Real-Time Sensor Integration and AI-Driven Decision-Making for Autonomous Driving: Steering Prediction, Speed Adaptation, Lane Changes, and Parking

Rami Simaan

January 2025

Before we start, we highly recommend you to watch our video via this link. You can get the source code via Github by clicking here

1 Abstract.

In this paper, we present the development of a simulation-based autonomous driving system using the Godot game engine. The system integrates state-of-the-art machine learning and computer vision techniques, including the Nvidia DAVE-2 model for steering prediction and YOLO (You Only Look Once) for object detection. Our simulation features seven test tracks, each designed to evaluate various autonomous driving capabilities such as lane merging, speed sign detection, obstacle avoidance, lane changing, and autonomous parking. The test results demonstrate strong performance, despite the server-side computations being executed on a CPU. With CUDA-compatible GPU hardware, the system's efficiency and real-time responsiveness would significantly improve

2 Table of content.

3 Introduction

Autonomous vehicles are a rapidly growing field, with applications ranging from personal transportation to logistics. This project aims to simulate an autonomous driving system capable of handling real-world challenges, such as lane change, lane merge, obstacle avoidance and parking. Using the Godot game engine, we implemented a simulation with seven tracks designed to test

various driving scenarios. The system combines End-to-end learning for steering prediction(Dave-2) with object detection (YOLO) and trajectory planning (Bezier curves) to achieve robust performance.

4 System Design and Architecture.

Like we have mentioned before, the system simulates an autonomous driving environment with multiple tracks, each designed to test a specific capability of the car. This simulation aims to replicate real-world challenges by integrating advanced AI models. The core components of the system include:

- Perception: Using a front view camera to capture images and send them to Dave-2 to predict steering angle. Sensors to identify the existing of objects, then using Yolo for object detection.
- Planning: Using Bezier curves to plan LC and Parking maneuvers.
- Control: using Dave-2 model for steering angle prediction

Regarding the architecture of the system, it consists of two main inputs, sensors and cameras. Front view camera's images are fed into Dave-2 to predict steering angle and sent it to control to apply it. Sensors are being used to identify the existence of object in roads to then use YOLO for object detection.(we will discuss later the reason behind this, why not run YOLO each frame). The diagram below demonstrate our autonomous system.

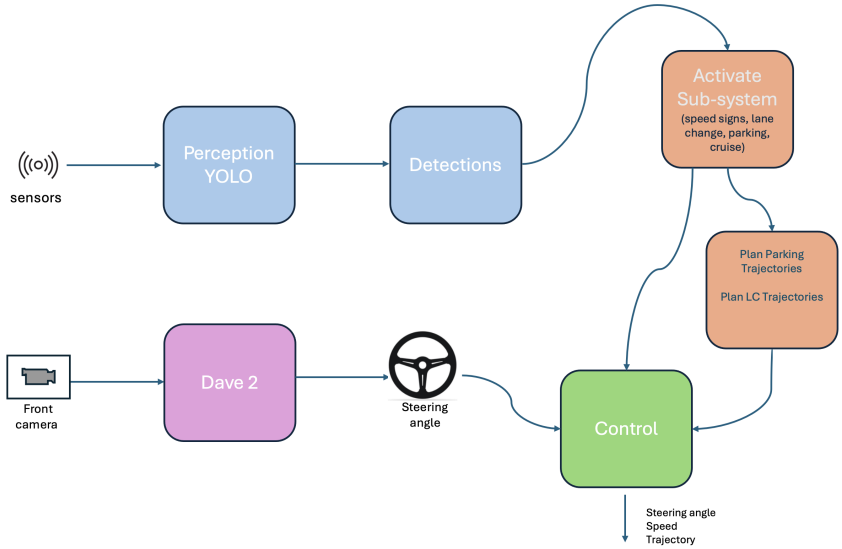

Figure 1: Autonomous System.

The system processes an image from a front view camera each frame, every frame is equal to 0.15 second at it fed to Dave-2 for steering angle prediction.

The predicted steering angle is sent to control and apply it. We will talk thoroughly about Dave-2 later. Sensors are being used to detect existence of object around the car. There are multiple sensors in the car each one has its job for a specific system.

4.1 Traffic Sign System

For predict traffic signs system, we have a sensor that scan from forward direction of the car to 30 degrees to the right and 10 meters forward. For illustration:

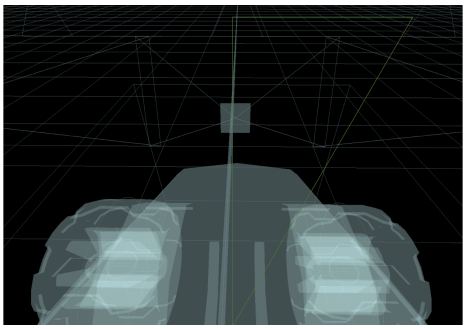

Figure 2: Sensor senses only the 30 degrees to the rigth.

