedruckte Spritzschutz einen ausreichenden Schutz bieten.

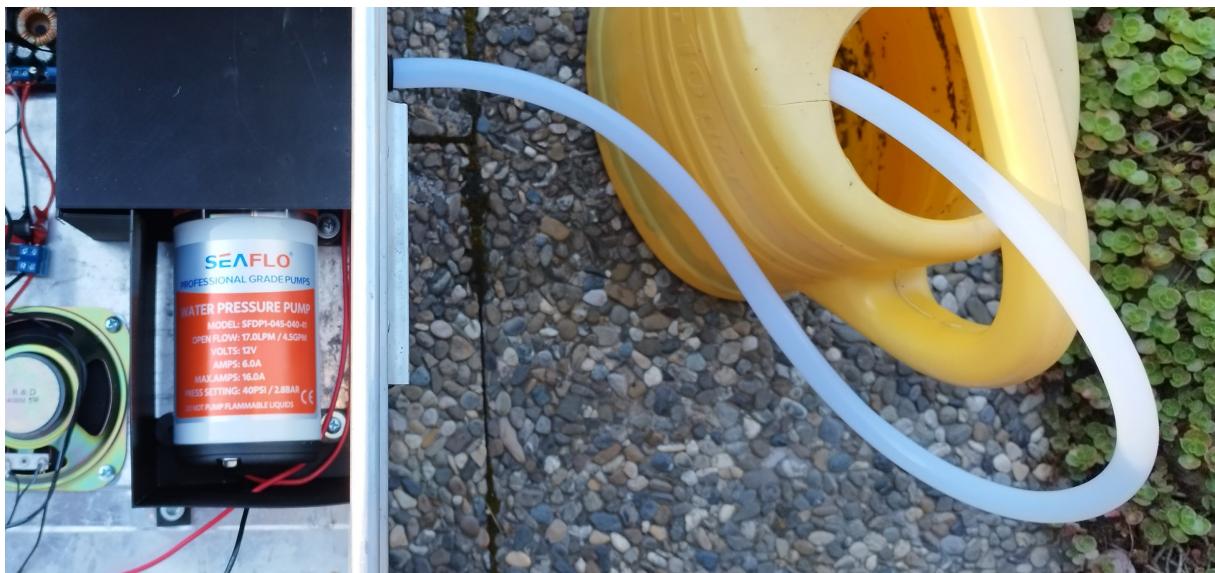


Abbildung 3.2.: Eingebaute Wasserpumpe mit Wasserversorgung und Spritzschutz

3.1.3. Mikrocontroller

Durch die derzeitigen Liefermangel an Mikrocontrollern ergab sich die Auswahl eines passenden Mikrocontrollers für den Einsatz im Abschrecksystem ebenfalls als schwierig. Getestet worden sind drei verschiedene Controller mit unterschiedlichen Erfolg, welche hier beschrieben werden.

Eine weitere Anforderung an das Abschrecksystem besteht darin, den Einsatz von *Stereo Vision* für das korrekte Zielen zu ermöglichen. Alle drei Mikrocontroller sind daher mit zwei Camera Serial Interface (CSI)-Anschlüssen ausgestattet. Dies war von besonderer Bedeutung, da Multi-Camera Adapter wie von *ArduCam* ([10]) und auch USB-Kameras keine ausreichende Zeitsynchronität und Anpassbarkeit an das System ermöglichen. Nicht synchronisierte Kameras verfälschen die Tiefenberechnung sehr. Ein genaues Zielen wäre demnach nicht länger möglich wie aus der Arbeit von Shimizu und Co. hervorgeht. [43]. Durch den direkten Anschluss an die CSI-Anschlüsse wird dieses Delay minimiert. Die Mikrocontroller unterstützen teilweise die zeitgleiche Aufnahme von Bildern auf beiden CSI-Anschlüssen. Im Falle des *Jetson Nanos* ist eine Zeitsynchronität jedoch nicht vollständig gewährleistet. Dies geht aus dem *ArduCam* Artikel von [25] hervor. Ein eigener Test ergab jedoch, dass die Kameras ausreichend synchron Bilder aufnehmen. Die Tiefenberechnung wäre demnach nicht zu stark beeinträchtigt, wie es mit dem *MultiCam* Adapter der Fall wäre.

Um die Versorgungsspannung der 12V Autobatterie nutzen zu können, wird ein Spannungswandler benötigt, da die Mikrocontroller mit 5V betrieben werden. Der Spannungswandler muss in der Lage sein, kurzfristig eine ausreichende Leistung bereitzustellen. Bei der Objekterkennung können für einen kurzen Zeitraum hohe Leistungsabfragen entstehen. Wenn die Leistung dann nicht schnell genug bereitgestellt werden kann, kann es zu Zwangsabschaltungen des Mikrocontrollers kommen. Dadurch könnte auch der Controller selbst Schaden nehmen.

Beim Aufsetzen und Testen des Jetson Nanos kam es wiederholt zu Stromabschaltungen. Der Nano wurde dabei mit einem Raspberry Pi Netzteil betrieben, das eine angegebene Leistung von 12,5 Watt hatte und daher für den Einsatz geeignet schien. Jedoch traten bei hohen Anforderungen, wie der gleichzeitigen Objekterkennung auf beiden Kameras, dennoch Stromabschaltungen auf. Im Gegensatz dazu traten beim Betrieb mit dem Spannungswandler und der Autobatterie keine unerwünschten Abschaltungen auf. Dies lag vermutlich daran, dass der einstellbare Spannungswandler von *AzDelivery* laut den Angaben in [4] bis zu 60 Watt Leistung für den Mikrocontroller bereitstellen kann.

Radxa CM 3

Durch den Liefermangel bestimmt waren Ende 2022 keine der nachfolgend beschrieben Mikrocontroller verfügbar. Daher ist das vom chinesischen Startup Radxa vertriebene *Radxa Compute Module 3* für den Einsatz im Abschrecksystem getestet worden. Der Mikrocontroller zeichnet sich besonders durch die eingebaute Neural Processing Unit (NPU) sowie die verwendete Hardware wie Serial AT Attachment (SATA)-/USB 3.0-Anschlüsse und die Anzahl von 50 General Purpose Input/Output (GPIO)-Pins aus. [40]

Allerdings war das eigens für das CM 3 konzipierte *IO Board* zum Zeitpunkt der Untersuchung nicht erhältlich. Laut *Radxa* ist das CM 3 jedoch kompatibel mit dem *Raspberry Pi 4 IO Board*. Mit diesem Board kann das CM 3 jedoch nur mit eingeschränkter Hardware genutzt werden.

Radxa bietet ein eigenes Betriebssystem für die Verwendung mit dem Raspberry Pi (RPi) IO Board an. Bei der Einrichtung des Systems traten jedoch bereits Probleme auf. Zunächst konnte der Embedded Multi Media Card (eMMC)-Speicher nicht erfolgreich geflasht werden. Nur mit erhöhtem Aufwand war die Inbetriebnahme möglich.

Im weiteren Betrieb traten weitere Probleme auf. Verschiedene Anwendungen und die Steuerung der Hardware (wie Kameras und GPIO-Pins) waren sehr fehleranfällig und stürzten wiederholt ab.

Auch konnte die NPU des CM 3 leider nicht mit den gängigen ML-Frameworks wie Tensorflow genutzt werden. Stattdessen bietet *Radxa* eine auf Linux basierende Toolchain an, um ML-Projekte den Zugriff auf die NPU zu ermöglichen. Die Einrichtung dieser Toolchain gestaltet sich jedoch als sehr umständlich, und viele Nutzer haben im *Radxa*-Forum um Hilfe gebeten. [40]

Aufgrund dieser Probleme konnte das *Radxa Compute Module 3* nicht für das Abschrecksystem verwendet werden.

Raspberry Pi CM 3

Auch RPi-Geräte sind von Lieferengpässen betroffen. Die Geräte sind entweder nicht verfügbar oder nur zu überhöhten Preisen erhältlich. Glücklicherweise hatte das Unternehmen *softwareinmotion* noch ein CM 3+ auf Lager, dass für die Studienarbeit ausgeliehen werden konnte.

Es ist jedoch zu beachten, dass das CM 3 bereits veraltet ist, da seit 2020 das CM 4 auf dem Markt ist. Der Hauptunterschied besteht darin, dass das CM 3 mehr GPIO-Pins hat, jedoch nur mit 1 GB DDR2 RAM erhältlich ist. Das CM 4 hingegen verfügt über 1 bis 8 GB DDR4 RAM und ist mit USB 3.0 ausgestattet. [56]

Die CM-Mikrocontroller haben vor allem deshalb an Beliebtheit gewonnen, weil sie die Möglichkeit bieten, eigene Carrier Boards zu entwerfen. Diese Boards eröffnen neue Flexibilität, um die Hardware an spezifische Anforderungen anzupassen und maßgeschneiderte Lösungen zu entwickeln. Ein Beispiel dafür ist der *StereoPi*, ein Carrier Board, der für die CM Reihe entwickelt wurde. Der *StereoPi* beschränkt Größe und Funktionalität des Compute Module auf die Stereo-Vision-Funktion, was ihn für bestimmte Anwendungen, wie die Tiefenberechnung besonders geeignet macht. Das *StereoPi* Carrier Board hätte sich ideal für das Abschrecksystem geeignet. Bedauerlicherweise war es aufgrund der neuen Version und des Liefermangels nicht verfügbar, weshalb das standard Carrier Board von Raspberry Pi verwendet wurde. [44]

Der RPi CM 3+ ließ sich im Gegensatz zum CM 3 von *Radxa* problemlos in Betrieb nehmen. Allerdings ist es erforderlich, Jumper-Kabel an einige GPIO Pins anzuschließen, um die Kameras nutzen zu können. Zudem müssen die Kameras über ein RPi Zero Flachbandkabel mit dem CSI-Anschluss am CM verbunden werden. [32]

Der CM 3 Mikrocontroller eignet sich daher für den Betrieb, wurde jedoch aus anderen Gründen, die in Kapitel 3.4.6 beschrieben sind, nicht verwendet.

Nvidia Jetson Nano

Der Jetson Nano Mikrocontroller hat eine ähnliche Größe und Form wie die Standard RPis. Im Vergleich zu den RPis verfügt der Jetson Nano jedoch nicht über WLAN oder Bluetooth. Dennoch besitzt er einen entscheidenden Unterschied: eine integrierte Grafikkarte. Aufgrund dieser Grafikkarte eignet sich der Jetson Nano besser für Aufgaben im Bereich des Machine Learnings. [36]

Grafikkarten (GPUs) spielen eine essenzielle Rolle bei der Anwendung von ML in der IT-Branche. Sie ermöglichen die parallele Verarbeitung von Daten und speziell Matrixoperationen. Diese Operationen werden in ML-Algorithmen und DNNs intensiv verwendet. Im Vergleich dazu können herkömmliche CPUs diese Operationen nur mit erheblichem Rechenaufwand bewältigen. Die hohe Rechenleistung und Parallelverarbeitungsfähigkeit von GPUs machen sie daher attraktiv bei ML-Anwendungen. Auch ermöglichen sie eine verbesserte Manipulation und Auswertung von Bilddaten. Dies ist besonders vorteilhaft für das Abschrecksystem, da es in Echtzeit Bilder verarbeiten soll. Die Graphics Processing Unit (GPU)-basierte Verarbeitung kann dabei helfen, die Objekterkennung und Bewegungserfassung effizient auszuführen. Dadurch kann das Abschrecksystem schneller und präziser auf die Kleintiere reagieren. [39]

Dennoch traten auch bei diesem Mikrocontroller Probleme auf, wie später in verschiedenen Kapiteln beschrieben wird. Der Jetson Nano hat sich jedoch aufgrund seiner Leistungsfähigkeit als „*Mini AI-Rechner*“ für das Abschreckssystem am besten geeignet.

3.1.4. Sensorik-Kameras

Viele unliebsame Kleintiere sind nachtaktiv. Daher muss das Abschreckssystem auch unter schwierigen Belichtungsbedingungen, insbesondere bei Nacht, die Kleintiere ebenso gut erkennen können wie bei Tageslicht. Aus diesem Grund ist es erforderlich, dass die eingesetzten Kameras eine Nachtsichtfunktion besitzen. Dafür werden infrarotsensible Kamerasensoren verwendet. Für den Raspberry Pi gibt es eine Variante der *OV5647*-Kamera, die für diese Anwendung geeignet ist. Bei der "NachtsichtVariante fehlt im Vergleich zur herkömmlichen Variante der Infrarotfilter. Im Tageslicht enthalten sind nämlich Infrarotstrahlen, die sonst für Rotstich-Aufnahmen sorgen.

Doch allein dadurch ist es noch nicht möglich, auch bei Nacht „sehen“ zu können, da es nachts keine natürliche Infrarotstrahlung durch das Sonnenlicht gibt. Die Nachtsichtvariante der *OV5647*-Kamera ist daher mit zwei zusätzlichen Infrarotstrahlern ausgestattet.

Dadurch entsteht jedoch ein neues Problem: Rotstich-Aufnahmen führen sowohl tagsüber als auch nachts zu einer Verschlechterung der Bildqualität, was auch die Objekterkennung beeinträchtigen könnte. Um tagsüber keinen Rotstich zu erhalten, verfügt die *OV5647*-Kamera über einen zuschaltbaren Infrarotfilter. Bei Nacht lässt sich der Rotstich dennoch nicht vollständig vermeiden. [54]

Die Kamerasensoren sind jedoch nicht für den Jetson Nano geeignet, da laut einem Eintrag im Nvidia-Forum aus dem Jahr 2021 die Verarbeitung der reinen Kameradaten nicht öffentlich zugänglich ist. [17]

3.2. Dreidimensionales Zielsystem

Für das dreidimensionale Zielsystem ergaben sich mehrere Herausforderungen. Einerseits musste eine präzise Steuerung des Zielsystems gewährleistet sein, andererseits musste die Reichweite von zehn Metern für den *Wasserwerfer* erreicht werden.

Die präzise Steuerung ist insbesondere wichtig, da nur das anvisierte Ziel mit Wasser getroffen werden soll und nicht alles herum mit Wasser bespritzt wird. Dies ermöglicht es, Wasser nicht zu verschwenden und einen maximalen Vertreibungseffekt zu erzielen.

In Tabelle 2.1 sind einige Elektromotoren und ihre Vor- und Nachteile beschrieben. Die

Elektromotoren eignen sich alle für den Einsatz im Zielsystem, haben aber Nebeneffekte, die bei der Auswahl mit berücksichtigt worden sind.

Bei den Schrittmotoren stellte die präzise Ansteuerung ein Problem dar, da gängige Varianten nur eine Schrittweite von $1,9^\circ$ haben. Dies würde zwar ausreichen, um größere Tiere gezielt anvisieren zu können, jedoch würde bei einer Entfernung von zehn Metern jeder Schritt eine Bewegung von 33 cm verursachen. Daher wäre ein Untersetzungsgetriebe erforderlich. Aufgrund des begrenzten Raums für das Zielsystem ist jedoch eine Implementierung eines solchen Getriebes nicht ohne weiteres möglich.

Auch die DC-Elektromotoren konnten nicht verwendet werden, da auch ihr Einbau viel Platz erfordern würde. Um die Elektromotoren präzise auf eine bestimmte Position zu fahren, wären zudem Positionssensoren erforderlich. Sie müssten im Zielsystem verbaut werden und würden einen umfangreichen Aufbau erfordern. Darüber hinaus müssten für beide Motortypen zusätzliche Treiberplatinen beschafft und eingebaut werden, was die Kosten erhöhen und viel Platz in Anspruch nehmen würde.

Deshalb wurden kleine Servomotoren für das Zielsystem verwendet. Die eingebauten Servomotoren von *Seamuing* sind normalerweise für den Einsatz in RC-Modellen vorgesehen, verfügen jedoch mit 245 N/cm über ein sehr hohes Drehmoment. Dieses Drehmoment wird benötigt, um den Gegenkräften des Wassers beim Zielen standzuhalten. Die Servos ermöglichen zudem eine präzise Auflösung von $0,09^\circ$. Somit wäre selbst im äußeren Zielbereich nur eine Abweichung von wenigen Zentimetern zu erwarten. Die Servos können zudem einfach über PWM angesteuert werden, sind jedoch nur im Bereich von 0° bis 180° einstellbar. [9]

Die Servomotoren werden jedoch mit der im Modellbau üblichen Spannung von 7,4V betrieben. Hierfür eignet sich derselbe Spannungswandler, der auch für den Mikrocontroller verwendet wird. Bei dem verwendeten Spannungswandler kann nämlich die Ausgangsspannung über eine Stellschraube angepasst werden.

Die zweite Herausforderung bestand darin, die geeignete Wasserpumpe zu finden und einzustellen. Hierbei wurde, wie im Kapitel 3.1.2 beschrieben, die Hausleitung und Sprinkleranlagen als Orientierung genutzt.

Für die Zielberechnung wurde ebenfalls auf bestehende Sprinklersysteme zurückgegriffen. Deren Düsenöffnungen haben nur einen Durchmesser von wenigen Millimetern. Daher wurde der Durchmesser des "Laufsäuf zwei Millimeter verengt. Ein Test ergab, dass mit dieser Einstellung und der verwendeten *SEAFLO* Pumpe eine Reichweite von 10 Metern erreicht werden kann.

