Final Report

Autonomous Driving via Reinforcement Learning with Object Detection

Celina Polanco
Charnay Hatten
Brandon Fitch
Ory Wickizer
Firas Abou Karnib

April 26th, 2024

Supervised by: Prof.Mohammad Nafea

Table of Contents

UNIVERSITY OF DETROIT MERCY

Introduction

As Artificial Intelligence (AI) continues to expand, the solution to a fully autonomous vehicle (AV) becomes more achievable. The levels of AV's range from zero to five with level 5being fully autonomous. A level 5 AV is defined as a vehicle that can perform all parts of the dynamic driving task (DDT) [1]. To achieve this level of autonomy, it requires multiple systems working together to mimic the DDT performed by a human. In 2023, Waymo introduced their private transportation service utilizing a fully automated vehicle [4]. While this is an impressive feat, these vehicles are not available for public purchase as they are still being tested to develop a more robust system. Until then, developing such a large system is a topic that many researchers are interested in.

The main systems of an AV are 1) sensing, 2) localization and mapping, 3) planning and driving policy, and 4) control [2]. Each system can be developed in unique ways. For sensing, the most common sensors utilized are the camera, LiDAR, radar, and ultrasonic sensor. By combining the inputs from these components along with a computer vision system, the surrounding environment becomes less abstract. For detection, machine learning architectures such as a convolutional neural network (CNN) and a single shot detector (SSD) are used [3]. For localization and mapping, sensors such as GPS and IMU are used to determine where the vehicle is with respect to the world. The planning and driving policy system receives input from the first two systems to determine how to maneuver the vehicle. Finally, the control system is driven by commands from the planning and driving policy system to steer and accelerate the vehicle as needed. Of these systems, the project focuses on the sensing of the environment, including object detection, and low level planning and control based on the sensed environment.

Within the sensing system, object detection algorithms such as You Only Look Once (YOLO), are instrumental. YOLO's unique approach to object detection facilitates real-time detection by processing images in a single evaluation to predict bounding boxes and class probabilities. This capability is vital for autonomous systems, where the timeliness of object detection and classification directly impacts the vehicle's ability to respond to dynamic road conditions. YOLO's efficiency not only enhances the vehicle's perceptual accura cy, but also integrates with the vehicle's broader decision-making framework, underscoring the importance of advanced object detection in AV development. In this project, we employ YOLO for object detection and localization.

Parallel to the evolution of object detection is the domain of reinforcement learning (RL), which offers a robust framework for developing a decision-making agent. RL algorithms enable agents to learn optimal actions through direct interaction with their environment, guided by a reward system . For autonomous driving, the agent is the vehicle, and the reward aligns with key objectives such as safety, efficiency, and compliance with traffic regulations. The iterative process of action, feedback, and adaptation in RL is ideally suited to the complex and varied scenarios encountered in autonomous driving. This methodical approach to learning and adapting exemplifies the potential of machine learning to enhance the strategic capabilities of autonomous systems. We adopted an RL framework for the planning component of our project.

The convergence of YOLO's object detection with RL algorithms represents a significant stride in the quest for fully autonomous navigation. Recent research demonstrates that the integration of these technologies improves the vehicle's navigation and decision-making in complex environments [11]. By leveraging YOLO for detailed environmental perception and informing the RL agent's decision-making process, this synergistic approach enables more sophisticated navigation strategies. Such advancements not only improve the vehicle's ability to navigate and respond to obstacles, but also ensure adherence to traffic rules. This illustrates the collaborative potential of perception algorithms and machine learning in driving the future of autonomous vehicles.

Overall, in this project, we successfully implemented the YOLO object detection system and the RL framework for path planning, within the well-known CARLA simulator [15]. Their integration is left for future development. The rest of the report is outlined as follows. We begin with an initial design which was an outcome of our research. We then update this design with the specific solutions we chose and their implementation. We then state the results and their implications. We discuss future directions next, and we conclude with statements about ethical considerations of our work and its alignment with existing standards.

Initial Design

By deciding which systems the project will focus on, the objective became clearer: follow traffic laws and avoid collisions. Creating a design of how the intended system was needed to derive engineering solutions. Below are the requirements driving the effort of the project.

- The system will efficiently detect stop signs, traffic signals, pedestrians, and vehicles and localize them with respect to the vehicle.
- The vehicle will stop 3 feet before stop signs and remain stopped for 5 seconds.
- The vehicle will stop at red lights.
- The vehicle will slow down to a stop at yellow lights.
- The vehicle will proceed on green lights.
- The vehicle will detect pedestrians and other vehicles in its direct path towards destination, maneuver around these objects by switching lanes or stopping when needed.

Having these requirements in mind, a block diagram shown in Figure 1 was developed to show how the systems will work together. The diagram shows the main systems of the project which are sensing, detection & localization, and motion planning. The project will use these systems to achieve the design requirements stated above.