After the sensor senses an object, the system captures an image using the front view camera of the car. This image is being sent to the perception system(YOLO), using YOLO in real-time object detection is efficient for detecting objects in the environment. We used Yolo v8n which is trained on German Traffic Sign Recognition Benchmark from Kaggle. There was a notebook and trained model which we used on this system. We tried to collect our custom data of traffic signs ourselves and train the YOLO model, but unfortunately, the model performance was poor in some cases, it was detecting false positives. After many attempts to improve our model, we chose to use the one from Kaggle. By utilizing such state-of-the-art Computer vision model, we managed to detect traffic speed signs successfully in the simulation and then use the predicted speed and send it back to the control system in order to apply the speed limit of the vehicle. When the front senor detect the existence of an object, it notifies the system to send request to capture an image and use YOLO on it to detect speed limit signs. We are using this approach to reduce the interference time and remove the necessity to run YOLO at each frame. Figure 3 shows an example of speed limit detection:

4.2 Lane Change System (LC)

4.2.1 Sensors

For the Lane change system, we have five sensors, three of them at the front view. And the other two centered at left side and right side of the car. We

Figure 3: Detection of speed limit 50 sign.

designed a control system for this system. You can see an illustration in figure 4:

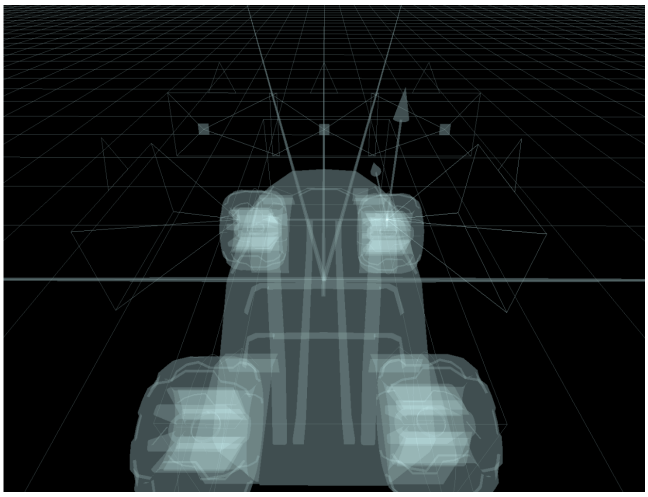

Figure 4: Sensors in LC system.

The control system is responsible for detection of objects around the car, when LC mode is activated the system perception is centered around the system above. LC mode is activated if the centered front sensor senses an object, then based on the control logic it checks which lane it must change to. This decision is done by checking the other two front left and right sensors. Based on the result, for instance, left sensor does not detect anything, which means the car can manage a left lane change successfully, accordingly the same for right sensors. Before starting LC maneuver and calculating it. The system checks if the maneuver can be done without any dangerous collisions, for that the side sensors are being used.

4.2.2 Path Planning

After the system confirms LC change, The planning of the maneuver trajectory is calculated based on Bezier curve and many factors about the environment. To smoothly plan the trajectory there are main factors that must be considered. **Lane width**, it determines the boundaries within which the car must operate, it's a crucial factor for determining the end position of the trajectory. **Lateral offset**, it is the horizontal offset distance between the car's current position(center of the lane), and its target position, it is crucial to determine the control point of the Bezier curve. These two main factors are crucial to calculate a smooth trajectory using Bezier Curve. To Calculate the trajectory given a start point (car's current position) and an endpoint. We use a **quadratic Bezier curve formula**, by defining a control point which is calculated based on start point and end point and an offset to create the curvature for smooth lane change. Still, we did not talk about how to find this end point. Our approach is based on the **lateral offset** and **forward offset**. The lateral offset is constant and depending on the lane width like we talked about. But forward offset is dependent on the detections in the environment. When a LC is activated, it is being done by the middle front sensor according to the distance from the detected obstacle or car. This distance has a role of defining the forward offset, a distant car which is like 15 meters afar will have a smooth trajectory, meaning the forward offset which controls the end point will have larger value. But when the distance is small, the forward offset will become shorter, which will result in a sharper trajectory. The forward offset is controlled by a hyperparameter called sharpness, which sets a sharpness value based on the distance from the target car.

