UiA University of Agder

Master

# Development of System for Deterrence of Animals using Artificial Intelligence

OLE HENRIK HAAKSTAD

SUPERVISOR
Andreas Klausen, Morten Kjeld Ebbesen

**University of Agder, 2021**
Faculty of Engineering and Science
Department of Engineering Sciences

# Abstract

This thesis is part of the Mechatronics masters program at the University of Agder (UiA). It describes the development of an animal deterrence system, using computer vision with embedded systems for edge computing.

Household cats can run freely around in neighborhoods, but all neighbors may not appreciate it. This project seeks to develop a harmless and accurate system for keeping cats away.

The solution is to use a state-of-the-art object detector to look for cats in an area. Using an embedded system to run the object detector and tracking method. Design a two-axis turret with servo motors and a camera.

The object detector used for this project was YOLOv4-tiny. It was trained to detect cats, dogs, humans, and hedgehogs. The YOLOv4-tiny object detector was implemented on an NVIDIA Jetson Nano development board and ran at an average of 23 FPS. Robot Operating System (ROS) was used to run the control algorithm servo motors and the object detector. A two-axis turret, which uses two servo motors, was designed and built to make the camera track the cat. A two-axis turret, which uses two servo motors, was designed and built to make the camera track the cat. A component housing was also designed and built to have the whole system in a small enclosure.

# Contents

**Abstract** **i**

**1 Introduction** **1**
1.1 Challenges . . . 1
1.2 Goals . . . 2
1.3 Product Specification . . . 2

**2 Theory** **5**
2.1 Artificial Intelligence . . . 5
2.2 Neural networks . . . 5
2.3 Convolutional Neural Networks (CNN) . . . 6
2.3.1 Convolution . . . 6
2.3.2 Convolution Layer . . . 6
2.4 You Only Look Once (YOLO) . . . 7
2.4.1 Intersection over Union . . . 8
2.4.2 Detection . . . 8
2.4.3 Loss Function . . . 9
2.4.4 YOLOv4 . . . 9
2.4.5 The Architecture of YOLOv4 . . . 10
2.4.6 YOLOv4-tiny . . . 11
2.4.7 Calculating accuracy . . . 11
2.5 Simple Online and Real-time Tracking . . . 12
2.6 Edge Computing . . . 13
2.7 State of the Art . . . 14

**3 Methods** **15**
3.1 Object Detector . . . 15
3.1.1 Datasets . . . 15
3.1.2 Darknet . . . 18
3.1.3 YOLOv4 Training Setup . . . 18
3.1.4 Overfitting . . . 20
3.2 Mechanical System/Hardware Setup . . . 21
3.2.1 Nvidia Jetson Developer Kit . . . 22
3.2.2 Raspberry Pi NoIR Camera . . . 25
3.2.3 PDI-1171MG Servo Motor . . . 25
3.2.4 Adafruit PCA9685 Servo Driver . . . 26
3.2.5 Arduino 5mW laser . . . 26
3.3 Turret Design . . . 28
3.3.1 Operating Area . . . 29
3.4 Control Software . . . 29
3.4.1 TensorRT . . . 29
3.4.2 Transferring Data . . . 30
3.4.3 Object Detector Implementation . . . 30
3.4.4 ROS . . . 30

3.4.5 Control Algorithm . . . 31
3.4.6 DeepSORT . . . 33
3.4.7 Testing Accuracy of Object Detector . . . 33

**4 Results** **34**
4.1 Object Detector . . . 34
4.1.1 Real-life Detection . . . 35
4.1.2 Raspberry Pi NoIR . . . 37
4.1.3 Frames Per Second - Jetson Nano . . . 37
4.2 Object Tracking . . . 37
4.3 Turret Design . . . 38
4.4 Final Specification . . . 39
4.5 Video of results . . . 41

**5 Discussions** **42**
5.1 Object Detector . . . 42
5.1.1 Dataset . . . 42
5.1.2 Real life evaluating of object detector . . . 43
5.1.3 YOLOv4-tiny . . . 43
5.1.4 Object Detecting at Night . . . 43
5.1.5 YOLOv5, PP-YOLO, and Other Versions . . . 43
5.2 Object Tracking . . . 44
5.2.1 Control Algorithm . . . 44
5.2.2 Importance of FPS . . . 44
5.2.3 DeepSORT . . . 44
5.2.4 ROS and Implementing Software . . . 45
5.3 Hardware . . . 45
5.3.1 Nvidia Jetson Nano . . . 45
5.3.2 Turret System . . . 45
5.4 Water Gun System . . . 46

**6 Conclusions** **47**
6.1 Object Detection . . . 47
6.2 Control Software . . . 47
6.3 Turret Design . . . 47
6.4 Targets Reached . . . 47

**7 Further Work** **48**
7.1 Object Detector . . . 48
7.1.1 custom dataset . . . 48
7.2 Object Tracking . . . 48
7.2.1 Distance Sensor . . . 48
7.3 Water Gun Design . . . 48

**Bibliography** **49**

# Chapter 1

# Introduction

Household cats are predatory animals and are active during the night to hunt and explore [50] [9]. This can be problematic if they roam in unwanted areas, fight with other cats, and make unwanted mess and noise. Many creative solutions have been used, but it is desired to create a device with a high success rate.

Ultra-sonic solution [29] One previous solution is by using motion sensor and ultrasonic sound to scare the cat. The system works when the cat enters the range of the sensor and the sound is played. But the cat learns that it can ignore the sound or just avoid the range/area of the sensor. One big disadvantage of motion sensors is that it will engage everything that passes by [49]. Water guns with motion detection has also been tested previously, but the same problem with the motion sensor occur, but a direct hit of water seems to work in most cases. Even if its not a direct hit it scares the cat [45].

The application in this thesis is to identify cats and keep them away from an area. The device will use the real-time object detection You Only look Once algorithm (YOLO) to identify if the area contains a classified object and identify if it is a cat with a certain precision. The unwanted cat will be tracked by using deepSORT (Simple Online Real-time tracker). These methods will be implemented on an NVIDIA Jetson AGX Xavier Computing board for computation on site. A object detector and tracker will be developed and tested on a NVIDIA Jetson Board.

Mechatronics is a multi-discipline field in engineering which combines electrical, mechanical, computer and control engineering [48]. Typical application in mechatronics is to combine elements from each field for industrial application, prototyping and design for innovative solutions. The cat detection and spooking device will build upon these elements by using Computer Vision (CV), Artificial intelligence (AI), control algorithm, electrical components, edge computing and mechanical design to create the prototype. The field of AI is becoming more popular and has a huge potential in everything from economics* to Industrial applications, and this thesis will showcase elements from the subfield of AI, Machine Learning (ML), and the use of real-time CV in a physical application.

## 1.1 Challenges

A challenge will be to identify the cat in both low light and daylight conditions. A control algorithm to track the cats movement and position should also be implemented. The implementation of real-time object identification and tracking on a physical device will be a challenge, the main use of these applications is usually theoretical and is applied on sampled video and images. The prototype should be tested both inside and outside, should be able to handle outside climate for a day in the testing period.

## 1.2 Goals

The main goal for this thesis is to create prototype of the cat spooking device and be able to successfully identify and track the cat when it enters the area. Implement a spooking method which can spray water on the target and keep it out of the area.

## 1.3 Product Specification

When developing a new product, its useful to develop a Product Specification to find and specify consumer needs. The Product Specification is usually determined by the client or constructed for the target consumer by the developer in an easy language, which later will be translated to technical terms by engineers or designer.
These specifications are only preliminary and can change before final specifications are set, the final specifications are covered in chapter 4.4.

# Product Specification

| Name: | Date: | Project: | Target | Optional |
|---|---|---|---|---|
| Group 8 | 12.01.2021 | Master Thesis 2021 | | |

| | | | |
|---|---|---|---|
| **1 Function/Requirements** | **Note** | --- | --- |
| ***1.1 Process*** | | --- | --- |
| 1.1.1 Detect cat using Image recognition | | X | |
| 1.1.2 Estimate position of cat | | X | |
| 1.1.3 Estimate pose of cat | | X | |
| 1.1.4 Track Movement and targeting system | | | X |
| ***1.2 Product*** | | --- | --- |
| 1.2.1 Reliable and durable in outside environment (IP-rating, must handle rain) | | | X |
| 1.2.2 Fast recognition of cat | | X | |
| 1.2.3 Wireless, using battery and Wi-Fi | | | X |
| 1.2.4 Usable at night (night vision camera) | | | X |
| ***1.3 Control*** | | | |
| 1.3.1 Autonomous | | --- | --- |
| | | | X |
| **2 Surroundings** | | --- | --- |
| ***2.1 Indoor*** | | --- | --- |
| 2.1.1 Test purposes | | X | |
| ***2.2 Outdoor*** | | --- | --- |
| 2.2.1 Outdoor testing and use | | | X |
| **3 Parts** | | --- | --- |
| 3.1 Nvidia Jetson Computer board | | X | |
| 3.2 Camera | | X | |
| 3.3 Mounting/housing | | X | |
| 3.4 Water gun system | | | X |
| **4 Project Plan** | | --- | --- |
| 4.1 months, Master Project deadline | | X | |
| **5 Cost** | | --- | --- |
| 5.1 Development: Within Master Budget | | X | |
| 5.2 Production: not defined | | | X |
| **6 Product/assembly** | | --- | --- |
| 6.1 Simple production | | X | |
| 6.2 Design of Prototype | | X | |
| 6.3 Full scale Prototype | | | X |
| **7 Safety** | | --- | --- |
| 7.1 no human target or other animals | | X | |
| 7.2 High accuracy for wanted target | | | X |
| 7.3 low force on target | | X | |
| 7.4 No injury possibilities | | X | |

# Target Specification

*The Target specifications is developed from the Product specification. #id number is given from where its defined in the Product Specification.*

## 1 Function/Requirements

| #id | Metric | Unit | Ideal | Marginal |
|---|---|---|---|---|
| 1.1.1 | Identification Accuracy for identifying if it is a cat or not. (CNN usually gives accuracy in percentage) | % | 90 | 80 |
| 1.1.2a | Furthest detection range | m | 20 | 10 |
| 1.1.2b | Closest possible detection range | m | 1 | 2 |
| 1.1.2c | Accuracy of cat placement in space XYZ-direction | cm | 1 | 5 |
| 1.1.4 | Tracking the cat inside the range matching the speed of the walking cat | m/s | 13 | 5 |
| 1.2.1a | Product is operating outside and should be given an estimated IP-rating (certification not needed) | IP | 54 | 43 |
| 1.2.2 | Object detector performance, measured in Frames Per Second | FPS | 60 | 10 |
| 1.2.4 | Able to work in low light conditions | | | |
| 1.3.1 | Starting procedure | | One on/off button | In Terminal |

## 6 Production/Assembly

| #id | Metric/Description |
|---|---|
| 6.1 | Parts used to make the prototype will be created with additive manufacturing and will be designed using simple shapes and as few parts as possible, saving printing time and material. |
| 6.2 | Design a Prototype in CAD software |
| 6.3 | Print and Assemble the Prototype |

## 7 Safety

| #id | Metric/Description |
|---|---|
| 7.1 | Only Cats will be targeted by the system, and all other interference will be ignored |
| 7.2 | For safety reasons the system will not target if it is under **85% certainty** if it is the wanted target |
| 7.3 | Implemented water-gun will have low force, target should fear the sudden hit of water, and not experience any harm. |

# Chapter 2

# Theory

This chapter gives an overview of the theory behind the technology used in this thesis. This will cover the general understanding of AI, Convolution Neural networks, Creation of data-sets for deep learning, Theory of YOLO, and deepSORT. This chapter should give the reader a better understanding of the algorithms and applications used in this thesis and understand the choices and framework for continuing the work.

## 2.1 Artificial Intelligence

The basic understanding of artificial intelligence (AI) is a software (program) capable of making choices from the input/data. This can be basic operations where a robot can choose where to move by input from distance sensors to Deep Learning, where the output is a learned function by looking at data of similar objects or behavior as the wanted output. This thesis will use an object detection algorithm that uses convolution neural networks and the principle of deep learning.

## 2.2 Neural networks

Neural network is a set of neurons, where each neuron contains a weight which is learned by the input and wanted output. The neural network can contain several layers, inputs and outputs.
The activation function defines the relation between the sum of the weighted inputs from different nodes. The most common activation function is the Rectified Linear Activation (ReLU) [21]

$$f(x) = x^+ = \ max(0, x) \tag{2.1}$$

The relevance here is, the input would be a image and the output would be the predicted object. The neural networks layers would contain all the weights trained in the learning phase.

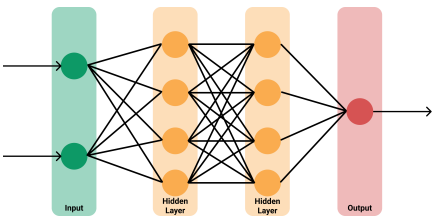



Figure 2.1: Simple neural network

## 2.3 Convolutional Neural Networks (CNN)

CNNs is a class of neural network which is specialized to process data from multidimensional arrays. CNNs are mostly used in machine- and deep-learning with images to create object detectors and image classifiers [**cnn**]. The most basic form of CNNs consists of three stages; Convolution-, Detection- and pooling-stage. Figure 2.2 show a simplified 3D representation of a Convolution neural network.

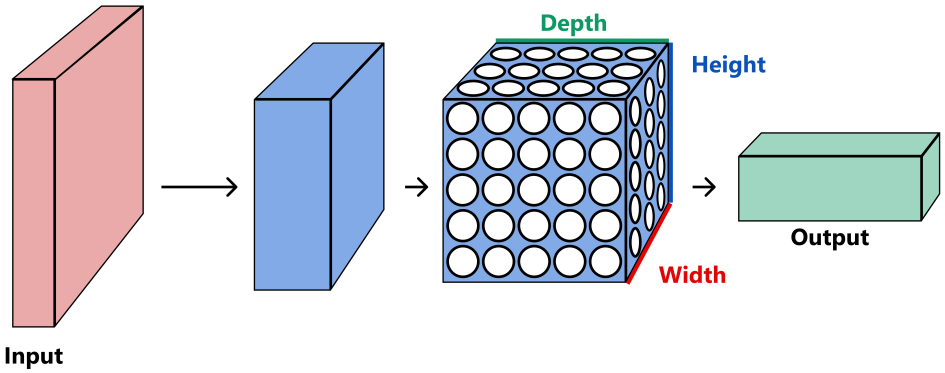



Figure 2.2: Convolution Neural Network: 3D representation of Figure 2.1

### 2.3.1 Convolution

Convolution is an mathematical operation on two functions which produces a third function which can be described as

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.2}$$

where $x(a)$, the first argument, is referred to as the input. $w$ is the second argument and is referred to as the kernel, and $t$ is the time index. The notation $*$ is the convolution operator. The output of the convolution function is sometimes referred to as the feature map[19].
The function 2.2 is a general expression for a convolution function for discrete time in one-dimension. Convolution Neural Networks is usually used in machine learning and deep learning when using image data. Image data is usually three-dimensional, where the image color is separated in three channels (red, green and blue) and the pixels are a two-dimensional array of data. The channels are usually split, since its easier to calculate with a two-dimensional data array.
The input image $I$ is two-dimensional, the kernel $K$ should also be two-dimensional and the new convolution function will be equation 2.3

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{2.3}$$

### 2.3.2 Convolution Layer

One layer in an CNN typically consists of three stages. The convolution stage, detector stage and pooling stage [19].