Dennoch muss für das präzise anvisieren die Flugbahn des Wassers bestimmt werden. Um die Berechnung zu vereinfachen wird die Formeln für den *schrägen Wurf* angewendet, bei den äußeren Bedingungen wie Luftwiderstand und Windgeschwindigkeit nicht beachtet

werden. Dabei wird die Formel nach dem Startwinkel umgestellt. Die benötigten Parameter für die Berechnung sind dann die *Anfangsgeschwindigkeit* sowie die *x*- und *y*-Koordinate des Ziels.

Um die Formel anwenden zu können, fehlt nur noch die Anfangsgeschwindigkeit. Nachfolgend in Kapitel 3.5.3 ist, wie die *x*- und *y*-Koordinate bestimmt werden können.

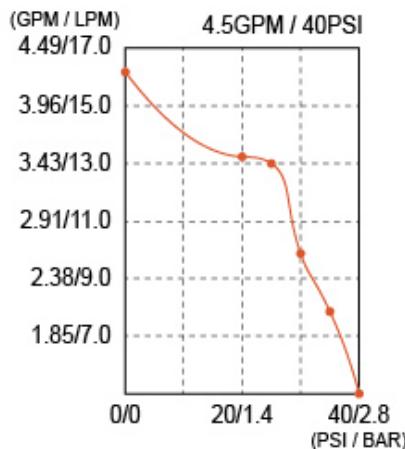


Abbildung 3.3.: Diagramm der Fördermenge gegenüber des Drucks. [42]

Um die Anfangsgeschwindigkeit zu bestimmen, wurde zunächst angenommen, dass die Pumpe nahe ihrer maximalen Leistung betrieben wird. Diese Annahme basiert auf den Tests mit der zwei Millimeter Öffnung, bei denen auch die Druckabschaltung der Pumpe getestet wurde. Dabei wurde festgestellt, dass die Pumpe nahe der Druckabschaltung von etwa 2,8 Bar betrieben wird.

Basierend auf dem Diagramm in Abbildung 3.2 lässt sich ableiten, dass die Pumpe bei dieser Einstellung eine Fördermenge von etwa 5 Litern pro Minute haben sollte. Mit diesen Daten kann die Austrittsgeschwindigkeit des Wassers berechnet werden. Dafür wird lediglich der Volumenstrom und der Querschnitt des Laufs benötigt. Allerdings ergibt die Formel

$Geschwindigkeit = \frac{Volumenstrom}{Strömungsquerschnitt}$ aus [13] keine korrekte Berechnung der Austrittsgeschwindigkeit, da ein Wert von über 265 Metern pro Sekunde berechnet wird. Es ist sehr wahrscheinlich, dass die Geschwindigkeit durch den Luftwiderstand stark reduziert worden ist.

Daher ist erneut die Formel für den *schrägen Wurf* verwendet worden und nach der Anfangsgeschwindigkeit aufgelöst worden. Dabei stellte sich heraus, dass die Anfangsgeschwindigkeit in etwa 10 Meter pro Sekunde beträgt.

Mit diesen Daten ist nun ein präzises anvisieren des Ziels möglich.

3.3. Ansteuerung

Zum Testen aller Funktionalitäten ist ein Arduino UNO verwendet worden. Der Arduino kann problemlos verwendet werden, um die Transistoren zu schalten und die Servomotoren zu steuern, da alle erforderlichen Bibliotheken bereits auf dem Board vorhanden sind. Der Testcode kann im Anhang unter A.1 eingesehen werden.

Der Arduino schaltet alle Aktoren - Ton, Licht, Pumpe und Servomotoren - in bestimmten

Zeitabständen ein und aus. Währenddessen werden die Servomotoren in inkrementellen Schritten angesteuert und ein Hochfrequenzton an zwei Pins ausgegeben. Bei diesen Tests stellte sich jedoch heraus, dass der erzeugte Ton zu leise war, um die Tiere effektiv zu stören. Daher wurde eine Verstärkerplatine hinzugefügt, die eine Ausgangsleistung von bis zu zehn Watt ermöglicht. Aus den Informationen von [7] geht hervor, dass die Platine direkt ohne Spannungswandler von der Autobatterie mit Strom versorgt werden kann. Der Verstärker verfügt über einen Klinkenanschluss. Da der *Jetson Nano* keinen Klinkenanschluss besitzt, wird ein USB-auf-Klinke-Adapter benötigt, um einen Ton abspielen zu können.

Der *Jetson Nano* besitzt 40 GPIO Pins, mit denen die Transistoren geschaltet und die Servomotoren bedient werden können. Der Nano verfügt aber nur über zwei PWM Pins. Eine Ansteuerung von mehr als zwei Servomotoren ist daher nicht möglich. [34]

Das Steuern der Aktoren wird durch die von *Nvidia* zur Verfügung gestellte *Jetson.GPIO* Python Bibliothek ermöglicht. [35]

Damit das Toggeln der Pins die Objekterkennung nicht blockiert, wurde eine threadbasierte Klasse erstellt, die jedem Pin einen eigenen Thread zuweist. Aus dem Hauptprogramm kann das Toggeln jederzeit durch den Befehl *Pin.start_toggle()* gestartet und durch *Pin.stop()* pausiert werden. Auch kann die Frequenz des Toggelns durch das Multithreading dynamisch in dem Thread selbst verändert werden. Besonders für die Änderung der Schaltfrequenz für das Abschrecklicht eignet sich diese Funktion.

Für die Steuerung der Servomotoren wurde eine ähnliche threadbasierte Klasse erstellt, bei der der Ansteuerungswinkel einfach und ähnlich wie im Arduino-Code aus A.1 angewendet werden kann.

Beim Testen der Toggle-Funktion wurde jedoch ein irreführendes Verhalten beobachtet. Das Schaltverhalten entsprach nicht immer den Vorgaben. Nach ausführlicher Fehlersuche und einen Blick in das Datenblatt des Jetson Nanos ergab, das die Spannung, die an den meisten GPIO Pins anliegt, lediglich 1,8V beträgt und nur teilweise 3,3V. Die ausgewählten Transistortreiber sind jedoch für eine Schaltlogik von 3,3V bis 5V ausgelegt. Zudem wurde im Datenblatt der Transistoren festgestellt, dass bei Verwendung einer 3,3V-Logik der Strom auf unter 10 Ampere begrenzt werden sollte, da sich der Transistor sonst möglicherweise nicht abschalten lässt. Dieses Verhalten konnte beim Betrieb der Wasserpumpe festgestellt werden, da sie sich nur stark verzögert ausschaltete.

Es traten noch weitere Probleme bei der Verwendung der PWM-Steuerung der Servomotoren auf. Es schien, als ob die Servomotoren kein gültiges Signal vom Nano erhalten und keinerlei Bewegung zeigten. Bei der Messung der PWM stellte sich heraus, dass keine PWM erzeugt wurde. Selbst die Troubleshooting-Anleitungen von Nvidia brachten keine Lösung. In einigen Fällen wurde sogar versucht, wie von Nikola Jelic in [26] beschrieben,

Registereinträge auf dem Nano vorzunehmen. Leider führten auch diese Versuche nicht zum gewünschten Erfolg. Daher muss für die Steuerung der Servomotoren eine alternative Lösung gefunden werden.

Doch durch den Versuch die PWM auf den Nano zu verwenden traten noch weitere Nebeneffekte auf. Das Toggeln der Pins war nicht mehr bis zum neustarten des Nanos möglich. Deshalb wurde die gesamte Hardwareansteuerung auf den Arduino UNO ausgelagert. Dieser hatte sich bereits durch den Funktionstest bewährt.

Für die Ansteuerung der Aktoren durch den Arduino wurde das Testprogramm aus A.1 angepasst. Der Arduino kann nun über den seriellen Port (USB) Befehle und Werte für die Steuerung des Zielsystems empfangen. Das Toggeln der Aktoren liegt vollständig in der Verantwortung des Arduino. Sobald er Werte für die Ansteuerung der Servomotoren erhält, wird er aktiv und steuert die Aktoren an. Die Steuerung wird solange ausgeführt, bis das Kommando „g“ vom Nano empfangen wird. Das „g“ steht dabei für *gone* und signalisiert, dass das Ziel den Zielbereich verlassen hat.

Auf der Seite des Jetson Nanos wurde außerdem eine threadbasierte Steuerung für den Arduino entwickelt. Die serielle Kommunikation mit dem Arduino benötigt deutlich mehr Zeit als die vorherige direkte Ansteuerung über die GPIO-Pins, weshalb das Multithreading unbedingt erforderlich ist. Der Effekt auf die zeitliche Interferenz mit der Objekterkennung ist jedoch gering und es konnte keine signifikante Verbesserung durch das Multithreading festgestellt werden.

Den gesamten Schaltplan können Sie in Abbildung 3.3 einsehen. Die Kamerasensoren werden über den Jetson Nano betrieben und ihre Steuerung erfolgt hauptsächlich über Software. Daher werden sie erst in den nachfolgenden Kapiteln genauer beschrieben.

3. Umsetzung

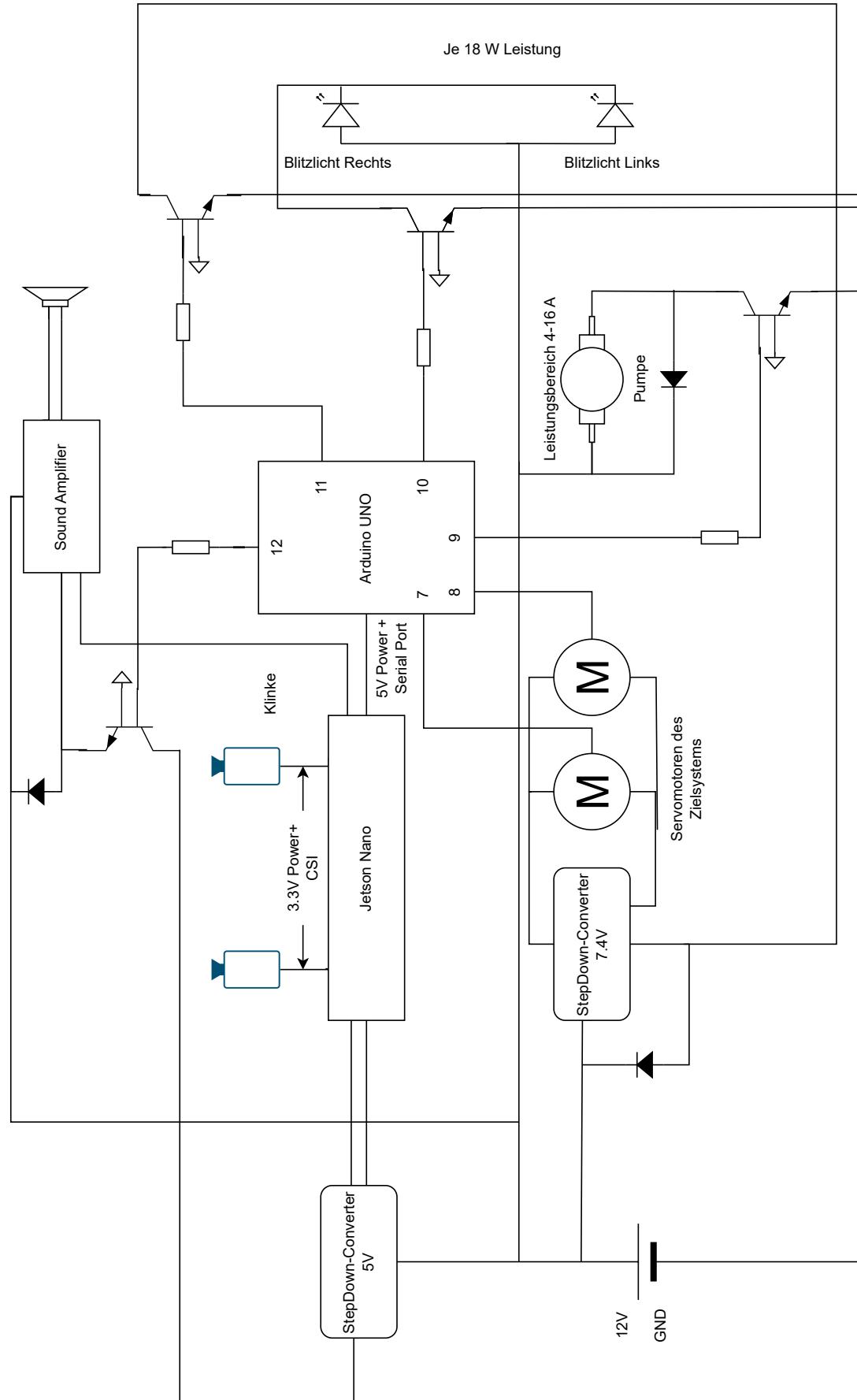


Abbildung 3.4.: Schaltplan der Hardware

3.4. Objekterkennung und -Verarbeitung

Der Hauptteil der Arbeit bestand in der Entwicklung der Software, insbesondere der Implementierung der Objekterkennung und der Interaktion und Synchronisation der Kameras. Für das Training und Deployment der Objekterkennung wurden hauptsächlich Frameworks verwendet. Dennoch wurden auch einige Hilfsprogramme entwickelt und eingesetzt, welche im Folgenden beschrieben werden.

Für die Objekterkennung wurden verschiedene Modelle verwendet. Darunter zwei Modelle aus dem *TensorFlow Model Zoo* und ein Modell von Google für ihren USB-Accelerator *Coral*.

Überwiegend wurde jedoch mit dem *YOLOv8*-Objekterkennungsmodell gearbeitet. Dieses Modell bot klare Vorteile gegenüber den anderen Frameworks. Insbesondere das sehr einfache Setup für das Training sowie die Exportmöglichkeit in verschiedene ML-Formate überzeugten. Eine detaillierte Beschreibung dieser Vorteile wird im weiteren Verlauf gegeben.

3.4.1. Ordner- und Datenstruktur

TensorFlow bietet in [47] ein eigenes Tutorial an, welches zeigt, wie man die Objekterkennung API nutzt und ein Objekterkennungsmodell trainiert und evaluiert. Diese Ordnerstruktur wurde für das Trainieren der Objekterkennungsmodelle übernommen und auf die eigene Nutzung angepasst. Die gesamte Ordnerstruktur ist in Abbildung 3.4.1 zu sehen.

Aus der Struktur und dem Ablauf der Studienarbeit ergaben sich vier Kernbereiche, die auch in der Ordnerstruktur widergespiegelt werden. Das Projekt besteht aus den folgenden vier Ordnern: **Dokumentation**, **Dispositionspapier**, **Deployment** und **Detection_training** für das Training der Objekterkennung. Im folgenden werden aber nur die Inhalte des **Deployment** und **Detection_training** Ordners näher betrachtet.

Deployment

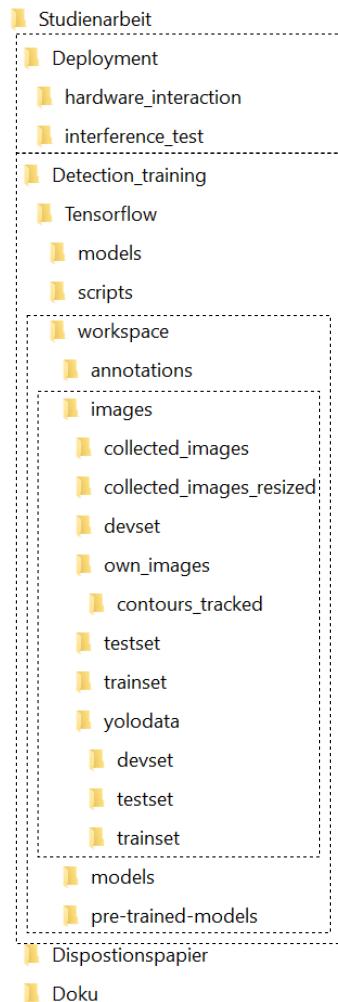
Der **Deployment** Ordner enthält mehrere Unterordner, die zur besseren Übersicht und Einordnung der Funktionen dienen. Darunter befinden sich die **Hardwareansteuerung**. Diese beinhalten auch die Arduino-Dateien des Funktionstestprogramms, sowie das Programm zur externen Steuerung des Zielsystems. Zudem ist in diesem Ordner das Python-Modul abgelegt, das die externe Ansteuerung des Arduinos ermöglicht.

Das verworfene Python-Modul zur direkten Ansteuerung der Aktoren über die GPIO-Pins des Jetson ist ebenfalls in diesem Ordner zu finden.

Des Weiteren gibt es einen separaten Ordner für die zeitlichen Inferenztests. In diesem Ordner befindet sich das Python-Skript, das die Ausführungszeit der verschiedenen Objekterkennungsmodelle und ihrer Formate testet. Die Auswertung der Inferenzzeit wird zudem in eine Markdown-Datei speichert.

Die weiteren Dateien, die in diesem Ordner abgelegt sind, werden in denen folgenden Kapiteln beschrieben.

Detection training



Im **Detection_training** Ordner befinden sich zunächst die Jupyter-Notebooks für die Aufbereitung der Daten und der Auswertung der verschiedenen Modelle. Für die verschiedenen Trainingsmöglichkeiten sind jeweils eigene Notebooks erstellt worden.

Des Weiteren enthält der Ordner die modifizierte TensorFlow Ordnerstruktur. Darin befindet sich ein Skriptordner, welche einige Hilfsprogramme für das Trainieren der Objekterkennungsmodelle enthält. Die meisten dieser Hilfsprogramme sind jedoch für die Konvertierung der Bildannotationen in verschiedene Dateiformate zuständig.

