Simulation-Based Autonomous Driving in Crowded City

Pranav Athrapullikal, TUM

***Abstract*—A self-driving car is one that can sense its environment and drive without human input. The purpose of this project is to build an AI agent for an autonomous driving simulator. The car must drive around in the simulated city while following traffic rules and avoiding collisions with other cars and trucks.**

***Index Terms*—AI, Autonomous Driving, Python, Object Detection, Lane Detection**

I. Introduction

THE purpose of this project is to build an AI agent for an autonomous driving simulator. It must drive around in the simulation while following traffic rules and avoiding collisions with other cars and trucks.

Self driving cars are transforming travel as we know it. It is making driving safer and more efficient. It required the deep learning revolution to happen and reap its benefits.

These cars can reduce traffic congestion, cut transportation costs, improve walkability and reduce Co2 emissions.

These days companies like Tesla deliver all the software requirements continuously to your car, helping make you safer. It produces a lot of value for drivers and society. That said, there is still a lot of work to be done.

Today, most cars only include basic autonomous features, but major advancements in capabilities are on the horizon. Vehicles will ultimately achieve SAE Level 5.

Full Automation is still under active research. At level 5, the car is able to perform all driving tasks under all possible conditions in the environment. It does not require any human attention. A level 5 car does not require geofencing. This refers to the restrictions on the geographical region where the vehicle is safe to function.

By exploring this topic, we aim to contribute to the ongoing discourse surrounding self driving cars and build a model for the given simulator.

II. Components

The driving decisions are computed either in a modular pipeline, or in an end-to-end learning fashion, where sensory information is directly mapped to control outputs.

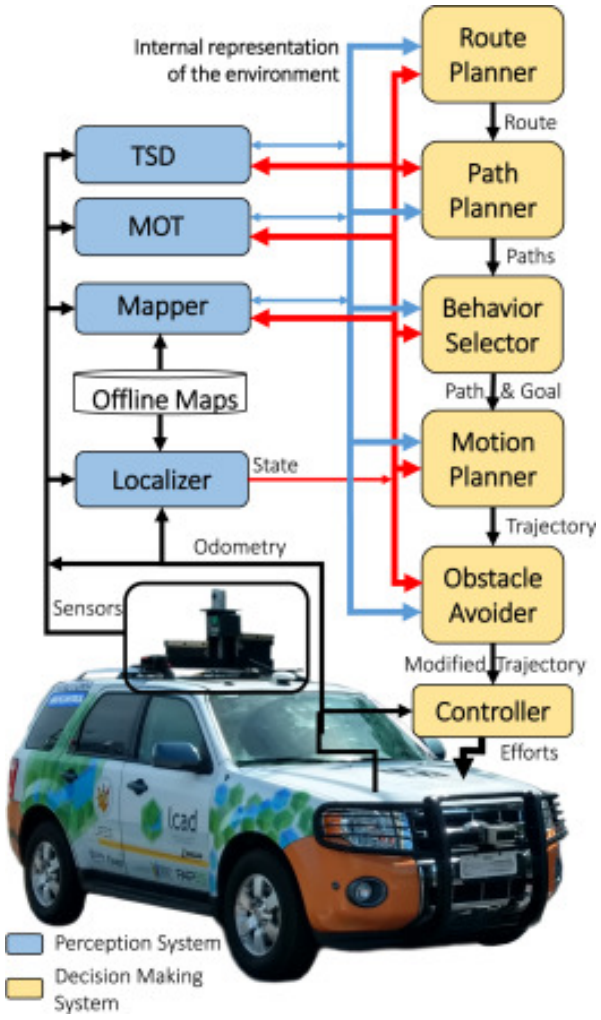

Fig. 1. The modules involved in a self driving car stack

The components of the pipeline can be designed either based on deep learning or using classical non-learning approaches. For example, a deep learning based object detector provides input to a classical A* path planning algorithm.

A. Pipeline

A self driving car uses different sensors around the vehicle, to create and maintain a map of the car's

surroundings. Some sensors include radars, lidars and cameras.

They also often employ sensor fusion techniques to integrate data from various sensors to create a complete and accurate representation of the environment. This improves the reliability of object detection.

