CIENE 9101

Research Report

Nihar Mauskar, Kyle Coelho

Introduction

Driving a self-driving car in the real world is an understandably daunting task. To that end, the original outset of this research project was to find and define a simulator for modelling autonomous driving. We dove into CARLA to identify and see if it was a possible simulator that would facilitate this. After discovering it was, we went and developed a rudimentary autonomous driving algorithm to discover the limitations of CARLA and found out what workarounds existed. We discovered how to extract usable LiDAR and RGB camera data as well as how to utilize that data while running CARLA to have multiple neural networks consume that data in real time to make driving decisions for the simulated vehicle.

Background Research

In order to discover the proper simulator for this task, we examined both SUMO and CARLA to see which traffic simulator is the better one for testing realistic autonomous driving. Both were open source systems which we greatly appreciated as it very quickly allowed us to verify whether certain issues were implementation dependent or inherited from the library used. Additionally, both were easily extensible using python which allowed for easy integration with existing machine learning libraries as most of those are implemented in python. We then examined the individual features of each system to see which would make for better modelling. Here there was a distinct split at SUMO was created for modelling large-scale traffic networks and as such did not model the sensors necessary for autonomous driving. CARLA conversely was made for simulating a smaller-scale environment but did have capability for modelling all sensors as well as a way to obtain ground-truth labels for the objects in the environment. Here we also looked to see what projects utilized each system and found that CARLA was designed for exactly this type of simulation (Dosovitskiy et al., 2017). From there we started looking into the various types of data we could stream from CARLA and found that it allowed for real-time streaming of a

variety of sensors but most importantly for our use cases, LiDAR point-cloud data and RGB camera images.

Implementation

To test the quality of these sensors we implemented a recording system to save the data that was being streamed and saved the recordings of them, these are viewable in the Google Slides that we have also shared and in the figure below.

Figure 1: Lidar Sensor on camera with Mayavi visualization

For the LiDAR data we examined various image segmentation networks utilizing three-dimensional point-cloud data and decided to use SqueezeSegV3 (Xu, Chenfeng et al., 2020) because it provided the highest accuracy for segmentation at the time and was one of the smaller networks allowing for a high throughput of data in the streaming setting. We initially tried live streaming data directly to the network as it was coming in but due to hardware constraints we found that such a method was infeasible. Our second attempt involved utilizing our recording system to save point-clouds and then utilize SqueezeSegv3 for image segmentation after the fact. Here we had more success in that we were able to save these point-clouds and feed them into SqueezeSegv3 in the format it required however we were once again beset by hardware constraints as each point-cloud contained upwards of 5000 points and running inference on them was very computationally intensive. Looking deeper into the paper for SqueezeSegv3 we found it was intended to run on a

distributed network of GPUs to ease the burden of computation instead of the single GPU we were attempting to run it on. Despite this not resulting in real-time inference on our machines we were able to confirm that the data recovered from CARLA, with a powerful enough system, would be able to create usable data for autonomous driving via LiDAR segmentation.

The next sensor we tested was the RGB Camera sensor. Here we utilized our recording system again to save images from the camera and then run object detection on them using Yolov5 (Jocher et al., 2021). We picked Yolov5 specifically because it is known to be a high accuracy object detection neural network with very low computational overhead. Here we were able to successfully run the network and identify the various objects within the frame. Some examples are indicated in the figures below:

Figure 2: YoloV5 outputs from RGB camera mounted on CAR

Images and videos for this can also be seen in the Google Slides provided. Having now proven that the objects in RGB images could be identified we attempted to do so in real time. Again we were met with success as we were able to implement real time object detection with bounding boxes for a vehicle driving around in the CARLA simulation. From there we decided to take things one step further and attempt to actually use this data to drive a vehicle autonomously. Initially, we did so via very rudimentary and inaccurate conversions however after seeing that the results were not what we wanted we decided to do more research into various methods of steering and throttle control.

First we researched how automated steering and driving systems were used in the real world and came across various methods including Stanley, PID, and others however these methods all required us to have a 3D ground truth that could give a location for the vehicle to move to. As we were using 2 dimensional data this proved to be a halting point because there was no way to transform the 2 dimensional data we were receiving in the form of camera images to the 3 dimensional data required by these control systems without a prior ground truth conversion which we cannot guarantee exists in the real world. For future work, this may be possible using LiDAR but that is purely speculative as we are unsure how accurate the conversion process would be or how to go about converting from a point-cloud space to a target location. After seeing the problems with such control processes we went to see what prior work existed for generating steering and throttle values from images and came across PilotNet (Bojarski et al., 2020) which was a neural network architecture developed by NVIDIA for exactly this type of problem. Examining their results we found that they were able to run their network with minimal computational overhead and generate useful values for steering and throttle given a region of interest in the image which is exactly what we were looking for.