Der *models* wird benötigt um die Objekterkennungs-API von TensorFlow nutzen zu können. In diesem sind die Installationsdateien, sowie Hilfsprogramme für die TensorFlow Objekterkennungs-API gespeichert.

Es ist auch von TensorFlow vorgesehen, Speicherorte für vortrainierte Objekterkennungsmodelle sowie für nachtrainierte oder selbst trainierte Modelle anzulegen. Diese werden im Unterordner *workspace*.

Zusätzlich wird für das Trainieren mit TensorFlow eine Umwandlung der Bildannotationen in das TFRECORD-Format vorgesehen. Die konvertierten Annotationen sind daher im *annotations* Ordner unter dem *workspace* Ordner zu finden.

Abbildung 3.5.: Ordnerstruktur des Projektes

3. Umsetzung

Ein weiterer Unterordner des *workspace* Ordner ist der *images* Ordner, der alle Bilddaten enthält. Darin befindet sich ein Ordner mit der Bezeichnung *collected_images*. Er enthält alle Bilder und Annotationen, die für das Abschrecksystem verwendet wurden.

Zudem wird für das Trainieren der Modelle eine Zerteilung der annotierten Bilder in Train-, Test-, und Devset benötigt. Die Verteilung erfolgt mit den angegebenen Prozentsätzen: 85% der Daten werden dem Trainingsset zugeordnet, 10% dem Devset und 5% dem Testset. Dadurch können die Modelle mit unterschiedlichen Datensätzen trainiert, validiert und getestet werden.

Da YOLO eine leicht veränderte Datenstruktur erwartet, wird diese separat im Unterordner *yolodata* abgespeichert. Der Unterschied liegt dabei jedoch im Annotierungsformat.

Für die Aufteilung werden die Daten aus dem *collected_images_resized* Ordner entnommen. Der Ordner enthält alle Bilder und Annotationen, die auf eine einheitliche Höhe normiert wurden. Dies Normierung ist notwendig, da die Objekterkennungsmodelle bestimmte Bilddatenformate erwarten, wie beispielsweise 320x320 Pixel. Durch die Normierung der Bilder auf eine einheitliche Größe wird sichergestellt, dass während des Trainingsprozesses keine zusätzliche Zeit und RAM-Speicher für die Größenanpassung benötigt wird.

Einige der Bilder im Originalformat haben eine hohe Auflösung, wie in etwa 4K-Qualität. Diese erfordern entsprechend viel Speicherplatz und erhöhen die Ausführungszeit während des Trainings. Durch die Umwandlung der Bilder von 4K auf 320x320 Pixel vor dem Training der Modelle wird der Speicherbedarf erheblich reduziert. Die Größe der Bilder nach der Umwandlung beträgt weniger als ein Prozent der ursprünglichen Größe. Dadurch wird eine schnellere Ausführung während des Trainingsprozesses ermöglicht und der Speicherplatzbedarf erheblich verringert.

Unter den Bildern befinden sich auch eigene Bilder, die vom Abschrecksystem selbst aufgenommen wurden. Sie sind im Ordner *own_images* gespeichert. Dabei wird auch unterschieden, ob es sich bei den aufgenommenen Bildern um erkannte Tiere handelt, die mittels Objekterkennung identifiziert wurden, oder ob es sich um eine Bewegungserkennung mittels Kontur-Tracking handelt. Die Disjunktion zwischen den beiden wird später beschrieben.

3.4.2. Sammlung der Trainingsdaten

Zu Beginn der Arbeit wurde zunächst nach vergleichbaren Modellen und Datensätzen für die *Objekterkennung* gesucht. Dabei fiel auf, dass es für einige unliebsame Kleintiere wie den *Marder* leider keine geeigneten Datensätze gab.

Es gab jedoch bereits ein Modell für die Objekterkennung von Waschbüren. Dat Tran hatte im Jahr 2017 ein TensorFlow-Lite-Modell für die Erkennung von Waschbüren erstellt.

3. Umsetzung

Allerdings hatte er bei der Entwicklung eine andere Absicht. Die Tiere waren seine Lieblingstiere und er wollte wissen, wann ein Waschbär vor seiner Haustür auftaucht. [51]

Mit etwas anderer Absicht lässt sich dies aber auch für die Studienarbeit nutzen, da Dat Tran seine 200 handgelabelten Bilder online zur Verfügung gestellt hat. Diese können für die Objekterkennung des Waschbären verwendet werden und dienen als Grundlage für das Training des Modells in der vorliegenden Arbeit.

Die Anzahl der Bilder erscheint für eine erste Betrachtung und Einarbeitung in das Trainieren eines eigenen Objekterkennungsmodells als geeignet. Ein Vergleich der Trainingsmodelle kann in Kapitel 3.4.6 eingesehen werden.

Der Marder und der Waschbär waren aber nicht die einzigen Tiere, die aus dem Garten ferngehalten werden sollten. Auch Katzen, Füchse und Eichhörnchen sollten aus dem Garten ferngehalten werden. Die Absicht, Katzen und Eichhörnchen fernzuhalten, begründet sich damit, dass wir Vogelliebhaber sind und verhindern möchten, dass diese Tiere die Vögel stören oder ihnen schaden.

Bei den Füchsen sieht es ähnlich aus. Sie sollen von den heimischen Beeten ferngehalten werden, da sie bekanntermaßen Krankheiten übertragen können und somit eine potenzielle Gefahr darstellen.

Um auch Marder und andere Tiere in die Erkennung aufzunehmen, wurden mehrere Datenlabeling-Programme getestet. Das Programm *Label Studio* hat dabei am meisten überzeugt.

Label Studio bietet einen interaktiven Workflow und ermöglicht die Zusammenarbeit mehrerer Personen an einem Projekt. Tasks können verschiedenen Personen zugewiesen und übersichtlich dargestellt werden. Die benutzerfreundliche Drag-and-Drop-Oberfläche erleichtert das Labeln von Bildern und das Zeichnen von Bounding Boxes. Ein Beispiel für das Labeln eines Bildes wird in Abbildung 3.4.2 exemplarisch dargestellt. Es ist jedoch zu beachten, dass das Bild bereits zu einem späteren Zeitpunkt der Studienarbeit stammt, bei dem das Abschreckssystem bereits funktionsfähig ist. [23]

Die gelabelten Daten stehen anschließend in dem XML-basierten *Pascal VOC* Datenannotationssformat zur Verfügung.

3. Umsetzung

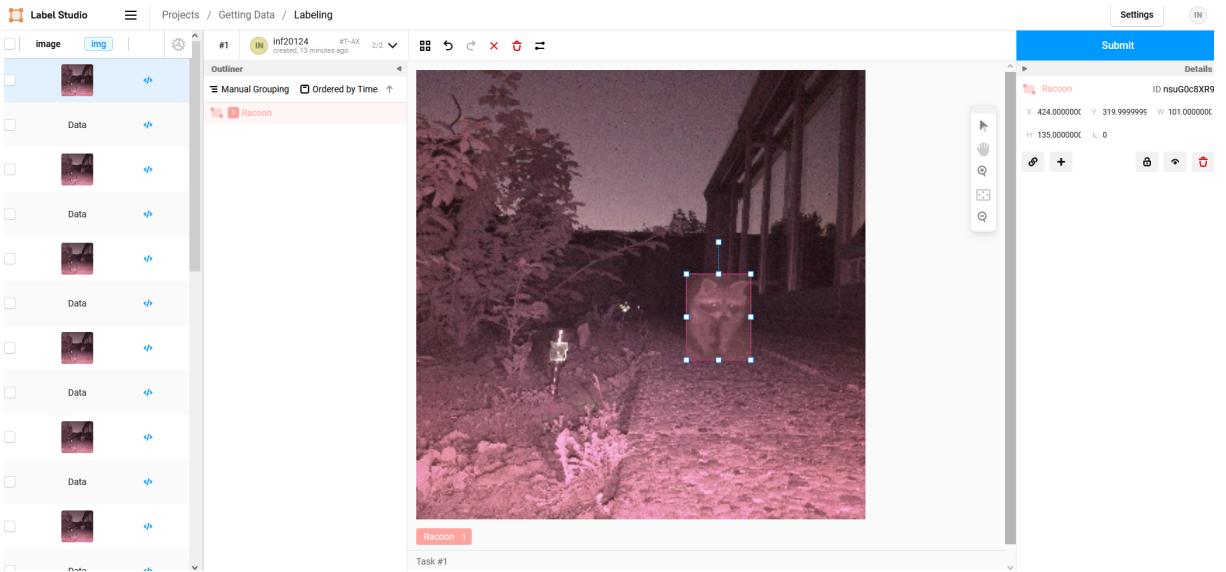


Abbildung 3.6.: Einzeichnen einer Bounding Box mit dem Programm Label Studio

Das Problem beim manuellen Erstellen solcher Datensätze ist jedoch, dass dies viel Zeit in Anspruch nimmt. Ein ausreichend großer Datensatz kann daher innerhalb des vorgegebenen Zeitrahmens nicht erstellt werden. Aus diesem Grund wurde die Suche nach zusätzlichen Datensätzen erneut aufgenommen.

Dabei stachen die Daten von *Google Open-Images-V7* heraus. Der Datensatz enthält 16 Millionen Bounding Boxes auf 1,9 Millionen Bildern. Von den 600 verfügbaren Klassen sind auch die für die Arbeit relevanten Tiere enthalten. Allerdings ist ihre Verteilung ungleichmäßig. Ein Großteil der relevanten Bilder handelt von Katzen, während es weniger als tausend Bilder von Waschbüren gibt. Daher ist zu erwarten, dass die Waschbüre im Endprodukt schlechter erkannt werden als die Katzen. [19]

Von den 1,9 Millionen Bildern in *Google Open-Images-V7* sind nur etwa 17.000 für das Abschrecksystem relevant, da sie Waschbüre, Katzen, Füchse oder Eichhörnchen enthalten. Nur diese Bilder sollten dem Datensatz hinzugefügt werden.

Hier kommt *FiftyOne* ins Spiel. *FiftyOne* ist eine Open-Source-Bibliothek und Plattform zur Datenanalyse von Computer-Vision-Modellen. Eine Funktion von *FiftyOne* ist das Extrahieren und Herunterladen von Computer-Vision-Datensätzen.

Allerdings werden die Bilder und Bounding Boxes nicht getrennt von den nicht benötigten Daten heruntergeladen. Stattdessen werden CSV-Dateien heruntergeladen, die alle Bounding Boxes aufgeteilt in Train-, Test- und Devset enthalten. Diese CSV-Dateien werden anschließend von *FiftyOne* ausgewertet, und die entsprechenden Bilder werden nachträglich heruntergeladen. [50]

Das Herunterladen der Bilder kann je nach Bandbreite des Netzanbieters eine sehr lange Zeit in Anspruch nehmen. Besonders da die Größe der CSV-Datei, die alle Trainingsdaten beschreibt, bereits mehr als 2,1 GB groß ist. Zusätzlich zur Download-Zeit der CSV-Dateien kommt die Auswertungszeit hinzu, um festzustellen, welche Bilder heruntergeladen werden sollen, sowie die Download-Zeit der eigentlich ausgewählten Bilder.

Nach dem Herunterladen der ausgewählten Bilder und den dazugehörigen Bounding Boxes steht der Datensatz immer noch nicht direkt zur Verwendung bereit. Die Bounding Boxes sind zwar vorhanden, jedoch sind sie weiterhin mit allen anderen Bounding Boxes kombiniert. Dabei beträgt die Größe der CSV-Datei für die Trainingsdaten 2,1 GB, was der Hälfte des Speicherplatzes der heruntergeladenen Bilddaten entspricht.

Weitere Schritte sind daher erforderlich, um den Datensatz für das Training verwenden zu können. Unter dem Skriptordner ist daher ein in *Rust* geschriebenes Konvertierungsprogramm mit der Bezeichnung *csv_conv* abgelegt. Dieses Skript konvertiert die heruntergeladenen CSV-Dateien in ein CSV-Format, das mit *TensorFlow* kompatibel ist. Dabei werden auch alle nicht relevanten Bilddaten und Bounding Boxes herausgefiltert. Die Größe der resultierende CSV-Datei, die alle Trainings-, Test- und Devset-Bounding Boxes enthält, beträgt danach weniger als 1 MB.

Zusätzlich wurden irrelevante Daten entfernt, wie die Datenquelle sowie die Informationen *IsOccluded*, *IsTruncated*, *IsGroupOf*, *IsDepiction*, *IsInside* und die *Confidence*. Die *Confidence* gibt an, mit welcher Wahrscheinlichkeit ein Objekt erkannt wurde. Jedoch spielt dieses Feld nur eine Rolle, wenn ein Objekterkennungsmodell die Bounding Box vorhersagen würde. Da dies bei den vorliegenden Daten nicht der Fall ist, kann das Feld ebenfalls gelöscht werden.

Durch die Deserialisierung können diese Felder automatisch ohne Mehraufwand in der Programmierung entfernt werden. Als ein letzter Schritt werden danach die Bounding Box Daten, welche in Prozent angegeben sind, in eine Ganzzahl konvertiert und die Klasse des Objektes von einer ID zu einem Namen (zum Beispiel *Racoon*) aufgelöst.

Nun gibt es jedoch ein weiteres Problem: Die Daten von Dat Tran und die von *Google-Open-Images* liegen in unterschiedlichen Dateiannotierungsformaten vor. Allerdings hat Dat Tran in [51] bereits ein Python-Skript mit dem Namen *xml_to_csv* erstellt, das die XML-Dateien in eine CSV-Datei konvertiert. Dieses Skript wurde an dieser Stelle verwendet und befindet sich ebenfalls im Skriptordner.

Mit den nun vorliegenden Daten kann die Objekterkennung angegangen werden.

3.4.3. Bildmanipulation und -augmentation der gesammelten Bilder

Objekterkennungsmodelle lernen, Objekte anhand von Mustern, Farbtönen und Positionen im Vergleich zu anderen Objekten zu erkennen. Dies führt jedoch häufig zu „Auswendiglernen“ oder Overfitting der Trainingsdaten. Wenn diese Modelle anschließend auf Daten angewendet werden, die nicht in den Trainingsdaten enthalten sind, können sie diese nur mit geringer Zuversichtlichkeit erkennen. Die Augmentierung der Trainingsdaten versucht, dieses Problem zu lösen, indem sie die Trainingsdaten manipuliert, um das Modell vielseitiger zu machen und unbekannte Daten besser zu erkennen.

Gängige Möglichkeiten zur Durchführung von Bildmanipulationen sind:

- Rotation der Bilder
- Ein- und Auszoomen der Bilder
- Abschneiden eines Bildteils
- Hinzufügen von zufälligem Jitter zu den Bildern

Es gibt jedoch viele weitere Möglichkeiten. Zum Beispiel bietet das TensorFlow-Framework 39 verschiedene Augmentierungsmöglichkeiten an. Auch andere Frameworks bieten ähnliche Möglichkeiten an. [18, 46]

Im vorherigen Abschnitt wurde beschrieben, wie die Daten für das Training der Modelle generiert wurden. Diese können jedoch nicht ohne weitere Anpassungen direkt für das Training verwendet werden, da sie nur geringfügig den erwartenden Bildern durch das Abschrecksystem entsprechen. Ein Beispiel dafür ist in Abbildung 3.4.3 dargestellt.

Links ist ein Bild aus dem Datensatz von Dat Tran zu sehen, während rechts ein Bild vom Abschrecksystem zu sehen ist. Die Bilder wurden bereits auf die gleiche Höhe normiert. Wie im vorherigen Kapitel 3.4.1 beschrieben, führt diese Bildmanipulation zu einer erheblichen Reduzierung der Zeit- und Ressourcenanforderungen während des Trainings.

Die Größenanpassung selbst wird mithilfe eines Python-Skripts durchgeführt, das sich im Skriptordner befindet. Die Anpassung der Annotationen an die neue Bildgröße erfolgt mit einem separaten Rust-Programm.