$$(a) s_p = Current\ car\ position$$

$$(b) e_p = s_p + lateral_{offset} \cdot sharpness + forward_{offset} * lane_{width}$$

$$(c) curve_{offset} = \frac{lateral_{direction} \cdot lane_{width} \cdot direction}{2}$$

$$(d) c_p = \frac{s_p + e_p}{2} + curve_{offset}$$

After successfully planning a trajectory of the LC maneuver The Car must pursue this trajectory, the trajectory consists of a set of points in 3d, in our case is 40 points. To make the car travel from point a to point b in the 3d space we must calculate the steering angle needed that the car must apply to get from point a to point b. To achieve that we used the **Pure Pursuit algorithm**. It's purpose is to calculate the which is the angle of the front wheel that must be taken in order to reach a target point. The algorithm is as follows, given a path that a car must follow, a point is identified on that path, ld is the lookahead distance which is calculated based on the distance between the current vehicle position and the target point, L is the wheel base (the distance between the rear wheels) and is the angle between the current position and the target position. After creating the equations and simplifying them, we got a simple equation

which calculates the degrees in angles in which the front wheel of the car must take to eventually reach the target point. The formula is:

$$\delta = \arctan\left(\frac{2L \cdot \sin(\alpha)}{l_d}\right)$$

Where: - δ represents the angle of the front whe Which represents the angle of the front wheel. So, using this algorithm to calculate the needed steering an-

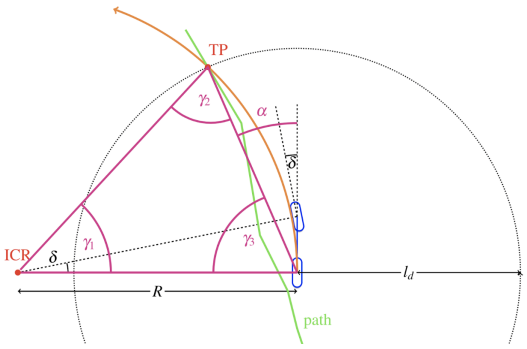

Figure 5:

gle to reach from point a to point b can be applied to each point in the trajectory that we found earlier. By applying it to each point the car can travel on this trajectory from start to end, meaning that the car can move from the start point of the maneuver and reach the end point of it successfully to complete the lane change maneuver. We would like to express our gratitude to our Friend Alaa Khamis(Also taking the lab this semester)about his help regrading path planning techniques. While our projects differ in scope, exchanging ideas provided valuable perspectives that contributed to the development of this system.

4.3 Autonomous Parking System

Like the LC System, the parking System utilized planning trajectory maneuvers using Bezier curves and utilized a pure pursuit algorithm to enable the car to complete the trajectory.

4.3.1 Sensors

This system utilizes state-of-the-art YOLO v8n model which is fine-tuned on our own custom parking lot dataset. By using this version of YOLO, the car can successfully detect an empty slot and an occupied slot and then notify the planner side of the system to plan trajectories to park. Now we will discuss the process in more detail. Given that the car know that it is a parking lot, the main system activates the parking system, after it initialized (set a low speed for the car), the left and right-side cameras of the car search for empty slot this

being done by capturing images by the left and right cameras every half second. Those images are being sent to the YOLO model that is trained on detecting empty parking slots, once the model detects an empty parking slot the system returns an answer of detection empty slot and by which camera. By gaining this information the parking system now can start planning the parking maneuver.

4.3.2 Path Planning

parking maneuvers start from the center of the detected empty slots, it consists of two trajectories, one which plans a straight trajectory in the forward direction of the car, the forward offset is set based on the width of the slot. (in simulation we assume that the width of each slot is equal and constant). The second trajectory is for the parking maneuver. It starts where the first trajectory ends. In this trajectory the lateral offset is key, we calculate the end point of the trajectory that must be at the center of the slot near the rear lane. To calculate this end point of the trajectory as we said, we use the lateral offset, and at which side we want to park at left or right. We use main factors like width and height of the slot in our case it is 4 and 5 respectively. By calculating the target point in the slot, we use Bezier curve to plan a trajectory like in LC, here we used a quadratic Bezier equation and a control point with offset to add a curvature to the trajectory to ensure smoothness in parking maneuver. After each trajectory is planned, the system sends them to control In order to use the Pure Pursuit algorithm. Does this approach always ensures a successful parking? No. There are cases when the parking slot YOLO model detects the free slot prematurely, meaning before the car reaches the center to the slot, hence the new trajectories which are planned will not be precise to ensure a successful parking. To overcome this, we added four more sensors at left back, left front, right back and right front positions. Let's say that the parking system detects a free slot using the left camera, then the left back and left front sensors are activated, they search for adjacent car, when the left back sensor senses the adjacent car it will notify the system that yet the planning can't start now, when the car reaches the area which both left back and left front sensors do not sense a thing meaning the car reached a point where it is parallel to the slot center point, then the parking system will activate the start planning trajectories system, when the parking lot is empty, the success rate of perfect parking will drop because the enhancement that we just described won't work on this case, but at all cases the parking will end with no collisions and no danger. In the figure below you can see un illustration of parking Lot YOLO model detecting an empty slot

4.4 Cruising system

For cruising system, no detection are notified for control. So, Dave-2 will keep sending steering commands to the control.