### Convolution Stage

The broad description of the job of the Convolution stage is to extract features from the data. The feature information is extracted by applying filter on the input image. The convolution stage uses the trained filter and passes it over the image to create a feature map from the input image. A basic example will be a 5x5x1 image matrix convolved with a 3x3x1 kernel(filter) will give a 3x3x1 feature as output [44]. When multiple filters are used, i.e. 5 filters, the feature output will be a volume of the size 3x3x5[13], see figure 2.3. The filter moves over the input image from top right over the whole image to all pixels are covered. Stride defines how many pixels it will step at a time.

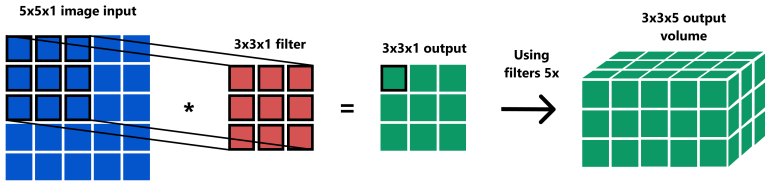



Figure 2.3: Visualization of Convolution

### Detector Stage

The detector stage is where the linear system from the convolution stage is passed through a non linear activation function. Non linearity is important to describe data patterns more detailed[19].

### Pooling Stage

The pooling stage will down sample the feature map more. This is done to make the computation more effective and highlight dominant features. The two most used pooling features are max pooling and average pooling. Max pooling will store the highest value from the neighbours, and average pooling will store the average off all the neighbours[19].

### Fully Connected Layer

The Fully Connected Layer (FC) is the last classification layer which is connected to every activation point in the previous layer[13]. In the YOLO architecture, see figure 2.4, the two last layers are the FC layers. This layer is responsible for output the predicted class.

## 2.4 You Only Look Once (YOLO)

YOLO is the state-of-the-art algorithm for object detection, when it comes to real-time detection. Previous object-detection algorithms such as R-CNN, would propose potential bounding boxes and run image classification inside the bounding boxes. YOLO was reframed as an regression problem and would compute bounding box probability and class prediction directly from the pixel input using only one single convolution network. Figure 2.4 showes the convolution neural network architecture for the first version of YOLO.

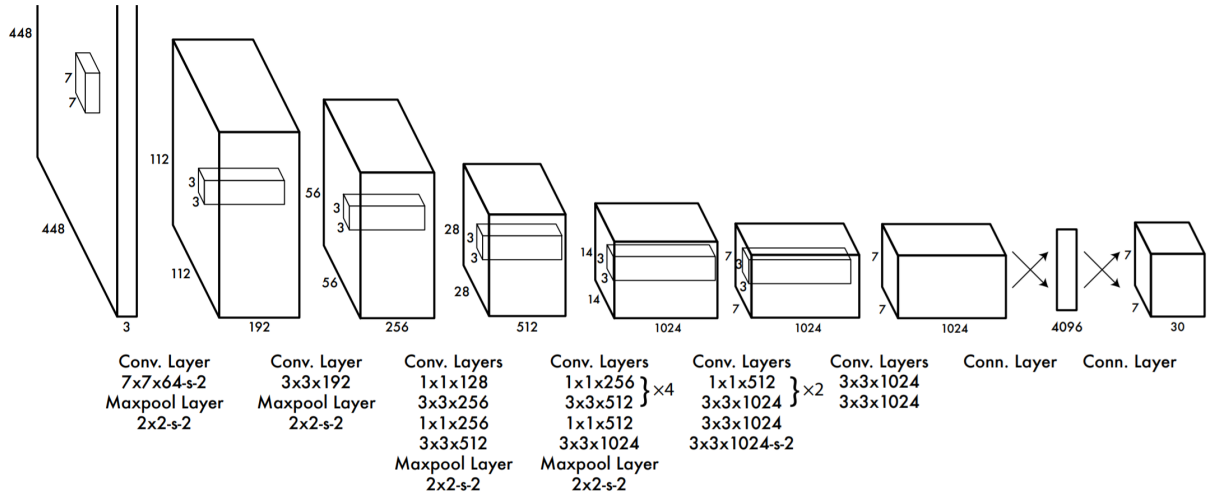



Figure 2.4: The YOLO Architecture [27]

### 2.4.1 Intersection over Union

Intersection over union (IoU) determines the accuracy of the predicted bounding boxes and the ground truth box. IoU is calculated by finding the area of the intersection and dividing it by the union of the bounding boxes, as seen in figure 2.5. A IoU with value 1 would be a perfect result.

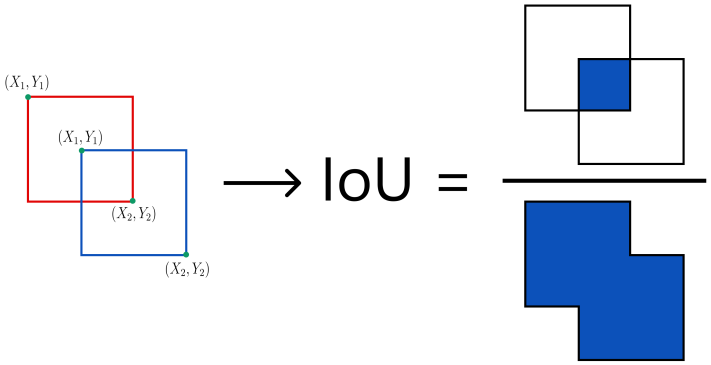



Figure 2.5: Intersection over Union

### 2.4.2 Detection

The input image is divided in to a $SxS$ grid. Each grid cell predicts Bounding boxes $B$ and its confidence score. Bounding box prediction consists of five parameters $(x, y, w, h, confidence)$, where $(x, y)$ is center of the Bounding box, and $(w, h)$ is width and height.
Prediction is defined as $Pr(object) * IOU^{truth}_{pred}$, and each grid cell also predictes class probabillites $C$ as $Pr(Class_i|Object)$. One grid cell only holds the prediction of one class.

$$Pr(Class_i|Object) * Pr(Object) * IoU^{Truth}_{Pred} = Pr(Class_i) * IoU^{Truth}_{Pred} \tag{2.4}$$

Equation 2.4 gives the specific class prediction for each bounding box [27].

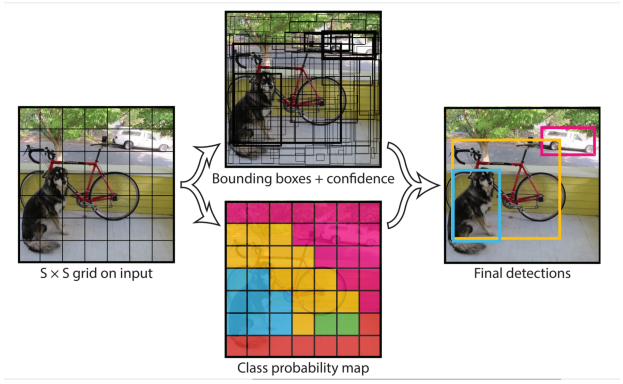



Figure 2.6: The YOLO model; Input image is divided in $SxS$ grip and YOLO predicts bounding boxes and class probabilities for each cell[27]

### 2.4.3 Loss Function

Simply the loss function is applied during training to penalize deviation between the predicted bounding box and the ground truth.

$$
\begin{aligned}
loss = \lambda_{coord} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
+ \lambda_{coord} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
+ & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_{ij} - \hat{C}_{ij})^2 \\
+ \lambda_{noobj} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
+ & \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.5}
$$

Where $\mathbb{1}_i^{obj}$ is set to 1 if there is an object in cell $i$, and $\mathbb{1}_{ij}^{obj}$ denotes the responisble bounding box predictor in cell $i$[27]. $\lambda_{coord}$ and $\lambda_{noobj}$ is parameters for changing the confidence predictors to increase the loss from bounding boxes and decrease loss from boxes that do not contain objects[27]. $x_i$ and $y_i$ is the center location of the ground truth bounding box, and $\hat{x}_i$ and $\hat{y}_i$ is the predicted position. $w_i$ and $h_i$ compared the width and height of the bounding box to the predicted width and height $\hat{w}_i$ and $\hat{h}_i$. $C_ij$ is the confidence score for a present object. This is compared to $\hat{C}_ij]$ and is the Intersection over Union (IoU) calculated for the prediction. The last term in the loss equation is not explained in the research paper, but $p_i(c)$ and $\hat{p}_i(c)$is the class predictions, which calculates the error for the detection of correct class.

### 2.4.4 YOLOv4

YOLOv4 is the fourth iteration of the original YOLO object detector. YOLOv4 was created by a new team of researchers taking over the work on YOLO and the Darknet framework. YOLOv4 did not have significant architectural changes, and the essence of the YOLOv4 research paper[8] was to test out different techniques to improve YOLOv3. Different data augmentations, using different feature detectors and activation functions. Most object detectors uses the same dataset when training to compare their performance. Figure 2.7 compares YOLOv4 to YOLOv3 and other object detectors.

The most significant change to YOLO was the introduction of anchor boxes in YOLOv2. YOLO was directly predicted bounding box coordinates using the fully connected layers, but this was removed to accommodate anchor boxes [40]. Anchor boxes are pre-defined bounding boxes specific for each class, usually with a pre-defined height and width to match the class. The anchor box location is defined by the network output for every grid cell, meaning multiple (can be thousands) of anchor boxes across the whole image. Using IoU and the confidence score calculates which anchor boxes will be part of the predicted bounding box [6]. Changes to the image size were necessary when implementing anchor boxes and are now 416x416 and scalable by 32 for custom image sizes[40]. YOLOv3 had some minor improvements compared with YOLOv2 and can be explored by reading the research paper [41].

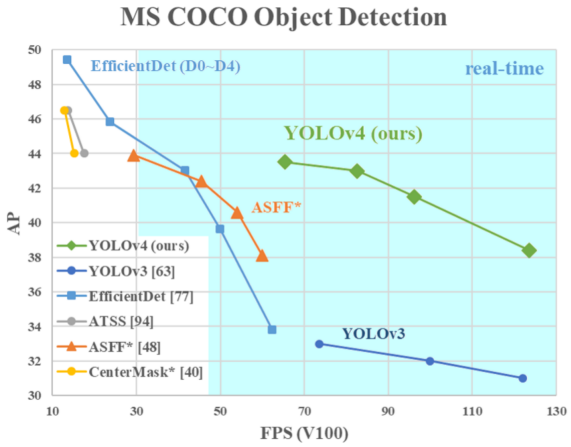



Figure 2.7: Comparison between YOLOv4 and other object detectors and classifiers[8]

### 2.4.5 The Architecture of YOLOv4

The Architecture of YOLOv4 was created by testing different types of methods. Figure 2.8 showes a general depiction of an object detector, such as YOLO[27], SSD[33] and RetinaNet[32].

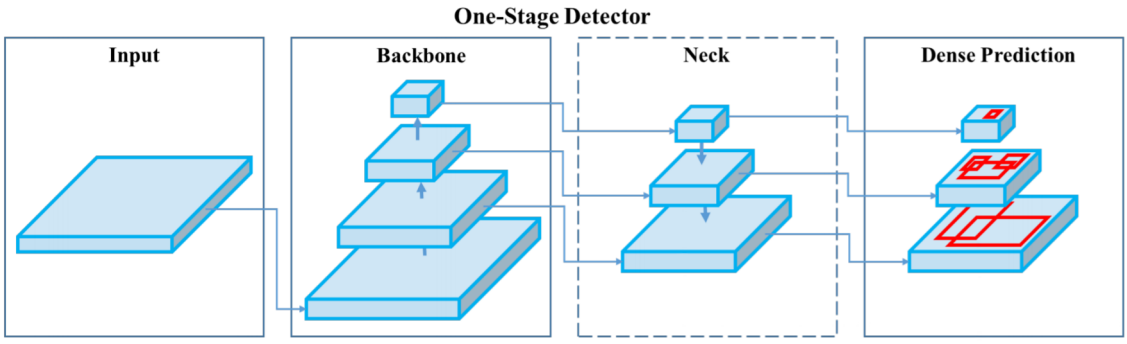



Figure 2.8: General One-Stage Object Detector [8]

#### Backbone

The backbone of the object detector is the convolution layer, also known as the feature extractor. There are many different frameworks to use for the feature extractor. In YOLOv4, the researcher compared Darknet to ResNext and EfficientNet to see if it would gain performance. The result showed that Darknet was the best option [8]. Specifically, CSP-Darknet53 was the result.

#### Neck

The Neck contains extra layers after the backbone, which extract features from different stages in the backbone structure (Vizual explanation see figure 2.8). A method often used is the Feature Pyramid Network (FPN). This method was implemented in YOLOv3 [41]. Each down-sample layer is done in the backbone. The FPN extracts the feature map, up-sample the feature map, and merges each up-sample into one feature map(see figure 2.8) Neck). For YOLOv4, they changed the Neck to Spatial pyramid pooling (SPP) and Path Aggregation Network (PAN)[8].

#### Dense Predictions (Head)

The Head is responsible for bounding box prediction estimation with the anchor box method discussed previously. This stage will output the four values for the bounding box and the probability for the class. The Head was not changed from previous versions of YOLO. The number of filters is calculated based on the network scale and is calculated by equation 2.6

$$Filters = (n_{classes} + P(class) + (x, y, w, h)) \cdot n_{anchors} \tag{2.6}$$

Number of filters for YOLOv4 using COCO weights is $255 = (80 + 1 + 4) \cdot 3$

#### Bag of Freebies and Bag of Specials

Bag of freebies and bag of specials is a term used in the YOLOv4 research paper to summarize different methods to improve YOLO. Bag of freebies is data augmentation on the dataset with rotation, cropping, changes to contrast, and saturation. The integration of cutout where part of the image is removed.
Bag of specials talks about inclusion of methods such as Squeeze-and-Excitation and Spatial Attention Module. YOLOv4 changed the activation function from leakyReLU to Mish, used in the backbone[8].
These method was applied to the already improved YOLOv4 to extract more performance. Most of the Bags of Freebies methods can be changed or deactivated when configuring the training parameters for YOLOv4, since not all is suitable for every situation.

### 2.4.6 YOLOv4-tiny

YOLOv4-tiny is a compressed version of YOLOv4 meant to decrease computational load at the cost of accuracy but able to use the object detector at embedded devices and low-performing systems.

