Vision-Based Algorithms for Obstacle Detection and Avoidance in Autonomous Driving

Ryan Diaz

August 2024

Contents

1 Overview

1.1 Introduction

This document serves as a comprehensive overview of the work done by Ryan Diaz on the Minicity Autonomous Vehicle project (also worked on by Ahron Bloom and Caitlyn Johnson) that took place in Summer 2024 as part of the WashU CSE REU program, advised by Dr. Eugene Vorobeychik. It will provide some details and motivations for the methods, links to repositories containing relevant code, and suggested directions for moving forward.

1.2 Previous Work

A large part of the project was built off of the work done by Owen Ma during the previous year. He has provided his own documentation (linked in the "Resources" section of the document), and it is highly recommended to read through it, especially if one plans to work with the F1Tenth car in the real world Minicity. Owen's work itself builds upon earlier efforts made by George Gao, David Brodsky, and Angelo Benoit, and documentation relevant to those projects can also be found in the "Resources" section.

1.3 Project Outline

As the title may suggest, this project consists of two major parts: obstacle detection and obstacle avoidance. Here, obstacle detection refers to the classification and localization of relevant objects (such as pedestrians and other vehicles) in the environment around the ego vehicle on a windshield point-of-view image. The task of obstacle avoidance then refers to the ego vehicle performing actions to maneuver around static obstacles in its path while adhering to the original desired route as much as possible. Ultimately, given start

and goal locations on the map, the vehicle should be able to a) construct a high-level plan to navigate the Minicity and reach the destination, b) avoid collisions with any obstacles that might be present in the vehicle's path, and c) do all of this while adhering to basic traffic laws such as stopping at stop signs and stop lights, following road markings, and maintaining a reasonable speed.

Project development occurs in two major places: the CARLA simulation environment [1] and the real-world WashU Minicity environment. Although most of the development for this project took place in CARLA, the hope is that these models can be transferred to the real-world setup with minimal fine-tuning. Discussion on this can be found in the "Next Steps" section of this document.

2 Object Detection

2.1 Overview

We focus on two major parts of the obstacle detection task: collecting an annotated dataset and training a model on that dataset. For both the simulation and real-world environments, we use the YOLOv8 model from Ultralytics as it is easy to set up and performs very well for our task once it is fine-tuned. This model can be substituted out for other object detection models if necessary (in particular, if the new Ultralytics AGPL-3.0 license becomes too restrictive). The dataset collection process, on the other hand, is specific to the environment (either in simulation or in the real world), and will be detailed in the following sections.

As of right now, we focus on detecting four classes of objects: **vehicles**, **pedestrians**, **stop lights**, and **traffic signs** (with a specific focus on stop signs). These classes can be extended or broken up to categorize new sets of objects, but this will require recollecting and/or re-annotating the training set. The next two sections describe the process of collecting and annotating data, and running a trained model in both the CARLA simulation and on the F1Tenth vehicle.

2.2 CARLA Object Detection Model

We provide a tool to automatically collect and annotate large datasets of images collected from the windshield point-of-view of our vehicle in the CARLA simulation. Follow the instructions in the linked repository's `README` file to collect a dataset, train a YOLOv8 model on the data, and evaluate the model's performance. Currently, the repository only supports automatic annotation of the four classes mentioned above, but has instructions for adding additional classes if one so chooses.