With this knowledge, we saw that we could combine the two networks together, using Yolov5 to denote an area of interest, namely a vehicle ahead that our vehicle was attempting to follow, and using PilotNet to generate the values for steering and acceleration that we could pass onto CARLA. This proved to be fairly successful as while the neural networks were receiving data at a rapid rate, we could accurately control the vehicle to follow the one ahead without crashing into it or running off course. Unfortunately here we ran into a problem that is inherent to CARLA itself. We discovered that after the first ~20-30 frames of data, the rate of streaming would drop to approximately 1 frame per second. For reference this means that instead of being able to process ~15 frames of data per second, the algorithm was only able to receive and process 1 which naturally led to a steep drop-off in the quality of autonomous driving. Additionally, because the steering and throttle values were only updated upon sensor input the rate of change of these values dropped drastically leading to the vehicle swerving off track or hitting the car ahead because it did not receive any data allowing it to realize the car was closer or farther than expected. Examples of collisions are shown in the figure below.

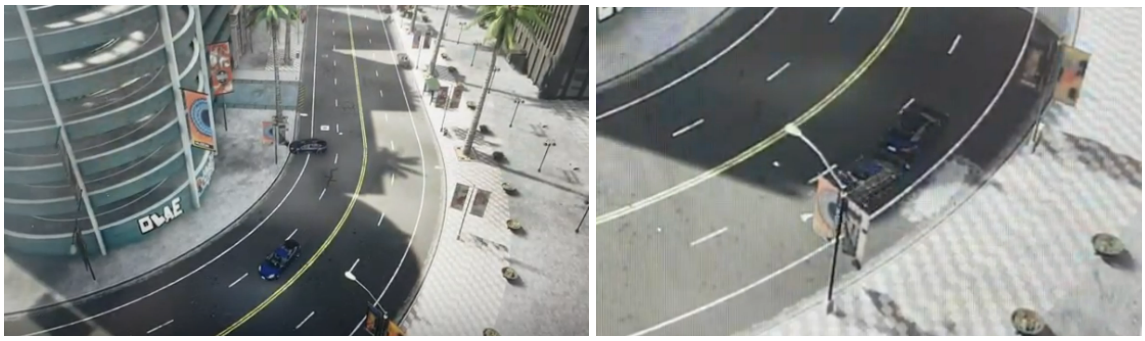

Figure 3: Examples of collisions due to poor frame rate. Collision into scenery (left) and into car ahead (right)

This is a problem that we did resolve however it took a massive restructuring of our code and converting from an asynchronous callback based infrastructure to a synchronous one. After this restructuring we found that our algorithm required a few tweaks to ensure it continued following the same vehicle and not lose track of it. Adjusting our outputs manually we found a set of transformations that allowed us to prevent such occurrences and the final results of these can be found in the slides. Ultimately the algorithm we have is not “road-ready” but is good enough to prove that CARLA is ready and capable of simulating something as complex as autonomous driving. Despite that, we found that the new algorithm could only run on sufficiently computationally capable computers so it is not the case that any device would be able to run these simulations.

We also experimented with earlier versions of YOLO such as YOLOv3 because many applications might already be running this popular version of the NN. Additionally, we were curious to see how previous versions of this network would fare in terms of accurate detections. It does not perform as well as YOLOv5 because it misclassifies other objects as cars much more frequently. As a result, it tends to steer more extremely in either direction depending on the confidence of the wrong prediction that it made. If the true lead vehicle stays within frame continuously, we have confidence it can work well in the long run but that would only be in very controlled environments like these. Videos for these observations are added to the Google slides as well.

Conclusion

Overall, we found that though CARLA is a good simulator for driving in the real world, it still has various bugs that need fixing. We have opened an issue to the CARLA team regarding our main problem with their platform (Issue) however we haven't received a response from the official CARLA team yet. CARLA is, however, a great method to gather data to train various neural networks for autonomous vehicles as it does provide a variety of differing types of sensor data and can be used to very easily generate this data. Additionally, after adjusting various settings within the simulator we were able to run a simulation of long-term autonomous driving so it is possible to do so with the current incarnation of CARLA, though highly inconvenient to do so. The simulation also makes various assumptions about the processing rate of the world, i.e. the synchronicity, which is unrealistic but necessary for such a simulation to function. This project could be extended in various ways, from training neural networks from the ground up on data exclusively generated by CARLA to working with the CARLA platform team to fix the bugs that come with asynchronous processing.

Bibliography

Bojarski, Mariusz, et al. “The NVIDIA PilotNet Experiments.” *ArXiv:2010.08776 [Cs]*, Oct. 2020. *arXiv.org*, http://arxiv.org/abs/2010.08776.

Dosovitskiy, Alexey, et al. “CARLA: An Open Urban Driving Simulator.” *ArXiv:1711.03938 [Cs]*, Nov. 2017. *arXiv.org*, http://arxiv.org/abs/1711.03938.

Jocher, Glenn, et al. *Ultralytics/Yolov5: V6.0 - YOLOv5n “Nano” Models, Roboflow Integration, TensorFlow Export, OpenCV DNN Support*. v6.0, Zenodo, 2021. *DOI.org (Datacite)*, https://doi.org/10.5281/ZENODO.3908559.

Xu, Chenfeng, et al. “SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation.” *ArXiv:2004.01803 [Cs]*, Apr. 2021. *arXiv.org*, http://arxiv.org/abs/2004.01803.