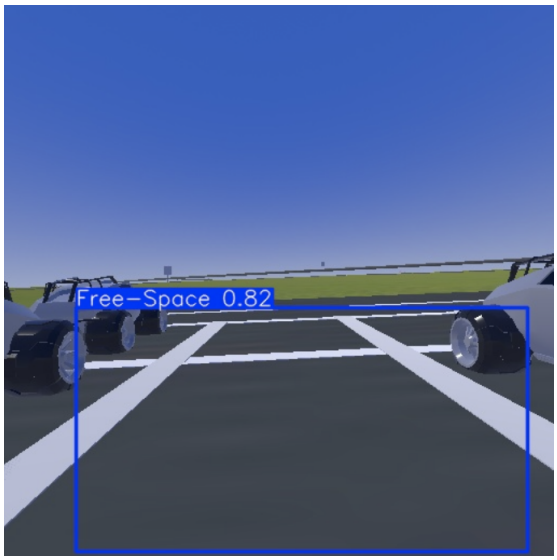

Figure 6: Detection of free slot

4.5 Dave-2

Moving on the core of our system, the Dave-2 designed by Nvidia. By utilizing such state-of-the-art model like Dave-2 we managed to train a CNN to predict steering outputs. Using a CNN in this kind of field and a backbone is very effective and shows promising results , in autonomous driving we want a way to detect lanes boundaries and this can be done by manual feature extraction, but designing a system which manually do that step by step(edge detection, lane detection , ROI etc.) can be time consuming and can lead to errors when the environment begin to get more complex, like introducing shadows curves and complex weather. Hence it would not work perfectly for autonomous driving, because our sole purpose is to drive good and safely. So, the CNN comes and automate this process by learning hierarchical features directly from data, there are multiple set of convolutional layers, the lower layers learn basic features like edges, middle layers start to learn road boundaries and higher layers will start to learn lane structure, this can be done directly from raw pixel data. By using this set of convolutional layers the model can capture spatial relationships within the image. Hence, CNN allows end-to-end learning for autonomous driving where the entire pipeline is optimized in a single step, feeding it one input image and predicting steering angle. Let's discuss how Dave-2 system works, like shown in the figure below:

At each frame of collected data, 3 images are collected from front center, left center and right center cameras. Those images undergo augmentations, like shifts rotations reduce brightness, padding etc.... after augmenting the data these images are fed into the CNN, the network computes a streeting command

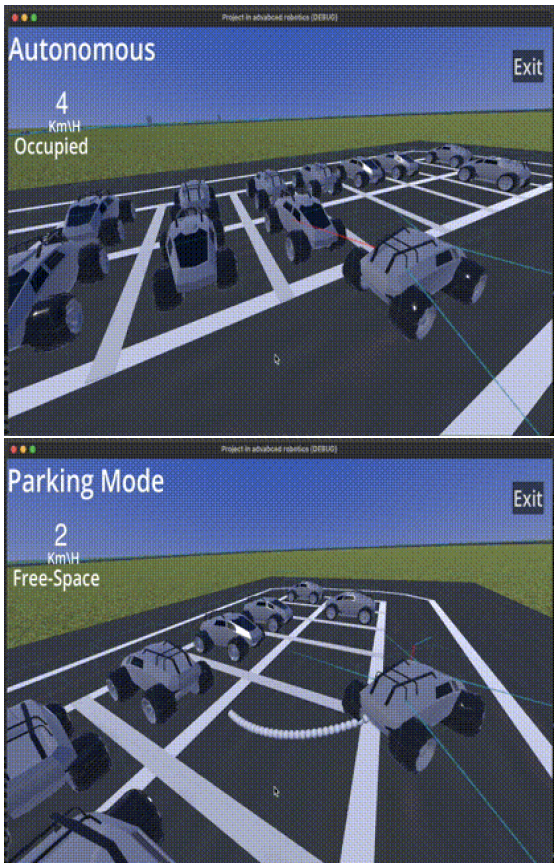

Figure 7: Parking system in real time

then it being compare with actual steering command by using MSE loss function, error has computed, then the network do back propagation and adjust weights. This is the process of training the model. To use the model in real time all is needed is to use the front camera as input, feed it to the model, and it predicts a steering angle. As simple as that.