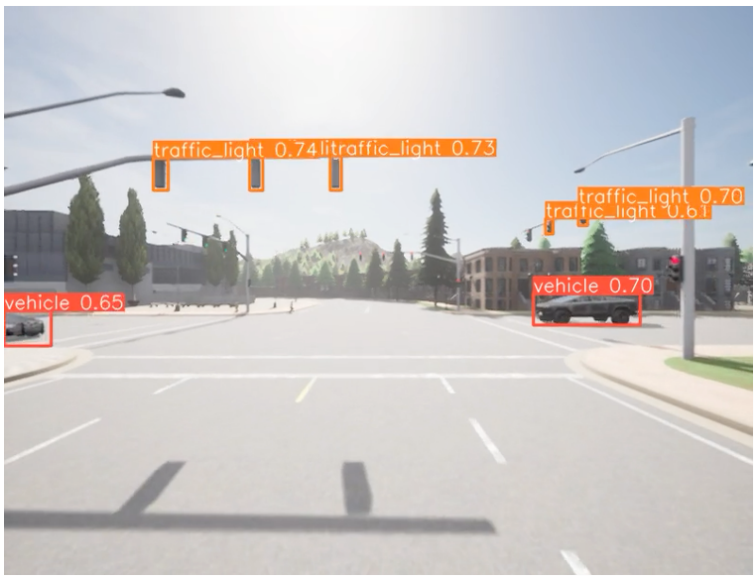


Figure 1: Object detection results in CARLA.

2.3 Minicity Object Detection Model

On the F1Tenth car, we have built a ROS node dedicated to collecting data and running inference in real time using a trained model. To run this node, make sure you first run the `usb_cam` and `image_transport` nodes (steps 1 and 2 of the image processing pipeline in Owen Ma's documentation). Then, run the following command in the terminal:

```
ros2 run yolo_image_processor yolo_image_processor_node
```

You may need to modify the code (located in the same folder as the core image processing modules on the vehicle) depending on whether you want to collect image data for training a model or perform inference with a trained model. If you are performing inference, we provide a trained detection model (pre-trained on CARLA simulation data and fine-tuned on real-world data) that can be used. If you are collecting a dataset, you will need to manually annotate your collected dataset using any online annotation tool.

Figure 2: Object detection results in the real-world Minicity.

3 Obstacle Avoidance

3.1 Overview

In this project, the task of obstacle avoidance is formulated as follows: an agent (known as the "ego vehicle") starts at a certain location in the map (either a CARLA map or the Minicity) and is tasked with traveling to a separate destination point at another location on the map. Given a roadmap of the environment, the ego vehicle is able to construct a high-level plan of what route to follow in order to reach the destination. The vehicle may then travel along the pre-planned route. However, it must follow certain rules while traveling:

1. The vehicle must stay within the lane it is traveling on (except when changing lanes or turning in an intersection) and maintain a reasonable speed.

2. The vehicle must obey basic traffic laws, such as stopping at stop lights and stop signs.

3. The vehicle may encounter static (unmoving) obstacles in its path, such as vehicles or pedestrians. It must be able to avoid these obstacles while adhering to the original route as much as possible.

For each step in the environment, the agent should take the high-level plan and windshield RGB images as input and should output steer, throttle, and brake commands for its action. We discuss our method for building such an agent in the following sections. It is worth noting that our method is largely based off of the work done by Zhang et al. [2] (which will be referred to as "Roach") that trained a visuomotor agent to drive in realistically busy road conditions. We highly recommend one to read this paper as it provides some additional context for our methods. The code for our work in this section can be found in this repository.

3.2 Knowledge Distillation

To train an agent that can learn to avoid obstacles using image inputs, we elect to use a framework of "knowledge distillation", building off of the work done in the Roach paper. Knowledge distillation consists of two models: a more "knowledgeable" expert model and a more "realistic" student model. The expert model is more powerful and more capable of learning the desired behavior, either by having a larger and more complex architecture than the student model, or by taking more semantically useful information, such as ground truth environment states, as input.

However, the complexity of the expert model or the inability to source the required data on the fly may make it difficult to deploy the expert model in a realistic environment. The student model then represents the realistic model that can be deployed in the environment of interest. The goal of knowledge distillation is to find a way to "transfer" the knowledge of the expert model to the student model, such that the student model is able to achieve performance comparable to that of the expert without having access to the information and/or learning capabilities of the expert.

3.3 Network Architecture

We discuss the application of the knowledge distillation framework to the obstacle avoidance task in CARLA. Figure 3 depicts a summary of the overall framework.