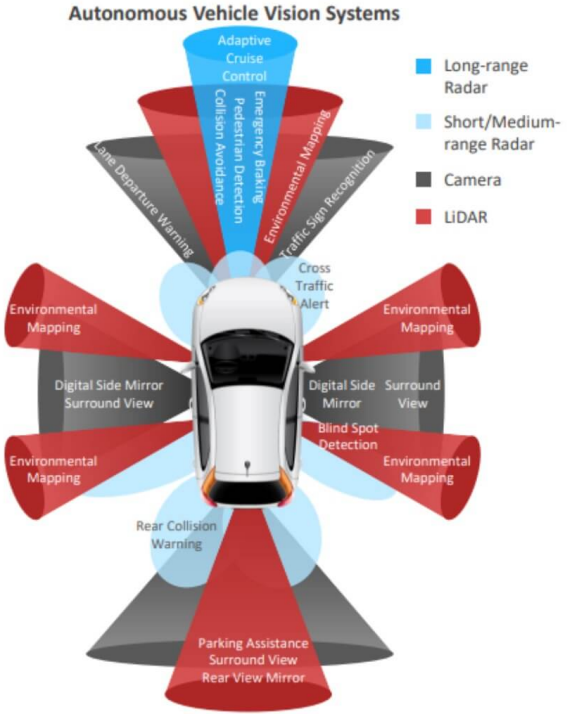

Fig. 2. Sensors on a Self Driving Car

Once the data is collected path planning, predictive modeling, hard-coded rules and object recognition help us navigate obstacles.

Computer vision can be considered the eyes of the car, and it helps us figure out what the world around it looks like. Sensor fusion is a way of incorporating the data from various sensors such as radar and lidar to gain a deeper understanding of our surroundings.

Once we have a deeper understanding of what the world around it looks like, we want to know where we are in the world, and localization helps with this.

After understanding what the world looks like and where we are in the world, we want to know where we would like to go, so we use path planning to chart the course of our travel.

A safety monitor is designed to assure the safety of each module.

Finally, control helps with turning the steering wheel, changing the car's gears, and applying the brakes.

B. Object Detection and Tracking

Every obstacle on the road needs to be detected by the car, if they are to be avoided. Tesla cars rely on data from 6 forward facing cameras and ultrasonic sensors.

Other companies use different configurations. This stage is crucial for localizing objects such as pedestrians, traffic lights/signs, other vehicles, and barriers in the car's vicinity.

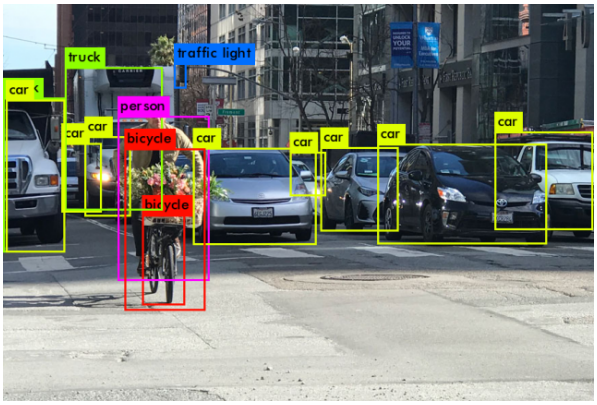

Fig. 3. An example of Object Detection in a driving scenario

YOLOv8 and RAANet are some 2d and 3d detectors that are at the forefront. Detecting an object on every frame in a video from the car's cameras slows down detection.

Semantic segmentation is also a crucial computer vision task in this context. It involves classifying each pixel in an image into one of several predefined classes, such as road, sidewalk, vehicles, pedestrians, traffic signs, and buildings. It helps us to understand and interpret the environment. There are several types of segmentation, and in the case of semantic segmentation, no distinction is made between unique instances of the same object.

Alternatively, you can use instance segmentation.

It's important to note that instance segmentation can be computationally intensive, and its practical use may depend on the hardware capabilities. In many cases, a combination of object detection, semantic segmentation, and traditional tracking algorithms may be sufficient.

C. Object Tracking

An important research area is to use the temporal and spatial correlations between consecutive frames to detecting objects and increase the speed of the model.

Object detection systems must operate in real-time to provide timely information for making decisions. High-performance hardware and optimized algorithms are necessary to achieve this.

In dynamic environments, objects move, so they can be partially or fully occluded by other objects or obstacles. Effective tracking algorithms must handle occlusions to maintain accurate object positions. Object tracking is used to maintain a continuous association between detected objects over time.

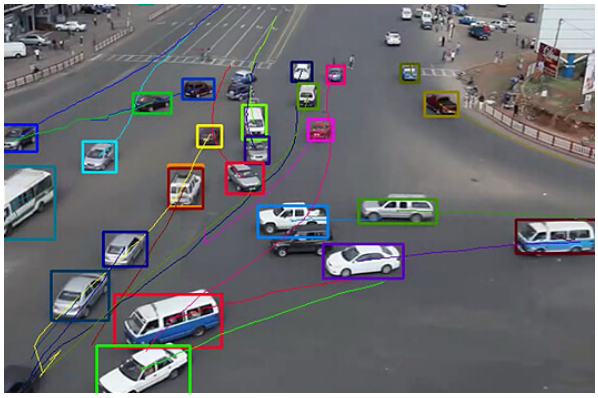

Fig. 4. An example of Object Tracking

These trackers predict the future positions and velocities of tracked objects. These predictions are essential for safe and anticipatory driving behavior.

Deep learning techniques, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), have shown promise in object tracking. They can capture complex temporal dependencies in object motion.

Tracking information is used in conjunction with behavioral prediction models to anticipate the future actions of tracked objects, such as predicting if a pedestrian will cross the road or if a vehicle will change lanes.