### 2.4.7 Calculating accuracy

How the object detector performs can be calculated by using different methods. Instead of subjectively look and evaluate it can be calculated and compared with different object detectors to see the accuracy and performance.

#### Average Precision

Average Precision (AP), as seen on figure fig:yolov4fast(y-axis), is a metric for measuring the accuracy of object detectors. AP computes the average precision value for the recall value between 0 and 1 [22].
Precision measures how accurate the prediction is, see equation 2.7.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2.7}$$

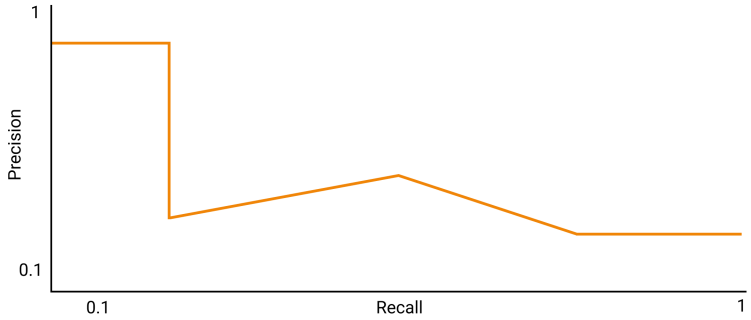



Figure 2.9: Precision-recall curve[22]

Recall measures the accuracy for finding all positive results, see equation 2.8.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2.8}$$

Then the definition on the Average Precision for Object detectors will be the area under the function $Precision(Recall)$, see equation 2.9.

$$AP = \int_0^1 P(r)dr \tag{2.9}$$

Where P(r) is the function of the precision-recall curve, see figure 2.9.

#### Mean Average Precision

Mean Average Precision (mAP) generally defined by calculating the AP of each class and take the average of AP for every class.[35].
Common Objects in Context (COCO) is a dataset used by most researchers to compare their object detectors performance against others. And in COCO context AP and mAP is the same[15].
One more important precision metric, also used with COCO, is $AP@.50$ or $AP@.75$. This simply means AP for a detecteon with IoU with value $0.5$ or $0.75$[15].

## 2.5 Simple Online and Real-time Tracking

Simple Online and Real-time Tracking (SORT) is a tracking framework which can track multiple objects between frames. [56] Yolo and other object detectors showes where an object is and the bounding boxes is not associated with the detected object, so the object detector does not account for if the object has been detected the previous frame or if its new. SORT in combination with YOLO is shown to work well[34].
SORT performes Kalman filtering in image space and frame-by-frame data association using the Hungraian method with an association metric that measures bounding box overlap
The kalman filter tracking uses eigth state-space vectors $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$, where $(u, v)$ is is the center position of bounding boxes, $\gamma$ is the aspect ratio, $h$ is the height and $(\dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ are the velocity of the image coordinates.
SORT achieves 260Hz (or FPS) on an unspecified Intel i7 2.5GHz CPU with 16 GB of memory[7]. The computation speed of SORT makes it possible to use in real-time and can be matched with YOLO.

#### DeepSORT

DeepSORT is the improvement of the SORT approach. DeepSORT uses a pre-trained CNN that matches appearance in the bounding boxes. Using Kalman filter and methods from SORT algorithm deepSORT is a more robust object tracker that can track an object for up to 30 Frames without new input from the object detector [56]

## 2.6 Edge Computing

Edge Computing is a computing framework which brings computations and data storage closer to data sources[23].
Edge computing offers a more efficient alternative to computing. The data is processed closer to the point where it's created and stored. Since the data does not travel over a network to a an off-site server, latency is significantly reduced. enables faster and more comprehensive data analysis, and creating the opportunity for faster response times. From vehicles, robots and industry 4.0, the amount of data from devices being generated is higher than ever,and most of this IoT data is not exploited or used.

## 2.7 State of the Art

There are many possible methods to spook animals. Most technological systems for this use motion sensors, as stated in the introduction (chapter 1), and will trigger the system for any animal or object passing the sensor. It is important to have control over what animal passes the deterrence system. Using computer vision and deep learning, it is possible to distinguish between different animals.

Computer vision is a heavily researched and popular topic in mechatronic engineering. The current state-of-the-art in computer vision is object detectors and image classifiers using deep learning. Object detectors can identify an object in an image and approximately where in the image it is placed. Using a state-of-the-art object detector, it is possible to find what animals are passing the deterrence system.

You Only Look Once (YOLO) is an object detector famous for being accurate and fast. There are other object detectors that are faster or more accurate, but YOLO is famous for having both attributes. Depending on which YOLO model is used and the system it is running on, YOLO can achieve fast detection at real-time speeds.

Animals are intelligent and have tendencies to learn how to avoid stationary systems. Using the output from object detectors makes it possible to implement control algorithms and then target the cat directly to make it harder to avoid.

Object detector has usually been implemented on computers with high-power graphics cards. with the ever-increasing efficiency in object detector algorithms and processing power makes it possible to implement object detectors on an embedded device. Using embedded computers with graphics cards from Nvidia Jetson, creating a small system with high computing power can be moved and put where it is needed without being connected to a server or an expensive computer.

# Chapter 3

# Methods

This chapter will elaborate on how the system was developed and the different tools and software used.
This chapter will elaborate on how the system was developed and the different tools and software used. There are three main topics in this thesis and is divided into how the object detector was trained, implementing the object detector on an embedded device, and lastly, building an enclosure with motors to have a targeting system

## 3.1 Object Detector

YOLOv4 is considered the state-of-the-art object detector, with a good combination between fast detections and prediction accuracy, and is at the time of writing the best performing object detector when combining both accuracy and computation time. It is important to mention that YOLOv5 exists at this time and is discussed later in this thesis. YOLOv4 is an object detector that will output multiple detections on a single image and provide bounding boxes around the detected target to show wherein the image the detected object is. YOLOv4 can be trained to detect almost anything and perform at real-time speeds. This last feature can be helpful to develop control to target the detected object.

### 3.1.1 Datasets

The dataset needs to fit the wanted outcome of the AI/computer vision algorithm. The YOLOv4 method works like any other CNN or machine learning algorithm using an extensive dataset of images of the desired result. YOLOv4 comes with pre-trained weights out of the box, which can detect up to 80-classes base on the COCO dataset. It is fast and easy to use the pre-trained system if it has a suitable class, But it is also possible to train for a custom class and improve the detection accuracy.

#### Image Annotation for Custom Dataset

The dataset created needs to contain ground truth boxes (bounding boxes label to tell the algorithm where the object is). It is vital to have the correct type of label or ground truth to match the right algorithm

#### Bounding Boxes

Bounding boxes are a popular version where the object is marked with a rectangular box with two anchor-point, top left corner and bottom right corner, which shows where the object is on the picture. This method will also include some objects or backgrounds.


Figure 3.1: Images with Bounding Boxes

### Google Open Images Dataset

There are many famous datasets used for training object detectors. Different deep learning competitions, such as Pascal VOC and MS COCO, have a standard dataset that the participants use to compare algorithms. Google's Open Images Dataset (OID) is a database with prelabeled images which has over 600 different image classes. OID was used to create a dataset for training the object detector.

| **Library** | **Number of Images** | **Number of Classes** | **Latest Update** |
|---|---|---|---|
| PASCAL VOC | 11.5K | 20 | 2012 |
| COCO | 330K | 80 | 2017 |
| OIDv6 | 9M | 600 | 2020 |

Table 3.1: OID vs COCO vs Pascal VOC

### Number of Images

The recommended number of images is around 2000 for each class. At the beginning of the thesis, YOLOv4 was tested by using every cat image from OID (almost 2000) and trained with the standard YOLOv4 configuration. These weights reached an mAP of 94% using a validation set of 300 images, also gathered from OID. YOLOv4, trained and validated from on the COCO dataset, reached approximately 90% mAP **??**. These two different weights were compared by using a video of cats to if there was a difference. The custom trained showed a better result, and it was decided to continue with custom training.

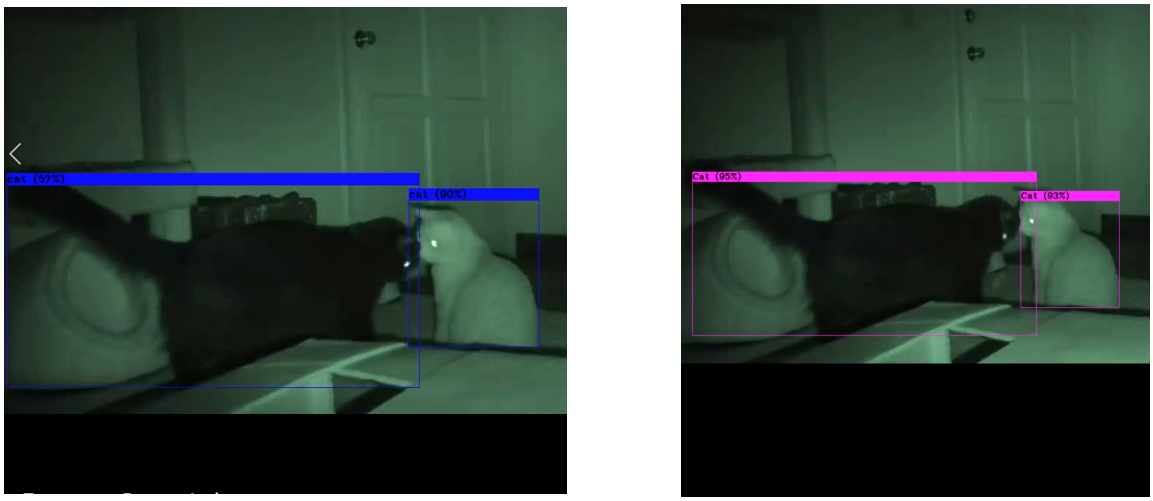



Figure 3.2: YOLOv4 (left) compared with custom training using 2000 images from OID (right)

### OIDv4 Toolkit

OIDv4 Toolkit is software created to make it easy to download object detection and image classifier datasets. It uses the API from Google's Open Image Dataset to download the classes with related bounding boxes. It uses one command to download multiple classes and organizes the folder structure to be ready to use for training[51]. The YOLO bounding box converting script was acquired from $theAIGuysCode$ in the $OIDv4\_toolKit$[47].

### Cats and other animals

There are multiple types of animals that roam around in Norwegian woods and neighborhoods. It is essential to consider the animals that can be detected and be targeted for legal reasons. Hedgehogs are not directly red-listed, but it is common knowledge that hedgehogs are off-limits. The Norwegian database for animal species [1] shows what animals are off-limits, and it should be check when making a system that will interfere with wildlife.
In this case, the pre-trained class did not contain a hedgehog, which needs to be included (see chapter 3.1.1). Creating a custom object detector gives more control over what is detected and can be tailor-made to fit the project's criteria. Since there are datasets with hedgehogs available in the OID library, hedgehogs will be included in the training of YOLOv4 for more accurate training results and possible avoidance of targeting the species. To make sure that YOLOv4 can differentiate and not target anything other than what is set by the control method. Hedgehogs, dogs, and humans are also included in the dataset to differentiate between the animals. It is easier to use the control algorithm to target cats when we have multiple outputs than to have cats as the only output and trust that YOLOv4 has false positives.

### Dataset

OID on only contained 350 images of hedgehogs. Compared to cats that have 2000 images, it is possible for the algorithm to have a bias towards classes with higher image counts. Choosing 500 images for each class (350 for hedgehog) will decrease training time but also accuracy. Using 500 images should be sufficient but not optimal.

[1] https://artsdatabanken.no/Rodliste

| Class | Number of images |
|---|---|
| Cat | 500 |
| Dog | 500 |
| Human | 500 |
| Hedgehog | 350 |

Table 3.2: Classes and number of images in the dataset

### 3.1.2 Darknet

Darknet is an open source neural network framework which supports CPU and GPU computations with different models as YOLO, ResNet, ResNeXt, RNNs, DarkGo and custom classifiers[58]. Darknet can be used to run or train an YOLO detector. In this project Darknet is used to train YOLOv4.
Darknet can be downloaded from the darknet Github-page[5].
Important dependencies when using Darknet is OpenCV and CUDA.

#### Open Source Computer Vision Library

OpenCV is an open-source real-time computer vision library, which provide free computer vision and machine learning application[1].
OpenCV is used in darknet to manipulate images and display images for validation and training. OpenCV is also used to access camera on the NVIDIA Jetson

#### Compute Unified Device Architecture

Compute Unified Device Architecture (CUDA) is a computing platform and application programming interface (API) which allows access to perform calculations by using the system Graphical Processing Unit (GPU) for faster Computations with NVIDIA GPUs.

### 3.1.3 YOLOv4 Training Setup

When training a custom YOLO object detector with Darknet, The training files need to fit the training objective. Five files need to be configured to fit the dataset and training outcome. For this thesis, YOLO is used on an Nvidia Jetson Nano, and to have enough FPS to be close to the target of 30 FPS, YOLOv4-tiny was used.

#### Data and Names-files

The $.data$ file is a path to file names and directories for the training and validation dataset. This file contains the number of classes, the names of the classes, and where to store the backup weights. Under training, the algorithm needs the file location for every image used. These are separated into .txt files for both the training and validation datasets. The data file points to a names file with the name for each class. The order must match the same as for the annotated dataset.

#### Training Parameters

YOLOv4 can be changed by editing the config-file. This is necessary when training on a custom dataset, to get the right network size and parameters.
In Darknet, the number of iterations is dependent on the parameters $max_{batches}$. After instructions, the researchers of YOLOv4 recommend using the following calculations for training iterations. It is possible to have a longer training period, but then the risk of overfitting can occur.

$$max_{batches} = N_{classes} \cdot 2000 \tag{3.1}$$

This gives the number of $max_{batches}$, $4classes \cdot 2000 = 8000$ for the training of this object detector[58].
Changes to step is defined by $max_{batches}$, where the fist part should be 80% of $max_{batches}$ and the second part should be 90% so:

$$steps = 6400, 7200$$

#### Network size

With a larger network size, the image becomes more accurate, but its FPS performance will suffer. Table 3.3 shows how the network size will influence the outcome after training the model. The data is taken from the darknet GitHub repository[5]. The comparison is made by using the same yolov4.weigth file, only changing the network size, and tested with the same GPU.

| Size | mAP@0.5 | FPS |
|---|---|---|
| 608 | 65.7% | 34 |
| 512 | 64.9% | 45 |
| 416 | 62.8% | 55 |
| 320 | 60% | 63 |

Table 3.3: How network size influences performance. The FPS measured using an NVIDIA RTX2070[5]

Usually, the standard network size is 608 for YOLOv4, but it can be customized by changing it with steps of 32[5]. For this project, YOLOv4-tiny is used, and it has a 416x416 image input size to increase the FPS by the cost of accuracy. These changes are done in the config-file, and there are three layers in the standard YOLO-config, but only two in the TINY-version that needs to be changed.
The filters should be change according to the classes.

$$filter = (classes + 5) * 3 \tag{3.2}$$

[58] so in this case with 4 classes the filters in each layer should be 27.