4.5.1 Data collection

In the paper they collected data from various scenarios and roads, in our project, we collected data only from one track which is the first track in simulation, and then we tested it on the other tracks. The first track data were collected while driving on road with forward and reverse direction with multiple lanes; while collecting the data, the car drove on the right lane. This track has multiple complex scenarios, like merging from two lanes to one, or splitting from one lane to two lanes, curvy roads and different lane boundaries. We collected around

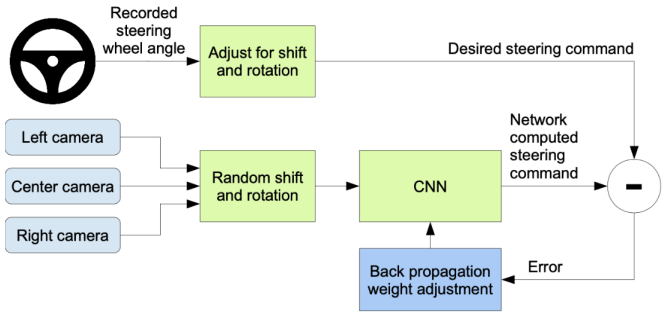

Figure 8: Dave-2 system

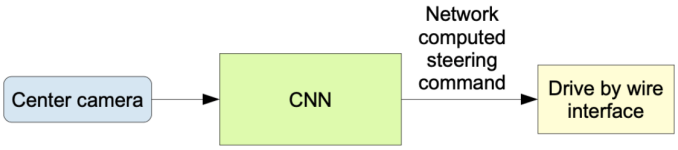

Figure 9: Dave-2's application in real time

15,000 images, we did not use all of them, we filtered some of the straight driving data, to not make the model biased regarding driving straight.

4.5.2 Network Architecture

We trained the weights of our network to minimize the mean squared error between ground truth steering angle and the one from the network. The architecture consists of 8 layers, 5 convolutional layers and 3 fully connected layers (we skipped the normalizing layer because the images are normalized when they are processed). The images are converted to YUV planes and fed to the network. The convolutional layers are designed to perform feature extraction, for the first three we used strided convolutional layers each one with stride of 2x2 and kernel size of 5x5, the last two without strides and with kernel size 3x3. For the first three we used a 5x5 kernel in order to extract main features and for the latter two layers we used a 3x3 kernel size to extract more finer and detailed features. The fully connected layers are being used to balance the value of the predicted steering angle. Each image that is fed to the network is resized to 200x66 and then fed to the network. You can see in the figure below the architecture of the model.

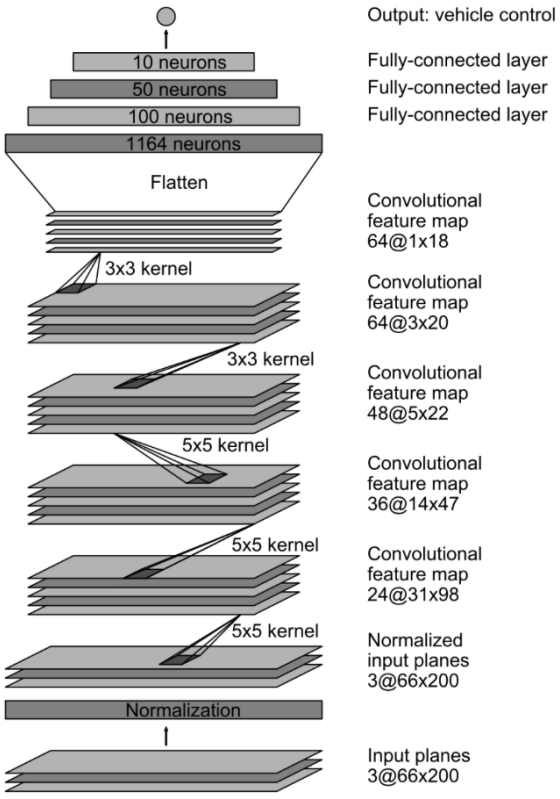

Figure 10: Network layers

5 Implementation

To implement such a system, we need multiple things, we need to simulate an environment to test our system, for that we used Godot game engine, our choice of using Godot is due to it being lightweight on our available hardware. Due to limitations of GPU and hardware incompatibly we could not use CARLA simulator for simulation and showcase our system, there is another simulator inspired by Udacity, we chose not to use because of similar reasons. By using Godot, we managed to build a car, with a motor, wheels and write a code to drive it. In addition, we designed a world with 7 tracks, each one testing each system of the car. In each track the car drives autonomously, using Dave-2 CNN model for steering prediction. To implement this world and car systems and interaction with the world we coded in C# to create this simulation and physics process needed to make it run as smoothly as possible. To utilize the computer vision and machine learning part in our simulation, we used Flask to run a server that connect between the two worlds: python libraries, model,