D. Lane Detection

Lane detection is required to properly navigate in urban scenarios and on highways. Lane Detection is a computer vision task that involves identifying the boundaries of driving lanes in a video or image of a road scene.

The goal is to accurately locate and track the lane markings in real-time, even in challenging conditions such as poor lighting, glare, or complex road layouts.

There are mainly two types of models: segmentation-based and anchor-based. Segmentation methods attempt to identify each lane pixel in the image.

On the other hand, anchor-based lane detection only searches for a limited number of key features based on predefined anchor points or boxes which improves computation time.

Another related task under active research is Lane classification. It refers to categorizing lane markings into

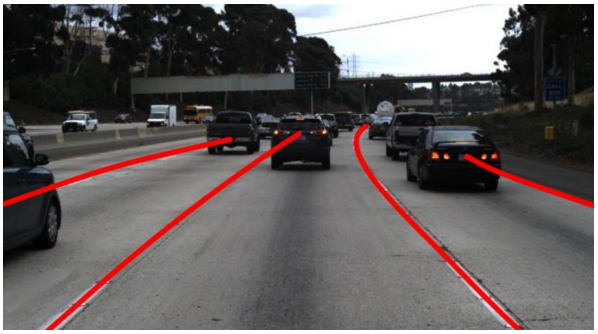

Fig. 5. An example of deep learning based Lane Detection

different classes, such as solid lines, dashed lines, or double lines. This helps us understand the proper lane uses and follow rules correctly.

More advanced systems can go beyond simple lane detection. They can detect lane changes and merging vehicles. This is crucial for ADC and lane-keeping systems.

E. Maps

Self-driving cars rely on various types of maps to operate safely and effectively. HD maps are central to precise localization and scene understanding, while semantic, localization, 3D, and weather maps provide additional context and information to enhance the vehicle's perception and decision-making capabilities. These maps collectively form a crucial part of the autonomous driving stack.

HD maps are very accurate maps of the road that includes details about lane, traffic light position, traffic signs, road markings, width of the road and so on. They are captured using lidar, GPS and aerial imagery.

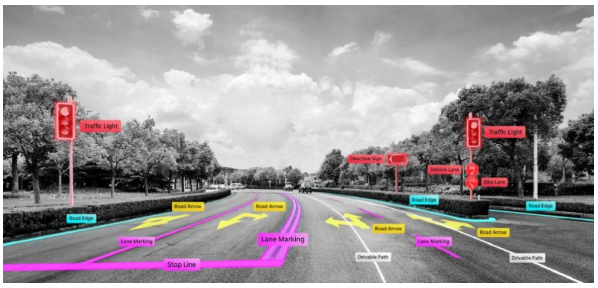

Fig. 6. An example of a motion planner deciding the car's trajectory

These maps improve perception, makes localization accurate and improves path planning to safely execute any movement.

HD maps provide redundancy for our perception. Even when the sensor data is limited due to poor weather conditions or sensor failures, the we can rely on the map data to maintain accurate navigation.

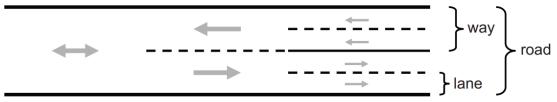

Fig. 4: Definition of road, lane and way

Fig. 7. The structure of the road

The downside is that it is not possible to create a HD map of the entire planet, much less keep it up to date. Integrating HD maps with existing perception and planning modules is an important branch of research.

You can also consider localization maps to serve as a reference for determining the vehicle's precise position on the road. They often contain information about landmarks features that can be detected by sensors. These maps are created during the mapping process. They help the ego car align its sensor data with the known landmarks.

F. Path Planning

The map created by the perception system serves as the basis for path planning. It typically has information about the roads, lanes, traffic signs, signals, and of course other objects.

Path planning starts with defining a clear goal, which can be the vehicle's destination or a any intermediate point it must go to.

Using a cost function and the map, the planning algorithm generates a set of potential paths from the vehicle's current position. These paths may be continuous or be made using waypoints.

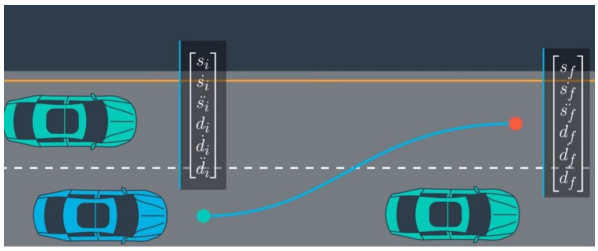

Fig. 8. An example of a planner deciding the car's future path

It must incorporate obstacle avoidance strategies to ensure that the path avoids collisions with other vehicles, pedestrians, and obstacles. This often involves predictive modeling of the behavior of other road users.

The planning activities can be described by using a top-down approach containing a hierarchy of three layers - route, behavioral, and motion planning.