#### YOLOv4 Performance enhancers

YOLOv4 introduced different methods for enhancing performance, see section 2.4.5. Parameters such as $angle$, $saturation$, $exposure$ and $jitter$ was unchanged. The function $random$ changes the network size randomly. This caused crashed during training and was turned off. This can increase performance and should be left on if possible.

### 3.1.4 Overfitting

Overfitting can occur if training is performed too long at the same dataset. The algorithm is trained to detect cats, but if the algorithm is overfitted, it would not be able to detect other images than the images used for training the object detector
Overfitting can be a problem, but when following the training guide, it should not. it is good practise to check the previous iteration with the validation set after training. Figure 3.3 shows that the training data would be more accurate over time, but when testing the detector on the validation set, detection accuracy could be worse.

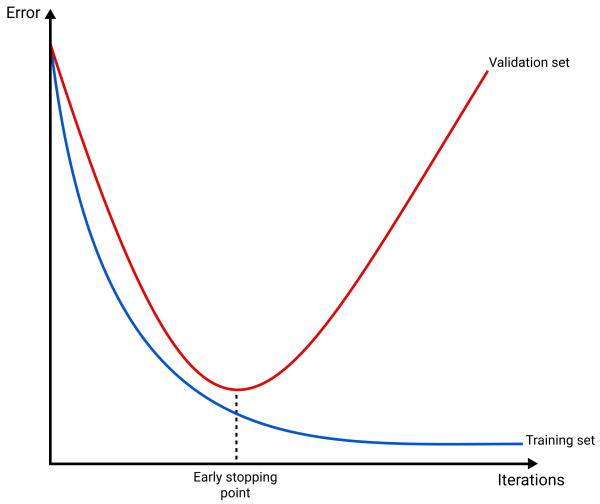



Figure 3.3: Overfitting curve show an approximation of how it would look when training longer then necessary[58]

## 3.2 Mechanical System/Hardware Setup

The main target of this thesis is to spook a cat away from an area. The main thought for the mechanical system for the spooking device was to make a water-gun turret to shoot water on the cat, which is safe and known to scare cats. Due to time constraints and complexity, the water gun system will be simulated using a small laser turret that can be used to test the object detection and control software.

The mechanical system in this thesis was made to test the object detection and control system implemented on the NVIDIA Jetson and should be redesigned for further development with more testing or implementation with other equipment.

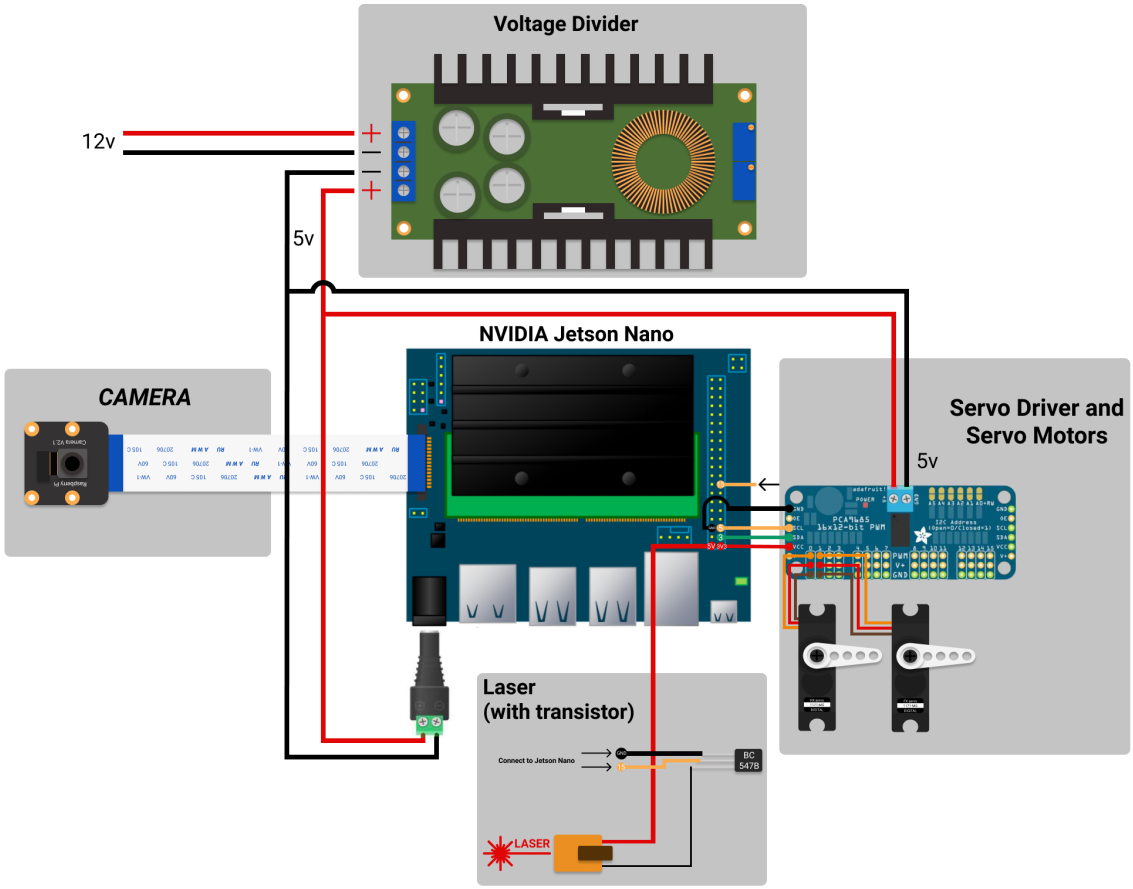



Figure 3.4: Component layout

| Equipment | Name | Comment |
|---|---|---|
| Controller/Computer | Nvidia Jetson Nano | — |
| Servo motor | PDI-1171MG | 2x |
| Camera | Pi NoIR Camera V2 | Night Camera |
| AC Adapter | YU-0805 | Converts 230v AC to 12v DC |
| Voltage Adapter | — | Converts 12v DC to 5v DC |
| Motor Controller | AdaFruit PCA9685 | 12-Channel servo motor controller |
| 3D-printed Housing Structure | — | — |

Table 3.4: Equipment/part list

### 3.2.1 Nvidia Jetson Developer Kit

Nvidia Jetson is developed to provide powerful GPUs to be used for robotics and AI in edge computing. The developer kits are primarily used for prototyping and testing software and hardware implementation. The integrated GPU makes the Jetson line-up more suitable for this project compared to teensy 4.1, Arduino, and raspberry pi 4. In this thesis, both Jetson Nano and AGX Xavier will be used. Xavier is the more powerful computer, but the lack of easy camera implementation made it unusable for the prototype, and Jetson Nano was used instead.
The Jetson Line-up uses Ubuntu 18.04 and comes preinstalled with OpenCV and CUDA to access GPU computation.
In production, the developer board should not be used and should be exchanged with NVIDIA Jetson Module. The main difference between developer kits and modules is the provided I/Os on developer kits. Modules come only as Computer-boards without I/Os.

**NVIDIA Jetson Nano**

The Jetson Nano is the smallest and has the least computing power in the Jetson Line-up. In this thesis, the Jetson Nano Developer kit is used to get easy access to ports and pinout for SPI and I2C communications. Jetson Nano has a built-in GPU and can use CUDA for faster computations, which is important with Real-time object detection. [18]

| | |
|---|---|
| GPU | 128-core NVIDIA Maxwell |
| CPU | Quad-core ARM A57 @ 1.43 GHz |
| Memory | 2 GB 64-bit LPDDR4 25.6 GB/s |
| Storage | microSD (Card not included) |
| Video Encode | 4Kp30 \| 4x 1080p30 \| 9x 720p30 (H.264/H.265) |
| Video Decode | 4Kp60 \| 2x 4Kp30 \| 8x 1080p30 \| 18x 720p30 (H.264/H.265) |
| Connectivity | Gigabit Ethernet, 802.11ac |
| Size | 100 mm x 80 mm x 29 mm |
| Camera | 1x MIPI CSI-2 connector |
| Deployment | Module (Jetson Nano) |

Table 3.5: NVIDIA Jetson Nano Specifications[18]

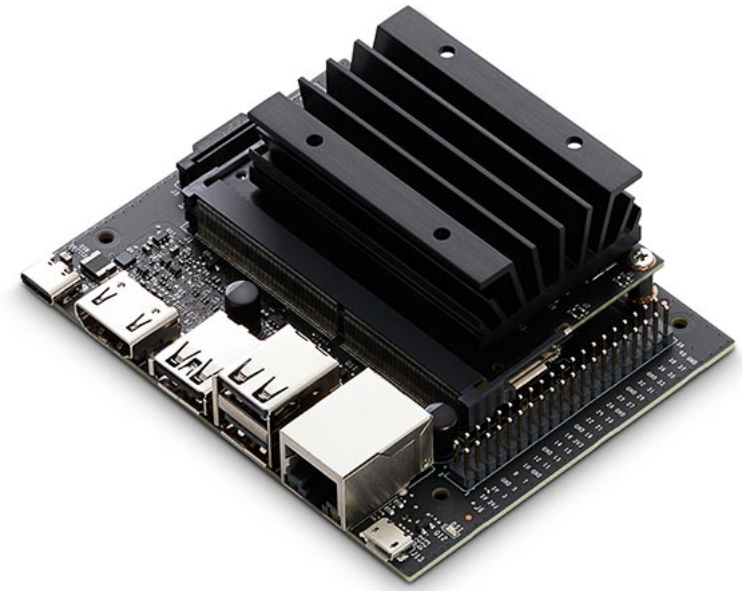


Figure 3.5: NVIDIA Jetson Nano Developer Kit[18]

**NVIDIA Jetson AGX Xavier**

AGX Xavier is the most powerful computer in the Jetson developer kit lineup[17]. It is designed specifically for autonomous machines, is power efficient. It is supported with Nvidia JetPack which contains CUDA, cuDNN and TensorRT libraries.
The Developer Kit is used in this Thesis. The developer kit, has ports to access Ubuntu User Interface and I/O-pins with SPI, I2C for communication with servo motors. [16]

| | |
|---|---|
| GPU | 512-core Volta GPU with Tensor Cores |
| CPU | 8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3 |
| Memory | 32GB 256-Bit LPDDR4x \| 137GB/s |
| Storage | 32GB eMMC 5.1 |
| DL Accelerator | (2x) NVDLA Engines |
| Vision Accelerator | 7-way VLIW Vision Processor |
| Encoder/Decoder | (2x) 4Kp60 \| HEVC/(2x) 4Kp60 \| 12-Bit Support |
| Size | 105 mm x 105 mm x 65 mm |
| Deployment | Module (Jetson AGX Xavier) |

Table 3.6: NVIDIA Jetson AGX Xavier Specifications[16]


Figure 3.6: NVIDIA Jetson AGX Xavier Developer Kit[16]

### 3.2.2 Raspberry Pi NoIR Camera

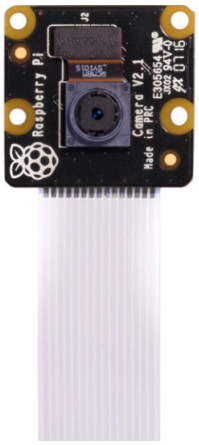


Figure 3.7: Raspberry Pi NoIR

| | |
|---|---|
| Camera module | Sony IMX219 8-megapixel |
| Video | 1080p30, 720p60 and VGA90 video modes |

Table 3.7: Raspberry Pi Camera Module 2 [10]

The Raspberry Pi NoIR Camera is the Raspberry Pi Camera Module V2 without Infrared Filter. The camera works well for prototyping and fits with the NVIDIA Jetson Nano. The camera can be used with the Picamera Python library or with OpenCV [38]. 60 FPS @ 720p also works well for this project and can be changed to 1080p @ 30FPS if needed. [38]

The camera module comes with a flexible cable as standard. This is ideal for this design since the camera is rotating, and the wire will experience some twisting.

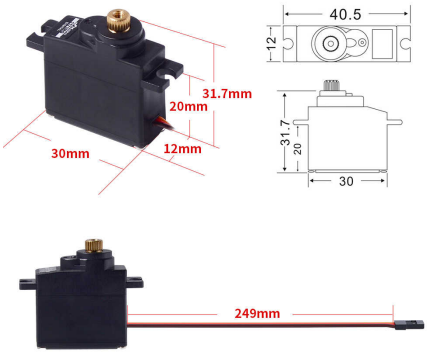
When designing the camera mount, the camera's field of view (FOV) was considered to remove as much disturbance from the image as possible. The FOV of the Raspberry pi NoIR is 62.2°in horizontal and 48.8°vertical.

### 3.2.3 PDI-1171MG Servo Motor

The PDI-1171MG servo motor is a small and accessible servomotor that can theoretically rotate 180°but is limited to 120°in this project due to issues when the built-in mechanical brake stops the motor instead of using the signal as a break. The servo motors can be used with the Adafruit PCA9685 servo driver (chapter 3.2.4). When limited to approximately 120°, the motors have steps resolution from 204 to 816 steps over 120°.

#### Connection

Connecting the servo motors to the servo driver is straght forward. Make sure the color code on the driver and the cable from the servo motors match. The connector is 5v, GND and signal for the PWM frequency.




| | |
|---|---|
| Dead band | $1\mu s$ $1520\mu s/330hz$ |
| Voltage range | 4.8V - 6.0V |
| Motor | High quality core motor |
| Operation Speed (4.8V) | 0.13sec 60° |
| Operation Speed (6.0V) | 0.11sec 60° |
| Stall Torque (4.8V) | 0,29 Nm |
| Stall Torque (6.0V) | 0,34 Nm |
| Dimensions | 30x12x31.7mm |
| Weight | 17.5g |

Figure 3.8: JX Servo PID-1171MG [57]

### 3.2.4 Adafruit PCA9685 Servo Driver

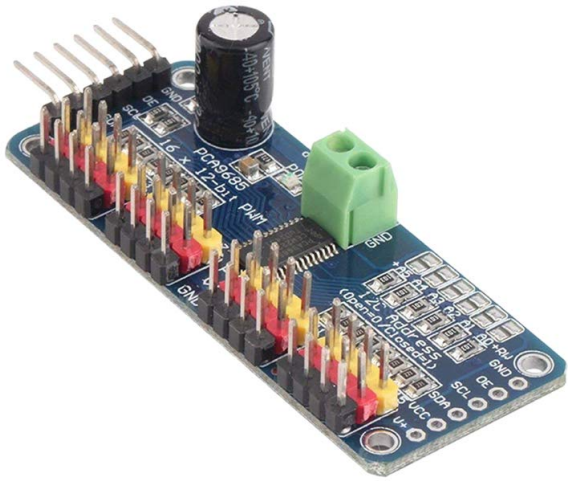


Figure 3.9: Adafruit Servo Motor Driver[3]

The Adafruit PCA9685 Servo Driver is used to drive up to 16 servo motors with Pulse width Modulation (PWM). It uses the Adafruit PCA9685 Library, which is available for both C and Python, and the communication with the microcontroller is by I2C protocol[3]. This servo driver is used to power two servo motors.