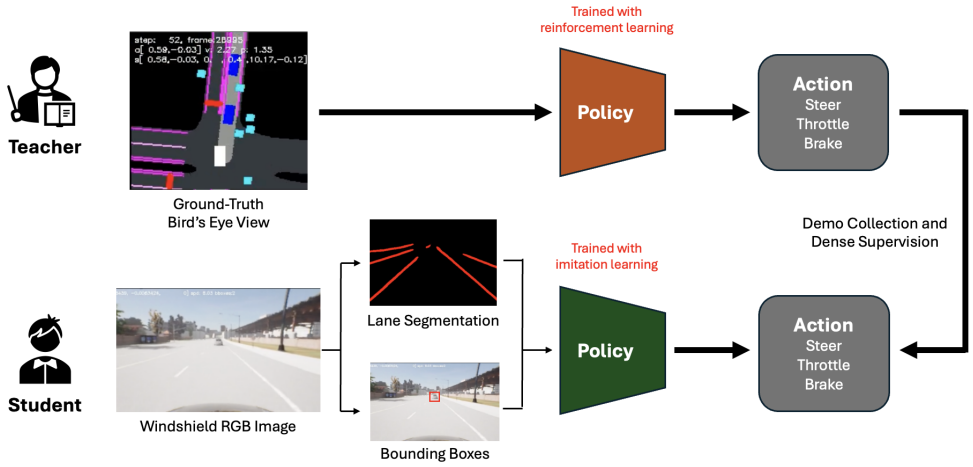
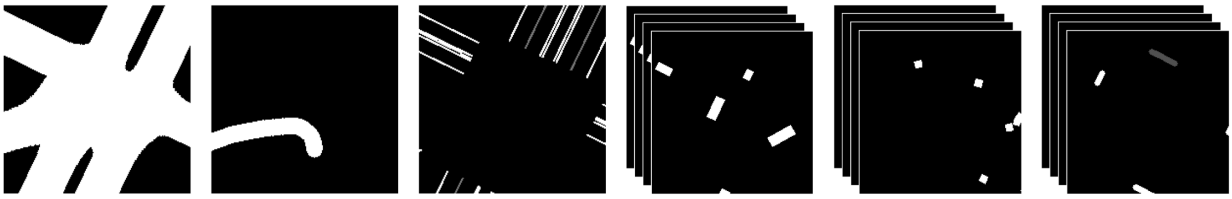


Figure 3: Overview of the knowledge distillation framework applied to the obstacle avoidance task.

3.3.1 Privileged Expert Agent

Our expert agent takes as input a simplified bird's eye view (BEV) rendering of the environment around the ego vehicle, which encodes ground-truth locations of important features such as roads, lane markings, stop light states, and a rendering of the desired route from the high-level plan. It also takes in other vehicle measurements such as steer angle and vehicle speed. The expert then has two outputs: a value estimation of the current state (used during reinforcement learning training) and an action consisting of steer, throttle, and brake commands for driving the vehicle.

Since the BEV is obtained through the CARLA simulation's API, it represents privileged information that can only be accessed by the expert. The expert can then more easily learn the desired behavior, since it only receives the information most relevant for the task (unlike image inputs which may include noise and other irrelevant aspects of the environment).



(a) Drivable areas (b) Desired route (c) Lane boundaries (d) Vehicles (e) Pedestrians (f) Lights and stops

Figure 4: The BEV representation used by the expert agent (figure taken from [2]).

3.3.2 Visuomotor (Student) Agent

Our student agent takes as input an RGB image taken from the windshield point-of-view of the ego vehicle and the same low-level vehicle measurements that the expert agent receives. We preprocess the image input before feeding it into the action network: first, we use a LaneNet [3] model trained by David Brodsky and Angelo Benoit and fine-tuned by Owen Ma to extract segmentations of lane markings on the road. Then, we use our trained object detection model described in the "Object Detection" section of this document to gather bounding box coordinates of objects in the four classes detected by the model. We feed the lane segmentations and bounding box coordinates into separate encoding networks and concatenate the output representations to use as input for the policy network that will output the action command for the vehicle.