Under mission planning, use algorithms like Dijkstra's algorithm or A* to find a high level path. One of the most challenging problem to solve is predicting the behavior of dynamic objects in the environment.

The result of mission planning is a set of road network segments or waypoints that the car needs to follow from its current position.

G. Behavioral Planning

The next planning step is behavioral planning, which decides how best to reach the next waypoint under the actual local driving context, i.e., with regard to the current road geometry, obstacles, other traffic participants, traffic rules, etc.

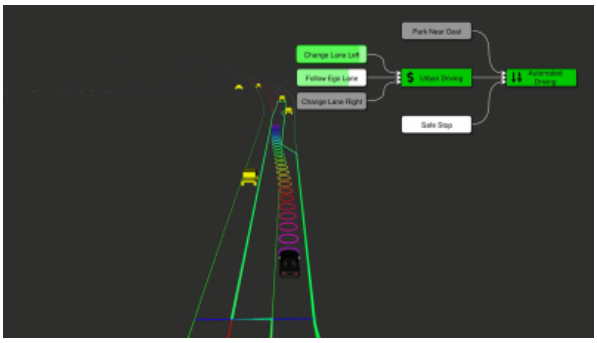

Fig. 9. An example of a planner deciding the car's trajectory

While path planning deals with the path of the vehicle, behavioral planning determines the vehicle's overall behavior, including actions such as lane changes, merging onto highways and yielding to other vehicles.

So the result of this planning step is a high-level decision, such as changing lane, lane following, merging, overtaking, etc.

One simple option for behavioral planning is to have a rule-based system, relying on predefined sets of rules and heuristics. On the other hand, we can use decision theory and consider probabilities and uncertainties to make optimal decisions.

Behavioral planning also involves dealing with ethical dilemmas regarding how the vehicle should respond in morally challenging situations with potentially deadly consequences.

H. Motion Planning

Motion planning is the problem of finding a sequence of valid configurations that moves the object from the source to destination.

The motion planner provides a set of driving commands for the vehicle controller over time, typically in the form of steering wheel angle, brake and throttle level. We must consider time in this step.

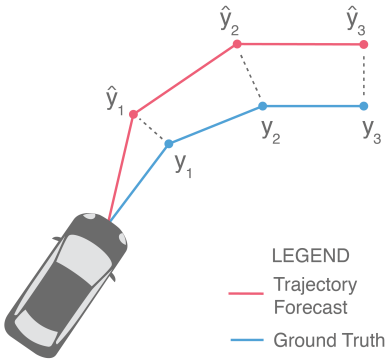

Fig. 10. An example of a planner deciding the car's trajectory

These algorithms generate continuous trajectories for the vehicle, which describe its path through space and time. These trajectories are composed of positions, velocities, and accelerations that the vehicle should follow.

This project does not need any motion planning algorithms as the simulator does not support it yet. In the meantime, we can possibly try CARLA or other open source simulators.

I. Depth Estimation

Depth estimation is a critical task that involves determining the distance from the AV to other objects in the environment. It is obvious that accurate depth estimation is important for real world scenarios.

In Monocular Depth Estimation, we use a single camera and estimate depth from image cues such as object size and perspective. This is often combined with machine learning algorithms to improve accuracy.

A more expensive option is to use Lidar. These sensors emit laser pulses and measure the time it takes for the pulses to bounce back after hitting objects. This information is used to create a 3D point cloud of the environment, which can be used for precise depth estimation.

Depth estimation can be challenging in adverse weather conditions and low-visibility scenarios. Sensor limitations, such as sensor noise and occlusions, can also affect accuracy.

This project does not need any depth estimation algorithms as the simulator does not support it yet. For now, we are using the object detection bounding boxes to estimate distance to the other cars.

III. Simulator

We use a simulator for careful testing and validation of driving models, and for the refinement of algorithms and systems to enhance the realism of the car responses.

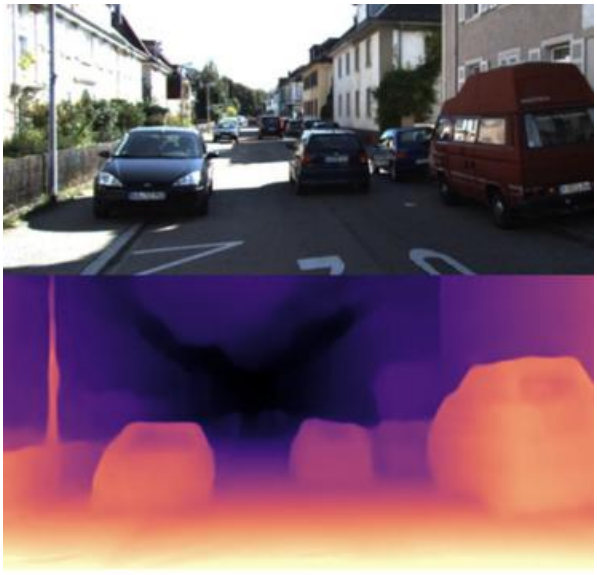

Fig. 11. Depth Estimation Example

During the test mode the simulator sends images, current speed, current steering angle and throttle information of the car to your python code which is responsible to sending the new steering angle, throttle and breaking signals.