#### Connection

The Adafruit is connected to the I2C bus1 on the Jetson Nano (would be the same on AGX Xavier). That is pin-3 and pin-4 for SCL and SDA, and pin-1 (5v) for VCC and pin-6 GND. It also needs 5v from the power supply, which is connected to the screw terminals.

### 3.2.5 Arduino 5mW laser

The laser comes from an Arduino Uno kit and is of unknown origin: The laser is used to point at the Target and is centered close to the camera to target the center of the camera image.
The implementation of a laser serves two purposes. The first is to use it for targeting and visualization of the control algorithm. The second is the see if the cat is scared away when it is pointed directly at it.

**Connection**

Nvidia Jetson Nano does not output enough current from its I/O pins to drive the laser directly from the I/O pins. A transistor with the name $BC547B$ from an Arduino kit was implemented. The transistor base was connected to I/O-pin 15 on the Nvidia Jetson Nano Board, while the emitter was connected to the laser GND and collector to Nano I/O-pin ground.

## 3.3 Turret Design

The physical build of this thesis was made to house all the components and have servo motors mounted to have a two-axis system where the camera can be mounted and the control algorithm tested. The main housing will hold the NVIDIA Jetson Nano, motor driver, and voltage divider. To simplify the construction of a motor bracket for holding the motor responsible for movement around the Y-axis (see figure 3.10), it was designed to be directly mounted on the servo motor. The X-axis motor holds a small bracket made to hold both the camera and laser. These parts are designed to be light and 3D-printed with 20 percent infill.

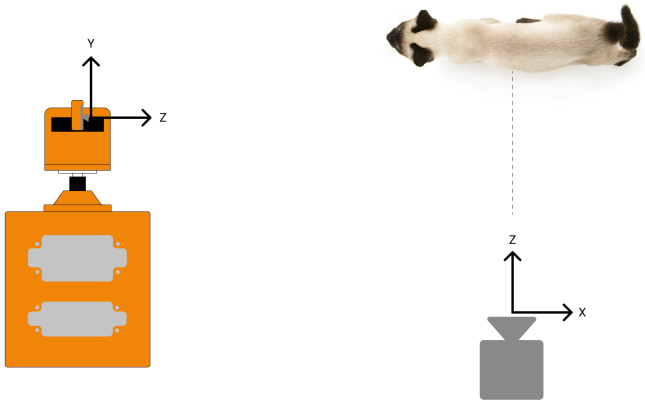



Figure 3.10: Coordinate system

The system was designed in Solidworks CAD, 3D-printed and assembled. Figure 3.11 shows a render of the assembled model in Solidworks.

Figure 3.11: Render of Solidworks Model

Both motors are rotating around Y- and X-axis and has 2 degrees of freedom. The Raspberry pi camera has the camera is mounted as close to the rotation-center of both motors as possible. Due to construction restraints with the cables and Wires, its rotation will be

limited to 180 degrees to not damage camera and motor cables.

### 3.3.1 Operating Area

The view area of the system is decided by two variables, Camera FOV and servo motor range. The servo motor range is 120°and the field of view of the camera 62°.

$$total_{range} = servo_{range} + camera_{range} \tag{3.3}$$

The total operating range is equal to 183°. When the servo is at the end of its range, the camera can see 31.5°more in that direction, making the total viewing range equation 3.3 Since the laser is mounted center and will follow the movement of the motor, since the target laser is set to be in the middle of the camera image, the target range for the system is 120°and not the full 183°that the camera can see.

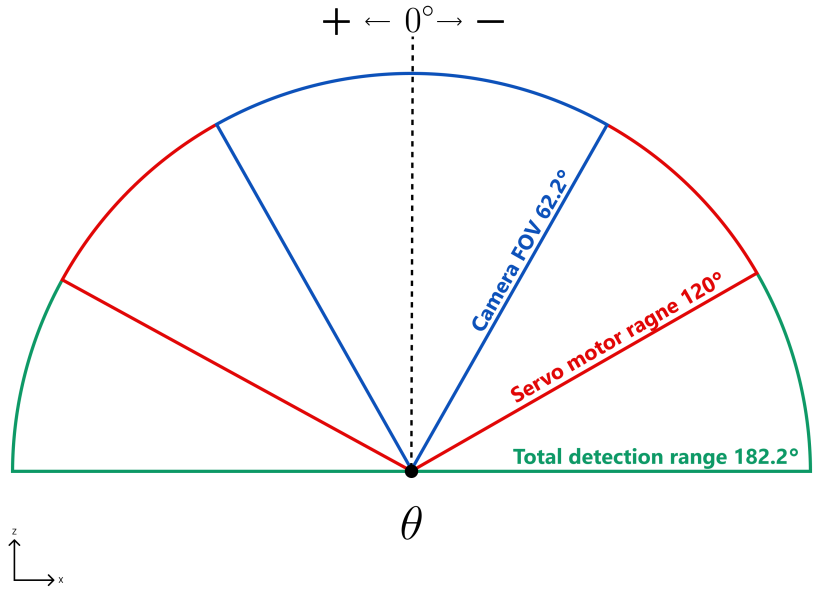



Figure 3.12: Diagram over camera FOV, servo motor range and the servo motor range

## 3.4 Control Software

This section will cover the development of the control software for the system. The goal for the control system is to be able to target the object detected by the detector created and shown in the previous section. The control algorithm will be developed and tested on the Nvidia Jetson Nano, with the hardware from section 3.2.1.

### 3.4.1 TensorRT

Nvidia Jetson Nano and Xavier come out of the box with TensorRT, TensorRT is an software developer kit for high-performing deep learning inference[37].TesnorRT is used for running the object detector on the Jetson Nano with the built-in GPU. YOLOv4 weights will only work when using Darknet. To utilize TensorRT for more GPU performance, the YOLOv4 weights must be converted to TensorRT weights. The convertion was done by using the script from GitHub-page tensorrt_demos[28].

### 3.4.2 Transferring Data

TensorRT and Tensorflow both working on python 3.x, while ROS Melodic works on python 2.7. For that reason, it is not possible to use TensorFlow and YOLO functions directly in ROS. The workaround for this problem was to open a pipe by importing the built-in OS library. A pipe in this context is a function that sends information between different processes running on the system. The function is started by calling os.pipe(), which defines two pipes, read and write, for transferring data.
The whole software for this system can be divided into two main processes. ROS is responsible for the control algorithm and control of the servomotors, while there is a separate python 3 process running the object detector and camera. The control algorithm only needs four values from the object detector to work. These values were converted to a string and sent over the pipe to ROS.

### 3.4.3 Object Detector Implementation

This section will go through implementing the object detector into working code that can be sent to ROS. The main script for the object detector was created by modifying the functions from $jkjung - avt$ GitHub repository $tensor\_demos$[28], primarily using the "utils" folder, which contains the functions to run YOLOv4 with TensorRT.

#### Camera

Implementing a camera to send images to the object detector was implemented using the "gstreamer" framework that comes built-in on the Jetson line-up. By calling the "gstreamer" function, it will detect the camera connected to the MIPI-port, which is located on the Jetson nano. Using OpenCV the data streamed from the camera is passed to the detection function.

#### Sending Data to ROS

The object detection algorithm returns four numbers for the position of the target. These numbers are the bounding box parameters that show the object's position in the image. The bounding box parameters were combined into a string which can be sent over the pipe to ROS and be used by the control algorithm. The data from the detector function is a list containing four integers where each number represents a corner from a bounding box. The list was converted to a string to be able to send it over to ROS, and it was converted back to a list in ROS.

### 3.4.4 ROS

Robot Operating System (ROS) is a framework for writing software for robotic applications. ROS aims to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS enables the communication between different computers and systems with a different programming language if ROS is supported.

#### Nodes

ROS nodes are the same as processes. By dividing processes into several parts, more can be done in parallel, and therefore increasing efficiency **??**. It also helps troubleshoot where a problem can be specified to a specific node instead of looking through a massive chunk of code. In this project, there are two nodes, see table 3.8. It is also possible to look at the object detector script as a third node but is not inside the ROS environment, so it is not defined as a ROS node even though it is working in parallel to the ROS nodes.

| Node | Description |
|---|---|
| catdetController | Functions for control algorithm, servo motors, and motor driver |
| mainController | Main script running the system |

Table 3.8: List of nodes in ROS

The catdet-controller node is responsible for communication with the Adafruit servo driver and the servo motors. This node also contains the class with the calculations for the control algorithm. The mainController script contains the main parts of the scripts and how the system will behave. It runs the control algorithm and starts the object detector subprocess. It receives bounding box data and passes it to the control algorithm.

#### Topics

Topics are messages which send data between nodes. Different nodes can publish data to a topic or subscribe to receive the topic data stream **??**. A topic can be used by multiple nodes, which increases data communication in the ROS environment **??**. This project only uses one topic 3.9.

| Topic | Description |
|---|---|
| servoSetpoints | Message for servomotors with step number |

Table 3.9: List of topics in ROS

This topic sends the servo motor position calculated by the control algorithm to the catdetcontroller script, which outputs the value to the servo motors. There are also data communication between the object detector process and the mainController.py script, which sends the string containing the bounding box data. This data stream would have been a topic if it was inside the ROS environment.

#### Packages

Package is a method for ROS to organize software in the ROS environment. The package contains nodes, libraries, data, and useful files. Packages are helpful for reusing software later or integrating it in different systems with other packages **??**. In this project, all software in ROS is in one package named "catdet".

### 3.4.5 Control Algorithm

The control algorithm will control the two servo motors implemented on the turret system. The object detector outputs wherein the image targets are and what class the object is. That information will be used to create the control algorithm and make the laser mounted on the turret aim at the target.

#### Targeting

The goal is to keep the detected target in the middle of the screen. The laser is mounted above the camera **??** and can target approximately in the center of the camera image. The target will be the middle of the screen and calculated as following:

$$center_x = \frac{image_{width}}{2}$$

and

$$center_y = \frac{image_{height}}{2}$$

.

### Centroid

The centroid is the middle point in the bounding box (see figure 3.4.5). The bounding box comes as a list with the image pixel position for each corner. Equation for calculating centroid see equation 3.4.

$$\begin{aligned} x_{Centroid} &= (x_{min(bbox)} + x_{max(bbox)}) \\ y_{Centroid} &= (y_{min(bbox)} + y_{max(bbox)}) \end{aligned} \tag{3.4}$$

Bounding boxes are square, and animals are often different shapes so that bounding boxes will cover more than the exact target. Using the centroid will, with high probability, always cover the target inside the bounding box. Using the average between the min and max position will also minimize the movement of the centroid if the bounding box is fluctuating in size due to detection error.

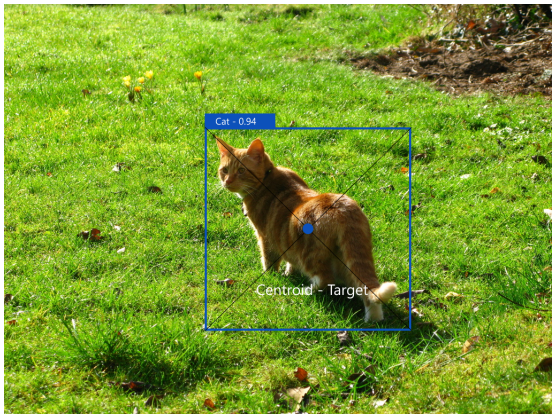



Figure 3.13: Vizualisation of Centroid

### Image position to Real world position

The control algorithm works by converting the image position to a real-world position.

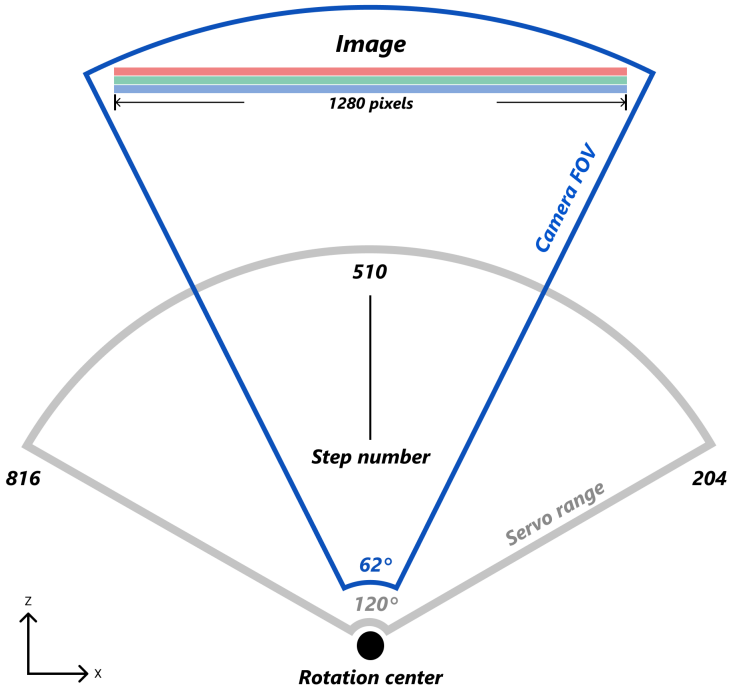



Figure 3.14: Visualization of control algorithm for one servo motor

The servo motors have a known step-count inside the operation area. In this case the there are 612 steps inside the range of 120°of rotation. The camera documentation also gives an approximation of the field of view (see section 3.2.2) in degrees. Using this information, it is possible to find the step-count inside the camera's field of view.

$$steps_{image} = \frac{(step_{max} - step_{min})}{angle_{max} - angle_{min})} \cdot camera_{FOV} \tag{3.5}$$

Equation 3.5 outputs how many steps there are in the camera field of view. Since the centroid should be in the center of the image, changing the difference between where the centroid is and where it is supposed to be is possible. By using a mapping function (equation 3.6), it can output the position difference between the detected target and where the center is. The $input$ in the function is the centroid pixel position, the range of the $input_{min}$ and $input_{max}$ is the pixel resolution for the camera. The output range, $output_{min}$ and $output_{max}$, from the result of the steps in the camera field of view.

$$output = output_{max} + \frac{(output_{max} - output_{min})}{(input_{max} - input_{min})} \cdot (input - input_{min}) \tag{3.6}$$

The output of the map function is the offset the servo motor has to move to target the centroid. Using the servo motors' current position and adding the offset, the control algorithm will return the new step target. The control algorithm will update the new target for every centroid received from the object detector and have the same calculation frequency as the object detector FPS.
This algorithm will utilize the whole servo range of 120°, but it can still only detect and receive information inside the camera field of view. This method is implemented on both servo motors, only changing the camera field from 62°to 49°of view and pixel resolution (from 1280 to 720)to match camera specs for the motor revolving around x-axis, see figure 3.10.