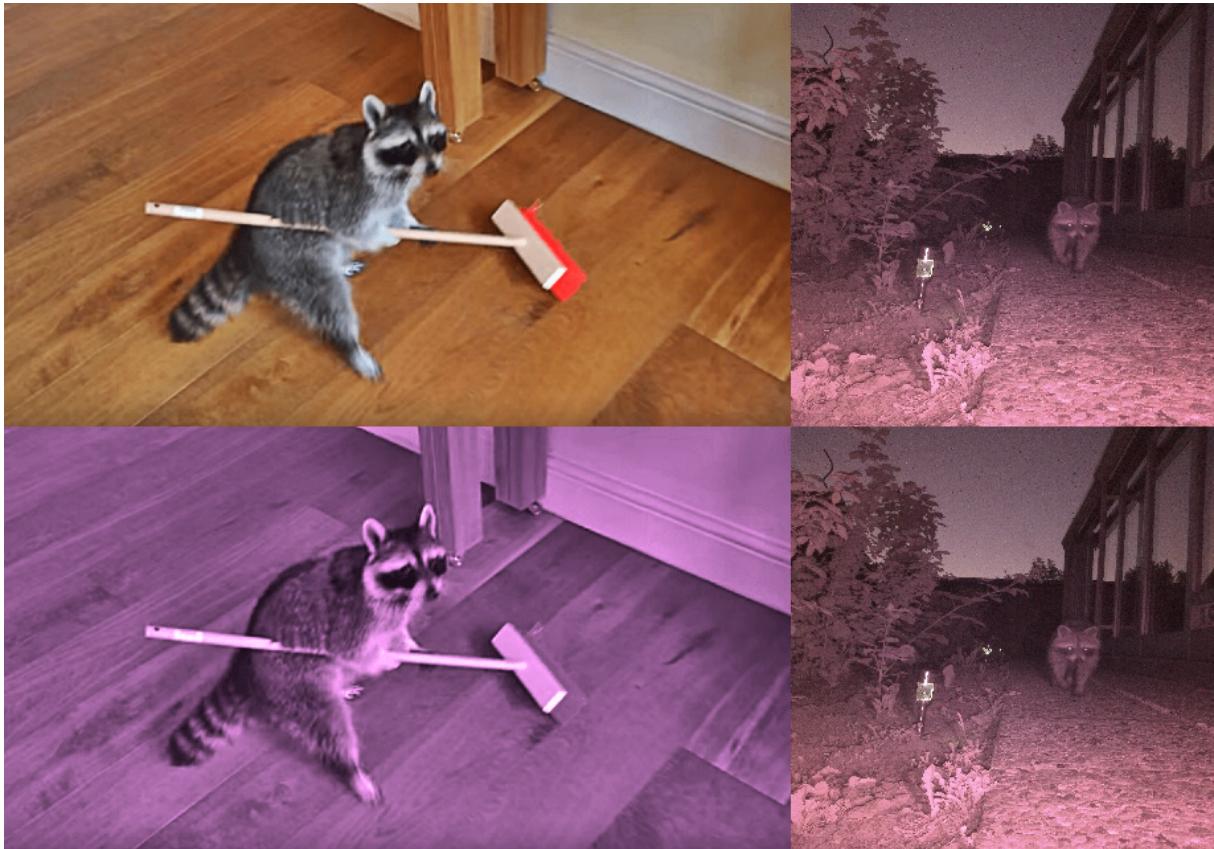


Abbildung 3.7.: Vergleich eines augmentierten und nicht augmentierten Bildes aus den Trainingsdaten (links [51]) und ein Bild das vom Abschrecksystem aufgenommen wurde

Die verschiedenen Frameworks, die für das Training der Modelle verwendet werden, verfügen über diese Augmentierungsmöglichkeiten, die Trainingsbilder an die tatsächlichen Bilder anzugeleichen. Dazu gehören manuelle Anpassungen an den Farbwerten, der Sättigung und der Dunkelstufe (HSV) an den einzelnen Bildern. In den jeweiligen Frameworks können diese verwendet werden, um ein Bild während des Trainings anzupassen. Auf diese Weise können die Bilder dem pinkfarbenen Erscheinungsbild des Abschrecksystems angepasst werden. Dadurch wird die mAP der Modelle auf den heruntergeladenen Datensätzen zwar deutlich verringert, jedoch haben kleinere Tests mit den Bildern des Abschrecksystems gezeigt, dass die Zielobjekte besser erkannt werden. Eine beispielhafte Anpassung der Bilder aus dem Datensatz kann in Abbildung 3.4.3 betrachtet werden.

3.4.4. Training

In der Studienarbeit wurden drei verschiedene Frameworks verwendet, um verschiedene Objekterkennungsmodelle hinsichtlich ihrer Inferenzzeit und Genauigkeit zu evaluieren. Im

Folgenden sind die Workflows beschrieben, die für das Training der Modelle erforderlich sind.

TensorFlow Model Zoo

TensorFlow bietet zwei Möglichkeiten, Modelle für die Objekterkennung zu trainieren. Zum einen kann ein eigenes Modell mithilfe der *Keras*-API definiert und trainiert werden. Zum anderen besteht die Möglichkeit, vortrainierte Modelle aus ihrer Modellsammlung *Model Zoo* weiter zu trainieren und auf die eigene Nutzung anzupassen.

Wie später in Kapitel 3.4.4 beschrieben wird, eignen sich vortrainierte Modelle besser für die Entwicklung der Objekterkennung im Abschrecksystem, da sie während des Trainings schneller eine höhere Genauigkeit erreichen.

In der TensorFlow-Bibliothek sind auch die Inferenzzeit und Genauigkeit der vortrainierten Modelle auf dem *COCO-2017*-Datensatz angegeben. Dadurch kann ein Entwickler bereits entscheiden, welche Modelle für das Anwendungsgebiet relevant sein könnten. Bei den meisten Modellen steigt die Genauigkeit mit der Ausführungszeit und Bildgröße. Das genaueste Modell im Model Zoo ist das *EfficientDet*-Modell mit einer mAP von 51,2. Das Modell ist mit Bildern der Größe 1536x1536 Pixel trainiert worden.

Das schnellste Modell, welches auf eine Bildgröße von 320x320 Pixel trainiert worden ist, ist das *CenterNet MobileNetV2*-Modell mit einer Ausführungszeit von nur 6 Millisekunden. Obwohl es nur etwa 2 Prozent der Ausführungszeit des *EfficientDet*-Modells benötigt, fällt die Genauigkeit auf unter 24 mAP ab.

Basierend auf diesen Werten wurde entschieden, dass die beiden Modelle *SSD MobileNet V2 FPNLite 320x320* und *EfficientDet D0 512x512* für das Abschrecksystem in Betracht gezogen werden sollen. Diese Modelle weisen eine geringe Ausführungszeit auf und ihre Genauigkeit ist nicht zu niedrig. [18]

Der Aufwand, die Modelle nachzutrainieren, ist gering, da TensorFlow die erforderlichen Jupyter Notebooks in ihrem GitHub-Repository ([18]) hochgeladen hat. Mit kleinen Anpassungen kann das Notebook für das Training des Abschrecksystems verwendet werden. Der Ablauf eines Trainings mit dem Jupyter Notebook ist in dem Programmablaufplan (PAP) in Abbildung 3.4.4 dargestellt.

Zunächst müssen alle erforderlichen Pfade definiert und die entsprechenden Ordnerstrukturen erstellt werden. Ein Großteil dieser Ordnerstrukturen wurde bereits mithilfe des Python-Skripts aus Kapitel 3.4.1 erzeugt. Es ist lediglich die spezifische Benennung des Modells erforderlich.

Anschließend wird das vortrainierte Modell heruntergeladen, sowie die TensorFlow Object Detection API installiert.

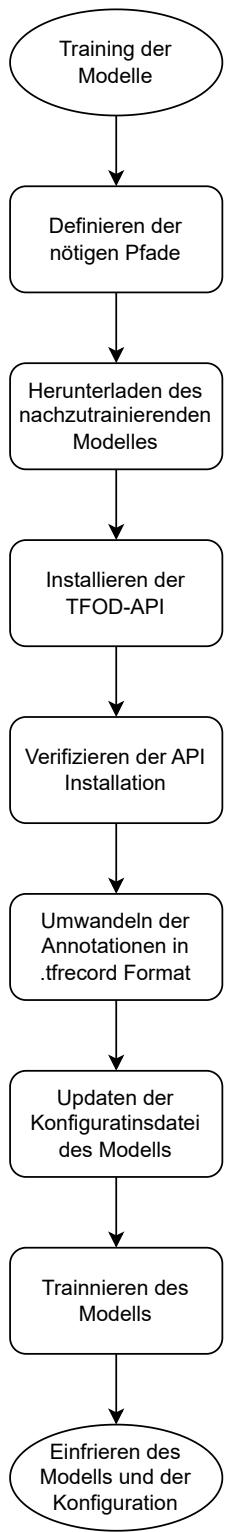


Abbildung 3.8.: PAP TensorFlow Flow Nach-training

Bevor das Training beginnen kann, müssen die Annotationsen, die in der Trainings-CSV-Datei enthalten sind, in das TFRecord-Format umgewandelt werden. TensorFlow bietet hierfür ein Python-Skript an. Allerdings musste dieses Skript angepasst werden, da es nicht direkt verwendet werden konnte. Das angepasste Skript iordnest unter dem Skriptordner enthalten.

Anschließend wird die Konfigurationsdatei des zuvor heruntergeladenen Modells aktualisiert. Dies bedeutet, dass die Pfade zu den Bilddaten und der TFRecord-Datei angepasst werden müssen. Darüber hinaus können weitere Trainingsparameter geändert oder hinzugefügt werden. Zum Beispiel können auch Augmentierungen hinzugefügt werden und die Anzahl der Objekttypen kann an den Anwendungsfall angepasst werden.

Erst nach Abschluss dieser Schritte kann das Training gestartet werden. Die Installation der TensorFlow Object Detection API sollte jedoch nur einmal erforderlich sein. Für das Training wird dann nur noch die Anzahl der Trainingsschritte benötigt. Diese gibt an, wie lange und intensiv das Modell nachtrainiert werden soll. Basierend auf der angepassten Konfigurationsdatei wird das Nachtraining gestartet.

Nach Abschluss des Trainings wird das Modell eingefroren. Das eingefrorene Modell kann dann für die Objekterkennung verwendet werden.

Training für Google Coral

Da es nicht von Anfang an möglich war, einen Jetson Nano mit integrierter Grafikkarte zu erhalten, wurde zunächst der Ansatz mit dem Raspberry Pi verfolgt. Es war jedoch von Anfang an klar, dass die Ausführung der Objekterkennung auf dem Raspberry Pi nur zu einer geringen Anzahl von FPS führen würde. Daher war zusätzliche Hardware erforderlich, um eine Objekterkennung auf dem Raspberry Pi zu ermöglichen. In diesem Fall kommt Googles USB-Accelerator *Coral* zum Einsatz. Die über USB anschließbare Edge-TPU ist speziell für KI-Inferenzanwendungen entwickelt worden. Mit seiner

Hardware ermöglicht der USB-Stick die Ausführung von ML-Algorithmen auf Geräten mit begrenzter Rechenkapazität.

Der Coral USB-Stick ist in der Lage, die Objekterkennung in Echtzeit durchzuführen. Dank seines geringen Stromverbrauchs von zwei Watt und seiner Mobilität eignet sich der Coral USB-Stick auch für die Anwendung in dem Abschrecksystem. [20]

Auf der Coral-Webseite ([20]) werden ebenfalls eigene Trainingsmodelle und Jupyter-Notebooks zur Verfügung gestellt, um das Training durchzuführen. Zusätzlich wird eine Auflistung angeboten, wie die Modelle auf der Edge-TPU performen.

Coral greift dabei auf Modelle von TensorFlow Version 1 und 2 zurück. Allerdings ist die Auswahl deutlich eingeschränkter als bei TensorFlow ihren *Model Zoo*. Coral selbst bietet nur ein Notebook für die Nutzung mit TensorFlow Version 2 an. Weitere Notebooks sind für TensorFlow Version 1 ausgelegt. Da weder Colab noch die über den JupyterHub der DHBW zur Verfügung gestellte GPU TensorFlow Version 1 unterstützen, konnte nur dieses eine Modell getestet werden. Das Modell benötigte die Annotationen im PASCAL-VOC-Format. Dies ist das ursprüngliche Format der Waschbärbilder von Dat Tran. Unter dem Skriptordner wurde daher ein weiteres Rust-Programm entwickelt, um die CSV-Datei in XML-Dateien zu konvertieren.

Um nun mehrere Modelle und deren Performance vergleichen zu können, müssen das *MobileNet*-Modell von TensorFlow und das *YOLO*-Modell für die Edge-TPU konvertiert werden. Bei *YOLO* muss dafür nur der Exportbefehl mit dem Argument `format=edgetpu` aufgerufen werden. Die Konvertierung war jedoch zunächst erfolglos. Richard Aljaste hat den Grund dafür in einem GitHub-Issue unter [1] gefunden und einen Pull Request mit einer Lösung erstellt. Dieser Pull Request wurde jedoch noch nicht in das offizielle *YOLO*-Repository aufgenommen.

Die Veröffentlichung des Pull Requests erfolgte Ende März und war erst im April ausgereift genug, um angewendet zu werden. Die daraus resultierende Ergebnisse können in 3.4.6 eingesehen werden.

Die Konvertierung aus TensorFlow heraus verlief jedoch erfolglos. Auf ihrer GitHub-Seite wird dieses Problem immer wieder gemeldet. Es scheint, dass die für die Edge-TPU erforderliche Integer-8-Quantisierung wiederholt Schwierigkeiten verursacht. Das Modell wird dabei zwar konvertiert, aber anscheinend können die Enden der Modellarchitektur nicht immer korrekt quantisiert werden. Dies führt zu einer Senkung der Zuversichtlichkeit der Objekterkennung auf unter 5 Prozent und auch die Position der Bounding Boxes wird inkorrekt. Daher kann das Model nicht mit der Integer-8 Quantifizierung verwendet werden.

YOLOv8

Das Framework *Ultralytics* ermöglicht durch einen High-Level Zugriff ein schnelles und einfaches trainieren von Klassifikations-, Segmentierungs- und Objekterkennungsmodellen. Es basiert auf dem *pytorch*-Framework, das bereits einen High-Level-Zugriff auf ML-Modelle und deren Entwicklung ermöglichte. Ultralytics vereinfacht diesen Prozess noch weiter. Für das Training werden neben den Daten nur noch die Pfade, eine YAML-Datei und der zu trainierende Modelltyp benötigt.

Durch den High-Level-Zugriff wird das Deployment auf sämtlichen Plattformen ermöglicht. Mit einem einfachen Exportbefehl kann das trainierte Modell für gängige Frameworks wie TensorFlow, TensorFlow-Lite, PyTorch und ONNX konvertiert und in diesen verwendet werden. [52]

Da das Training zunächst für TensorFlow vorgesehen war, mussten die Bildannotationen von der CSV-Datei in das für das Framework geeignete TXT-Format konvertiert werden. Hierfür wurde ein selbstgeschriebenes Python-Skript im Skriptordner abgelegt.

Das Training auf *Google Colab*

Für das Training der Modelle wurde zunächst *Google Colab* verwendet, jedoch ist die kostenlose GPU dort nicht besonders leistungsstark. Zudem ist die maximale Trainingsdauer auf wenige Stunden begrenzt. Colab beendet sämtliche laufende Prozesse spätestens nach zwölf Stunden und entzieht vorher die kostenlose GPU. Darüber hinaus werden Prozesse nach zehn bis zwanzig Minuten Inaktivität beendet. Um als aktiv zu gelten und die Trainingsdauer zu verlängern, wurde ein Python-Skript entwickelt, das alle paar Minuten einen Mausklick simuliert. Dadurch kann die Trainingsdauer verlängert werden, ohne dass man aktiv am Training teilnehmen muss. Erklärt werden, da man nicht aktiv beim Training etwas tun muss.

Zudem mussten die Bild- und Annotierungsdaten auf Colab hochgeladen werden. Glücklicherweise verfügt Colab über eine Integration von *Google Cloud*, wodurch die Bilddaten in der Cloud gespeichert und für mehrere Sitzungen verwendet werden konnten. Andernfalls müssten die Daten bei jeder Sitzung erneut hochgeladen werden, da sie bei Beendigung einer Sitzung gelöscht werden, sofern sie nicht in der Cloud gespeichert sind. Außerdem wurde festgestellt, dass bei wiederholter hoher Auslastung während einer Sitzung die Zeit bis zur Sitzungsbeendung durch Colab weiter verkürzt wird. Ein effektives Training von Modellen ist dadurch nicht möglich.

Zu einem späteren Zeitpunkt konnte die DHBW über den eigenen JupyterHub eine leistungsstarke GPU zur Verfügung stellen. Mit dieser konnte die maximale Trainingszeit von wenigen Stunden auf mehrere Tage ausgedehnt werden. Zudem verkürzte sich die

Trainingszeit von einigen Stunden auf weniger als 30 Minuten, was zu einem erheblichen Leistungssprung führte.

Allerdings hatte der Wechsel seine Nachteile. Aufgrund unzureichender Berechtigungen zur Installation aller benötigten Bibliotheken - TensorFlow (Objekterkennungs-API), Edge-TPU-Konvertierung und TensorFlow-Lite Model Maker (Google Coral) - war es nur noch möglich, YOLO-Modelle zu trainieren.

3.4.5. Deployment

Für das Deployment wurden verschiedene Modelle und Laufzeitumgebungen erstellt. Daher war es erforderlich, fünf Laufzeitumgebungen einzubinden bzw. zu unterstützen, um die Modelle auswerten zu können. Nachfolgend sind die einzelnen Umgebungen beschrieben. Die verschiedene Ergebnisse bei der Nutzung der Umgebungen werden in Kapitel 3.4.6 beschrieben.

TensorFlow

TensorFlow speichert seine Dateien als .pb-Dateien ab, die mit der normalen TensorFlow-API in das Programm geladen werden können. Eine Installation der Object Detection API ist nur für das Training der Modelle erforderlich.

Anschließend muss das Eingabebild an das Modell angepasst werden. Wenn das Bild zu groß ist, z.B. mit einer 4K-Auflösung, muss es zuerst verkleinert werden. Außerdem benötigen alle Bilder eine zusätzliche Dimension. Diese Dimension kann einfach mit dem `expand_dims`-Befehl von NumPy hinzugefügt werden.

Danach kann das Bild dem Modell zur Verarbeitung übergeben werden. [18]

Die Ausgabe der TensorFlow-Modelle muss dennoch nachbearbeitet werden, da sie alle Erkennungen bis zur eingestellten maximalen Anzahl ausgibt. Bei der Standardkonfiguration von MobileNet wären das bis zu 100 Erkennungen. Um unerwünschte Erkennungen herauszufiltern, z.B. solche mit einer Zuversichtlichkeit von nur 1%, wird die Ausgabe gefiltert und in ein Dictionary gespeichert.