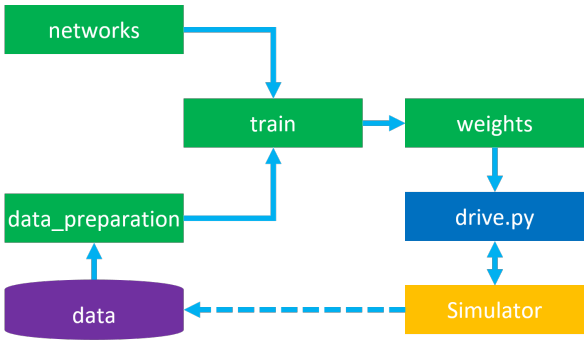

Fig. 12. Simulator

The provided simulator has a train mode so that data can be collected from the simulator.

A. Purpose of using simulators

Simulators can generate a lot of data, including sensor data, vehicle steering data, etc. This data is valuable for analyzing the performance of autonomous systems and refining algorithms.

Some, like Carla, even provide a scene editor that enables users to design and create custom driving scenarios. This is obviously useful for testing and validating driving algorithms under any conditions you wish.

They can simulate realistic traffic scenarios, pedestrian movements, and other road users. This helps us to

evaluate how autonomous cars interact with complex traffic scenarios.

They also provide us with many weather conditions for us to test for example, the breaking system.

When comparing multiple algorithms, we must establish a standard for evaluating performance. For this a simulator can be very useful. Some open source simulators that are available to us are Carla, Autoware, etc.

IV. Project

This section explains the techniques used in the project.

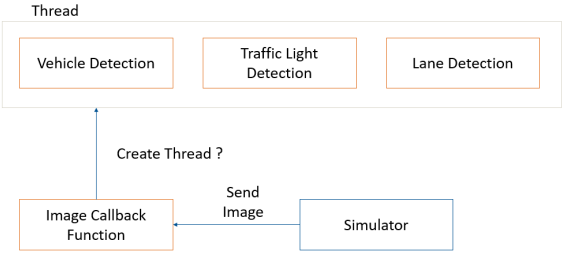

Fig. 13. Project Pipeline - Receiving and Processing Input Image

This chart is a general overview of the project. Initially, the simulator sends an image to the python code. Here, we decide whether to start a thread or not. This is done to help deal with the low FPS. When an image is already being processed by a thread, we do not create a new one. We simply drop the image.

Within the thread, we have all out components like object detection, traffic light detection and lane detection.

A. Obstacle Detection

RCNN operates in two stages, first proposing regions of interest and then classifying and refining object bounding boxes within those regions. This approach tends to yield high accuracy but is computationally intensive.

On the other hand, YOLO takes a single-stage approach, directly predicting object bounding boxes for the entire image in a single pass, making it significantly faster but it can potentially provide lower accuracy boxes.

The project uses a YOLOv8 model for object detection. It is a one stage detector, and as we just explained, it is faster than an R-CNN model.

When a R-CNN model was tried, it took around 10 seconds to complete inference when tested with the simulator. So the YOLOv8 was chosen over it.

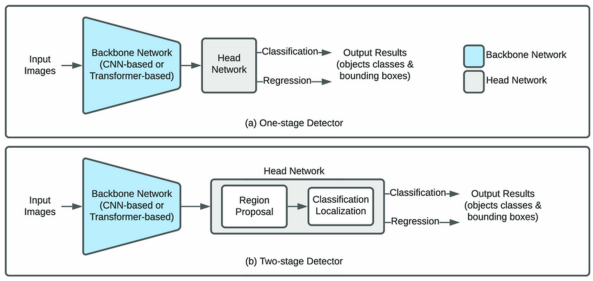

Fig. 14. Attempts with RCNN

The original pretrained YOLOv8 model detected non-existent traffic lights in the simulator. This must be due to domain change of input data. The original was likely trained on real-world images. When used in a simulator image, it was not accurate enough.

A domain shift or change is a change in the distribution between a training dataset and the data it encounters when deployed. These domain shifts are common in practical applications of artificial intelligence.

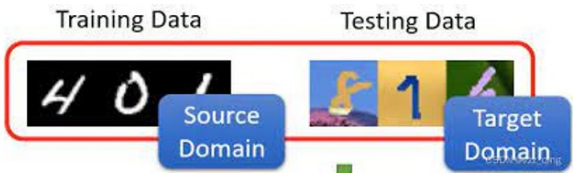

Fig. 15. Domain Change Issue

To fix this, the model was fine-tuned on a separate data set similar to the target domain.

The decision to apply the break, is done in a quite straight-forward way. Just calculate the center of the bounding box. If it is too far to the left or right, then it is not blocking your car.