### 3.4.6 DeepSORT

The purpose of using deepSORT, is to associate the detected bounding boxes to one object and keep track of that object. YOLO only provides detection for each image input and outputs if there is a detected object and where it is. The YOLO algorithm does not give any information on if it is the same object or another one. deepSORT will create an ID for the detected object using a simple feature extractor to associate the detected object from YOLO with the object on the next frame. By using kalmanfilter, it will try to predict the next frame and will give the output bounding box more stability and will be easier to use as a target.

### 3.4.7 Testing Accuracy of Object Detector

When testing the object detector and tracker a random image of cats on pictures and from a mobile phone.

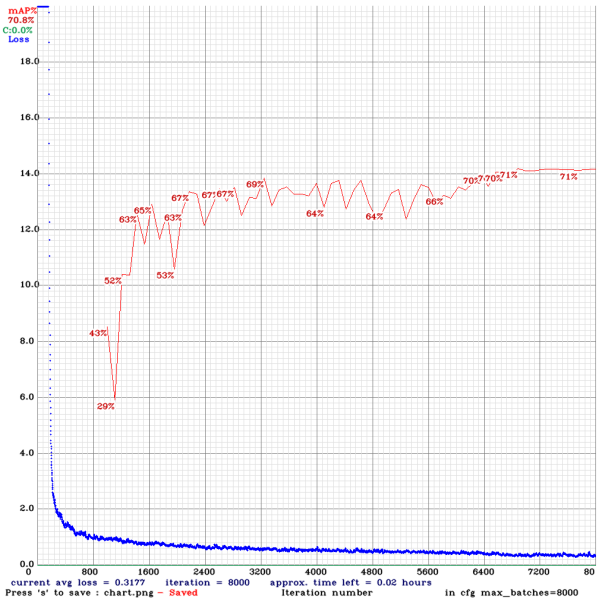
# Chapter 4

# Results

This chapter shows the results of the developed animal deterrence system. This will aid as a reference for further development, improvement, or recreation of the system. This chapter will be organized into three central parts: object detection, Object tracking (Control algorithm), and hardware.

## 4.1 Object Detector

This section will show the result of training the detector and how accurate the trained YOLOv4-tiny. The object detector is validated using a separate validation dataset which assures that it doesn't contain images used from training. The calculation for accuracy is discussed in section 2.4.7.




```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
1344
 detections_count = 10231, unique_truth_count = 1890
class_id = 0, name = Cat, ap = 79.43%           (TP = 309, FP = 156)
class_id = 1, name = Dog, ap = 81.92%           (TP = 494, FP = 217)
class_id = 2, name = Human, ap = 24.60%         (TP = 327, FP = 594)
class_id = 3, name = Hedgehog, ap = 97.10%      (TP = 21, FP = 8)

 for conf_thresh = 0.25, precision = 0.54, recall = 0.61, F1-score = 0.57
 for conf_thresh = 0.25, TP = 1151, FP = 975, FN = 739, average IoU = 41.07 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.707659, or 70.77 %
Total Detection Time: 9 Seconds
```

Figure 4.1: Result and validation of YOLOv4-tiny

The left graph in figure 4.1 shows the the average precision and loss for all classes during training. It behaved as expected and the average precision increased over time, while the loss,(see theory section2.4) decreased over time.
The info, right in figure 4.1, gives a more detailed look at the different classes. The cat and dog class has an average precision of approximately 80%. The human class shows a low average precision. The human class also has 594 false positives (FP) detections, which is not a satisfactory result. The hedge class has high average precision, but it has only 29 positive detection, which is low compared to the other classes.

### 4.1.1 Real-life Detection

Calculating the precision from the validation dataset will only tell the accuracy with a specific type of image used in the dataset, typically high resolution and with low noise. Good engineering practice would validate real-time detection and compare the theoretical result with what the system detects.

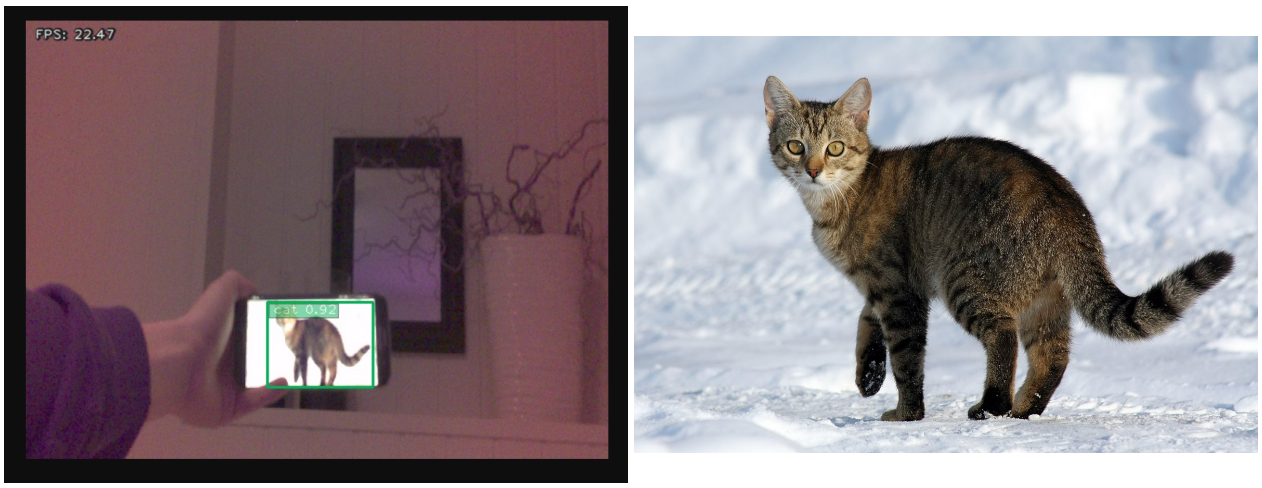



Figure 4.2: Result of Real-world test on the cat class(left), image used as test(right)[52]

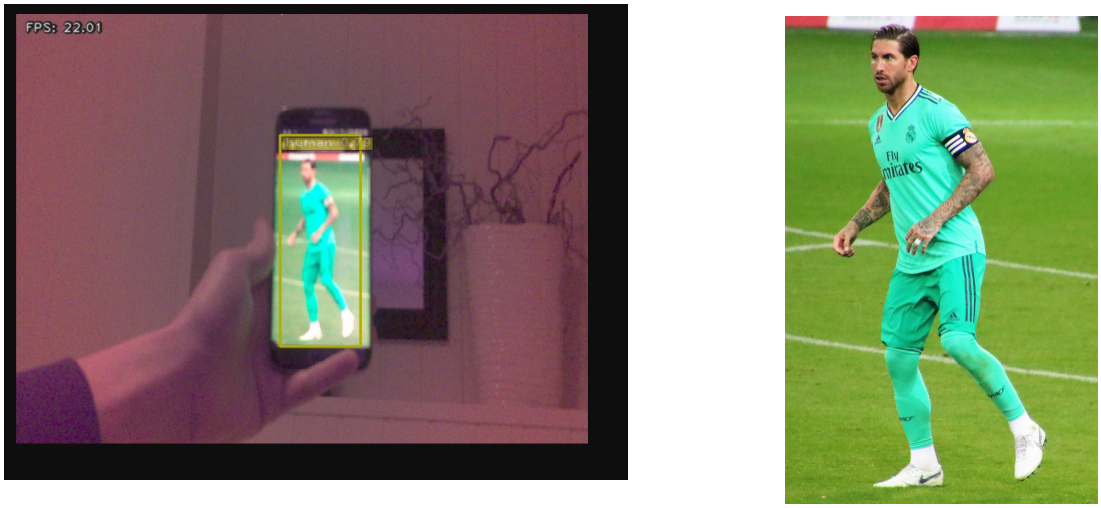



Figure 4.3: Result of Real-world test on the human class(left), image used as test(right)[54]

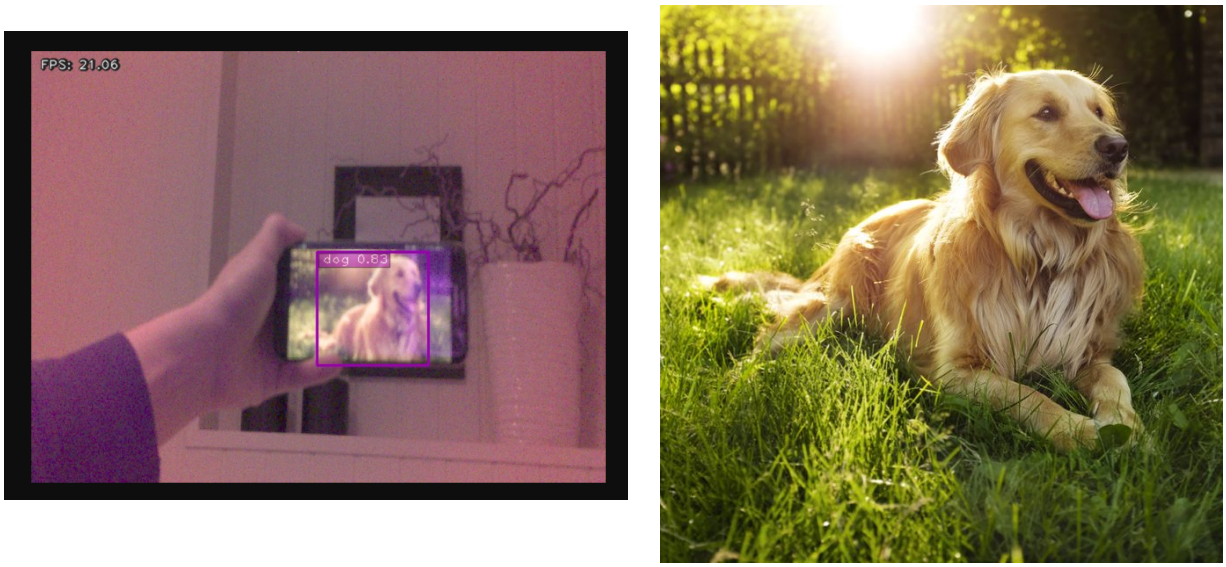



Figure 4.4: Result of Real-world test on the dog class(left), image used as test(right)

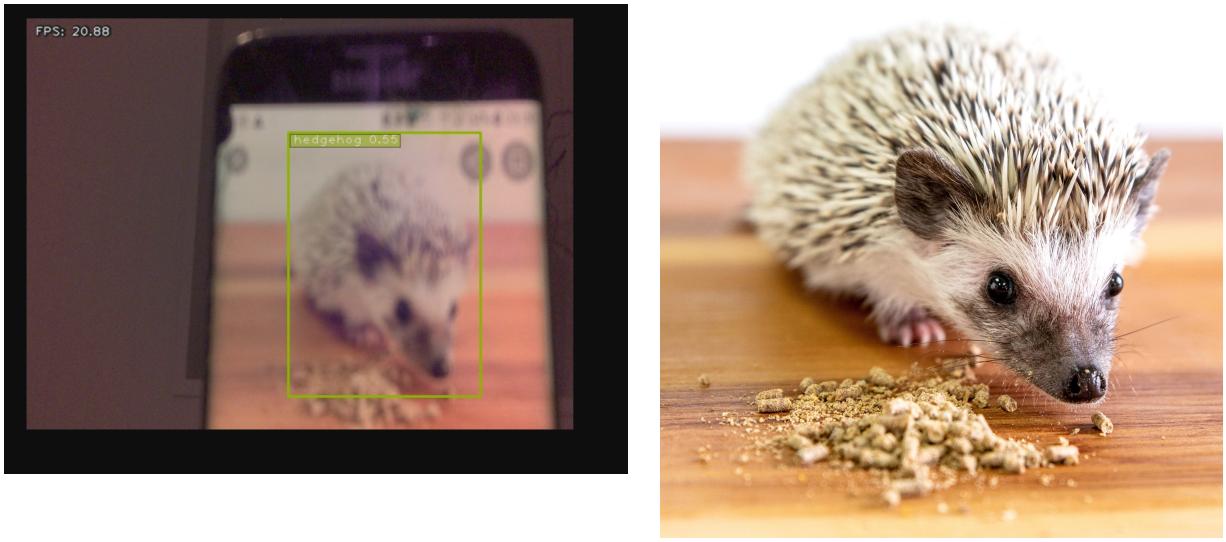



Figure 4.5: Result of Real-world test on the hedgehog class(left), image used as test(right)[31]

Calculating the precision from the validation dataset will only tell the accuracy with a specific type of image used in the dataset, typically high resolution and with low noise. Good engineering practice would validate real-time detection and compare the theoretical result with what the system detects.
The images were held as far away from the camera where they still had detection. The background light from the phone is turned back to 30% not to interfere or help more than necessary.
The real-world test shows some interesting results compared to the theory. The cat class was by far the best. Moving the image fast would not lose detection. Looking at figure 4.2 the appears small, meaning it could be targeted from afar. The cat also had reasonably high accuracy with 0.92 probability.
The human class was satisfactory. The images lost detection whit fast movement and had to be held closer than the cat image and with lower certainty. Compared to the 25% Average Precision, mainly due to false positives, it performed as expected real world. Dog images would often be detected as cats. If the image were further away than shown in figure 4.4, it would be detected as a cat. When moving the image rapidly, it would switch

between detecting a cat or a dog. The worst performing class for this evaluation was the hedgehog class. For the detector to work on a hedgehog, the image needed to be close and still.

### 4.1.2 Raspberry Pi NoIR

The Raspberry Pi camera without an infrared filter was implemented to use the system at night. The camera has a lot of background noise, so much it detects objects that are not there. Using two infrared light sources to aid the camera only made the image pink, and it did not help. Figure3.2 shows the difference between the NoIR camera and a regular phone camera with the same lighting. There is a considerable difference between the two cameras, and the NoIR camera has way more visibility.