Damit ist die Erkennung mittels der TensorFlow API abgeschlossen und die Ausgabewerte können vom Hauptprogramm ausgewertet werden.

TensorFlow-Lite und Google Coral

Die TensorFlow Lite-Umgebung basiert auf TensorFlow. TensorFlow Lite-Modelle sind jedoch kompakter als die .pb-Modelle von TensorFlow. Allerdings erfordern sie mehr Aufwand für die Initialisierung von Modellen, da beim Laden des Modells explizit das Format der Ein- und Ausgabeparameter aus dem Modell geladen werden müssen. Diese werden anschließend in der Ausführung und Nachbearbeitung verwendet.

Ein Vorteil gegenüber den TensorFlow-Modellen besteht darin, dass sie besser für die Ausführung auf einer CPU geeignet sind. Dies liegt darin, dass sie eine geringere Rechenintensität besitzen. TensorFlow bietet drei Quantisierungstyps dafür an: Float-32, Float-16 und Integer-8.

Die Bezeichnungen beschreiben die Art der Gradientenoptimierung. Während Float-32 und -16 Gleitkommazahlen beschreiben, trifft dies bei Integer-8 nicht zu. Bei Integer-8 handelt es sich um eine Ganzzahl im Bereich von 0 bis 255. Die Zahlen 8, 16 und 32 geben die Anzahl der Bits an, mit denen ein Gradient gespeichert wird. Das Modell hat also den kleinsten Speicherbedarf wenn eine Integer-8 Quantifizierung der ursprünglichen .pb-Modelle vorgenommen wird.

Die Wahl des Quantisierungstyps hängt dabei von den spezifischen Anforderungen und den verfügbaren Ressourcen ab. Float-32 bietet die höchste Genauigkeit, benötigt jedoch mehr Speicherplatz und kann mehr Ausführungszeit benötigen. Float-16 bietet eine gute Balance zwischen Genauigkeit und Ressourcenverbrauch. Integer-8 hat den geringsten Speicherbedarf und kann auf Hardwareplattformen mit geringerer Präzision wie Mikrocontrollern effizienter ausgeführt werden.

Bei der Quantifizierung der Modelle treten allerdings auch Genauigkeitsverluste auf. Dieser Verlust könnte unter Umständen nicht im toleranten Bereich der Anwendung sein. Im Kapitel 3.4.6 werden auch diese Verluste genauer betrachtet. [18]

Auf dem USB-Accelerator Coral können allerdings nur Integer-8 Quantifizierte Modelle betrieben werden. Die anderen Modelle werden daher auf der CPU des Mikrocontrollers ausgeführt. Um den USB-Accelerator nutzen zu können muss allerdings lediglich der TensorFlow-Lite Umgebung dies mitgeteilt werden. [20]

ONNX

Um Objekterkennungsmodell im ONNX-Format auf dem Jetson nutzen zu können gibt es drei Möglichkeiten: OpenCV, ONNX-Runtime und TensorRT. TensorRT wird im nachfolgenden Kapitel beschrieben, da hierzu weitere Schritte nötig sind.

Das ONNX-Format wird für die Nutzung der YOLO-Modelle verwendet. Die trainierten YOLO-Modelle können durch das Ultralytics-Framework direkt exportiert und verwendet

werden. Laut Ultralytics ermöglicht das konvertieren des Modells nach ONNX eine bis zu dreimal schnellere Ausführungszeit. [52]

Exportierte Modelle können auch direkt mit dem Ultralytics-Framework verwendet werden. Dabei gab es jedoch Probleme mit PyTorch, auf dem das Ultralytics-Framework basiert. Das Ultralytics-Framework kann zwar manuell über die Quelldateien auf dem Jetson Nano installiert werden, aber es treten bei der Ausführung Probleme. Um das Framework nutzen zu können, muss zusätzlich zu PyTorch die *torchvision*-Bibliothek installiert werden. Während dem Detektieren von Objekten trat das Problem auf, da das Detektieren von Objekten nur solange funktionierte nur solange, bis ein Objekt erkannt wurde. Sobald ein Objekt erkannt wurde, kam es zu einer Fehlermeldung. Diese besagt, dass nicht die richtige Version von *torchvision* installiert wurde, da sie nicht mit PyTorch kompatibel sei. Das Nvidia-Forum enthält mehrere Einträge zu diesem Thema. Das eigene Nvidia-Forum, dass die Installation von PyTorch und *torchvision* beschreibt, enthält über 1200 Einträge mit vielen Lösungsmöglichkeiten. Allerdings konnten viele davon das Problem nicht lösen, wodurch das nutzen des Frameworks auf dem Jetson Nano für das Abschreckssystem nicht möglich war. [37]

OpenCV verfügt ebenfalls über die Möglichkeit, ONNX-Modelle zu laden und eine Objekterkennung durchzuführen. Allerdings müssen die Bilddaten vorher mit der Funktion `blobFromImage` vorbereitet werden, um sie für die Verarbeitung dem Modell übergeben zu können.

Nachdem das Modell die Bilddaten verarbeitet hat, muss die Ausgabe nachbearbeitet werden. Dabei werden zunächst die Zuversichtlichkeiten der Bounding Boxen extrahiert und überprüft, ob dieser Wert ausreichend hoch ist. Diese Nachbearbeitung ist ähnlich wie bei TensorFlow. Allerdings konnte bei TensorFlow direkt über ein Dictionary auf die Zuversichtlichkeit zugegriffen werden. Beim ONNX-Format muss diese zunächst durch das Auswählen der entsprechenden Ausgabewerte extrahiert werden. Gleicher gilt für die Ermittlung der Bounding Boxen. [38]

Ein ähnliches Verfahren wird bei der Verwendung der ONNX-Runtime angewendet. Der Unterschied zwischen den beiden liegt jedoch in der Ausführungszeit. Die ONNX-Runtime verarbeitet die Eingabewerte schneller als OpenCV, wie aus den Ergebnissen in Kapitel 3.4.6 hervorgeht.

TensorRT

TensorRT ist eine Optimierungs- und Inferenz-Laufzeitbibliothek, die von Nvidia für ihre Grafikkarten entwickelt wurde. Der Zweck dieser Bibliothek besteht darin, DNNs für den Einsatz auf Nvidia-GPUs zu optimieren und die Ausführungszeit zu reduzieren. Dafür benötigt die Bibliothek bereits trainierte Modelle von TensorFlow oder PyTorch.

Anschließend werden auf diesen Modellen eine Reihe von Optimierungen durchgeführt, um die Inferenzgeschwindigkeit und -effizienz zu verbessern, ohne die Genauigkeit der Modelle zu beeinträchtigen.

Um diese Optimierungsmöglichkeit nutzen zu können, müssen bestehende Modelle in das ONNX-Format konvertiert werden. PyTorch-Modelle können ohne weiteres Zutun direkt in das ONNX-Format konvertiert werden. Für die Verwendung von TensorFlow-Modellen muss jedoch ein Zwischenschritt über die Bibliothek `tf2onnx` erfolgen. Es kann auch erforderlich sein, das exportierte Modell im ONNX-Format zu bearbeiten, falls Probleme bei der Optimierung des Modells auftreten. [48]

Wie auch bei TensorFlow mit TensorFlow-Lite gibt es auch hier die Möglichkeit, die Quantisierung auf Float-32, Float-16 oder Integer-8 festzulegen. Das ermöglicht die Reduzierung des Speicherbedarfs und die Verbesserung der Inferenzgeschwindigkeit auf den Nvidia-GPUs, sowie dem Jetson Nano.

Es ist möglich, die mit TensorRT optimierten Modelle sowohl mit TensorRT als auch mit PyTorch zu verwenden. Es wird sogar in [33] empfohlen, TensorRT in Kombination mit PyTorch zu nutzen, da es performanter sein kann.

Diese Möglichkeit kann jedoch aufgrund der bereits genannten Versionsprobleme derzeit nicht angewendet werden. Daher wurde die alternative Methode getestet, nämlich die Verwendung von TensorRT mit PyCuda. Dabei müssen die Eingabe- und Ausgabewerte manuell mit PyCuda gesetzt bzw. gelesen werden. Zusätzlich ist mehr Initialisierungscode erforderlich. Durch anfängliche Fehler bei der Konfiguration dieses Codes kam es daher zu keiner sinnvollen Ausgabe. Diese Probleme konnten jedoch im Nachhinein behoben werden.

Die Vorbereitung der Bilddaten und die Nachbearbeitung der Ausgabe sind nahezu identisch zur Verwendung des ONNX-Formats. Der einzige Unterschied besteht darin, dass die Ausgabe in einer anderen Form vorliegt und daher ein anderes Slicing benötigt wird.

3.4.6. Auswertung der Modelle auf ihre Eignung für das Abschrecksystem

Da für den Einsatz im Abschrecksystem das Modell präzise und effizient Arbeiten muss folgt eine detaillierte Auswertung der verschiedenen Modelle hinsichtlich ihrer Eignung für das Abschrecksystem. Dabei werden die Modelle auf Basis ihrer Inferenzzeit und Zuverlässigkeit analysiert und miteinander verglichen. Im Folgenden werden die Ergebnisse und Erkenntnisse dieser Auswertung präsentiert.

Inferenzzeit der Modelle

Ein wichtiger Aspekt ist die Inferenzzeit, die entscheidend dafür ist, Tiere in Echtzeit erkennen zu können. Daher wurden die Modelle auf verschiedenen Hardwareplattformen getestet, um die Leistung in Bezug auf die Inferenzzeit zu bewerten. Es wurden signifikante Unterschiede sowohl aufgrund der Hardware als auch aufgrund des Modellformats festgestellt. Ein Auszug der Ergebnisse ist in Tabelle 3.1 dargestellt.

Modell	Hardware	Format	Schlechtester Lauf	Bester Lauf	Durchschnitt
MobilNet SSD	I7-10th	pb	147	26	52
MobilNet SSD	I7-10th	TFLITE-f32	106	18	39
MobilNet SSD Threaded	I7-10th	TFLITE-f32	71	0	21
MobilNet SSD Threaded	I7-10th	TFLITE-f16	37	0	17
MobilNet SSD	Jetson	pb	681	104	111
MobilNet SSD Threaded	Jetson	TFLITE-f16	675	0	113
YOLOv8n	I7-10th	pb	47	13	17
YOLOv8n Threaded	I7-10th	TFLITE-f32	37	0	15
YOLOv8n Threaded	I7-10th	TFLITE-f16	31	0	13
YOLOv8n Threaded	I7-10th	TFLITE-i8	28	0	12
YOLOv8n	Jetson	pb	82	48	53
YOLOv8n Threaded	Jetson	TFLITE-f32	226	166	179
YOLOv8n OpenCV	Jetson	ONNX	47	41	42
YOLOv8n ONNX	Jetson	ONNX	43	33	39
YOLOv8n TensorRT	Jetson	ONNX	16	10	13
YOLOv8n Threaded	CM3	TFLITE-f32	686	559	591
YOLOv8n	CM3 + Coral	TFLITE-i8	89	63	68
YOLOv8n	Jetson + Coral	TFLITE-i8	20	16	17
YOLOv5n	I7-10th	pb	75	13	26
YOLOv5n	Jetson	pb	70	43	47
EfficientDet D0	I7-10th	pb	453	183	249
EfficientDet D0	Jetson	pb	1623	492	511

Tabelle 3.1.: Vergleich verschiedener Objekterkennungsmodelle auf Basis ihres Formates und Inferenzzeit in Millisekunden

In der Tabelle ist ein Ausschnitt der durchgeföhrten Inferenzmessungen zu sehen. Zunächst wurde das Modell *MobileNet* von TensorFlow getestet. Der Test wurde auf einem Laptop mit einem Intel i7-Prozessor der zehnten Generation durchgeführt. Dabei wurde festgestellt, dass das Modell durch die Quantisierung erheblich beschleunigt wurde. Die Verwendung der Float-16-Quantisierung führte zu einer Reduzierung der Ausführungszeit auf ein Drittel der ursprünglich benötigten Zeit. Die Beschleunigung war auch Multithreading zu verdanken. In der TensorFlow-Lite-Umgebung kann man nämlich die Anzahl an Threads, die dem Prozess zustehen, erhöhen. Durch die eingestellten vier Treads konnte daher ebenfalls eine Verringerung der Ausführungszeit ermöglicht werden, wie an den Float-32 quantifizierten

Modell zu erkennen ist.

Als derselbe Test auf dem Jetson durchgeführt wurde, konnte nicht die gleiche Beschleunigung beobachtet werden. Dies liegt daran, dass die TensorFlow-Umgebung auf die GPU zugreifen kann, während die TensorFlow-Lite-Umgebung dies nicht kann. Das quantisierte Modell läuft daher auf der CPU des Jetsons, was zu einer deutlichen Steigerung der Inferenzzeit führt. Deshalb haben das ursprüngliche Modell und das float-32-quantisierte Modell nahezu die gleiche durchschnittliche Ausführungszeit von 100 Millisekunden. Eine Inferenzzeit in diesem Bereich ermöglicht keine Echtzeit-Erkennung der Tiere, weshalb eine weitere Untersuchung des Modells nicht länger in Betracht gezogen wurde.

Das zweite TensorFlow-Modell, das *EfficientDet D0*, wurde ebenfalls verworfen. Aufgrund seiner höheren Auflösung lag die Ausführungszeit auf dem Jetson bei über 500 Millisekunden, was für das Abschrecksystem nicht geeignet ist. Die Möglichkeit, die Modelle in das ONNX-Format zu konvertieren und dadurch eine potenzielle Verbesserung der Ausführungszeit zu erzielen, wurde jedoch nicht getestet. Diese Entscheidung wurde aufgrund der Erkenntnisse aus Kapitel 3.4.6, die parallel gewonnen wurden, getroffen.

Durch diese Erkenntnisse sind nur noch die YOLO-Modelle in Frage gekommen.

Auch beim YOLOv8-Modell zeigt sich das gleiche Verhalten wie beim MobileNet-Modell. Auf dem Intel I7-Prozessor wird die Ausführungszeit durch die Quantifizierung stark beschleunigt, jedoch nicht auf dem Jetson. Da aber die Integer-8-Quantifizierung funktioniert, konnte die Ausführungszeit auf dem USB-Accelerator Coral in Kombination mit dem Raspberry Pi CM3 und dem YOLO-Modell getestet werden.

Mit einer durchschnittlichen Ausführungszeit von 68 Millisekunden im Vergleich zu den 17 Millisekunden auf dem Jetson, war das Ergebnis aber eher ernüchternd. Deshalb wurde das Raspberry Pi CM3 endgültig von der Liste der möglichen Mikrocontroller für den Einsatz im Abschrecksystem gestrichen. Der Grund für den extremen Unterschied liegt allerdings am USB-Anschluss des CM3. Das CM3 verfügt nämlich nur über einen USB 2.0-Port, während der Jetson Nano über USB 3.0-Ports verfügt. Dadurch geht der Datenaustausch zwischen dem Mikrocontroller und dem Coral USB-Stick beim Raspberry Pi CM3 langsamer vorstatten.

Das Ultralytics-Framework bot auch die Möglichkeit, ältere YOLO-Modelle wie das YOLOv5-Modell zu trainieren. Dieses Modell wurde ebenfalls kurz getestet, jedoch wurde kein Vorteil gegenüber dem neueren YOLOv8-Modell festgestellt. Daher wurde dem YOLOv5-Modell keine weitere Beachtung geschenkt.

Somit blieb nur noch die Konvertierung in das ONNX- und TensorRT-Format als Option übrig. Das mit TensorRT konvertierte Modell erreichte eine beeindruckende Ausführungszeit von 13 Millisekunden für die Verarbeitung der Eingabewerte. Allerdings ist bei genauerer Untersuchung das Problem aus Kapitel 3.4.5 bemerkt worden. Die Verwendung von

PyCuda in Verbindung mit der Vor- und Nachbearbeitung verschlechterte die Ausführungszeit erheblich. Dadurch erreichte das TensorRT-Modell letztendlich eine ähnliche Ausführungszeit wie das ONNX-Runtime-Modell. Allerdings war der Ressourcenverbrauch des TensorRT-Modells deutlich höher. Nahezu alle vier Kerne und die GPU liefen bei dem Inferenztest auf der maximalen Leistung. In der ONNX-Runtime war der Verbrauch etwas geringer, weshalb dieses Modell eher für die Verwendung im Abschrecksystem geeignet war.

Zuversichtlichkeit der Modelle

Ob die Modelle jedoch für das Abschrecksystem geeignet sind, hängt nicht allein von der Inferenzzeit ab, sondern auch von der Qualität des Modells selbst ab. Diese beschreibt, wie gut das Modell mit Daten aus der realen Welt zurechtkommt, das heißt, ob und wie gut das Modell die Tiere erkennen kann.

Hierbei kommt die in Kapitel 3.4.1 beschriebene Ordnerstruktur zum Tragen. Ein Teil der gesammelten Bilder wurde bereits von den für das Training des Modells vorgesehenen Daten getrennt. Diese Bilder bilden das Testset.

Während des Trainings lernt das Modell Erkennungsmethoden anhand des Trainingssets und überprüft deren Genauigkeit anhand der Daten im Devset. Das Testset bleibt dabei völlig unbekannt und kann daher für eine grobe Abschätzung der Performanz auf unbekannte Daten verwendet werden.

