Object Tracking on Mobile Raspberry Pi Robot with YOLOv8 and YOLO11

Stevie Bronsard

January 2026

Abstract

Object detection has long been a complicated problem in computer vision, and has only recently rapidly improved thanks to the creation of deep learning algorithms. YOLO is one such algorithm which creates bounding boxes over objects it is trained to identify. There have been many different versions of the YOLO algorithm over the past decade, from YOLOv2 to YOLOv26. This project aims to compare how the YOLOv8s and YOLO11s algorithms differ in efficiency when applied to a non-stationary robot using picture-based object tracking software when following a moving target. The vehicular robot, called *Follower*, runs Python on a Raspberry Pi 4 and used a small fixed camera to capture images. Those images are wirelessly sent to a nearby computer and processed with a YOLO model. Instructions are sent back to *Follower* so that it may find and track the target.

1 Introduction

Object tracking technology has developed exponentially the past few years, from basic goals like detecting objects on a video file to the creation of the first self-driving car. This software can also be used when studying wildlife: robots disguised as small animals are sent to infiltrate colonies in order to gather data like behavioral patterns. These robots are typically remote controlled, thus relying on a researcher controlling them nearby, which can impact the data if the subjects sense that the researcher is nearby. The presence of humans is well known to make wild animals behave differently, so making these robots autonomous allows for even more accurate data. However, there is a significant risk, as an autonomous robot could accidentally harm the community it is observing. Carelessness when coding the robot could lead it to step on an egg or accidentally push a subject, thus choosing the right algorithm is crucial to prevent damage. The robot requires rapid remote processing speed and high model accuracy to avoid these scenarios.

2 Literature Review

Ultralytics YOLO (You Only Look Once) [1] is an object detection algorithm which creates a single end-to-end regression problem that predicts bounding boxes and class probabilities in one pass through a neural network. YOLO works by creating a grid on an input, then running a Convolutional Neural Network model (CNN) to predict bounding boxes for each box in the grid. YOLO then creates a class probability map to find which adjacent boxes have a significant object overlap. Finally, YOLO performs non-max suppression (a process to filter out insignificant bounding boxes). The result consists of large bounding boxes enclosing the significant detected objects (5). YOLO has evolved many times over the years, like YOLOX, YOLOv8, and YOLO11 [2].

3 Hypothesis

In this paper we present our own open-source wireless object tracking software using a raspberry pi robot. We created two YOLO models which can be used for terrestrial vehicles using two different YOLO algorithms: YOLOv8s and YOLO11s. We then compare the efficiency of each and offer a guide to setting up the necessary software for the robot.

Based on the data from the Ultralytics website, YOLO11s tends to have slightly lower latency and a higher mAP than YOLOv8s ([3]). The mAP refers to the mean Average Precision, which is the evaluation metric for object detection model's accuracy. In essence, it is a way to evaluate how tight the bounding box around the detected object is. When used on the MS COCO dataset, YOLO11s had a latency of 2.5ms/img and an mAP of 47 while YOLOv8s had a latency of 2.66ms/img and an mAP of 44.9. This indicates that YOLO11s might be slightly faster and more accurate than YOLOv8s when tested with *Follower*.

4 Methods

The design of this project's robot takes place in four sections: assembling the robot parts, training our object detection models, implementing the runtime architecture, and testing the YOLO models. The two object detection algorithms we import are Ultralytics YOLOv8s and Ultralytics YOLO11s.

4.1 Assembling ***Follower***

We used the *OpenSpec GoPiGo* robot kit from Dexter Industries for mobility ([4]). We also implemented an *Arducam for Raspberry Pi Camera Module 3* to gather live footage from the robot. We followed the instructions on the GoPiGo website and plugged in the Arducam to the CSI camera port before attaching the GoPiGo red board. The camera was fixed in place for the duration of the testing.

4.2 Training the Models

To create our own YOLO models, we first gathered our own training data of the mobile target. The target in question is a remote-controlled red toy car of which we took 94 pictures of (2). Each picture was then labeled through Label Studio ([5]), an open source tool with which we made bounding boxes over the red car in each picture. This gives us a 'txt' file with labeling information associated with each picture.

Both the YOLOv8s and the YOLO11s models were trained with 60 epochs and 480 image resolution. The resolution is 480 instead of 640 in order for the model to run faster, which comes at the cost of lower accuracy.

4.3 Runtime architecture

Our system is divided into two components, the client and the server, both implemented in Python. The server is the robot *Follower* and the client is our laptop running the YOLO model. The GoPiGo has a built-in wifi that the laptop connects to in order to transfer data between the components. We use sockets to communicate wirelessly to the GoPiGo, sending the image from *Follower* to the laptop and to return the instructions. If the YOLO model does not detect any car with a confidence above 75%, then the robot is instructed to turn 30 degrees left. If the car is found, then the bounding box location determines the course of action. We measure the position of the box relative to the image grid, finding both the x and y positions of the box. From there we determine whether *Follower* should move right, left, or forward. Once the y position is sufficiently small, *Follower* determines it is too close to the car and fully stops, ending the program entirely.

4.4 Testing

To test the YOLO algorithms' ability to track a moving object, we remotely moved the car and made *Follower* chase it. The toy car follows a predetermined path designated by blue tape on the ground and stops once the path ends (3). *Follower* will attempt to follow the car until it gets about two feet away from the car. The car is bright red and the testing ground is in a gray and white room in order to create a good contrast. To keep the testing grounds small, a ring was formed using a corner of the room and brown and white boxes lined up in a wall. This allowed *Follower* to have as little trouble as possible when identifying the toy car.

Follower and the car are first positioned in the ring, with *Follower* positioned on the X mark facing south and the car facing north on the southernmost point of the blue path (3). *Follower* does not see the car until it turns about 60 degrees southwest (4), at which point it identifies the car and moves 15 inches forward. At this point we remotely move the car down the line, which makes *Follower* stop and turn 30 degrees right as it no longer finds its target. Once *Follower* locates the car again, *Follower* moves forward until it gets about two feet away from the car. This scenario is repeated and timed ten times for each YOLO algorithm. We then analyze the results by comparing the time it took *Follower* to reach the car for both models.

5 Results and Analysis

We first compared how accurate were the YOLOv8s and YOLO11s models and found little difference. The 60 epochs kept both programs highly accurate, with both YOLO models generally getting the same accuracy percentage when tested with the same images (5). We then ran the test ten times for each YOLO model and found that the YOLO11s model was on average 3 seconds slower than the YOLOv8s model (6).

While YOLO11s is considered more efficient and accurate than YOLOv8s, this does not necessarily seem to be the case with single-target pictures. It is also slightly slower than YOLOv8s when analyzing single-target pictures, thus resulting in the average YOLOv8s test run being quicker than YOLO11s.

Some weaknesses of our project were the use of an underpowered machine and an inconsistent object tracking test. The machine used was a 5-year old Lenovo Gen6 Thinkpad, which might be responsible for the YOLO11s model's underwhelming performance. When moving the remote-controlled car in the testing phase, the car would not always go exactly straight, resulting in some error when it came to its final location, thence the robot would require less or extra time depending on the car's final location.

6 Future Works

When we tested the YOLO algorithms, we noticed that the YOLO version didnt seem to affect the processing speed as much as the number of epochs used when creating the model. Fewer epochs seems to correlate to a faster picture analysis but reduced accuracy, thus a potential future research could entail finding how few epochs can be used while keeping as accurate a model as possible.

The goal of this project was to help future studies on making animal research robots autonomous, and we found which YOLO algorithm was better to that end. However, our setup requires wireless connection. Depending on the location, the animal research robots won't have access to service, so making them function while offline is necessary. This can be done by transferring our program to the Raspberry pi itself and let it run there, thus we encourage future research to test out various scenarios these robots may encounter using that setup, like obstacle avoidance.

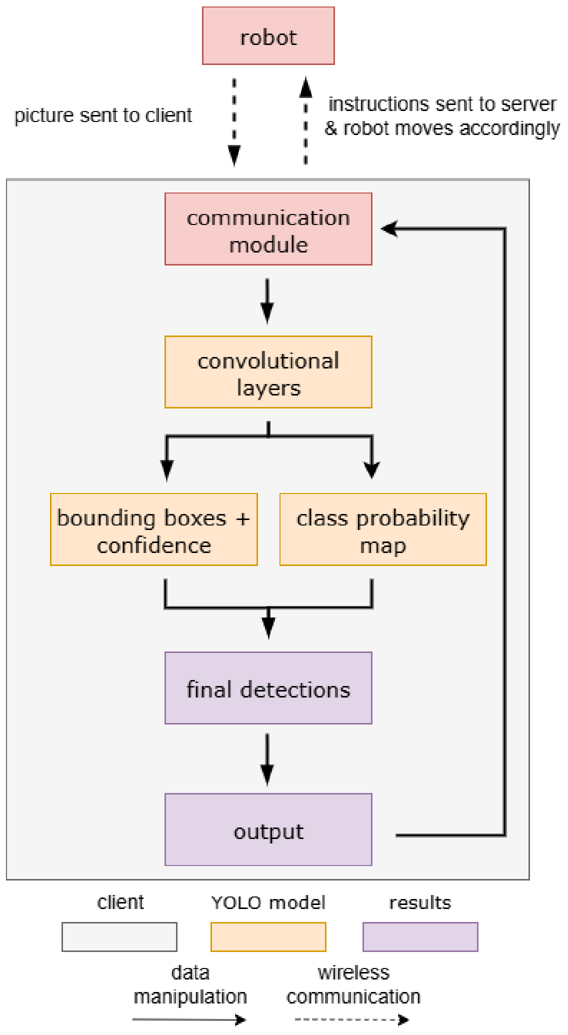

Figure 1: Data diagram of the entire process

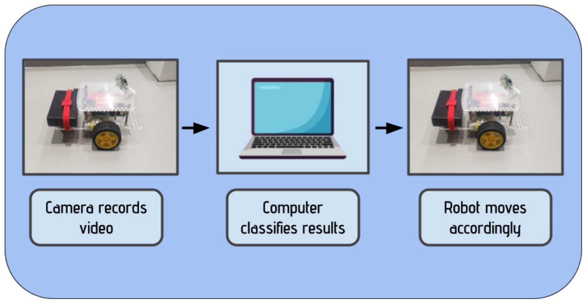

Figure 2: The general process for each round.

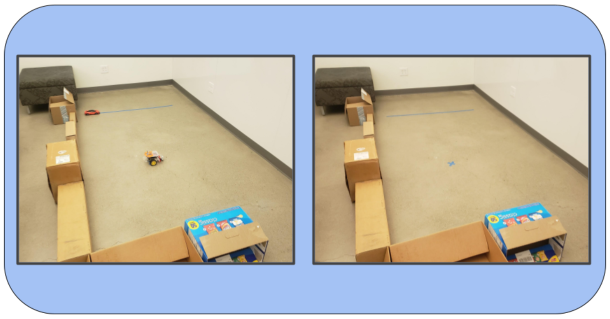

Figure 3: The testing setup.

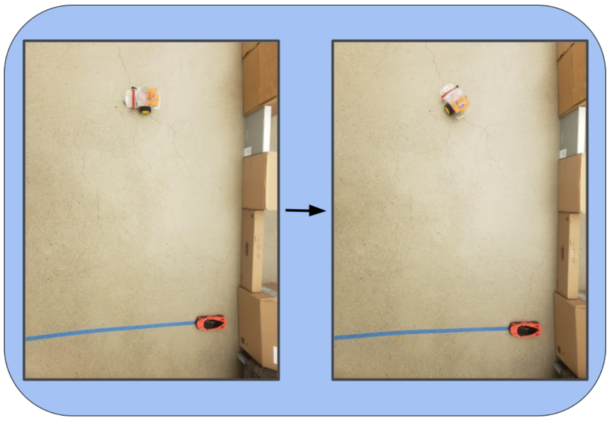

Figure 4: The robot adjusting to find the car.

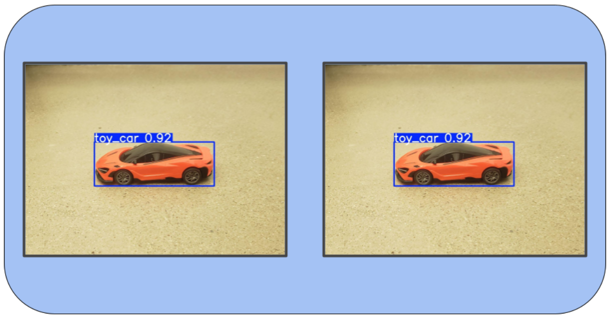

Figure 5: YOLOv8s (left) and YOLO11s (right) were tested on the smae picture. The resulting accuracy percentage was the same for both.

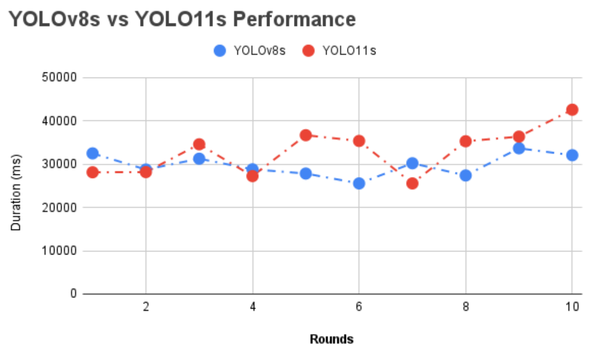

Figure 6: The duration in milliseconds the robot took to reach the target on each round based on the YOLO model.

References

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

[2] Reis, D., Kupec, J., Hong, J., & Daoudi, A. (2023). Real-time flying object detection with yolov8. *arXiv preprint arXiv:2305.09972*.

[3] Ultralytics yolov8 vs. yolo11: Architectural evolution and performance analysis. (2025). *https://docs.ultralytics.com/compare/yolov8-vs-yolo11/*.

[4] Modular robotics gopigo3 documentation. (2024). *https://gopigo3.readthedocs.io/en/master/*.

[5] Label studio. (2025). *https://labelstud.io/*.

[6] Gunnarsson, A. (2019). Real time object detection on a raspberry pi.

[7] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, 21–37.

[8] Varghese, R., & M., S. (2024). Yolov8: A novel object detection algorithm with enhanced performance and robustness. *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 1–6. https://doi.org/10.1109/ADICS58448.2024.10533619

[9] Li, M., Li, J., Cao, Y., & Chen, G. (2024). A dynamic visual slam system incorporating object tracking for uavs. *Drones*, *8*(6), 222.

[10] binti Rasidi, N. I., Al-Sanjary, O. I., Kashmola, M. Y., & Aik, K. L. T. (2022). Development on autonomous object tracker robot using raspberry pi. *2022 IEEE 10th Conference on Systems, Process & Control (ICSPC)*, 29–33.