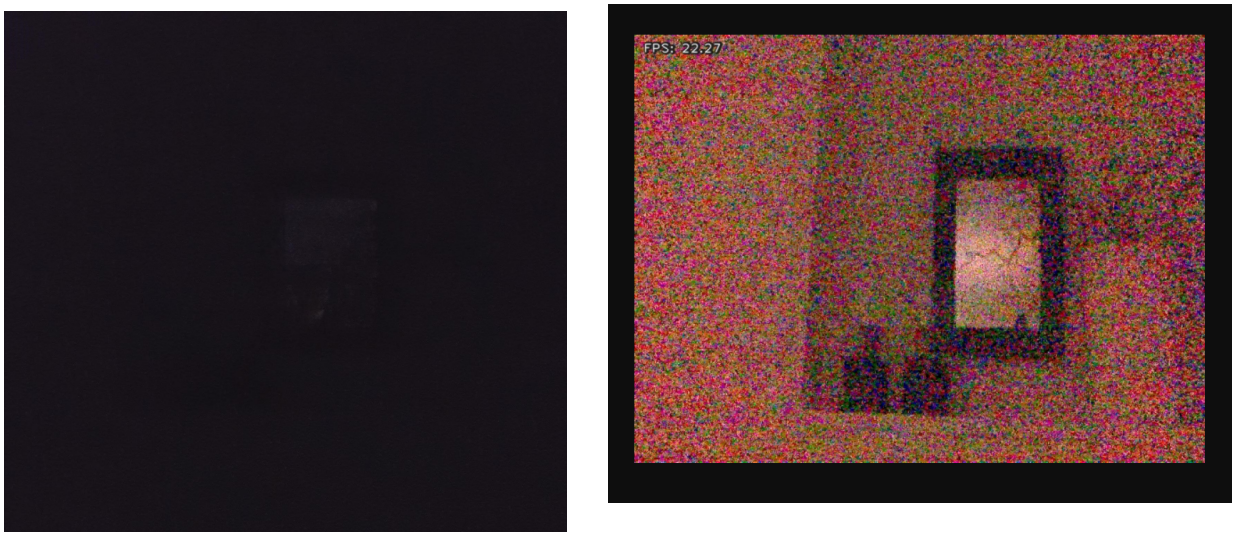



Figure 4.6: The images shows the difference between a regular camera and Raspberry pi NoIR at the same light level. Left: Phone camera, Right: Raspberry Pi NoIR

### 4.1.3 Frames Per Second - Jetson Nano

The target FPS from the target specification written at the beginning of this thesis was 60 FPS for an ideal result and 10 for marginal. The resulting FPS for YOLOv4-tiny trained with a custom dataset and implemented on Jetson Nano with TensortRT was 23 FPS. The reported FPS for Jetson Nano with YOLOv4-tiny TensorRT optimized is 25 FPS [28].

## 4.2 Object Tracking

The object detector was trained on Darknet. To get more performance out of the detector, the trained weights were converted to TensorRT.
YOLOv4 was integrated into ROS to use the bounding box values for the control algorithm from section 3.4.5. Motor control using the object detector and control algorithm was implemented in ROS.
Starting without motors, the control algorithm work and the output signals to the motors were as expected. Testing with servo motors showed unexpected behavior. The control algorithm only worked for a small step size when holding the image as center as possible. Trying to move the cat image to the edges, the motors either vibrate or set themselves to zero. Testing the servo motors without a control algorithm also showed not expected behavior. The motors would not move at specific steps or set themselves to zero, which is limited in the code. The motors were not replaced in time to make more extensive tests.

DeepSORT was not implemented in the system due to ram limitation on Jetson Nano. DeepSORT was tested on a windows computer to see how it behaves.
Code and video is sent in a zipped folder with the report.

## 4.3 Turret Design

A turret was designed with CAD software to house the components used for the animal deterrence system. Servo motors, a camera, and a laser were also mounted to make the development of a control algorithm easier. The build works as a two axis turret with 120 °of rotations around y-axis and x-axis. See figure4.7 and figure 4.8 for images of the animal deterrence system.

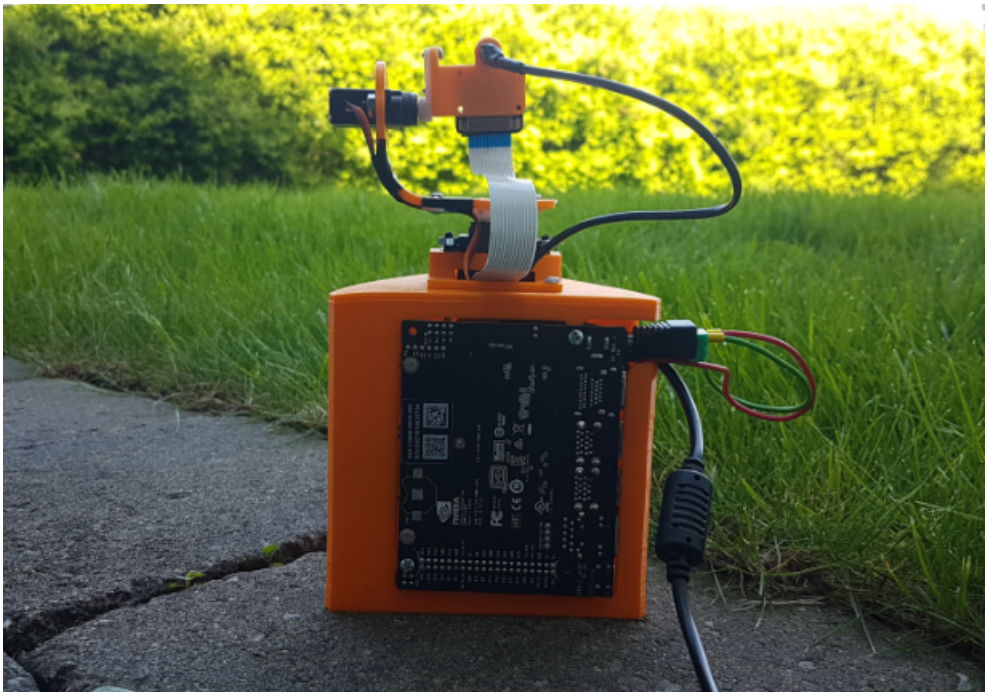
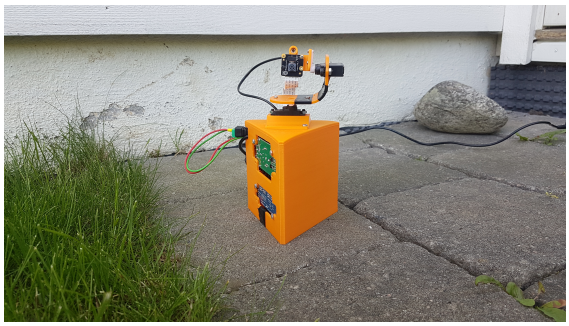


Figure 4.7: Animal deterrence turret (backside)



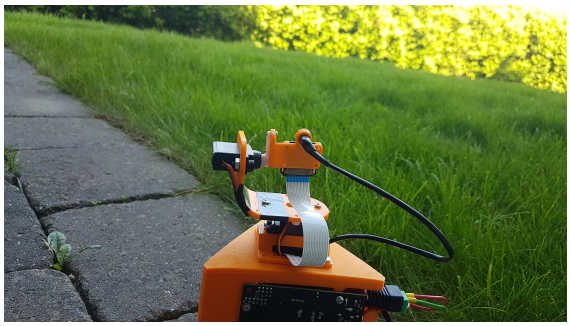


Figure 4.8: Animal deterrence turret

## 4.4 Final Specification

At the beginning of the thesis, product specifications and target specifications were made to aid the development of the animal deterrence system, see section 1.3. The following Final Specification is a summer of the reached targets.
For the Target specification range detection was not implemented. The turret is not water resistant.

# Final Specification

| Name: | Date: | Project: | Reached | Not-Reached |
|---|---|---|---|---|
| Group 8 | 10.06.2021 | Master Thesis 2021 | | |

| | | | |
|---|---|---|---|
| **1 Function/Requirements** | **Note** | --- | --- |
| ***1.1 Process*** | | --- | --- |
| 1.1.1 Detect cat using Image recognition | | X | |
| 1.1.2 Estimate position of cat | | | X |
| 1.1.3 Estimate pose of cat | | | X |
| 1.1.4 Track Movement and targeting system | | X | |
| ***1.2 Product*** | | --- | --- |
| 1.2.1 Reliable and durable in outside environment (IP-rating, must handle rain) | | | X |
| 1.2.2 Fast recognition of cat | | X | |
| 1.2.3 Wireless, using battery and Wi-Fi | | | X |
| 1.2.4 Usable at night (night vision camera) | | | X |
| ***1.3 Control*** | | | |
| 1.3.1 Autonomous | | --- | --- |
| | | | X |
| **2 Surroundings** | | --- | --- |
| ***2.1 Indoor*** | | --- | --- |
| 2.1.1 Test purposes | | X | |
| ***2.2 Outdoor*** | | --- | --- |
| 2.2.1 Outdoor testing and use | | | X |
| **3 Parts** | | --- | --- |
| 3.1 Nvidia Jetson Computer board | | X | |
| 3.2 Camera | | X | |
| 3.3 Mounting/housing | | X | |
| 3.4 Water gun system | | | X |
| **4 Project Plan** | | --- | --- |
| 4.1 months, master project deadline | | X | |
| **5 Cost** | | --- | --- |
| 5.1 Development: within master budget | | X | |
| 5.2 Production: not defined | | | X |
| **6 Product/assembly** | | --- | --- |
| 6.1 Simple production | | X | |
| 6.2 Design of prototype | | X | |
| 6.3 Full scale prototype with water gun system | | | X |
| **7 Safety** | | --- | --- |
| 7.1 no human target or other animals | | X | |
| 7.2 High accuracy for wanted target | | X | |
| 7.3 low force on target | | X | |
| 7.4 No injury possibilities | | X | |

## 4.5 Video of results

The video accompanying the report contains a short demonstration of how deepSORT would look like and work. The most important to notice is the ID that is given to the detected cat and human. This ID will make the control algorithm target a cat instead of just bounding boxes that can jump between cats or other targets. The number shows the ID instead of the confidence score. DeepSORT is running on a laptop without GPU and the FPS is very low compared to what it would with a GPU.


Figure 4.9: Image from deepSORT video-demo

The second part of the video shows that the object detector can follow the image of a cat even with fast movements. The value beside the cat shows the confidence score for the detected cat.

# Chapter 5

# Discussions

This chapter will cover different aspects of the methods used, cover challenges during the project, and the thesis's result.

## 5.1 Object Detector

### 5.1.1 Dataset

A good dataset is essential to train an accurate object detector. A good dataset would be lots of images, preferably over 2000 for each class[58]. The images should also be taken from different angles and distances. The optimal solution is to create a total custom dataset by taking pictures with the camera used in the project and ensure the labels are set correctly for each class. This would be difficult to achieve in the time frame for this project. Therefore OID was used to create the dataset.
One drawback of OID is that the images used are not optimal. They include paintings and vector art in the dataset. The difference between hedgehog images and human images is enormous. Even with more pictures with humans, the hedgehog mAP was much better when compared with the validation set. When looking at random samples from the dataset, the pictures in the hedgehog class were usually a hedgehog in the center, clear that the object was a hedgehog, and with no vector art or "funny" pictures.
The dataset for the human class usually contained multiple humans. It was hard to find all humans in the images when going through the dataset. the dataset for cats and dog was usually good, Even though there were some questionable images present.


Figure 5.1: This shows two images from the dataset. The picture to the left is an example of a bad picture (human class). The picture to the right is a good picture for this project(hedgehog class)

Using hedgehogs in the dataset made finding images and create a large dataset difficult. There were only approximately 350 images available of hedgehogs. This means that having 2000 images for the other classes would affect the Hedgehog class's performance, so the decision to only use 500 images for each class was made.

### 5.1.2 Real life evaluating of object detector

From results in section 4.1.1, the pictures show a more realistic result of the object detector than what the calculations can. In section 5.1.1 it was discussed that the dataset had questionable images that are not suitable to use. These images impacted the calculations and showed worse and better results (depends on the class) than what the object detector can perform. The hedgehog class showed good accuracy after training but was the worst-performing class when tested on the actual system with images. The human and the dog class did not perform to standard and was hard to detect with the camera. Both classes lost detection when moving the image. The cat class is performing better than expected and is considered a success. It had a good confidence score on all images tested and was stable when moving the image around in front of the camera.
Since the cat class performed well, re-training with improvements to the dataset will make the other classes perform as well or even better than the cat class.

### 5.1.3 YOLOv4-tiny

YOLOv4-tiny was used instead of YOLOv4 to increase FPS on Jetson Nano and reaching 23 FPS when optimized with TensorRT. Accuracy is compromised compared to YOLOv4, but the cat class was at a satisfactory level under testing. By optimizing the dataset and the object detector YOLOv4-tiny on Jetson Nano usable for this system.

### 5.1.4 Object Detecting at Night

Cat are often active at night, and it is wanted for the detector to work at night. The solution was to implement a camera that does not have an Infrared filter. The camera can read longer wavelength light that animals and humans can not see by using background IR-light or an additional light source. The camera outputs an image that is readable for humans or the object detector. The NoIR camera had too much noise, see result4.1.2. Under testing, the object detector detected different classes that were not present in the image. When applying more light with two IR light diodes, but the background noise got worse. A better quality camera with better light sources should make it achieve detection at night. The object detector is looking at shapes in the image, so the different color scales from the night camera should not impact performance a lot.
The camera out from low light condition is grey-scale, so a separate object detector was trained on gray-scale images to be tested with the NoIR camera to increase its performance. However, due to much noise, it was never tested.

### 5.1.5 YOLOv5, PP-YOLO, and Other Versions

YOLOv5 was first released in June 2020 and PP-YOLO in august 2020 and with many controversies. The controversies revolved around the lack of research papers and non-connections to the original researchers of YOLO and Darknet. The lack of documentation and validation made YOLOv4 the more attractive option since it still was relatively new. At the later stages of writing this thesis, many more exciting object detectors are coming to the market, Like YOLOv4-Scaled and PP-YOLOv2. Also, there are now many written articles supporting the validity of YOLOv5, but still no research paper. For further development, or new projects using object detector these new detectors is worth looking into.

## 5.2 Object Tracking

### 5.2.1 Control Algorithm

The control algorithm made for this system should work well for any similar system with the same known parameters. The positive with this approach is a fast and responsive system, which can be used for different motors and systems by only changing step count, resolution of the camera, or field-of-view parameters if parts are changed. The limiting factors for the accuracy of this method are mainly the servo motor's step resolution. Considering the distance to the target, the servo motor resolution will impact the accuracy the further away the target is. The distance to the target would be the radius of a circle around the system, and since the step resolution is the same, it means that the motor has to move less to cover the same distance as if the target was closer. Therefore the system could be improved by having higher stepper motor resolution for long-distance targeting. This would be the case for every control algorithm implemented.
During testing of the control algorithm, there were issues with the behavior of the system. The control algorithm did not move the way it was calculated to do. When developing the control algorithm, extensive checks were done to test if the theory was right. When implementing the algorithm on the system, more checks were done to see if it sent the right servoSetpoint to the motors, and it did. During testing, one servo motor broke, and the backup was installed. The control algorithm worked with a small step count with the backup motor but either started shaking or set the step to zero at random. Without more testing, it is difficult to blame the motor or the algorithm.
Implementing a PID-controller was also considered. PID-controller is a practical control algorithm to solve most control problems. Compared to the used approach, PID-controller will be slower and less accurate without proper tuning.
When a water gun system is implemented, it is important to choose a control method that match the system. It can be necessary to use a PID-controller or a state-space controller instead.