Following [2], the policy network consists of a branching architecture conditioned on discrete driving commands (such as `LaneFollow`, `ChangeLaneLeft`, and `ChangeLaneRight`). More specifically, a branch containing its own action predictor is constructed for each possible high-level command (though all of the action predictor models have the same architecture). After the input image is processed, the resulting embedding is fed through only the branch corresponding to the current command from the high-level plan for that step to generate an action. We have modified the conditional branching network architecture to add two more commands (`AvoidObstacles` and `ReturnToLane`) that can override the command from the high-level plan and facili-

tate the desired obstacle avoidance behavior:

- The agent receives the `AvoidObstacles` command when it has detected that it is sufficiently close to an obstacle (vehicle or pedestrian) in its path.
- The agent receives the `ReturnToRoad` command after it has steered out of the way of the obstacle, and continues to receive the command until it has returned to its original lane.

In this way, we can isolate the obstacle avoidance and recovery behaviors during the learning process and avoid disturbing the learning of the other branches whose desired behaviors may not be fully compatible with the actions necessary for obstacle avoidance.

3.4 Learning Pipeline

To facilitate knowledge distillation between the privileged expert agent and the visuomotor student agent, our training pipeline consists of three phases: training the expert using reinforcement learning, collecting demonstrations of obstacle avoidance from the expert agent, and using those demonstrations to train the student agent using imitation learning. We describe each of these steps below.

3.4.1 RL Expert Training

We train our expert agent with the proximal policy optimization (PPO) reinforcement learning algorithm, much in line with the original RL expert trained in the Roach paper. The agent receives a reward per step that reflects the conditions of the environment and the state of the agent with respect to the agent's ability to complete the task.

The reward is formulated as

$$r = r_{speed} + r_{angle} + r_{position} + r_{terminal}$$

and guides the agent towards specific behaviors. r_{speed} is based on the vehicle's current speed and guides it as close as possible to a desired speed depending on the state (which may decrease in some conditions such as stopping at stop lights or stop signs). r_{angle} and $r_{position}$ encourage the vehicle to follow the route outlined by the high-level plan (represented as a series of waypoints): r_{angle} makes sure the vehicle is pointed in the same direction as the route and $r_{position}$ ensures that the vehicle does not stray from the route in terms of lateral distance. $r_{terminal}$ is based on certain terminal conditions that may cause a vehicle to "fail" a run (such as running a red light or colliding with another vehicle), and provides a negative reward signal if any of these terminal events occur to discourage the agent from performing unsafe driving. For a more detailed overview of the reward calculation, we refer to Toromanoff et al. [4] which the Roach paper's reward signal is based off of.

Unlike the Roach paper, we train our expert agent in an environment with no other moving vehicles so that the agent can more easily learn a route-following policy without the complications of interacting with other drivers (as of now, the other vehicles in the Minicity environment will remain static). Our trained expert agent is able to follow a desired route (including appropriate lane-changes when turning at intersections), maintain a reasonable speed while lane following, and stop at stop signs and stop lights, which makes up the first two objectives for the obstacle avoidance task (see the "Overview" section).

3.4.2 Expert Data Collection

To transfer the route-following capabilities of our expert agent and also introduce the obstacle avoidance behavior, we collect demonstrations of the expert agent exhibiting obstacle avoidance behavior to use them for imitation learning training. During a demonstration, we collect a BEV representation of the environment to be used as input to the