B. Traffic Light Detection

The same YOLOv8 model predicts traffic lights and their active colors too. The detection is accurate and fast due to the use of YOLOv8.

Note that there could be multiple predictions of traffic lights. So for this project, we have chosen the most common color as the one meant for the ego car.

In order to counter this, we tried testing only a small region of the image for traffic lights. This proved to be ineffective as somtimes due to the FPS issue, even YOLOv8 would not detect the lights in time and the car just keeps moving forward. This usually leads to a crash.

A disadvantage is that is the car approaches the intersection at an unconventional angle, we also detect the traffic lights that are not meant for us.

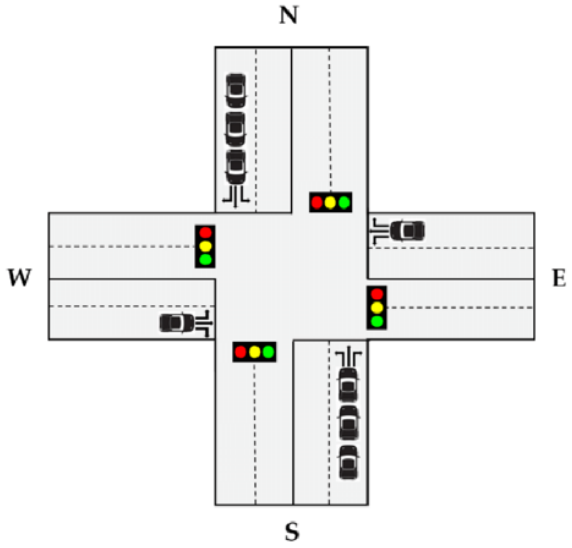

Fig. 16. Traffic Light Detection Issue

This can be resolved when global coordinates are provided or by using something like HD maps. In a more mature simulator, these issues can be resolved.

C. Lane Detection

Lane detection was first attempted using CNN based methods, but they all were too slow to be useful for the project. Each detection took around 15 seconds to predict lanes. This obviously caused a lot of crashes for the ego car.

So, they had to be abandoned and we went to the traditional methods of image processing to draw lane lines.

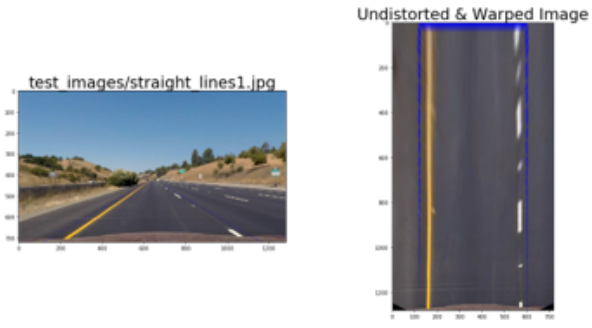

Fig. 17. Project Lane Detection

This involved performing edge detection using a Sobel filter. Then we used an histogram to detect the beginning of the lane lines.

Starting from the bottom slice, enclose a 200 pixel wide window around the left peak and right peak of the histogram.

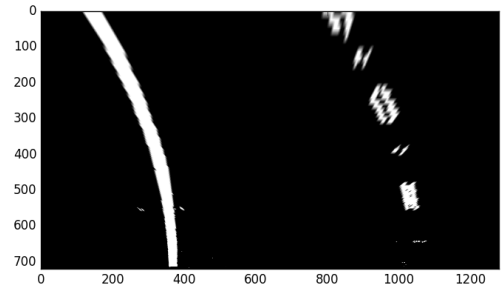

Fig. 18. Sobel Lane Detection

Go up the horizontal window slices to find pixels that are likely to be part of the left and right lanes, recentering the sliding windows.

Given 2 groups of pixels (left and right lane line candidate pixels), fit a 2nd order polynomial to each group, which represents the estimated left and right lane lines.

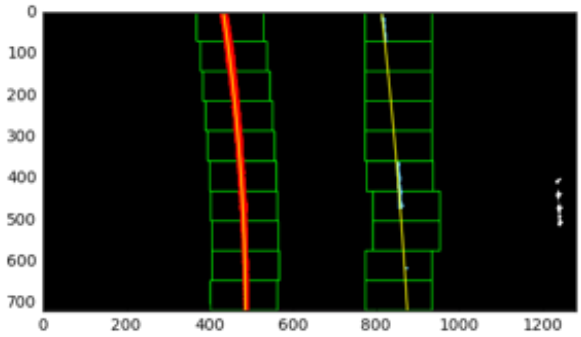

Fig. 19. Project Lane Detection

Given the polynomial fit for the left and right lane lines, we can calculate the radius of curvature for each lane line.

Also, given the polynomial fit for the left and right lane lines, we calculate the vehicle's offset from the lane center. This value is later used to calculate the steering angele for the ego car.

Finally, given all the above, we can annotate the original image with the lane area.

