Recent Advances in Machine Learning - Object Detection and Robot Control

University of Siegen

Shantanu Shirsath 1699551

Aakash Deshpande 1701953

Abstract

Object detection is one of the most prominent tasks in the field of computer vision that involves identifying and localizing objects present in a frame of image or video. This is a crucial method which finds applications in fields like autonomous driving, video surveillance, medical imaging, etc.

In this report, we focus on the implementation of real-time object detection and control of a robot. The project features integration of deep learning knowledge and hardware setup along with control engineering. Each segment of this project is well explained with the help of graphs or images.

1 Introduction

The main objective of this project is to enable a fischertechnik robot to detect and chase a specific object, a Cat in this case. A pre-trained deep learning model is retrained using a custom made dataset and deployed on a Raspberry Pi. The robot uses a webcam and the model to detect and track a cat in real-time. PID controllers compute motion commands based on the cat's position and size, enabling responsive and continuous pursuit.

We have combined various datasets to form a single integrated dataset that is well balanced, keeping in mind the deep learning aspect or retraining of a model. The topic of Dataset generation will put more light on this.

Selecting a deep learning model is a crucial task as the efficiency of the whole system depends on how well the model is able to predict or identify the object. Resons for selection of YOLOv8n[1] and retraining will be discussed in the subsequent sections.

Once the model is trained and a checkpoint point is generated, it needs to be integrated on a hardware, in this case, a Raspberry Pi-5. The conversion methods and the lightweight inference engine selected are very important to make the model more robust. We have selected a high performance neutral network inference framework named NCNN developed by Tencent. The details of which are discussed further.

Once the hardware is able to the real time inference and is ablet to classify the objects captured by the robot's webcam, the control aspect of this project comes into play. We have used 2 simple PID controllers to control the motion of the robot based on the errors generated. More about the Control engineering is mentioned in the section Control Engineering.

2 Dataset generation

For any Deep learning project which deals with training or retraining, the data plays a major role. A right dataset can help save countless hours spent fine tuning the model to increase its efficiency (collection of numerous parameters). Since we want our robot to detect a Cat, we have generated a mega-dataset of Cats. In initial stage, the dataset was not balanced and many issues were observed. The following table 1, lists the problems and action taken to solve that problem and table 2 shows how the dataset was distributed and balanced in the old and new dataset.

Problem	Solution
Model assumed a cat is always present.	Added non-cat images to reduce overfitting.
High false positives on cat-like objects.	Balanced cat and non-cat ratio to 50-50.
Model detected only one cat breed.	Added diverse breeds and custom cat images.

Table 1: Summary of key issues in the old dataset and corrective actions taken.

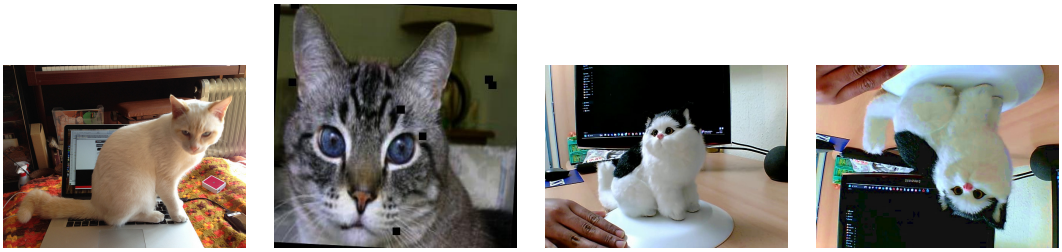
Source	Category	Old Dataset		New Dataset	
		Share (%)	Count	Share (%)	Count
COCO[2]	Non-cat	10%	478	50%	8,794
COCO	Cat	90%	4,298	25%	4,298
Roboflow[3]	Cat	–	–	14%	2,433
Custom	Cat	–	–	11%	2,063
Total		**100%**	**4,776**	**100%**	**17,588**

Table 2: Comparison of old and new dataset distributions used for cat vs. non-cat image classification.

The generation of a custom dataset included recording a video and extracting the frames. Then each frame was augmented as mentioned below.

- **A.OneOf([...], p=1.0)**
  Applies exactly one of the following three transformations (with 100% probability):
  - **A.HorizontalFlip(p=1.0)**: Flip the image left to right.
  - **A.VerticalFlip(p=1.0)**: Flip the image top to bottom.
  - **A.Rotate(limit=30, p=1.0)**: Rotate the image at random angle between $-30°$ & $+30°$.
- **A.RandomBrightnessContrast(p=0.5)**
  With 50% probability, randomly changes:
  - **Brightness**: Makes the image lighter or darker.
  - **Contrast**: Increases or decreases the difference between light and dark areas.
- **A.HueSaturationValue(p=0.5)**
  With 50% probability, randomly adjusts:
  - **Hue**: Adjusts the color tone.
  - **Saturation**: Modifies the intensity of colors.
  - **Value**: Alters the brightness component in HSV color space.

After the augmentation, each image was annotated using inference of YOLOv8n itself and incorporated in the main dataset.



COCO Cat | Roboflow Cat | Custom Cat | Augmented Cat

Figure 1: Sample cat images from various datasets

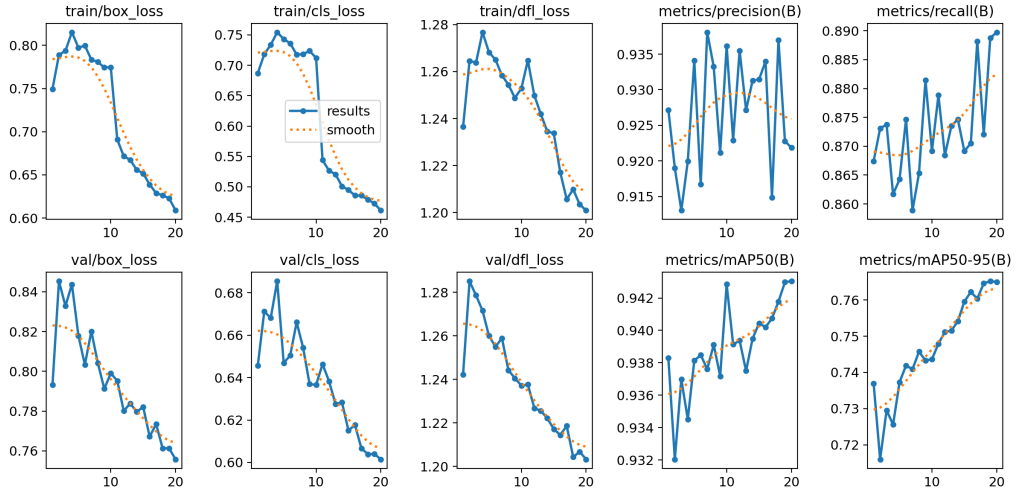
3 Selecting, training and Implementation of the model onto a Raspberry Pi-5

3.1 Selection and training

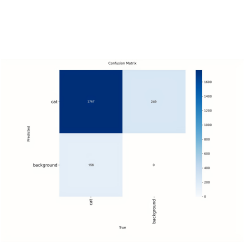
The selection of the model was dependent on the following things

- The architecture, model size and training simplicity.
- The Accuracy and speed of the inference.
- The Simplicity of exporting it onto Raspberry Pi.

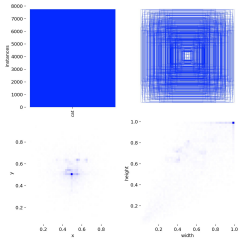
After comparing the models like YOLOv5, YOLOv8n, SSD(with mobilenet), Faster R-CNN, EfficientDet and RetinaNEt, we concluded that YOLOv8n is the best model to use for our application. Since the Model is already taught to extract the features on a gigantic dataset, there was no need to retrain it completely with our small dataset. The training was carried out by freezing the Backbone and Neck of the model which form the first 21 layers. The head which is the 22nd layer is responsible for classifying the objects and we fine turned the model to classify only 1 class i.e Cat. After splitting the data into train and validation (4:1), the training was carried out in 2 stages. First for 30 epochs and then for 20 more epochs. The results generated by the YOLO are as shown below:



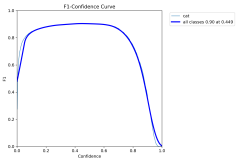
Training Curves



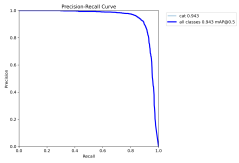
Confusion Matrix



Label Distribution



F1 Score vs Confidence



Precision–Recall Curve

3.2 Implementation on Raspberry Pi 5

For evaluating inference performance on Raspberry Pi, we tested four popular model formats with inference using the ultralitics library: **PyTorch (.pt)**, **TorchScript**, **ONNX**, and **NCNN**. Below is a brief overview of each:

- **PyTorch (.pt)**: Native PyTorch model format used during training and initial evaluation.
- **TorchScript**: A serialized and optimized representation of a PyTorch model, enabling deployment in C++ environments.

- **ONNX (Open Neural Network Exchange)**: An open format designed to allow interoperability between different deep learning frameworks.
- **NCNN**: A high-performance neural network inference framework optimized specifically for mobile and edge devices, especially those using ARM processors.

Inference Performance on Raspberry Pi

We benchmarked each format on four key metrics: preprocessing time, inference time, post-processing time, and achieved FPS (frames per second). The results are summarized in Table 3.

Table 3: Inference Performance Comparison

Format	Preprocess (ms)	Inference (ms)	Postprocess (ms)	FPS
TorchScript	5–8	456	0.5–1.5	1.5–2
PyTorch (.pt)	5–6	350	0.5–1	2.5–3
ONNX	5	300	0.5–1	3–3.5
NCNN	5–6	110	1–2	8–9

From the above comparison, it is evident that the NCNN format significantly outperforms others in terms of inference speed and achieved FPS on Raspberry Pi. This is primarily due to NCNN's design, which is optimized for ARM-based architectures and lightweight inference on edge devices. Therefore, for real-time deployment on Raspberry Pi, **NCNN was chosen as the preferred format** due to its efficiency and low-latency performance.

4 Control Engineering

We control two axis motion of the Robot x-Axis (lateral motion) and y-axis (longitudinal motion). In order for the robot to chase the cat, the control problem can be defined so as to minimize the distance between robot and cat along both x and y axis. The motion along x-axis defines the lateral distance or the direction in which the robot points while the motion along the y axis defines the heading direction of the robot.

4.1. Motion Along the X-Axis

To control the motion along x-axis we take help of the bounding box center. We calculate the bounding box center and try to minimize the distance between bounding box center and the frame center of the camera feed. This helps the robot to adjust its direction towards the object trying to keep the object always at the center of the frame.

4.1.1 lateral controller

To minimize the lateral distance between the center of the camera frame and the center of the bounding box (i.e., the cat's position), we first compute the error:

$$e(t) = x_{\text{center}}^{\text{frame}} - x_{\text{center}}^{\text{bbox}} \tag{1}$$

where:

- $x_{\text{center}}^{\text{frame}}$ is the x-coordinate of the center of the camera frame,
- $x_{\text{center}}^{\text{bbox}}$ is the x-coordinate of the center of the detected bounding box.

This error $e(t)$ represents how far off-center the object (cat) is in the horizontal direction. To correct this error and steer the robot to keep the object centered, we use a Proportional-Derivative (PD) controller.

A PD controller generates a control signal based on the present error and the rate of change of the error. The control law for a PD controller is given by:

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{de(t)}{dt} \tag{2}$$

where:

- $u(t)$ is the control signal (e.g., angular velocity or steering adjustment),
- K_p is the proportional gain,
- K_d is the derivative gain,
- $e(t)$ is the current error,
- $\frac{de(t)}{dt}$ is the derivative of the error with respect to time.

The proportional term ($K_p \cdot e(t)$) provides a correction proportional to the error, encouraging the robot to move such that the object is at the center of the frame. The derivative term ($K_d \cdot \frac{de(t)}{dt}$) helps dampen the response and reduce overshooting by considering how quickly the error is changing.

The computed control signal $u(t)$ is then sent to the motor controller to adjust the robot's orientation, ensuring that the object remains centered in the camera frame.

If we assume that the sampling time Δt is constant, we can approximate the derivative term using the finite difference method:

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-1)}{\Delta t} \tag{3}$$

Substituting this into the PD control law:

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{e(t) - e(t-1)}{\Delta t} \tag{4}$$

Since Δt is constant, we can define a new derivative gain:

$$K'_d = \frac{K_d}{\Delta t} \tag{5}$$

Thus, the simplified discrete-time control law becomes:

$$u(t) = K_p \cdot e(t) + K'_d \cdot (e(t) - e(t-1)) \tag{6}$$

This form eliminates the explicit time derivative, making it more suitable for digital implementation on a embedded system.

4.1.2. Motion Along the Y-Axis

To control the motion along the y-axis (i.e., forward or backward movement), we rely on the area of the bounding box detected around the object (the cat). Assuming the cat has a roughly constant physical size, a smaller bounding box area indicates that the cat is farther from the robot, while a larger area implies proximity.

We define a target area A_{target} that represents 80% of the total camera frame area:

$$A_{\text{target}} = 0.8 \cdot A_{\text{frame}} \tag{7}$$

The control objective is to maximize the bounding box area A_{bbox} such that it approaches A_{target}. To do this, we define the error in area as:

$$e(t) = A_{\text{target}} - A_{\text{bbox}}(t) \tag{8}$$

A positive error means the object is still too far away. To minimize this error, we apply a **Proportional (P) controller**, which adjusts the control signal based on the current error:

$$u(t) = K_p \cdot e(t) \tag{9}$$

where:

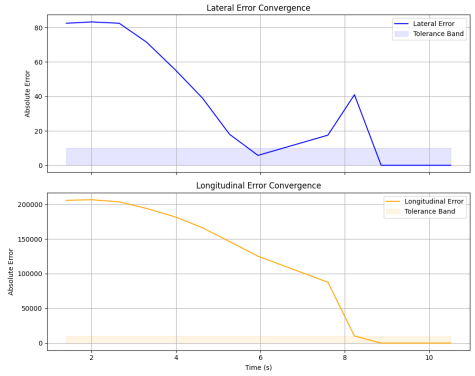
- $u(t)$ is the control signal (linear velocity sent to the motors),
- K_p is the proportional gain,
- $e(t)$ is the current area error.

The control signal $u(t)$ determines the forward motion of the robot. As the robot approaches the object and the bounding box area increases, the error decreases, causing the robot to slow down and helping it maintain a desired distance from the object.

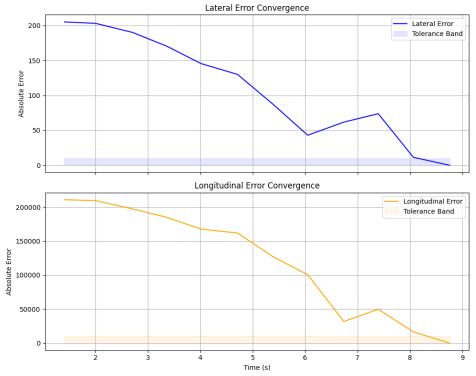
5 Controller tuning

5.1 Lateral PD Controller Behavior

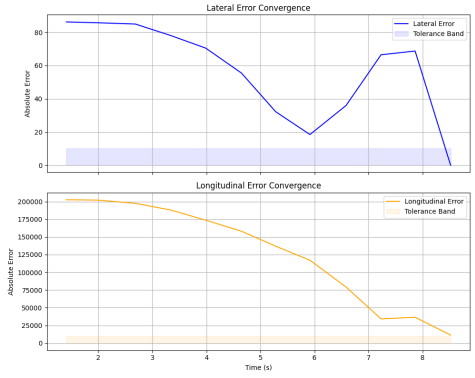
The PD controller for lateral motion was tuned by varying the derivative gain K_d while keeping the proportional gain K_p fixed. As shown in Figure **??**, increasing K_d initially improves response time by reducing overshoot. However, higher values also lead to increased oscillations and instability due to the derivative term's sensitivity to noise and sharp error changes.



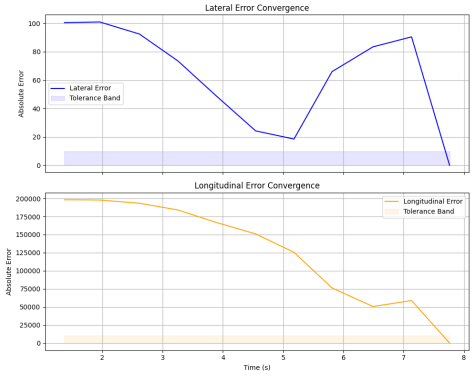
(a) $K_d = 0.1$



(b) $K_d = 0.3$



(c) $K_d = 0.6$



(d) $K_d = 0.9$

Figure 2: Comparison of Lateral and Longitudinal Error Convergence for varying K_d in the PD controller

As seen in the plots, a lower K_d results in smooth and stable convergence, whereas higher K_d values introduce fluctuations and delay in settling within the tolerance band.

5.2 Longitudinal P Controller Behavior

The longitudinal controller, being purely proportional, exhibits a monotonically decreasing error profile across all experiments. Its response is inherently slower but stable, as shown in all subplots of Figure 2. The absence of derivative or integral action avoids oscillations but limits responsiveness.

5.3 Conclusion

From the tuning experiments:

- Low to moderate K_d values offer a good trade-off between responsiveness and stability in PD control.
- High derivative gain can lead to undesirable oscillations and overshoots due to noise amplification.
- The longitudinal P controller provides stable convergence, but the system could benefit from adding derivative or integral components for faster performance.But due to the limitation of the responsiveness of our hardware we limit our controller to simple p controller.

References

[1] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics yolo, 2023.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[3] Mohamed Traore. Cats dataset. `https://universe.roboflow.com/mohamed-traore-2ekkp/cats-n9b87`, nov 2022. visited on 2025-07-10.