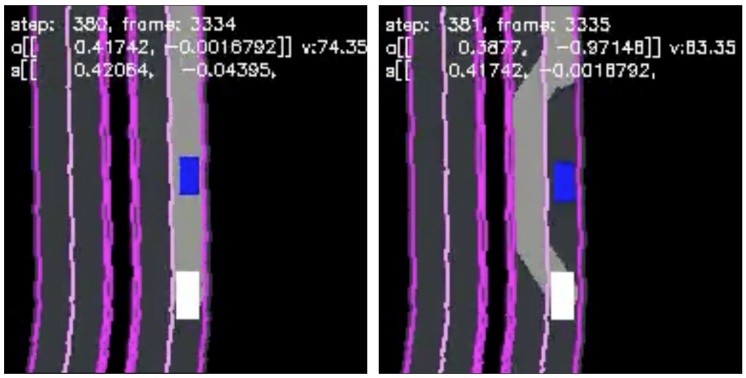


Figure 5: An example of the online path replanning performed during demonstration collection for obstacle avoidance.

expert, and record windshield RGB image data, the current high-level command, and the accompanying action output by the expert per step to supervise the training of the student agent.

Although the expert agent was not specifically trained to avoid static obstacles, we can utilize the route-following capabilities of the agent to guide it to avoid obstacles with online route replanning. If the agent encounters a static obstacle in its path (the same conditions for the `AvoidObstacles` high-level command), we generate new waypoints in a hard-coded manner on a lane adjacent to the vehicle's current lane that does not contain an obstacle. These new waypoints replace the next few waypoints in the existing route that the vehicle is about to travel to, updating the BEV rendering of the desired route and causing the vehicle to (slightly) change course in order to maneuver around the obstacle (see Figure 5). We capture this special behavior during demonstration collection by recording `AvoidObstacles` and `ReturnToRoad` commands when appropriate to allow the student model to specifically learn these behaviors.

3.4.3 IL Agent Training

With the demonstrations we have collected, we can use the image and action pairs generated by the expert agent as supervision signals when training the student model via imitation learning. More concretely, given a specific observation input, we aim for the expert and student models to output the same actions (or action distributions). In addition to the standard action supervision, the expert agent also provides other dense supervision signals (such as the value estimation, speed prediction, and latent feature matching) to help guide training. For more details on these supervision signals, we again refer to the Roach paper.

After this initial imitation learning stage (which can be seen as behavior cloning), there is also the option of introducing additional data collection and training stages to the learning schedule, turning the imitation learning algorithm something more akin to DAgger [5]. We collect an additional set of demonstrations using the trained *student* model (rather than the expert), with the expert annotating the observations with the correct actions. In this way, the student agent can learn to recover from mistakes and construct a more robust policy.

4 Next Steps

4.1 Overview

In this section, we discuss areas of improvement on our method in both object detection and obstacle avoidance, and also suggest some directions for future work (mainly in deploying the trained simulation models to the real-world Minicity environment).

4.2 Improvements to Object Detection

Our detector model is able to localize four broad classes of relevant objects in the environment, but we feel as if there is some room for refining these categories. Expanding these classes (for example, separating out specific types of traffic signs and distinguishing between red, yellow, and green lights as individual classes) can give richer semantic information to the student model and allow it to make more informed decisions. Additionally, some classes of vehicles such as bicycles and motorcycles were not included in our simulation dataset, and we believe it will be important to include these vehicles in our detection framework at some point down the road.

4.3 Improvements to Obstacle Avoidance

We have identified two specific aspects of our method and training process for the obstacle avoidance task that we believe can be improved:

- **Training Environment.** Currently, data collection and student model evaluation is being conducted on one short route, with a single vehicle placed in a randomized location along the route to act as the obstacle. Expanding the training environment to include multiple different routes and obstacle configurations can help the

student model learn a more robust and versatile policy. Additionally, we recommend collecting a larger demonstration dataset for training (our dataset consisted of 20 demonstrations on the same route), as the dataset is biased much more towards states corresponding to ordinary driving commands such as `LaneFollow` rather than `AvoidObstacles` or `ReturnToRoad`, which biases the behaviors that the student model learns in turn.