D. Intersection Detection

One task for improvement would be to use a model like SegFormer to perform road segmentation. It can help us prevent the ego car from driving onto the pedestrian lane.

Fig. 20. Project Lane Detection

Fig. 21. SegFormer Example

We can also explore whether it can help us detect intersections if maps and global coordinates are present. We can try to detect a large patch of on the screen and identify it as an intersection.

When this was initially attempted for this project, we faced the low FPS problem again and was not usable. The pretrained model also had the domain shift problem when tested on simulator images, so they were not accurate enough. More training would be required on appropriate data sets.

So, we trained it on a dataset based on the Udacity simulator. This reduces the domain change problem, but the model accuracy was still not high enough in our simulator for it to be used. This is an area to be improved in future iterations.

E. Handling FPS

Simulating a highly detailed urban environment with numerous vehicles, pedestrians and weather conditions is very computationally intensive. These simulators may experience lower FPS when handling such complexity.

In situations where you have limited FPS, making fast predictions is essential for ensuring that autonomous systems can respond effectively.

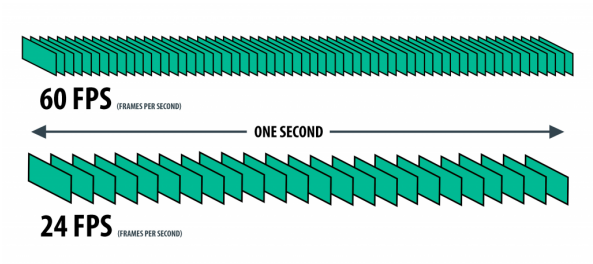

Fig. 22. What is FPS

To handle the issue with low FPS, a couple of logical changes were made. First, the speed limit of the car was reduced to reduce frequency of crashes. Probability was used to randomly skip calling slow functions or models.

Whenever an image reaches the code, one thread was spawned for each image decision. If another image arrived while this thread was running, it was skipped. No predictions would be made for it. This applies to all images received during this period.

For some agents, these changes may reduce prediction accuracy a little but will provide you with the necessary speed.

High-end GPUs can handle more complex simulations at higher FPS compared to lower-end hardware. So switch to them if possible. This code was run on a NVIDIA GEForce RTX with an Intel i7.

V. Results

The agent is able to detect other cars and trucks and stop when it has to. It can detect traffic lights and stop properly. The car is able to move along a straight lane as long as the FPS is handled well. If not, it tends to wear of the lane and moves randomly causing collisions. This was shown during the presentation.

Future work involves detecting intersections and making sure we stop at the lane limit line. Right now, the ego car keeps moving forward until it detects the red light. But sometimes it is too late and ends up in the middle of the intersection, only to be hit by oncoming traffic. But this is quite rare, as the fine-tuned YOLOv8 is very fast.

VI. Discussion

First, some words on the simulator. The simulator itself can be improved in some ways. For example, by providing the global coordinates of the cars, we can implement robust path planning algorithms for our agents.

We can also provide a map quite like an HD map to the agent to help us navigate.

Access to a map and global coordinates will allow us to stop the car properly at intersections. Right now, the car tends to stop further than it is supposed to. Occasionally it may get hit by oncoming traffic.

Once these issues are fixed, we can also add pedestrians and other obstacles to the simulator, to make it more complex. We will also require changes in weather conditions to more closely mirror the real world scenarios.

The future of self-driving cars holds the potential for safer, more efficient, and accessible transportation. But, addressing technical challenges and legal frameworks will be critical for attaining these benefits.

The potential benefits of self driving may be limited by disputes over liability, passenger concerns about safety, the implementation of a legal framework for self-driving cars, and security from cyber attacks. This will also have a negative impact on the large numbers of people currently employed as drivers.

When it comes to the agent, there are improvements that we can make. For example, we can try to use other lane detection techniques that are based on CNN. They will provide us with more accurate detections. From this we can calculate proper steering angles.

Pursuing an imitation learned based approach is not valuable here, because you cannot control the direction the car takes. It simply moves in the direction most found in the data set. So, we did not spend too much time with this area.

We can also pursue using finite state machines further. They are useful during the path planning for decision like changing lanes, etc. For this project, FSMs were not necessary, simple conditional logic did the trick.

One more improvement can be made to the capabilities object detection. We need to support mroe types of traffic lights, and also traffic signals. Different places obviously have many kinds of lights and signals. There are datasets that support this. So we can use them to fine-tune the Yolov8 model.

We can also consider using other sensors, maybe in other simulators, and improve the real world value for this car agent. Lidar and Radar are commonly used by companies. There are deep learning techniques associated with them. We can explore them in the next iteration.

Another important addition to make is to use object tracking. This will greatly improve the safety of the decisions the car makes because we will be able to model the future trajectory of the car. This is closely linked to behavior planning.

More testing in many weather conditions would also help a lot in increasing the trust and reliability of the agent.