Dafür wird der mean Average Precision (mAP) der Modelle ermittelt. Diese ist eine Bewertungsmetrik, die häufig zur Evaluierung von Objekterkennungssystemen verwendet wird. Dabei wird gemessen, ob die erkannten Objekte tatsächlich korrekt sind. Dabei können die Bounding Boxen auch bis zu einem gewissen Grad schwanken ohne den mAP zu verschlechtern. Dadurch kann eine umfassende Bewertung der Modellleistung durchgeführt werden. [46]

Um diesen Wert zu bestimmen, wird in TensorFlow das Programm TensorBoard verwendet. Mit diesem Tool kann die mittlere durchschnittliche Präzision (mAP) zu jedem Trainingszeitpunkt abgerufen werden. Standardmäßig wird jedoch nur beim letzten Schritt nach dem Training die Evaluierung durchgeführt.

Daher ist in Abbildung 3.9 die Präzision erst nach 50.000 Trainingsschritten dargestellt. Die aus den Daten entnommene mAP für die Präzession der Bounding Boxen im mittleren Bereich liegt demnach bei 36%.

Beim Ultralytics-Framework wird ebenfalls eine ähnliche Art der Evaluation durchgeführt. Dabei erfolgt standardmäßig nach jeder Trainingsepoch eine Auswertung. Die Ergebnisse der Evaluation werden dem Entwickler nach jeder Epoche mitgeteilt. Zudem erstellt

das Ultralytics-Framework automatisch ein Diagramm, das den Verlauf der mittleren durchschnittlichen Präzision (mAP) über die einzelnen Trainingsepochen anzeigt. Dadurch ist es nicht erforderlich, ein zusätzliches Tool zu installieren. Es werden auch weiter Metriken, wie z.B. den F1-Wert berechnet. Daher bietet das Framework eine schnelle und aussagekräftige Auswertung des Modells.

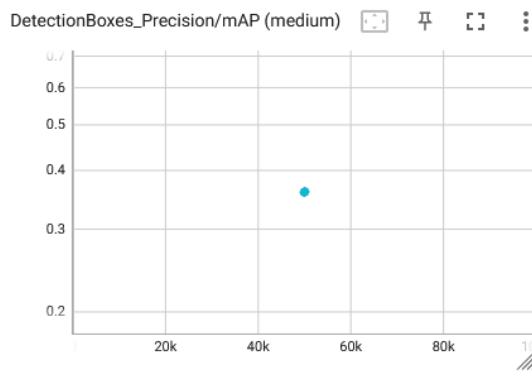


Abbildung 3.9.: mAP des MobileNet-Modells nach 50.000 Trainingsschritten

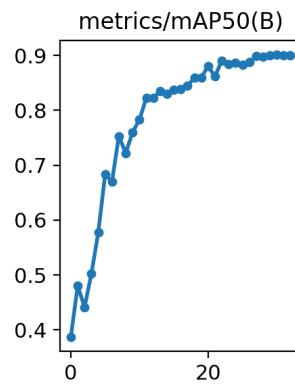


Abbildung 3.10.: mAP des YOLOv8-Modells nach 30 Epochen

Aus den Daten in den Abbildungen 3.9 und 3.10 lässt sich ableiten, dass das YOLO-Modell deutlich schneller eine höhere Präzision erreicht hat. Daher wurde weiterhin mit dem YOLO-Modell gearbeitet.

Beim Vergleich wurde darauf geachtet, dass die Modelle einen ähnlichen Ressourcenverbrauch während des Trainings aufweisen. Beide Modelle wurden für etwa dreieinhalf Stunden auf Google Colab trainiert. Bei gleicher Batch-Größe benötigte das MobileNet-Modell jedoch mehr RAM als das Konkurrenzmodell.

Aus der Auswertung dieser Untersuchung geht hervor, dass das YOLOv8-Modell als Basismodell verwendet werden sollte, da es die Anforderungen an das Abschrecksystem am besten erfüllt.

Da beim Konvertieren und Quantifizieren der Modelle eine Verringerung des mAP-Werts auftreten kann, wurden diese ebenfalls überprüft. Die konvertierten und quantifizierten Modelle können direkt in das Ultralytics-Framework geladen werden, was den Auswertung gegenüber der TensorFlow Modelle vereinfacht. Bei TensorFlow gibt es diese Möglichkeit nämlich nicht direkt. Ein Vergleich der Ergebnisse kann in Tabelle 3.2 eingesehen werden.

Daraus geht hervor, dass die Integer-8-Quantifizierung eine erhebliche Verschlechterung des mAP-Werts verursacht. Angesichts dessen wird die Verwendung von Integer-8-quantifizierten Modellen verworfen.

Beim Anwenden der Quantifizierung auf das ONNX-Modell wird ein ähnliches Verhalten

Format	Quantifizierung	mAP-50	mAP-50
pt	-	95,5	76,5
TFLITE	Float-32	88,9	66,4
TFLITE	Float-16	88,9	66,4
TFLITE	Integer-8	76,4	51,6
ONNX	-	92,3	71,8

Tabelle 3.2.: Vergleich des mAP-Wertes verschiedener YOLOv8-Quantifizierung und Modellformate

beobachtet. Da das Modell jedoch auch für die Verwendung von TensorRT vorgesehen ist, wird eine weitere Quantifizierung nicht verwendet. TensorRT bietet die Möglichkeit, zu bewerten, ob ein Knoten mit Integer-8 quantisiert werden soll. Dabei wird darauf geachtet, ob ein Präzisionsverlust durch die Quantifizierung auftritt. Diese automatische Auswahl der besten Optimierung des Modells ermöglicht die kürzeste Inferenzzeit bei vernachlässigbarem Präzisionsverlust.

Im Anschluss an diese Auswertung wurden zudem mehrere Testmodelle mit unterschiedlichen Augmentierungen erstellt. Diese Augmentierungen sollten die Modelle besser auf den Einsatz im Abschrecksystem abstimmen. Die Problematik wurde bereits in Kapitel 3.4.3 beschrieben.

Anschließend wurden die Modelle anhand einer handverlesenen Auswahl von Bildern getestet. Durch die Augmentierung ist zwar der mAP-Wert, der auf den Online-Datensätzen angewendet wurde, gesunken, aber die Erkennung der Tiere auf den ausgewählten Bildern wurde deutlich verbessert.

3.5. Zusätzliche Softwarekomponenten

Für das Abschrecksystem sind zusätzliche Softwarekomponenten von nötig. Die wichtigsten werden folgend beschrieben.

3.5.1. Erkennung durch Kontur-tracking

Ein Ansatz, um schnell Daten zu sammeln, bestand in der Verwendung von Kontur-Tracking mittels OpenCV. Dadurch konnten die Tiere bereits mit Bounding Boxen annotiert werden. Im Nachhinein musste das Tier zwar manuell bestimmt werden, aber auf diese Weise

konnten schnell repräsentative Daten gesammelt werden, auf denen das Modell evaluiert werden konnte.

Dabei wird OpenCVs `BackgroundSubtractor` verwendet. Diese berechnet eine Änderung von Pixel auf einem Eingabebild gegenüber eines vorherigen Bildes. Das Ergebniss wird in Abbildung 3.5.1 dargestellt.

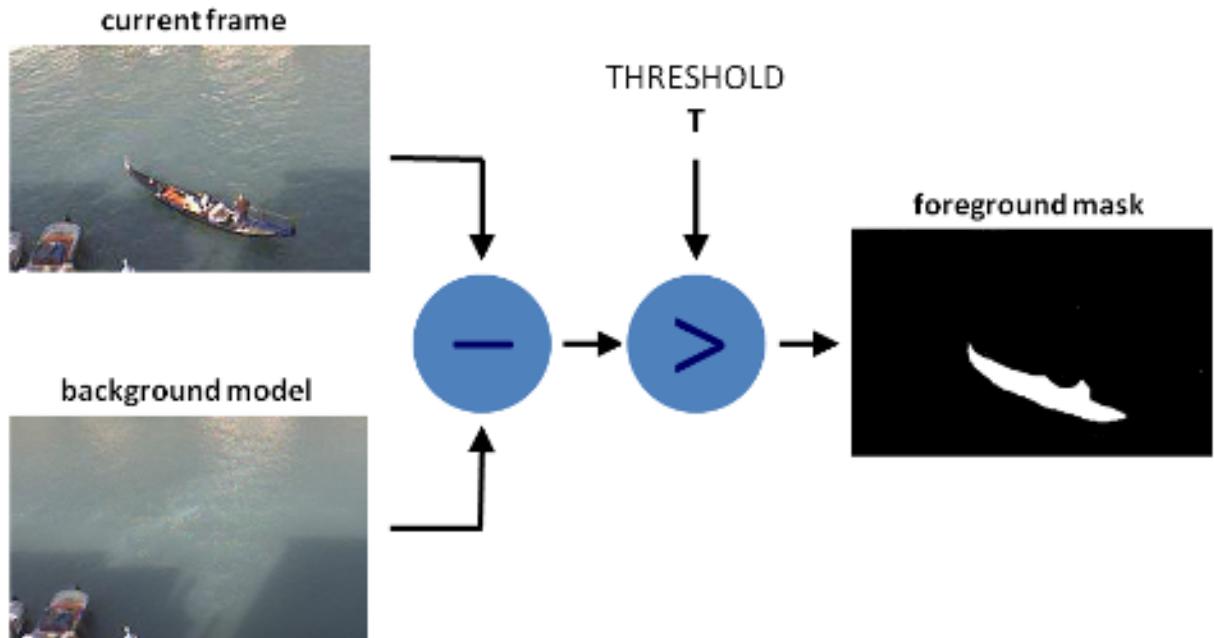


Abbildung 3.11.: Hintergrundsubtraktion auf eine gegebene Eingabe. Quelle: [12]

Dadurch können sich bewegende Objekte leicht erkannt und detektiert werden. Allerdings reicht diese Darstellung allein nicht aus, um eine Bounding Box zu erhalten. Bei der Hintergrundsubtraktion wird viel Rauschen aufgenommen, das nicht als Bounding Box erkannt werden soll. Aus diesem Grund werden morphologische Transformationen durchgeführt. Diese Transformationen werden auf das binäre Bild der Hintergrundsubtraktion angewendet. Mit der Funktion `morphologyEx` werden kleine Änderungen der Hintergrundsubtraktion entfernt.

Nachdem das Hintergrundrauschen entfernt wurde, können die Konturen von Objekten gesucht und für die Erstellung von Bounding Boxen verwendet werden.

Allerdings wurde bei Testläufen festgestellt, dass sich die Boxen häufig überlappen und in den meisten Fällen dasselbe Objekt darstellen. Dies würde den Aufwand für das manuelle Nachbearbeiten der Daten erheblich erhöhen. Aus diesem Grund wurden überlappende Bounding Boxen mit Hilfe eines Intersection over Union (IOU)-Algorithmus und der Slicing-Funktion von NumPy zusammengeführt. Durch diesen Schritt wurde die Anzahl

3. Umsetzung

der Bounding Boxen drastisch reduziert und die Genauigkeit der Bounding Boxes in Bezug auf ihre Position und Größe verbessert.

Trotz der morphologischen Transformationen blieb ein Problem bestehen, da immer noch kleine Bounding Boxes auf den Bildern vorhanden waren. Um dieses Problem zu beheben, wurden die Bounding Boxes mithilfe einer NumPy-Maskierung basierend auf ihrer Größe herausgefiltert. Durch diese Filterung wurden sowohl kleine Objekte als auch weiter entfernte Tiere nicht mehr erfasst und getrackt.

Mit diesen Methoden konnten viele Daten gesammelt werden. Allerdings wurden auch viele Vögel, Wolken oder sich durch den Wind bewegende Büsche durch das Tracking erkannt, was einen hohen Aufwand für das Filtern der Daten bedeutete. Dennoch konnten innerhalb weniger Tage etwa 400 Bilder von Waschbüchsen und Katzen erfolgreich mit Bounding Boxen versehen werden. Die Auswertung der Daten dauerte jedoch aufgrund der bereits genannten Gründe mehrere Tage.

Um das Bearbeiten der Daten zu beschleunigen, wurde ein Skript entwickelt, das mit der Tastatur gesteuert werden kann, um Bounding Boxen zu löschen oder Tierbeschreibungen hinzuzufügen. Die Bounding Boxen sind im Pascal-VOC-Format gespeichert, das eine standardisierte Darstellung von Objekten in Bildern ermöglicht. Das Skript erleichtert somit das schnelle und effiziente Korrigieren der Daten, indem es eine interaktive Methode zur Bearbeitung der Bounding Boxen und ihrer zugehörigen Tierbeschreibungen bereitstellt. Das Skript ermöglicht eine einfache Nutzung zur Bearbeitung der Bounding Boxen und Tierbeschreibungen. Ein Beispiel für die Nutzung ist in der Abbildung dargestellt. Durch die Verwendung der Zahlen 1 bis 4 können Tierbeschreibungen hinzugefügt werden, während durch Drücken der Taste "d" die Bounding Box gelöscht wird. Das Skript erkennt automatisch Dateien, bei denen nach der Bearbeitung keine Bounding Boxen mehr vorhanden sind, und löscht sie entsprechend. Dies erleichtert die effiziente Bearbeitung der Daten, indem nicht mehr benötigte Dateien automatisch entfernt werden.

Dadurch ist der Workflow ebenfalls schneller als das editieren der Daten über *Label-Studio*



Abbildung 3.12.: Anwendung zum herausfiltern der Bounding Boxen

3.5.2. Verbesserung der Bildqualität

Die IMX-219 Kamerasensoren, die für das Abschrecksystem verwendet wurden, unterstützen keinen zuschaltbaren Infrarotfilter wie die für den Raspberry Pi geeigneten OV5647 Sensoren. Dadurch sind die Bilder, insbesondere bei Tageslicht, von einer starken Rotfärbung geprägt. Dieser Unterschied zwischen den Kameras ist in Abbildung 3.5.2 dargestellt. Ein kurzer Testlauf mit einem breit trainierten Modell zeigte an das Erkennungen bei diesem Grad an Zuversichtlichkeit verlieren. Um dies zu verhindern sind daher verschiedene Methoden zur Verbesserung der Bildqualität betrachtet worden.

Zunächst ist die Nachbearbeitung der Bilder in Betracht gezogen worden, um sie natürlicher wirken zu lassen. Aufgrund der Ressourcenintensität der Nachbearbeitungsmethoden wurde davon aber Abstand genommen. Diese Nachbearbeitung würde zu einer erheblichen Verzögerung der Erkennung und Verlangsamung des gesamten Prozesses führen und ist daher für das Abschrecksystem nicht praktikabel.



Abbildung 3.13.: Makeraufnahme von IMX-219 (links) und OV5647 mit Infrarotfilter(rechts)

Daher musste eine andere Methode gefunden werden, die die Bilder natürlicher aussehen lässt und eine geringe Verarbeitungszeit besitzt. Dabei kam *GStreamer* ins Spiel. GStreamer ist ein leistungsstarkes Multimedia-Framework, das eine Vielzahl von Bildverarbeitungsprozessen bietet. Es ermöglicht die Echtzeitverarbeitung von Bildern und Videos, was ideal für das Abschrecksystem ist. Darüber hinaus bietet GStreamer die Möglichkeit der effizienten Parallelverarbeitung auf der CPU oder GPU, um die Ausführungszeit zu minimieren. Die Integration von GStreamer mit OpenCV erfordert nur geringfügige Anpassungen am Code. Ein weiterer Vorteil besteht darin, dass GStreamer bereits auf dem Jetson Nano durch das Betriebssystem installiert ist. Dies erleichtert die Nutzung und Integration in das Abschrecksystem. [21]

Für die Verbesserung der Bildqualität und der Nutzung von GStreamer ist es allerdings nötig Kameras via eines Kommandostrings zu öffnen. Der String ist wie folgend beschrieben:

```
nvarguscamerasrc saturation=0.5 awbblock=true wbmode=5 tnr-mode=2 tnr-
strength=1 ee-mode=2 ee-strength=1 sensor-id=%d ! video/x-raw(memory:NVMM),
width=(int)%d, height=(int)%d, framerate=(fraction)%d/1 ! nvvidconv flip-
method=%d ! video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx
! videoconvert ! video/x-raw, format=(string)BGR ! videobalance hue=-0.12
contrast=1.1 ! appsink drop
```

3. Umsetzung

Durch die Verwendung des Arguments *nvarguscamerasrc* werden die Bildverarbeitungsbefehle an GStreamer übergeben. Diese Befehle umfassen die Verringerung der Farbsättigung, sowie die Verbesserung der Kantenerkennung und die Reduzierung von Bildrauschen. Dadurch wird bereits eine deutliche Verbesserung der Bilder erkennbar. Ein weiteres wichtiges Argument ist *wbmode*, das auf den Wert "5" eingestellt ist. Dies bedeutet, dass GStreamer Bildoptimierungen basierend auf den Lichtverhältnissen die bei Tageslicht auftreten vornimmt. Durch diese Konfiguration kann GStreamer die Bildqualität verbessern und die Auswirkungen der Rotfärbung minimieren.

Nach dieser Konfiguration folgt die Einstellung der Kamera bezüglich ihrer Aufnahme und Wiedergabe der Bilder. Dabei kann direkt die Verkleinerung der Bilder für die Objekterkennung durchgeführt werden.

Der String schließt jedoch mit dem *videobalance*-Argument ab. Hierbei werden noch geringfügige Modifikationen der Bilder in Bezug auf Farbton und Kontrast vorgenommen.