YOLO, Dave-2 and Simulation in C#. Regarding the functionality of server and client side, Simulation is the Client, and, on the server, we run the computer vision and machine learning tasks. In simulation when the system needs to predict a new steering command for the next frame, it captures an image and converts this image to a byte array in order save its content more efficiently and send the prediction request to the server. The server accepts the request, it uses Dave-2 model to predict the steering command and then sends it back to the client as a json file, the client (simulation) parses the json file and extract the needed results. Then send the result to the control to apply them. The same when we use YOLO for speed sign detection or for parking lot free-space detections.

5.1 Server-side implementation

To implement the server we used python as a primary language. We used multiple python libraries, such as Socket io and eventlet to lunch the server, thread pool to create multiple threads for independent tasks like predicting steering commands and YOLO detection. As we mentioned before the GPU limitation, our hardware is not supported to work with Cuda, So our only option to use AI models in real time and launch a server from the computer is to utilize only the CPU. We tried other option, which to use Google colab as a server and ngrok as a bridge between it and Godot, but it was much worse, because of the high latency and unstable connection of ngrok. We used cv2, tensorflow and Ultralytics for AI models.

5.2 Client-side implementation

To implement the client side we used C# as a primary language. In addition, we used godot and System as main libraries to implement most of the simulation. We are not going to explain all the details, but what we can mention that apart of learning new graphical programing skills to implement the simulation, we used a asset libraries to create the graphical side of our simulation,

First track showcases the autonomous model capability as much as possible to drive on the center of the lane, driving in road with merging and splitting lanes and curvatures. The second track showcases the system capability to drive on high curvature roads with shadows although the dataset which the model was trained on have no data with shadows, with this track we test the model limitations with the data that we trained it on. The third track showcases the LC system. The track consists of one straight road with three lanes, we chose random places across each lane to place the Boxes on, and for some we chose the places by ourselves to make it harder for the system. By activating this LC system, the car can avoid collisions with obstacles at a high rate, even if the places of the boxes have changed. Fourth track, test the system ability to drive at a high speed and detect speed signs at a high speed and apply changes to the control of the car accordingly. Fifth track tests the system's ability to drive on roads with curvature and speed signs and show how the car can adjust

its control accordingly. Sixth track is like the third track but instead of using the LC system to avoid obstacles, the system uses it to drive autonomously in traffic. The seventh track tested the Parking system on perpendicular parking in a one-way entrance parking lot, we designed a parking lot and placed on random slots a few cars. To implement such things in Godot engine we had to improvise and learn new skills, search for asset libraries that can help us build an efficient simulation.

5.3 Training of models

This was a hard part, we collected the data by ourselves, it is not enough to collect data and move on, we must make sure that the data is good and has no noise and can help the model to generalize well on new data. After a lot of tries we managed to get the best possible data we can acquire to use it to train Dave-2 model for steering prediction. Due to our GPU limitation training model like this on CPU takes a lot of time so we had to use Google Colab to train the model on the GPU. On the GPU training of the final modeltook around 1 hour, we trained it for 10 epochs 80 percent for training and the rest for validation while training. To train each of the YOLO models we had to collect the data by ourselves and annotate them, for each task, traffic sign detections and free-slot detection we collected 1200 images for each task and annotated them manually using Roboflow. It took a lot of time to finish the annotation process. But the training time was fast about 10 mins.The code for training Dave-2 and YOLO models will be included in the main folder. Although, at the end we use the traffic sign trained YOLO model from kaggle, still our model will be available on the repository.

6 Results and Evaluation

To evaluate our system we designed the tracks that we mentioned before, and evaluated each system individually. You can check our video (link in 1st page)

6.1 Autonomous Parking

After many experiments on autonomous parking system, we found impressive results. The system always identifies a free slot.In cases when it finds a free parking space near an adjacent car the system planning of trajectories and parking maneuvers are well executed at a high rate if not all the time with no collisions. While when there are no adjacent cars near the detected free slot, the performance drops slightly because of the end position of the car after completing the parking,it is not always centered but always between the lanes. Either way no collision happens. Our experiments do not include dynamic obstacles like pedestrians and environmental factors such as poor lightning, rain. Our experiments were done during a sunny day in the simulation.

6.2 Traffic sign System

This system demonstrated high performance on the test tracks in our simulation. However, we encountered a challenge during performance testing: running the server-side processes on a CPU while simultaneously executing the computer vision and steering prediction models. Over time, this setup affected the simulation's performance, as the latency for each model to complete its task increased progressively during the simulation.

6.3 Lane Change System

This system showed a high performance on the LC test track, as you can watch in the Youtube video. In rare cases, when the latency of the tasks gets longer it affects the steering model prediction to not stay at the center of the lane, which will lead to an overshot or undershot of the planned trajectories.

6.4 Dave-2

To evaluate the steering prediction model which was trained only on the first track, we can test it on the other tracks, we collected another data of track 2 and track 5, and tested the model. As you can see in these results, the predicted

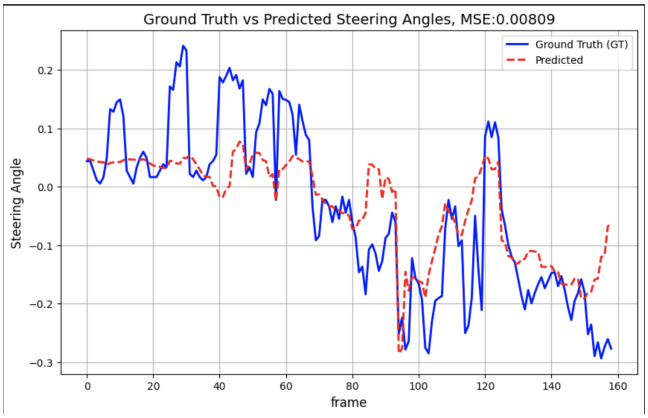

Figure 11: Track 2

steering in radians is very impressive, as you can see there are sometimes sharp values between the predicted and Ground truth values, this results from the fact that we as humans our reaction time to take a decision to a slight left or right rotation is longer than a computer than can do this in 30 ms using great GPU, maybe it can be done much faster using the best GPUS out there.

The Convolution layers of the CNN of Dave-2 system extract spatial features to identify boundaries of road, lane marking to eventually predict steering commands. Let us see example of the first and second activation of network.

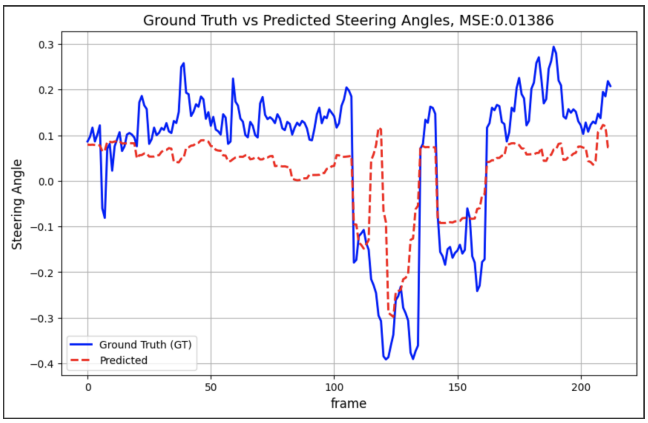

Figure 12: Track 5

7 Discussion

Our final product exceeded our expectations when we started. In the first month of working on the project, everything seemed impossible. Then we started to have doubts if we should continue working on this idea and switch back for the drone project. But we loved the idea of this project and really wanted to do something like this, which in our eyes is impressive. We managed to solve each problem at once, we divided the work between us and start working smart and not hard, we implemented each feature at a time, when we finish working on a feature, we start thinking about the next one and read a lot of papers to understand what is implemented in the real-world and what we can implement based on our hardware limitation, and then think about approaches, algorithms and models on how to implement the new feature. Our final product has only been tested on simulation, all of the big companies that develop autonomous car systems start with simulations to ensure a safe and an affordable way of testing the product. And then test in the real-world. In our case, due to a close deadline and unavailability of cars that have the number of sensors and cameras needed to implement this project in the real world, we stick to simulation for now. And to get such equipment that can effectively work in our system is very expensive. We are still thinking about continuing to develop this product and test it in the real-world.

7.1 Difficulties

Regrading difficulties that we faced, at the start was on how we should start, what simulator do we must choose, which one works on our computer effectively, on what available papers we must focus on, what is the final product must be. But after, we managed to solve each one at a time. And here we are :). Other difficulties that we faced are mainly performance related ones, all the simulation

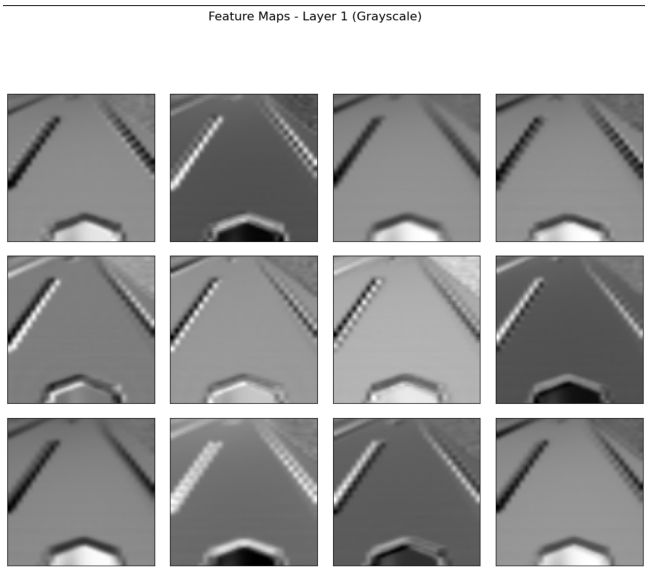

Figure 13: First activation

server side is running on the CPU and this kind of models like YOLO and DAVE-2 running on the CPU, each second when we running the simulation, 10 task are sent to the server and the CPU must complete them in a second, because real-time interference in autonomous driving is important, any lag in the system can cause a breakdown in the other systems. For instance, after running each track, the latency of the completed tasks done by the server from dave-2 model is about 30-35 ms but after a few minutes the CPU starts to get tired so those same tasks will take 60-70 ms and sometimes it reaches 100 ms, also the same about The YOLO model, they start with 70-90 ms and sometime they reach 150-200 ms. Hence, after a while with launching the simulation, it is expected that the performance will be lower. There were some cases when our CPU did not get tired in one track and the car kept driving for an hour autonomously. But if we ran the simulation on hardware with an excellent GPU supported by Cuda to lower the interference time the simulation experience will be much better and results in much more powerful performance.

7.2 Improvements

This product in our eyes is impressive. But there is always room for improvement. We can use a Dave-2 model which is trained on more diverse data in order to test it in the real-world, our approach of autonomous parking is great, but we think we can improve it with a new approach, which is to detect the tip point of two lanes that define the slot and according to those we can calculate the slot center more accurately. If not all, a very high percentage the parking

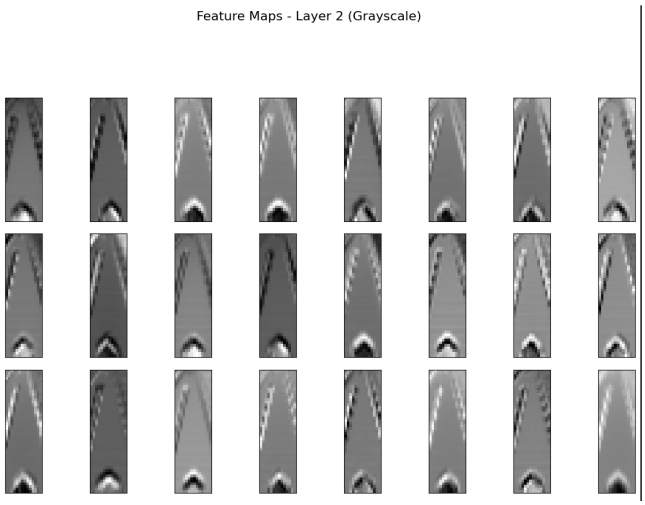

Figure 14: Second activation

maneuvers will be successful. we started on trying this approach but because of close deadline we could not yet find a solution. We can also use the Reinforcement learning approach to take decisions in an environment, but training such model like this need a powerful GPU and google colab will not work in our case due to previous encounter with it crashes during training, it is not stable and suitable for this tasks. Also Model Predictive Control can be a good improvement. For perception we can still improve the system by introducing a more complex sensor fusion approach.

8 Conclusions

Our product of autonomous car system performed great relative to our hardware limitations. The model , algorithms and approaches that we used in order to implement the system showed great results. Dave-2 by Nvidia , LC, perception and parking system showed remarkable results on the test track that we created in Godot simulation. By improving the hardware, we can get even much better results and even improve our approaches which leads to a better overall product. Which then we can use in real-world autonomous applications.

9 References

•Nvidia's paper. •YOLO paper. •German Traffic Sign Recongiton Benchmark. •Trained traffic sign YOLO model. •Roboflow for annotations. •We got help in implementing dave 2 and to understand the paper deeply using this playlist, In addition we used some of his code.. •Got help also to implement the code

using this repository. •Using Bezier idea for path planning was from. •Idea on how to implement ours own approach in autonomous parking came from. • Pure Pursuit Algorithms.