VII. Conclusion

We have seen the methods and models used to develop a self driving car agent for the simulator and discussed the potential future works to improve its reliability. Self-driving cars have the potential for safer, more efficient, and accessible transportation.

Acknowledgment

I would like to acknowledge and thank the volunteers and GitHub repository maintainers who worked on the simulator for the duration of this course and being responsive to our requirements.

I would also like to thank Lingo Zhou for clarifying my doubts while working on the project.

References

[1] Sumit Ranjan and Dr. S. Sentamilarasu, “Applied Deep Learning and Computer Vision for Self Driving Cars.
[2] Joel Janai, “Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art.
[3] Shaoshan Liu, “Creating Autonomous Vehicle Systems.
[4] Hanky Sjafrei, “Introduction to Self-Driving Vehicle Technology.
[5] Mariusz Bojarski, Davide Del Testa, “End to End Learning for Self-Driving Cars.
[6] Ekim Yurtsever, Jacob Lambert, “A Survey of Autonomous Driving.
[7] Sorin Grigorescu, Bogdan Trasnea, “A Survey of Deep Learning Techniques for Autonomous Driving.
[8] Claudine Baduea, Ranik Guidolini, Raphael Vivacqua Carneiro, “Self-Driving Cars: A Survey.
[9] Matt Hardwick, “Simple Lane Detection with OpenCV. Medium
[10] Sander Ali Khowaja, “Brief Timeline of YOLO models from v1 to v8
[11] Amit Chauhan, “OpenCV: Adaptive and Otsu Threshold in Image Processing with Python
[12] Hack the Developer, “Lane Detection OpenCV Python
[13] Automatic Addison, “The Ultimate Guide to Real-Time Lane Detection Using OpenCV
[14] Hugging Face, “SegFormer HuggingFace Blog
[15] Jeremy Jordan, “An overview of object detection: one-stage methods.
[16] Hugging Face, “Semantic segmentation HuggingFace Blog
[17] Pytorch, “Pytorch YOLOv5
[18] Pytorch, “Transforming and augmenting images
[19] Ultralytics, “ Ultralytics Documentation YOLOv8
[20] George Sung, “Advanced Lane Detection
[21] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation
[22] Jeremy Jordan, “An overview of semantic image segmentation
[23] Enze Xie, Wenhai Wang, “SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers
[24] Jeremy Jordan, “Evaluating image segmentation models
[25] Berkeley DeepDrive Dataset
[26] Bosch Small Traffic Light Dataset
[27] CityScapes Dataset
[28] KITTI Vision Benchmark Suite
[29] Udacity Self Driving Car Dataset
[30] Deep Learning with Python

[31] Grokking Deep Learning
[32] Hanky Sjafrie, “Introduction to Self-Driving Vehicle Technology
[33] Shaoshan Liu, Liyun Li, Jie Tang, “Creating Autonomous Vehicle Systems
[34] Joseph Howse, Joe Minichino, “Learning OpenCV 4
[35] Malik Ghallab, Dana Nau, Paolo Traverso, “Automated Planning - Theory and Practice
[36] Sumit Ranjan, Dr. S. Senthamilarasu, “Applied Deep Learning and Computer Vision for Self-Driving Cars
[37] Stanford, “CS231n: Convolutional Neural Networks for Visual Recognition
[38] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, Brian Lee, “A Survey of Modern Deep Learning based Object Detection Models
[39] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, Jieping Ye, “Object Detection in 20 Years: A Survey
[40] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, Rong Qu, “A Survey of Deep Learning-based Object Detection
[41] Elahe Arani, Shruthi Gowda, Ratnajit Mukherjee, Omar Magdy, Senthilkumar Kathiresan, Bahram Zonooz, “A Comprehensive Study of Real-Time Object Detection Networks Across Multiple Domains: A Survey
[42] Yuning Wang, Zeyu Han, Yining Xing, Shaobing Xu, Jianqiang Wang, “A Survey on Datasets for Decision-making of Autonomous Vehicle
[43] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, Weisong Shi, “A Survey on Simulators for Testing Self-Driving Cars
[44] Daniel Omeiza, Helena Webb, Marina Jirotka, Lars Kunze, “Explanations in Autonomous Driving: A Survey
[45] Zahra Soleimanitaleb, Mohammad Ali Keyvanrad, “Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics
[46] Ruize Han, Wei Feng, Qing Guo, Qinghua Hu, “Single Object Tracking Research: A Survey
[47] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, Francisco Herrera, “Deep Learning in Video Multi-Object Tracking: A Survey
[48] Alka Choudhary, “Sampling-based Path Planning Algorithms: A Survey
[49] Siyu Teng, Xuemin Hu, Peng Deng, Bai Li, Yuchen Li, Dongsheng Yang, Yunfeng Ai, Lingxi Li, Zhe Xuanyuan, Fenghua Zhu, Long Chen, “Motion Planning for Autonomous Driving: The State of the Art and Future Perspectives
[50] Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, Emilio Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles