Project Proposal

Stevie Bronsard

April 2025

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, like YOLO and SSD. This paper has for goal to create two different types of deep learning object tracking software which can be applied on a Raspberry Pi vehicular robot called *Follower* and compare their performances. The vehicle robot will be running Python on a Raspberry Pi 5 and will use a small fixed camera. The two object detection softwares will consist of Ultralytics YOLOv8, which is one of the most recent versions of YOLO, and SSD. *Follower* should be able to track any object as long as it is fully visible on the camera and has been processed in the model of whichever object detection software is running on *Follower*.

1 Introduction

Object tracking technology has developed exponentially the past few year, from basic goals like tracking wild animals to the creation of the first self-driving car. However, while object tracking technology functions is available to all through expensive products like drones, the ability to use this software without paying a large sum is really only accessible to those with knowledge on the subject. Thankfully, object detection software is often open source, so all that is really needed is to make programs which use object detection for land and air vehicles also available to all. This paper focuses on object detection for terrestrial vehicles and offers a guide to both assemble the robot in question and how its software functions.

2 Survey

2.1 YOLO

Of the object detection techniques that have been created using deep learning, one of the most impactful was You Only Look Once (YOLO) [1]. YOLO is an object detection program created in 2016. 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, thus identifying objects within each box. At the same time, YOLO 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. YOLO has evolved many times over the years, like YOLOX, YOLOv8, and YOLOv11 [2].

2.2 SSD

Another popular object detection software created in 2016 is the Single Shot MultiBox Detector (SSD) [3]. Like YOLO, SSD uses a single deep neural network to locate and classify objects in images. SSD runs a deep learning CNN with multiple layers to create feature maps, and defines points on the center of the objects detected by the feature maps, thus locating the objects on the picture. By using a fixed set of bounding boxes and analyzing the ratios in the feature maps, SSD can then predict object categories and bounding box adjustments. Thus, by using feature maps at different scales, SSD detects objects of different sizes and shapes.

2.3 Comparing YOLO and SSD

This paper is not the first to test the difference in performance between YOLO and SSD. The performances of SSDLite and YOLOv3-tiny were tested on a RAspberry Pi and compared by A. Gunnarsson in 2019 [4]. To be precise, the classification networks Gunnarsson used were **MobileNetV2** for SSDLite and **Tiny Darknet** for YOLOv3-tiny. By testing both models on a Raspberry Pi 3 B+, Gunnarsson found that SSDLite did outperform YOLOv3-tiny in both speed and accuracy. Unfortunately, the limited processing power of the Raspberry Pi 3 B+ hindered both models from achieving good performance, thus the results are not definite.

While the author states that SSD was empirically faster and generally more accurate, they do not delve into the architectural reasons why SSDLite with MobileNetV2 is more efficient than YOLOv3-tiny with Tiny Darknet. However, one potential reason mentioned for YOLO performing worse in accuracy might be related to the models' training sizes; the YOLO model was trained on 416x416 images, whereas the SSD model was trained on 300x300 images, and testing with smaller input sizes (96x96, 160x160, 224x224) might disproportionately affect models trained on larger images.

It should also be noted that the object used in the tests was a person in order to see how well the detector can perform for surveillance purposes. This means high speed is not essential, thus SSD with input sizes of 224x224 and 160x160 would both work well for that purpose. The robot in this paper will be identifying much smaller and faster objects, thus SSD will probably use different input sizes.

3 Design

The design of this project's robot will take place in three sections: assembling the robot parts, train our object detection models, and implement the object tracking software. The two object detection algorithms we will import will be Ultralytics YOLOv8 and SSD. We will rely on the PyTorch library to set up the necessary libraries.

3.1 Assembling *Follower*

I will be using the *OpenSpec GoPiGo* robot kit from the Dexter Industries for mobility. I will also implement an *Arducam for Raspberry Pi Camera Module 3* to gather live footage from the robot. The camera will either be installed onto a mini rotating platform on *Follower* itself or will be fixed in place. Follower will also have an object sensor in the front, so that the robot may identify when an obstacle is in front on it. However, if target object is found, the directive to go straightforward will override the object sensor until *Follower* is within two meters of the target.

The next step will be to set up the Raspberry Pi 5. This simply means install a fresh Raspberry Pi OS, specifically the 64-bit Raspberry Pi OS Bookworm version. The Raspberry Pi will need to have a General Purpose Input/Output (GPIO) script in order for *Follower* to know which pins on the circuit activate which part, like the direction the wheels will roll.

3.2 Training the Models

To create our own YOLO or SSD models, we will generate our own training model; we will take a large amount of pictures of whatever the object we want to track will be and use those images to train the model. We will then use Roboflow to create bounding boxes over the object we want to detect throughout multiple images. Roboflow is a computer vision tool which will allow the object detection program to identify the targeted objects. Using Roboflow, we will train our models directly with the images of identified objects. For this to work, we might need to use Google Colab, but, in theory, using one of Earlham's CS servers should work perfectly well. The result should be a functioning YOLO or SSD model which we can run on either images or videos at 640x480 resolution.

3.3 Implementing the Object Tracking Software

Both object detection models will result with a bounding box surrounding the target object, thus the box will have an id we can refer to in our python code. We will make the Raspberry Pi run the object detection model, and once the object is identified we can use the bounding box id to make *Follower* move towards the target. If the target moves behind a wall or large object, we will code *Follower* to stop, observe around, and resume the chase once the target pops back up.

4 Evaluation Plan

The project will consist of building *Follower* and then testing both object tracking software programs on it. To test the object tracking programs, we must first find an object for *Follower* to track. In our case, the models we will create will be aimed at identifying a red remote-controlled toy car. The car will be bright red and the testing ground will be in a gray and white room to create a good contrast. The testing ground will probably be in CST316, a room with many chairs and tables. This will allow *Follower* to have little trouble identifying the toy car but have a difficult time navigating the objects blocking the way.

The test will happen in the following fashion: the toy car will follow a pre-determined path, and *Follower* will attempt to follow it until it the toy car arrives at the destination and *Follower* catches up or once *Follower* fully loses sight of the toy car and can no longer find it. We will purposefully make the toy car navigate through the chairs to see how well will the robot do when confronted with obstacles. The test will be repeated ten times for each object tracking program, thus ten tests for YOLO and ten tests for SSD. We will then analyze the results by observing how much time did it take *Follower* to reach the toy car until the end and how closely did *Follower* follow the path the toy car took. The latter allows us to confirm how quickly will *Follower* track its target and how will it adapt to the target turning.

5 Contributions

5.1 Core Contributions

The results of this project should be an analysis of how do YOLOv8 and SSD compare on a Raspberry Pi 5 when used for the purposes of tracking a small object moving at a similar speed. This study is essentially building off the study by A. Gunnarsson previously mentioned [4], the main differences consisting of updated equipment and software, differing libraries, and the moving camera.

5.2 Auxiliary Contributions

This project will also let me see how much has YOLO improved over the years. Early versions of YOLO like YOLOv3-tiny tend to have more localization errors compared to other detection systems [**ctan**], so one of the goals is to see how well YOLOv8 does when used on a RaspberryPi 5 [5].

6 Risks

I believe I will encounter a lot of difficulties while designing this project. The main ones will be figuring out how to connect the bounding box to *Follower*'s instructions, ensuring a way for *Follower* to deal with obstacles and vision obstructions, and identifying how close must the toy car be to *Follower* for the latter to start following its target. We will figure out how to connect the bounding box to *Follower*'s instructions through further investigations of YOLO and SSD.

The way for *Follower* to deal with obstacles is currently to simply wait in place and look around. However, that relies on the target to continue to move or for the obstruction to not be large enough that it fully blocks the view of the target's travel. Previous research has found ways to to deal with this conundrum, but they require the installation of programs like SLAM that might impede the object tracking software [6]. A possible plan is to make *Follower* identify a path around the object, but that solution must be tested first. If obstacles become too much of a problem, they will simply be removed from the testing ground.

When it comes to the uncertain proximity of the target to *Follower*, that can be tested after the model is created. The goal is for the toy car to still be identifiable by *Follower* at least 8 to 10 meters away. If necessary, the models can be re-run to account for the distance and we can program *Follower* to stay farther away from the target to not get confused.

Another big risk is the Raspberry Pi being unable to run the software due to insufficient processing power. If that issue comes up, we can either find ways to increase the processing power or we can rely on a computer's system to run the object detection for *Follower* and transmit the results to it through a network connection.

7 Special Resources

	Resources
1	Raspberry Pi 5
2	Arducam for Raspberry Pi Camera Module 3
3	Open spec GoPiGo robot kit
4	Small Grey Room in CST

8 Timeline

This project will be completed over a 15 week semester. The primary deliverables will be a technical report, a poster presentation, and a project demonstration video. In the first 6 weeks of the semester, I plan to assemble *Follower* with the object tracking software programs implemented. The next 9 weeks would then consist of analyzing the data, writing the report, and creating the presentation.

Week	**Deliverables**
1	
2	Version 0 of the Technical Report
3	Version 0 of the Data Architecture Diagram
4	Version 0 of the Graphical Abstract
5	Version 1 of the Data Architecture Diagram and Graphical Abstract
6	Version 1 of the Technical Report
7	Version 2 of the Data Architecture Diagram and Graphical Abstract. Version 0 of the Poster
8	Version 0 of the Project Demonstration Video
9	Version 2 of the Technical Report
10	Version 1 of the Poster
11	Version 3 of the Data Architecture Diagram and Graphical Abstract
12	Version 1 of the Project Demonstration Video
13	Version 3 of the Technical Report
14	Version 2 of the Poster and Project Demonstration Video
15	Final Versions of All Deliverables

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] 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.

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

[5] 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

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

[7] 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.