- **Obstacle Avoidance Commands.** Currently, the agent determines when to execute the behaviors for avoiding obstacles by considering its distance to vehicles in its path. This process uses ground-truth data provided by CARLA concerning the positions of neighboring vehicles and pedestrians relative to the ego vehicle, which poses a problem when deploying to the real world as this information cannot be so easily obtained. Thus, we suggest turning the determination of when to enter an obstacle avoidance state into a classification problem. Specifically, one can add a classification head to the student model before the action prediction branches that classifies whether or not the vehicle should use the `AvoidObstacles` or `ReturnToRoad` branches for the action output given the latent representation as input. The classification head would be trained jointly with the rest of the student model during the imitation learning process, with the recorded high-level command serving as the supervision signal.

4.4 Real World Deployment

We think that the most obvious next step for this project is to construct a visuomotor agent that can follow routes and avoid obstacles in the real world Minicity, building off of the work done in the CARLA simulation.

Although the reinforcement learning and data collection processes are fairly straight-

forward in simulation, as the agent needs only to interact with a virtual environment, it is significantly more difficult to reliably carry out in a real-world environment. With this in mind, we suggest approaching the real-world deployment problem as a sim-to-real problem and directly deploying the models trained in simulation to the real world environment, rather than training new agents from scratch (an expert agent cannot be trained in a realistic environment anyways).

Achieving sim-to-real transfer will most likely involve fine-tuning the image preprocessing modules in the student model (both the lane segmentation and object detection models) on real-world data, as we cannot use the simulation's expert agent to collect demonstrations in the real-world environment. This process has already been done with the object detection model, though some additional work in re-annotation and retraining might be necessary if one wants to update the detected classes. A real-world LaneNet model had already been trained by David Brodsky and Angelo Benoit and refined by Owen Ma, but these models assume that no vehicles or other obstacles are blocking the path of the ego vehicle, which is not the case for the obstacle avoidance task. Due to the effect that the presence of other vehicles has on lane segmentations, it may be necessary to retrain the LaneNet model with images that depict other vehicles on the road.

5 Resources and Acknowledgements

5.1 Repositories

- **Object Detection Repository** [link]: This repository provides tools for collecting annotated data to use in training an object detection model.
- **Obstacle Avoidance Repository** [link]: This repository contains the code for collecting demonstration data and the training and evaluation of the expert and student models in the obstacle avoidance task.

5.2 Past Documentation

- **George Gao's Documentation** [link]: This documentation provides the foundations for the Minicity autonomous driving project.
- **David Brodsky and Angelo Benoit's Documentation** [link]: This documentation details improvements and additional work done on top of the previous iteration of the project.
- **Owen Ma and Ryan Chen's Documentation** [link (may need to download)]: This is the most up-to-date documentation regarding the state of the F1Tenth car. Owen Ma's work in the CARLA simulation can also be found in the repository that the documentation is located in.

5.3 Acknowledgements

I would like to acknowledge Caitlyn Johnson and Ahron Bloom, who had also been working on other aspects of the CARLA and Minicity projects. Caitlyn worked on a Minicity replica map in CARLA and trained an agent to perform high-level planning and decision

making using lane segmentation inputs, while Ahron worked on training an agent using LiDAR data, which may also prove helpful in real-world deployment. I highly recommend reading their documentations and other resources. Additionally, I would like to thank Dr. Vorobeychik and the members of the WashU CERL Lab for their valuable help and guidance on this project over the summer.

References

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning* (S. Levine, V. Vanhoucke, and K. Goldberg, eds.), vol. 78 of *Proceedings of Machine Learning Research*, pp. 1–16, PMLR, 13–15 Nov 2017.

[2] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-end urban driving by imitating a reinforcement learning coach," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 15222–15232, 2021.

[3] Z. Wang, W. Ren, and Q. Qiu, "Lanenet: Real-time lane detection networks for autonomous driving," *ArXiv*, vol. abs/1807.01726, 2018.

[4] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7153–7162, 2020.

[5] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, JMLR Workshop and Conference Proceedings, 2011.