Das Endargument *appsink drop* teilt GStreamer mit, dass er Daten die er von der Kamera bekommt nicht Zwischenspeichern soll. Ist dieses Argument nicht gesetzt erhält man eine Latenz von mehr als einer Sekunde, da zuerst die alten Bilder aus den GStreamer-FIFO-Buffer gelesen werden. Das Resultat dieser Modifikation ist in Abbildung 3.5.2 zu sehen. Dabei ist jedoch die Rotfärbung zu einer Gelbfärbung geworden. Dennoch wird die Bildqualität deutlich erhöht.



Abbildung 3.14.: Vergleich einer Szene, die ohne (links) und mit (rechts) GStreamer-Argumenten aufgenommen ist

Nachdem die Aufnahmeequalität der Kameras verbessert wurde, wurde die Synchronität der Kameras überprüft. Durch sequenzielle Abfragen der Kameras und das Warten auf die Bilder ergab sich das Problem, dass die Bilder nicht synchron aufgenommen wurden und das Warten blockierend war. Um dieses Problem zu lösen, wurde die Abfrage der Bilddaten in einen parallelen Kontext gestellt. Dabei wurde beobachtet, dass die zeitliche Abweichung der Aufnahmen der beiden Kameras von einer Abweichung von 0,5 Sekunden auf weniger als eine Millisekunde reduziert wurde. Die Kameras laufen somit synchronisiert und können durch den parallelen Kontext für die Tiefenberechnung verwendet werden.

3.5.3. Tiefenberechnung

Für die Tiefenberechnung ist es wichtig, genaue Positionen der Tiere in beiden Bildern zu haben. Zusätzlich müssen drei wichtige Daten der Kameras bekannt sein: die Kameraauflösung, die Blickweite und die Brennweite. Mit diesen Informationen und der epipolaren Geometrie aus Kapitel 2.2.4 ist es möglich, jeden Punkt, der auf beiden Kameras sichtbar ist, im dreidimensionalen Raum darzustellen.

Über die Objekterkennung können die Tiere perfekt in beiden Bildern erkannt werden und ihre Positionen können für die Tiefenberechnung verwendet werden. Der Entfernung auf der Z-Achse der Tiere zu den Kameras kann dann mithilfe der in Kapitel 2.2.4 beschriebenen Formel berechnet werden. Für das Zielsystem ist es jedoch auch erforderlich, die Winkel der Tiere in Bezug auf das Zielsystem selbst zu bestimmen. Hierbei werden die Kameraauflösung und die Blickweite benötigt.

Jeder Punkt auf dem Kamerabild kann als Winkel des Objekts zur Kamera selbst betrachtet werden. Mithilfe der Blickweite der Kamera und der Position des Punktes im Bild kann dann, basierend auf der bereits bestimmten Entfernung auf der Z-Achse, die Entfernung auf der X- und Y-Achse bestimmt werden. Dadurch können die Winkel des Ziels im Bezug auf den Kameras ermittelt werden.

Durch weitere Triangulation und Berücksichtigung der Geometrie des Zielsystems können auch die korrekten Winkel für die horizontale und vertikale Ansteuerung des Zielsystems errechnet werden. Dabei werden die Positionen des Tieres auf beiden Kamerabildern, die Entferungen und die Blickwinkel genutzt, um die genauen Steuerungswinkel für das Zielsystem zu bestimmen. Diese Werte ermöglichen es, das Zielsystem präzise auf das erkannte Tier auszurichten.

3.6. Kostenaufstellung

Folgend sind die Kosten und Verwendungszwecke aufgelistet.

Bauteil	Gesamtpreis in € (inkl. Mwst.)	Beschreibung
LED-Scheinwerfer	11.99	Die effizienten LED-Scheinwerfer sind für die Anwendung als Erweiterungsleuchten für das Fahrzeug gedacht. [8] Da die LEDs den hohen Belastungen beim Einsatz am Fahrzeug standhält, werden sie den Anforderungen an einem portablem Abschrecksystem gerecht. Sie werden als Blitzlicht für das Abschrecksystem verwendet.
Membranpumpe	73.35	Membranpumpen sind bei einfachen und kostengünstigen Anwendungen vertreten. Durch den geringen Verschleiß und einfache Wartbarkeit werden sie häufig in Frisch- und Abwascheranwendungen eingesetzt. [53] In der Arbeit wird die Pumpe wegen ihrem geringen Verschleißes und Anschaffungskosten verwendet.
Solarpanel	69.99	Das Solarmodul wird verwendet um die Portabilität und Autarken Eigenschaften der Abschrecksystems zu gewährleisten. Solange Sonnenlicht am Einsatzort verfügbar ist, kann das Abschrecksystem mit ausreichend Energie versorgt werden um die unliebsamen Kleintiere zu erkennen.
Autobatterie	59.90	Kombiniert mit dem Solarmodul versorgt die Batterie das Abschrecksystem mit der nötigen Energie. Tagsüber wird sie mithilfe des Solarmoduls aufgeladen, während sie Nachts das System mit Energie versorgt. [3]

Fortsetzung auf nachfolgender Seite

3. Umsetzung

Fortsetzung von vorheriger Seite

Diverse Kleinteile	35 + X	Diverse Kleinsteile werden in der Arbeit verwendet. Auch die Transistoren, die verwendet werden um die verschiedenen Aktoren an und auszuschalten fallen unter dieser Kategorie. Aber auch die Räder, Schläuche, Kabel, Steckverbindungen und Schrauben werden hier miteinberechnet. Zusätzlich kommen die, für das Abschrecksystem angefertigten 3D-gedruckten Elemente hinzu.
NOIR-Kameras	55.98	In der Studienarbeit sind mehrere Kameras verbaut. Dabei wurde zunächst in hinsicht auf den Raspberry Pi, die Kamerasensoren OV5647 angeschafft worden. Diese sind allerdings nicht mit dem Jetson Nano kompatibel, weshalb weitere Kamerasensoren bestellt werden mussten. Die IMX-219 Kamerasensoren sind zudem teurer, weshalb die Kosten für die Kameras von 39.98€ auf 55.98€ erhöhte.
Jetson Nano	189	Der Jetson Nano bildet den Kern des Abschrecksystems. Über ihn werden die Tiere erkannt und das Zielsystem und die weiteren Abschreckakoren angesteuert. Mit einer GPU ausgestattet ist sie in der Lage Echtzeiterkennung der Tiere zu ermöglichen. Da aufgrund von Ansteuerungsproblemen der Nano nicht in der Lage war die Hardware direkt anzusteuern, musste ein Arduino UNO diese Funktion übernehmen. Dadurch entstanden zusätzliche Kosten von 13.99€
Aluminiumkiste	109 DM	Die Aluminiumkiste ist Witterungsfest und besitzt eine gute Wärmeableitung. Alle Aktoren und Gerätschaften können in ihr vor Witterungsbedingungen geschützt untergebracht werden.

Fortsetzung auf nachfolgender Seite

Fortsetzung von vorheriger Seite

Tabelle 3.3.: Kostenübersicht und Erklärung der Funktion der verschiedenen angeschafften Hardware

Viele Kleinteile, sowie der Lautsprecher konnten aus dem eigenen Bestand verwendet werden. Daher wird unter dem Punkt *Diverse Kleinteile* eine Summe X angegeben. Die Kosten des Abschrecksystem berufen sich nach dieser rechnung auf mindestens 550€. Dabei kamen circa 200€ für das Verwenden eines autoarken Systems auf. Der Aufbau kann in Abbildung 3.6 betrachtet werden. Das 3D-Modell ermöglichte eine übersichtliche Darstellung aller Hardwarekomponenten.

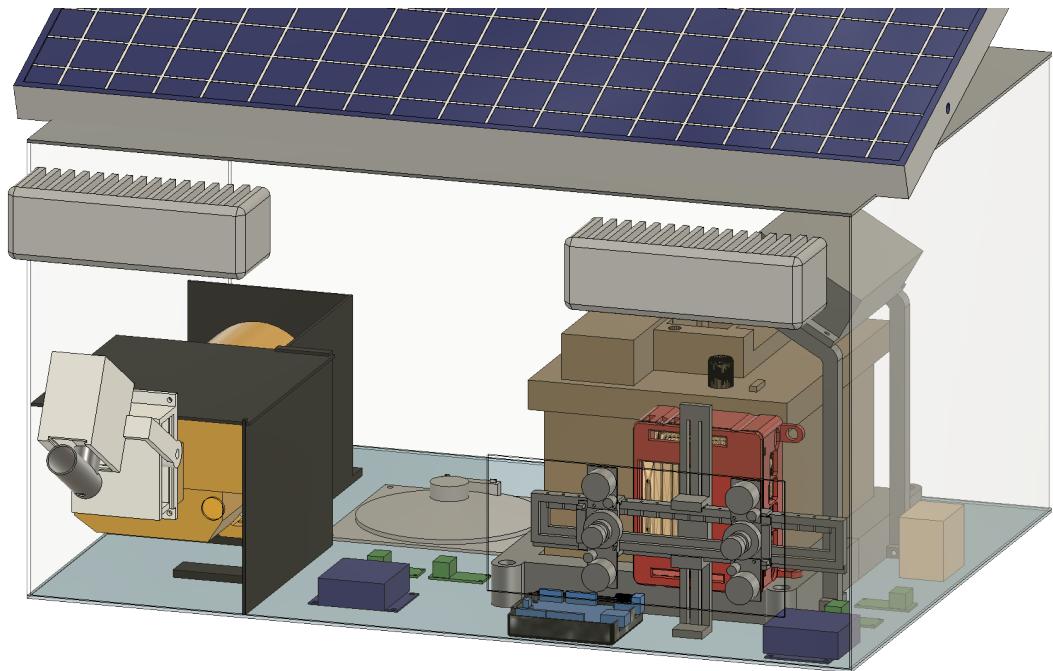


Abbildung 3.15.: Aufbau aller Hardwarekomponenten innerhalb eines 3D-Modells

4. Reflexion und Ausblick

Im Rückblick betrachtet war das Projekt eine Herkulesaufgabe. Anfangs stand aufgrund von Lieferengpässen kein Mikrocontroller mit ausreichender Leistung zur Verfügung, weshalb alternative Hardware gesucht und getestet wurde. Die Mikrocontroller konnten allerdings nicht für das Abschreckssystem verwendet werden, da die Objekterkennung auf den Geräten nicht in Echtzeit verfügbar war.

Als der Jetson Nano als letzte verfügbare Option für den Mikrocontroller-Einsatz verfügbar war, stellte sich heraus, dass er nicht mehr aktiv unterstützt wurde und es bei der Verwendung mit verschiedenen Anwendungen zu zahlreichen Problemen kam.

Dies liegt daran, dass der Jetson Nano ein veraltetes Modell der Jetson-Reihe ist, wodurch das Betriebssystem noch auf Ubuntu 18.04 basiert. Viele Python-Anwendungen sind jedoch auf neueren Python-Versionen ausgelegt, die auf dem Jetson nicht verfügbar sind. Obwohl ein Upgrade der Python-Version von 3.6 auf neuere Versionen möglich ist, kann die Grafikkarte häufig nicht mehr für die rechenintensiven Objekterkennungsmodelle genutzt werden, was die Verwendung des Jetsons für die Objekterkennung erschwert. Viele Bibliotheken mussten direkt aus den Quelldateien installiert werden. Zudem musste der Jetson zu Beginn häufig neu geflasht werden, da Installationsschritte einer Anwendung die Nutzung einer anderen negativ beeinträchtigten.

Deshalb wurde ein Installationsskript erstellt, das die wichtigsten Bibliotheken und Anwendungen erfolgreich und ohne Konflikte installiert. Allerdings konnte das Versionierungsproblem zwischen PyTorch und torchvision innerhalb der gegebenen Zeit nicht gelöst werden. Zudem konnten trotz Modifikation viele Bibliotheken nicht erfolgreich auf dem Jetson installiert werden. Dadurch ist viel Zeit darauf verwendet worden, die wichtigsten Bibliotheken und Anwendungen korrekt zum Laufen zu bringen.

Die Einrichtung des Jetson Nano mit Hilfe des Installationsskripts hat alleine schon fast drei Stunden gedauert. Um jedoch die neueste Version von OpenCV nutzen zu können, waren zusätzliche manuelle Schritte erforderlich, was die Gesamtzeit auf etwa 8 Stunden erhöhte.

Der Jetson Nano ermöglichte trotz der genannten Herausforderungen eine Tiererkennung innerhalb 13 Millisekunden unter Verwendung von TensorRT. Jedoch verlängerte sich diese Zeit durch verschiedene Vor- und Nachverarbeitungsschritte. Auch die Integration der Stereo-Vision und die Ansteuerung der Hardware führten zu einer Erhöhung der Inferrenzzeit um den Faktor 10. Um die Latenz zu verringern, wurden deshalb nahezu

alle Prozesse durch Multithreading beschleunigt. Allerdings konnte keine signifikante Verbesserung der Geschwindigkeit festgestellt werden.

Eine Verbesserungsmöglichkeit könnte das Auslagern von Codeabschnitten oder des gesamten Codes nach C++ sein. Artikel, die sich mit der Inferenzzeit von Objekterkennungsmodellen beschäftigen, behaupten, dass sie dadurch eine Reduzierung der Inferenzzeit beobachten konnten.

Zudem könnte es auch interessant sein, Nvidias DeepStream SDK zu nutzen. Dadurch sollen ebenfalls geringe Inferenzzeiten möglich sein.

Eine weitere Reduzierung der Inferenzzeit könnte durch die Änderungen an dem Stereo-Vision-Aufbau und der Tiefenbestimmung erreicht werden. Die derzeitige Implementierung sieht vor, dass die Objekterkennung gleichzeitig auf beiden Kameras durchgeführt wird. Bereits bei einer Ausführung der Objekterkennung auf einem Bild werden die Ressourcen des Jetsons nahezu vollständig aufgebraucht. Daher wäre es sinnvoll, das Ermitteln der Position des Tieres auf dem zweiten Bild durch alternative Methoden zu ersetzen.

Zusammenfassend lässt sich sagen, dass das Projekt eine große Herausforderung darstellte, sei es bei der Umsetzung oder der Beschaffung der Hardware. Das Projekt war ständig von der Suche nach Problemlösungen überschattet. Dennoch konnte ein funktionierender Prototyp fertiggestellt werden.

Während des Projekts konnte viel über Unix-Systeme, maschinelles Lernen, Objekterkennung und Code-Optimierung gelernt werden. Diese erworbenen Fähigkeiten werden sicherlich bei der Fortführung des Projekts von Nutzen sein. Da der Waschbär im Garten weiterhin sein Unwesen treibt, besteht persönliches Interesse daran, das Projekt abzuschließen und es auf das Optimum auszurichten. Nämlich der Vertreibung des Waschbären und anderen unliebsamen Kleintieren.

Literatur

- [1] Richard Aljaste. *Tflite and edgetpu exports fail on TensorFlow SavedModel*. 2023. URL: <https://github.com/ultralytics/ultralytics/issues/1185>.
- [2] Hamid R. Alsanad u. a. „YOLO-V3 based real-time drone detection algorithm“. In: *Multimedia Tools and Applications* 81 (2022), S. 26185–26198. doi: [10.1007/s11042-022-12939-4](https://doi.org/10.1007/s11042-022-12939-4).
- [3] amazon. *Autobatterie 12V 50Ah 480A/EN BlackMax Starter*. URL: [https://www.amazon.de/Autobatterie-12V-440-BlackMax-ersetzt/dp/B01M4JDLZ2/ref=sr_1_2?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1J6NCJAA6BK2P&keywords=bleiakku%2B12v%2B50ah%2Caps%2C105&sr=8-2&th=1](https://www.amazon.de/Autobatterie-12V-440-BlackMax-ersetzt/dp/B01M4JDLZ2/ref=sr_1_2?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1J6NCJAA6BK2P&keywords=bleiakku%2B12v%2B50ah&qid=1667134113&qu=eyJxc2MiOiIyLjkwlIwicXNhIjoiMS4zMCI&InFzcCI6IjAuMDAifQ%3D%3D&sprefix=bleiakku%2B12v%2B50ah%2Caps%2C105&sr=8-2&th=1).
- [4] amazon. „AZDelivery XL4016E1 DC-DC Step Down Buck Converter, Spannungsregler, Spannungswandler Einstellbar 5-40V auf 1,2-36V 12A Step Down Board kompatibel mit Arduino inklusive E-Book!“ In: (2023). URL: https://www.amazon.de/AZDelivery-Spannungsregler-Spannungswandler-Einstellbar-kompatibel/dp/B08T12J8KL/ref=sr_1_4?keywords=step%2Bdown%2Bconverter%2B10a&qid=1669129106&s=industrial&sPrefix=step%2Bdown%2Bconverter%2B%2Cindustrial%2C123&sr=1-4&th=1.
- [5] amazon. „BOJACK schottky dioden 10SQ045 (10A 45V) axial 10SQ045 (10 Amp 45 Volt) für solarpanel serien parallele reflow schutzdiode (Packung mit 20 Stück)“. In: (2022). URL: https://www.amazon.de/BOJACK-schottky-solarpanel-parallele-schutzdiode/dp/B07SQCCZFH/ref=sr_1_18?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=2WQIJIQTL24QW&keywords=schottky+diode+TC-SB340&qid=1669630149&sPrefix=schottky+diode+tc-sb340%2Caps%2C87&sr=8-18.
- [6] amazon. *CQUANZX 5 STÜCKE DC 5 V-36 V 15 A 400 W Dual-Hochleistungs-MOSFET-Triggerschalter-Antriebsmodul 0-20 kHz PWM-Anpassung Elektronische Schaltersteuerplatine*. 2022. URL: https://www.amazon.de/CQUANZX-Dual-Hochleistungs-MOSFET-Triggerschalter-Antriebsmodul-PWM-Anpassung-Elektronische-Schaltersteuerplatine/dp/B07VRCXGFY/ref=sr_1_6?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=ZTG49FQ3EVNJ&keywords=RLB3036PBF+MOSFET&qid=1669628922&sPrefix=rlb3036pbf+mosfet%2Caps%2C76&sr=8-6.

- [7] amazon. *LM386 Super Mini Verstärkerplatine, 3V-12V Leistungsverstärker Rauscharmer Stromverbrauch für Lautsprecher Soundsystem DIY lm386 Verstärker*. 2023. URL: https://www.amazon.de/LM386-Super-Verst%C3%A4rker-Board-V-12-Power/dp/B07BW8GRTD?pd_rd_w=VoNcs&content-id=amzn1.sym.83e2f3cd-483e-4714-8593-69ff6b513968&pf_rd_p=83e2f3cd-483e-4714-8593-69ff6b513968&pf_rd_r=G63SWJES44QA73GCZQ8J&pd_rd_wg=e8igU&pd_rd_r=43beb5a0-f240-4f1a-9f88-d04814047f30&pd_rd_i=B07BW8GRTD&psc=1&ref_=pd_bap_d_grid_rp_0_1_ec_t.
- [8] amazon. *NAIZY 2 x 18 W LED Work Light Square Offroad Floodlight Work Light*. URL: https://www.amazon.de/gp/product/B09MQG1W7Q/ref=ppx_yo_dt_b_asin_title_o08_s04?ie=UTF8&psc=1.
- [9] amazon. *RC Servo 25kg Digital Lenkservo - Standard Servo Wasserdicht High Torque Digitaler Metall Servomotor Waterproof Servos 180 Grad*. 2023. URL: https://www.amazon.de/dp/B08GLGJRK6/ref=promo_pdapsinfo?ie=UTF8&&psc=1&&m=AS8JK4HNTE2NO.
- [10] ArduCam. „Arducam Multi Camera Adapter Module V2.1 for Raspberry Pi 4 B, 3B+, Pi 3, Pi 2, Model A/B/B+, Work with 5MP or 8MP Cameras - Arducam“. In: (2023). URL: <https://www.arducam.com/product/multi-camera-v2-1-adapter-raspberry-pi/>.
- [11] Md. Shohel Arman u. a. „Detection and Classification of Road Damage Using R-CNN and Faster R-CNN: A Deep Learning Approach“. In: *Cyber Security and Computer Science*. Hrsg. von Touhid Bhuiyan, Md. Mostafijur Rahman und Md. Asraf Ali. Springer International Publishing, 2020, S. 730–741. ISBN: 978-3-030-52856-0. DOI: <https://doi.org/10.1007/978-3-030-52856-0>.
- [12] OpenCV Domenico Daniele Bloisi. „OpenCV: How to Use Background Subtraction Methods“. In: (2023). URL: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html.
- [13] Sabine Bschorer und Konrad Költzsch. *Technische Strömungslehre: Mit 262 Aufgaben und 31 Beispielen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2021. ISBN: 978-3-658-30407-2. DOI: [10.1007/978-3-658-30407-2_6](https://doi.org/10.1007/978-3-658-30407-2_6).
- [14] bussgeldkatalog.org. *Marderschreck Vergleich 2022: Aktuelle Empfehlungen im Überblick*. 2022. URL: <https://testsieger.bussgeldkatalog.org/marderschreck/>.
- [15] Google Colab. *Willkommen bei Colaboratory - Colaboratory*. 2023. URL: https://colab.research.google.com/?utm_source=scs-index.
- [16] Dlf.Mytyta. „Auto-tracking Water Blaster : 9 Steps - Instructables“. In: (2018). URL: <https://www.instructables.com/Auto-tracking-Water-Blaster/>.

- [17] nvidia forum. „does Jetson Nano support CSI camera with sensor ov5647?“ In: (2021). URL: <https://forums.developer.nvidia.com/t/does-jetson-nano-support-csi-camera-with-sensor-ov5647/74911>.
- [18] TensorFlow Model Garden. *TensorFlow Model Garden*. <https://github.com/tensorflow/models>. 2020.
- [19] Google. „Open Images V7“. In: (2023). URL: https://storage.googleapis.com/openimages/web/factsfigures_v7.html.
- [20] Coral Google. *Coral*. 2023. URL: <https://coral.ai/>.
- [21] GStreamer. *GStreamer: features*. 2023. URL: <https://gstreamer.freedesktop.org/features/index.html>.
- [22] Plattenspieler Guru. „Wie viel Leistung sollte ein Verstärker haben?“ In: (2023). URL: <https://www.plattenspieler-guru.de/wie-viel-leistung-sollte-ein-verstaerker-haben/>.
- [23] Inc. Heartex. *Open Source Data Labeling / Label Studio*. 2023. URL: <https://labelstud.io/>.
- [24] Helen. „DC Motor vs Stepper Motor vs Servo Motor - Which to choose?“ In: (2019). URL: <https://www.seeedstudio.com/blog/2019/04/01/choosing-the-right-motor-for-your-project-dc-vs-stepper-vs-servo-motors/>.
- [25] Lee Jackson. „Synchronized? Issues& Solutions about the 2 Camera Connectors (J13& J49) on Jetson Nano B01 Dev Kit - Arducam“. In: (2020). URL: <https://www.arducam.com/jetson-nano-stereo-camera-sync-issue-arducam/>.
- [26] Nikola Jelic. *Strange behaviour with Jetson Nano DevKit PWM signal*. 2022. URL: <https://forums.developer.nvidia.com/t/pwm0-on-jetson-nano-rel-32-5-not-working/204037/26>.
- [27] Karl Jousten und Jürgen Dirscherl. „Oszillationsverdrängerpumpen“. In: *Handbuch Vakuumtechnik*. Hrsg. von Karl Jousten. Springer Fachmedien Wiesbaden, 2018, S. 1–20. ISBN: 978-3-658-13403-7. DOI: [10.1007/978-3-658-13403-7_13-1](https://doi.org/10.1007/978-3-658-13403-7_13-1).
- [28] katzenverscheuchen. „Wasser Katzenschreck Sprinkleranlage - Katzen Verscheuchen“. In: (2023). URL: <https://www.katzenverscheuchen.de/wasser-katzenschreck/>.
- [29] Johann Kodnar. „1/2 Zoll vs. 3/4 Zoll Gartenschlauch - Wann benötigt man welchen und welche Unterschiede gibt es? - Gartenbewässerungs-Ratgeber mit Do-it-yourself-Tipps zur Planung, Kauf und Installation“. In: (2023). URL: <https://www.bewaesserung-selbst-bauen.de/1-2-zoll-vs-3-4-zoll-gartenschlauch-wann-benoetigt-man-welchen-und-welche-unterschiede-gibt-es.html>.

- [30] Raja Kraus. *Mitteldeutsche Landkreise reagieren auf Wasserknappheit*. 2022. URL: <https://www.mdr.de/nachrichten/deutschland/panorama/trockenheitswasserentnahmeverbot-100.html>.
- [31] Hamid Laga u.a. *3D Shape Analysis : Fundamentals, Theory, and Applications*. John Wiley & Sons, Incorporated, 2019. ISBN: 9781119405191. URL: <http://ebookcentral.proquest.com/lib/dhbw-stuttgart/detail.action?docID=5625418>.
- [32] Raspberry Pi (Trading) Ltd. *DATASHEET Raspberry Pi Compute Module 3+*. Techn. Ber. Raspberry Pi, 2019.
- [33] Triple Mu. *GitHub - triple-Mu/YOLOv8-TensorRT: YOLOv8 using TensorRT accelerate !* 2023. URL: <https://github.com/triple-Mu/YOLOv8-TensorRT>.
- [34] nvidia. *DATA SHEET: NVIDIA Jetson Nano System-on-Module*. Techn. Ber. 2022. URL: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano_DataSheet_DS09366001v1.1.pdf?gfELa_1CRx9BgcBwEJ8Mk4oPbG4iiMUpANL5j0mWG-066jBzZy4znERTvSy84iRVGuQTSNNHCMpzEu_DRePb-Xip_qqt46ScuhZPdym0TK_3idcTKCx5QzTLEm1WWK9qjgIHXexIJGhsYXsPP0Ik-7vYN9Mu-bjiL6FXkyPQ061e_1LLsDlF35X35-Zuzg==.
- [35] nvidia. *GitHub - NVIDIA/jetson-gpio: A Python library that enables the use of Jetson's GPIOs*. 2023. URL: <https://github.com/NVIDIA/jetson-gpio>.
- [36] nvidia. *Jetson Nano Entwicklerkit für KI und Robotik / NVIDIA*. 2023. URL: <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-nano/>.
- [37] nvidia. *PyTorch for Jetson - Jetson & Embedded Systems / Jetson Nano - NVIDIA Developer Forums*. 2023. URL: <https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048/1294>.
- [38] OpenCV. *OpenCV: DNN-based Face Detection And Recognition*. 2023. URL: https://docs.opencv.org/4.x/d0/dd4/tutorial_dnn_face.html.
- [39] Jon Peddie. *The History of the GPU - Eras and Environment*. Cham: Springer International Publishing, 2022. ISBN: 978-3-031-13581-1. DOI: [10.1007/978-3-031-13581-1_2](https://doi.org/10.1007/978-3-031-13581-1_2).
- [40] radxa. *Radxa Rock3*. 2023. URL: <https://wiki.radxa.com/Rock3/>.
- [41] Christine Riel. *So können Sie Waschbären vertreiben*. URL: <https://www.gartenjournal.net/waschbaer-vertrieben>.
- [42] Seaflo. *SEAFLO Washdown Pumps for Boats / 17.0 LPM 12V Quick Connect On Demand Diaphragm Water Pump*. 2023. URL: <http://www.seaflo.com/en-us/product/detail/608.html>.

- [43] Shoichi Shimizu u. a. „A PSEUDO STEREO VISION METHOD FOR UNSYNCHRONIZED CAMERAS“. In: (2004). URL: https://www.researchgate.net/publication/253608837_A_PSEUDO_STEREO_VISION_METHOD_FOR_UNSYNCHRONIZED_CAMERAS.
- [44] StereoPi. „StereoPi | StereoPi - DIY stereoscopic camera based on Raspberry Pi“. In: (2023). URL: <https://stereopi.com/>.
- [45] Leonhard Stiny. *Aktive elektronische Bauelemente: Aufbau, Struktur, Wirkungsweise, Eigenschaften und praktischer Einsatz diskreter und integrierter Halbleiter-Bauteile*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019. ISBN: 978-3-658-24752-2. DOI: [10.1007/978-3-658-24752-2_11](https://doi.org/10.1007/978-3-658-24752-2_11).
- [46] Richard Szeliski. *Computer Vision*. Springer Viehweg, 2022. DOI: [10.1007/978-3-030-34372-9](https://doi.org/10.1007/978-3-030-34372-9).
- [47] TensorFlow. „Training Custom Object Detector — TensorFlow 2 Object Detection API tutorial documentation“. In: (2023). URL: <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html>.
- [48] Tensorrt. „Developer Guide :: NVIDIA Deep Learning TensorRT Documentation“. In: (2023). URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#overview>.
- [49] Juan Terven und Diana Cordova-Esparza. *A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond*. 2023. arXiv: [2304.00501 \[cs.CV\]](https://arxiv.org/abs/2304.00501).
- [50] Attaching Them To. *Downloading and Evaluating Open Images — FiftyOne 0.21.0 documentation*. 2023. URL: https://docs.voxel51.com/tutorials/open_images.html.
- [51] Dat Tran. „How to train your own Object Detector with TensorFlow’s Object Detector API“. In: (2017). URL: <https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9>.
- [52] Ultralytics. *Ultralytics YOLOv8 Docs*. 2023. URL: <https://docs.ultralytics.com/>.
- [53] Gerhard Vetter. „Stand und Trends bei der Entwicklung leckfreier oszillierender Verdrängerpumpen“. In: *Chemie Ingenieur Technik* 57.3 (1985), S. 218–229. DOI: <https://doi.org/10.1002/cite.330570306>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cite.330570306>.
- [54] Welectron. *Waveshare 12076 RPi IR-CUT Camera, 25,90 € - Welectron*. 2023. URL: <https://www.welectron.com/#>.

- [55] Westfalia. „Maßnahmen, um Waschbären aus Haus und Garten zu vertreiben bei Westfalia Versand Deutschland“. In: (2023). URL: https://www.westfalia.de/static/informationen/ratgeber/garten/tiere_im_garten/waschbaeren_vertreiben.html.
- [56] Elaine Wu. „Comparing Raspberry Pi Compute Module 4(CM4) and CM3+, What has been changed from CM3+? - Latest Open Tech From Seeed“. In: (2020). URL: <https://www.seeedstudio.com/blog/2020/10/30/comparing-raspberry-pi-compute-module-4cm4-and-cm3-what-has-been-changed-from-cm3/>.

Anhang

A. Arduino UNO Testcode

```
1 #include <Servo.h>
2
3     int horizontalServoPin = 10;
4     Servo horizontalServo;
5     float horizontalServoPos = 90;
6     float horizontalServoStepSize = 20;
7
8     int verticalServoPin = 9;
9     Servo verticalServo;
10    float verticalServoPos = 90;
11    float verticalServoStepSize = 10;
12
13    int powerServoPin = 7;
14
15    int powerLightPin = 6;
16
17    int powerPumpPin = 4;
18
19    int soundSignalPin = 3;
20    unsigned long currentSoundFrequency = 15000;
21    unsigned long UpperSoundFrequency = 44000;
22    unsigned long lowerSoundFrequency = 15000;
23    int frequencyStepSize = 100;
24
25    int powerSoundPin = 2;
26
27    void setup() {
28        // put your setup code here, to run once:
29        Serial.begin(9600);
```

```
30     horizontalServo.attach(horizontalServoPin);
31     verticalServo.attach(verticalServoPin);
32     pinMode(powerServoPin, OUTPUT);
33     pinMode(powerLightPin, OUTPUT);
34     pinMode(powerPumpPin, OUTPUT);
35     pinMode(powerSoundPin, OUTPUT);
36     pinMode(soundSignalPin, OUTPUT);
37     digitalWrite(powerServoPin, HIGH);
38 }
39
40 void loop() {
41   if (millis() % 500 == 0) {
42     moveSevo(horizontalServo, 0, 180, &
43               horizontalServoStepSize, &horizontalServoPos);
44     Serial.print("Moving horizontal Servo to position ");
45     Serial.println(horizontalServoPos); // Printing
46               the position of the horizontal servo
47   }
48   if (millis() % 1000 == 0) {
49     moveSevo(verticalServo, 45, 135, &
50               verticalServoStepSize, &verticalServoPos);
51     Serial.print("Moving vertical Servo to position ");
52     Serial.println(verticalServoPos); // Printing the
53               position of the vertical servo
54   }
55   if (millis() % 200 == 0) {
56     digitalWrite(powerLightPin, !digitalRead(
57                 powerLightPin));
58     Serial.println("Toggling powerLightPin."); // A
59               message indicating powerLightPin has been
60               toggled
61   }
62   if (millis() % 5000 == 0) {
63     digitalWrite(powerServoPin, !digitalRead(
64                 powerServoPin));
```

```
57     Serial.println("Toggling powerServoPin."); // A
      message indicating powerServoPin has been
      toggled
58 }
59 if (millis() % 1000 == 0) {
60     digitalWrite(powerPumpPin, !digitalRead(
61         powerPumpPin));
62     Serial.println("Toggling powerPumpPin."); // A
      message indicating powerPumpPin has been
      toggled
63 }
64 if (millis() % 10000 == 0) {
65     digitalWrite(powerSoundPin, !digitalRead(
66         powerSoundPin));
67     Serial.println("Toggling powerSoundPin."); // A
      message indicating powerSoundPin has been
      toggled
68 }
69 if (millis() % 100 == 0) {
70     if (currentSoundFrequency < lowerSoundFrequency)
71     {
72         currentSoundFrequency = lowerSoundFrequency;
73         frequencyStepSize *= -1;
74     } else if (currentSoundFrequency >
75         UpperSoundFrequency) {
76         currentSoundFrequency = UpperSoundFrequency;
77         frequencyStepSize *= -1;
78     }
79     tone(soundSignalPin, currentSoundFrequency);
80     currentSoundFrequency += frequencyStepSize;
81     Serial.print("Playing sound with frequency: ");
82     Serial.println(currentSoundFrequency); //
      Printing the frequency of the sound played
83 }
84 }
85
86 void moveSevo(Servo motor, uint8_t minPos, uint8_t
maxPos, float* stepSize, float* currentPosition) {
```

```
84     if (*currentPosition < minPos) {  
85         *currentPosition = minPos;  
86         *stepSize *= -1;  
87     } else if (*currentPosition > maxPos) {  
88         *currentPosition = maxPos;  
89         *stepSize *= -1;  
90     }  
91     motor.write(*currentPosition);  
92     *currentPosition += *stepSize;  
93 }
```

Listing A.1: Arduino Test Code um alle Funktionen zu überprüfen.