### 5.2.2 Importance of FPS

Performance is an essential topic in this thesis, and it has been written about accuracy and interference time (Frames Per Second). This system is dependent on operating at real-time speeds. Real-time speeds can have different meanings, but it is reasonable to be defined as 30Hz (30 FPS) and above in this case. The main reason to pursue higher FPS is to target fast-moving objects, like cats or dogs. Simply put, the faster the system can detect and calculate positions, the probability of hitting the target increases. If the system is slow, the target may have exited the area or image frame before engaging. The control algorithms' calculation frequency is dependent on the object detector's interference time. If it can calculate faster, it can send a signal to the servomotors at a shorter interval, and it can set a new position faster and reduce jerk and be more smooth.

### 5.2.3 DeepSORT

DeepSORT was not implemented in the system. When implementing DeepSORT on Jetson Nano, there fast not enough available memory on the system to run it. A video with a demonstration of how DeepSORT would look like was delivered with the thesis. The object detector with DeepSORT in the video ran with the YOLOv4 weights trained for this thesis. DeepSORT will solve two challenges with the object tracker. When the YOLOv4 detects an object, it has no association between each detection, meaning it does not know if it is the same cat or not. DeepSORT solves by using a feature extractor and compare the detection, and gives the object an ID [56]. This can be used to track the ID instead of the first bounding box with the suitable class. DeepSORT also comes with a more sophisticated tracking algorithm, using a Kalman filter to keep tracking even if the view

is obstructed or loss of detection. DeepSORT will track and keep the ID for the target for 30 frames before considering it lost or exited the frame[56]. Its important to note that the implementation of deepSORT will impact the FPS performance of the system. The performance impact will vary and cant be estimated or concluded before testing together with the total system

### 5.2.4 ROS and Implementing Software

Implementing the object detector in ROS was not straightforward. Running ROS with python 2.7 while the object detector runs on python 3.x. The solution to run the detector in a subprocess work but is not the desired solution. Having the object detector integrated into ROS makes it easier to extract bounding box and class information without converting them from numbers to strings.

## 5.3 Hardware

### 5.3.1 Nvidia Jetson Nano

The intent was to use NVIDIA Jetson Xavier AGX for the system. However, Xavier Developer Board is missing MIPI CSI-2 camera connector, which other Jetson model has. It is possible to buy a connector and good quality camera modules for Xavier AGX that could be implemented. At the time of writing, these cameras and connectors were too expensive to acquire and deemed not reasonable for this project. The solution was to use a Jetson Nano instead. Jetson Nano is a reasonable option in terms of price, but at the cost of performance. This choice opens up the opportunity to test the difference in performance and make a more qualified recommendation as to what system is worth using for the final specification. Reaching an average of 23 FPS is under the ideal target of 60 FPS and the real-time FPS mark of 30. However, the achieved 23 FPS is usable and will work for this system. Using Jetson Nano is at the limit of what is needed and can be recommended. Using YOLOv4, the non-tiny version, and the older version of YOLO will require a better GPU.

### 5.3.2 Turret System

The turret system built for this thesis is not a proposed solution for how a final version of the animal deterrence device is supposed to be. It was created to house the Nvidia Jetson Nano and test the control algorithm and object detector with the raspberry pi camera, and its purpose is to be a testing platform. At the current configuration, parts are mounted directly onto the servo motors without any support. This is considered bad engineering practice and should be avoided when designing a more permanent solution.

#### Camera

The Raspberry Pi NoIR camera did not meet expectations after testing. It was used with 720p to keep the frame rate high. As the results show, the camera is usable in the daytime. Cameras with lower quality are not recommended to use with object detectors. The low light images have lots of noise and distortion that impact the object detector's effect and even detect objects in the noise present. It was also tested by using too small IR LED to improve the image. The IR LED light was clearly visible and made the image lighter. However, the image quality and noise were still bad. It should be considered to implement bigger light sources if the cameras are to be used further.

#### Servo Motors

The servo motors used were helpful for testing and are cheap and accessible since they were already on hand. It is not recommended to use these for a final product. The JX 1171MG has limited documentation and only provided with a sticker on its packaging.

## 5.4 Water Gun System

The product specification called for using a water gun as the method for scaring cats. Water is safe, accessible, and known to scare animals and even people when surprised or sprayed with water. The final product will include a mechanism that sprays a jet of water directly on the target. Due to time limitations to develop the water gun system, a laser was implemented instead to visualize if the water spray would hit the target. It was also considered to eliminate the water gun idea and instead use the laser as the solution. A laser could damage eyes and skin, and there is no wish to harm any target. Even with a low-powered laser, there still would have been a risk. After careful consideration, water would still be the best solution.

# Chapter 6

# Conclusions

This project aimed to develop an animal deterrence system that uses state-of-the-art methods to make a system to keep cats away from an area.

## 6.1 Object Detection

An objected detector was trained and could identify the objects wanted in this thesis. The object detection accuracy for cat class and human class was reasonable compared to what's expected for YOLOv4-tiny, But the dog class and hedgehog class should be improved.
Reaching 23 FPS with Nvidia Jetson Nano was under the target set at the beginning of the thesis, but it can be concluded that 23 FPS is enough to make the system usable.
Detection at night was not reached. The Raspberry Pi NoIR camera did not perform as expected. Due to the noise level on the NoIR camera the object detector could not work.

## 6.2 Control Software

The object detector was successfully implemented with ROS. The object detector was not implemented inside the ROS environment due to different python versions but can now be used by scripts in ROS.
The control algorithm works by converting the detected object's position in the image and converting the position to setpoints sent to the servo motors. During testing, the control algorithm did not work as expected, and it can not be concluded as a success without more testing.

## 6.3 Turret Design

A Two-axis camera turret was designed. The servo motors are used to rotate the camera to follow the cat or animal. The turret was designed in CAD with four separate parts that are easy to 3D-print. The undersection/base of the turret holds all the necessary components to work.

## 6.4 Targets Reached

At the start of this project, a set of product specifications was set(see section 1.3). The reached targets were covered in section 4.4. The targets in Product specification were reached. The optional target should be used if it is wanted to continue the work.

# Chapter 7

# Further Work

This chapter goes through some topics that should be looked into if the work is continued.

## 7.1 Object Detector

### 7.1.1 custom dataset

The dataset used in this thesis only contained approximately 500 images for each class. For further development, a more extensive dataset with 2000 images for each class should be trained and tested against the current result.

## 7.2 Object Tracking

The tracking algorithm should be further tested. A PID-controller should also be implemented and tested.
DeepSORT is not necessary to have a good control algorithm but should make it easier to keep control if more animals are present in the image. DeepSORT should be integrated into the main code, and make use of the integrated Kalman filter should make the Control algorithm more robust. Since deepSORT adds a convolution neural network, the performance should be tested again to see if there is performance loss that will impact the system.

### 7.2.1 Distance Sensor

A distance sensor or distance algorithm would be a helpful implementation combined with the water gun system. If water should hit the cat up to 10m away from the water gun nuzzle, it needs to have an angle to counteract gravity. For a more precise hit, finding the range to the target is necessary to calculate the angle. One way would be to install a Radar or Lidar to the system. Another option would be to implement the Perspective-n-Point algorithm[39]. This method is most accurate for an object of the same size, and cat comes in different shapes and sizes. This method would be interesting to test out for this system.

## 7.3 Water Gun Design

Designing a water gun system for this project description should be done after the project specification in section 1.3. One solution would be to use the garden tap to access water and pressure. A water solenoid could release the water into a designed nozzle with a set output diameter to accurately calculate the water velocity. Water velocity is needed to find the distance the water would travel at different angles. This new targeting method should be integrated with the control algorithm when developing the water gun turret.

# Bibliography


[1] *About OpenCV*. 2020. URL: https://opencv.org/about/. (accessed:06.05.2021).

[2] *About ROS*. 2015. URL: https://www.ros.org/about-ros/. (accessed:06.05.2021).

[3] *Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685*. (accessed:15.04.2021).

[4] *ADVANCED EMBEDDED SYSTEMS FOR EDGE COMPUTING*. —-. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. (accessed:06.05.2021).

[5] AlexeyAB. *Yolo v4, v3 and v2 for Windows and Linux*. URL: https://github.com/AlexeyAB/darknet.

[6] *Anchor Boxes for Object Detection*. URL: https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html. (accessed:06.06.2021).

[7] Alex Bewley et al. "Simple online and realtime tracking." In: *2016 IEEE International Conference on Image Processing (ICIP)* (Sept. 2016). DOI: 10.1109/icip.2016.7533003. URL: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[8] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[9] B.O. Braastad. *Katt*. URL: https://snl.no/katt. (accessed:11.03.2021).

[10] *Camera Module V2*. URL: https://www.raspberrypi.org/products/camera-module-v2/. (accessed:12.05.2021).

[11] *Camera Module V2*. URL: https://developer.nvidia.com/cuda-zone. (accessed:15.05.2021).

[12] *Common Objects in Context*. URL: https://cocodataset.org/#home. (accessed:12.04.2021).

[13] *Convolutional Neural Networks (CNNs / ConvNets)*. URL: https://cs231n.github.io/convolutional-networks/. (accessed:22.05.2021).

[14] *DeepSORT: Deep Learning to Track Custom Objects in a Video*. (accessed:25.04.2021).

[15] *Detection Evaluation*. URL: https://cocodataset.org/#detection-eval. (accessed:01.06.2021).

[16] NVIDIA DEVELOPER. *Jetson AGX Xavier Developer Kit*. URL: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit. (accessed:14.04.2021).

[17] NVIDIA DEVELOPER. *Jetson Developer Kits*. URL: https://developer.nvidia.com/embedded/jetson-developer-kits. (accessed:15.04.2021).

[18] NVIDIA DEVELOPER. *Jetson Nano 2GB Developer Kit*. URL: https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit. (accessed:14.04.2021).

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[20] The AI Guy. *yolov4-deepsort*. 2020. URL: https://github.com/theAIGuysCode/yolov4-deepsort.

[21] *How to Choose an Activation Function for Deep Learning*. (accessed:22.05.2021).

[22] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. 2018. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a311731. (accessed:31.05.2021).

[23] IBM. *What is edge computing?* URL: https://www.ibm.com/cloud/what-is-edge-computing. (accessed:05.05.2021).

[24] *Image Annotation*. URL: https://lionbridge.ai/services/image-annotation/. (accessed:25.04.2021).

[25] Zicong Jiang et al. "Real-time object detection method based on improved YOLOv4-tiny." In: *CoRR* abs/2011.04244 (2020). arXiv: 2011.04244. URL: https://arxiv.org/abs/2011.04244.

[26] JONATHANDAVIDSTEELE. *Image of Dog*. (accessed:10.06.2021).

[27] Ross Girshick Joseph Redmon Santosh Divvala and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: (2016). URL: https://arxiv.org/pdf/1506.02640.pdf. (accessed:04.05.2021).

[28] JK Jung. *tensorrt _demos*. 2020. URL: https://github.com/jkjung-avt/tensorrt%20%5Ctextunderscore%20demos.

[29] Roland Kays. *Test of ultrasonic cat repellent*. URL: https://www.youtube.com/watch?v=mv13QDct9OY. (accessed:11.03.2021).

[30] Andreas Klausen. *bsc-cat-2021*. 2021. URL: https://github.com/andrek10/bsc-cat-2021.

[31] Adrienne Legault. *The Spruce*. (accessed:10.06.2021).

[32] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[33] Wei Liu et al. "SSD: Single Shot MultiBox Detector." In: *Lecture Notes in Computer Science* (2016), 21â€"37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[34] Vishal Mandal and Yaw Adu-Gyamfi. "Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis." In: (2020). URL: https://arxiv.org/ftp/arxiv/papers/2007/2007.16198.pdf.

[35] *mAP (mean Average Precision) might confuse you!* URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2. (accessed:01.06.2021).

[36] McKinsey and Company. *THE INTERNET OF THINGS: MAPPING THE VALUE BEYOND THE HYPE*. 2015.

[37] *NVIDIA TensorRT*. URL: https://developer.nvidia.com/tensorrt. (accessed:10.06.2021).

[38] *Pi NoIR Camera V2*. (accessed:15.04.2021).

[39] *Real Time pose estimation of a textured object*. URL: https://docs.opencv.org/3.4/dc/d2c/tutorial_real_time_pose.html. (accessed:06.06.2021).

[40] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[41] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[42] *Robot Operating System Wiki*. URL: http://wiki.ros.org/. (accessed:03.06.2021).

[43] Pierrick Rugery. *Explanation of YOLO V4 a one stage detector*. 2020. URL: https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7. (accessed:04.05.2021).

[44] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (accessed:26.05.2021).

[45] Daryl Sowers. *Water Sprinkler Soaks Cats - Very Funny Part 1*. URL: https://www.youtube.com/watch?v=IrgUqxFik7k. (accessed:11.03.2021).

[46] *The PASCAL Visual Object Classes*. URL: http://host.robots.ox.ac.uk/pascal/VOC/. (accessed:12.04.2021).

[47] theAIGuysCode. *OIDv4_ToolKit*. URL: https://github.com/theAIGuysCode/OIDv4_ToolKit. (accessed:10.06.2021).

[48] Sachin Thorat. *What is Mechatronics System*. URL: https://learnmech.com/what-is-mechatronics-system-application-of-mechatronics-system/. (accessed:13.03.2021).

[49] Turnah81. *How you can make a Cat Repellent for Under $15 in Parts!* URL: https://www.youtube.com/watch?v=ElcviGYMb3U. (accessed:11.03.2021).

[50] unknown. *Cats at night*. URL: https://www.cats.org.uk/help-and-advice/home-and-environment/cats-at-night. (accessed:11.03.2021).

[51] Angelo Vittorio. *Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset*. 2018. URL: https://github.com/EscVM/OIDv4_ToolKit.

[52] Von.grzanka. *Image of Cat*. URL: https://no.wikipedia.org/wiki/Tamkatt#/media/Fil:Felis_catus-cat_on_snow.jpg. (accessed:10.06.2021).

[53] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2020. arXiv: 2011.08036 [cs.CV].

[54] Werner100359. *Image of Human*. (accessed:10.06.2021).

[55] Nicolai Wojke and Alex Bewley. *Deep Cosine Metric Learning for Person Re-identification*. IEEE. 2018. DOI: 10.1109/WACV.2018.00087.

[56] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. IEEE. 2017. DOI: 10.1109/ICIP.2017.8296962.

[57] *X PDI-1171MG 17g Metal Gear Core Motor*. URL: https://www.aliexpress.com/item/4000811149213.html. (accessed:10.06.2021).

[58] *Yolo v4, v3 and v2 for Windows and Linux*. URL: https://github.com/AlexeyAB/darknet/blob/master/README.md. (accessed:27.04.2021).

[59] *YOLOv4*. URL: https://github.com/ultralytics/yolov5. (accessed:05.06.2021).