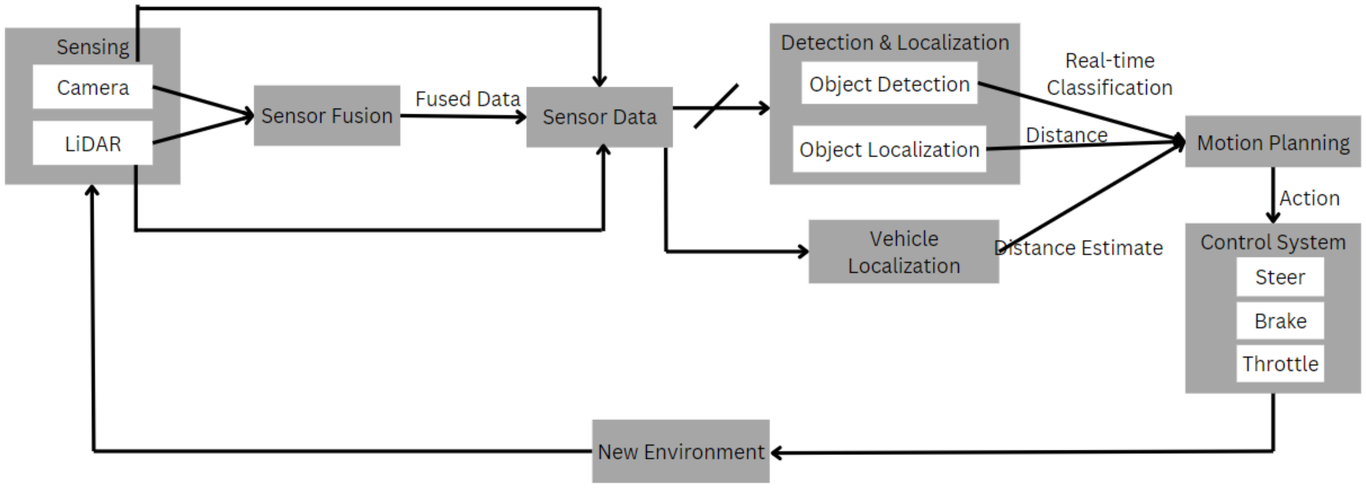


Figure 1 Initial Block Diagram
The sensing block has a camera and LiDAR whose data is sent to the sensor fusion block but also sent separately to have both the fused data and the individual data. Detection and localization uses this data to detect and localize the surrounding objects. To localize the vehicle, the same data is used. Having all of this information, the motion planning block can determine how to move the vehicle, sending commands to the control system which creates a new environment.

YOLO (You Only Look Once)

Prior to the advent of YOLO, detecting multiple objects in a single image typically involved multi-step approaches like selective search or Haar cascades, which first identify regions of interest within an image and then classify those regions using a convolutional neural network (CNN). In such frameworks, the CNN—a type of deep neural network especially adept at processing visual data—is repeatedly applied to various segments of the image to determine the presence and category of objects, a process which can be computationally intensive and time-consuming.

Introducing YOLO the state of art with a groundbreaking leap in object detection technology that reimagines this process with remarkable efficiency and speed. By design, YOLO unifies the two-stage process into a single-stage algorithm that simultaneously predicts multiple bounding boxes and class probabilities across the entire image in a single forward pass through its neural network architecture. This holistic approach, often referred to as real-time detection, significantly accelerates the task of object detection without compromising accuracy. It treats object detection as a simple regression problem, directly moving from image pixels to bounding box coordinates and class probabilities. This methodology allows YOLO to perceive and interpret the surrounding environment in real-time, which is a crucial advantage for applications requiring rapid and reliable decision-making, such as autonomous vehicle systems.With YOLO, our autonomous driving system benefits from a detection method that is not only swift but also powerful in its ability to generalize from training data to the dynamic conditions of real-world driving. By integrating YOLO into our system, we address a pivotal challenge in vehicle autonomy: enabling the vehicle to understand complex environments quickly and reliably, leading to safer and more efficient autonomous navigation.

The YOLO architecture, as illustrated in Figure 2, streamlines the process of visual recognition by dividing it into three purposeful segments: the backbone for feature extraction, the neck for feature processing, and the YOLO head for making predictions. Starting with the backbone, YOLO employs convolutional layers, each tuned to distill and analyze varying aspects of visual data. Here lies the unique C3CA configuration: 'C3' refers to blocks of 3x3 convolutional filters that process spatial data, and 'CA' implies the integration of Cross Stage Partial networks with attention mechanisms. These enhance the network's ability to discern complex features. The suffixes attached, like '_3' or '_6', indicate the depth of residual blocks and attention units, thus amplifying the model's learning capacity.

At the backbone's conclusion is the Spatial Pyramid Pooling for Features (SPPF) layer, which acts as a versatile connector, amassing features across different scales and contributing to the network's ability to generalize over various object sizes without being tied down by scale. The neck of YOLO's architecture is where feature refinement occurs. It employs upsampling techniques to retain the clarity of the finer details and concatenation to merge feature maps from different points of the network. Such fusion ensures the maintenance of detail crucial for the precise localization of objects within an image. Proceeding to the YOLO head, it comprises several specialized convolutional layers, each designed to detect objects at various scales within the same frame. This multi-scale detection is essential for YOLO's proficiency in dynamic environments, as it allows the network to identify a broad range of object sizes with impressive accuracy. Max pooling layers scattered within YOLO's design play a pivotal role in scaling down the feature maps. They selectively emphasize the most significant features, enhancing YOLO's computational efficacy and enabling the model to operate swiftly. Bottleneck layers, tactically placed, work to streamline information flow, compressing data to expedite processing while ensuring that critical information required for detection is preserved.

In essence, the architecture of YOLO, as depicted in Figure 2, is a harmonious blend of speed and accuracy. It converts raw visual data into actionable insights with exceptional rapidity, embodying the forefront of real-time object detection. This integration of advanced neural network design into our autonomous vehicle (AV) project not only elevates the current state of vehicular technology but also extends the frontiers of what can be achieved in terms of real-time perception and decision-making within the dynamic realm of autonomous navigation.

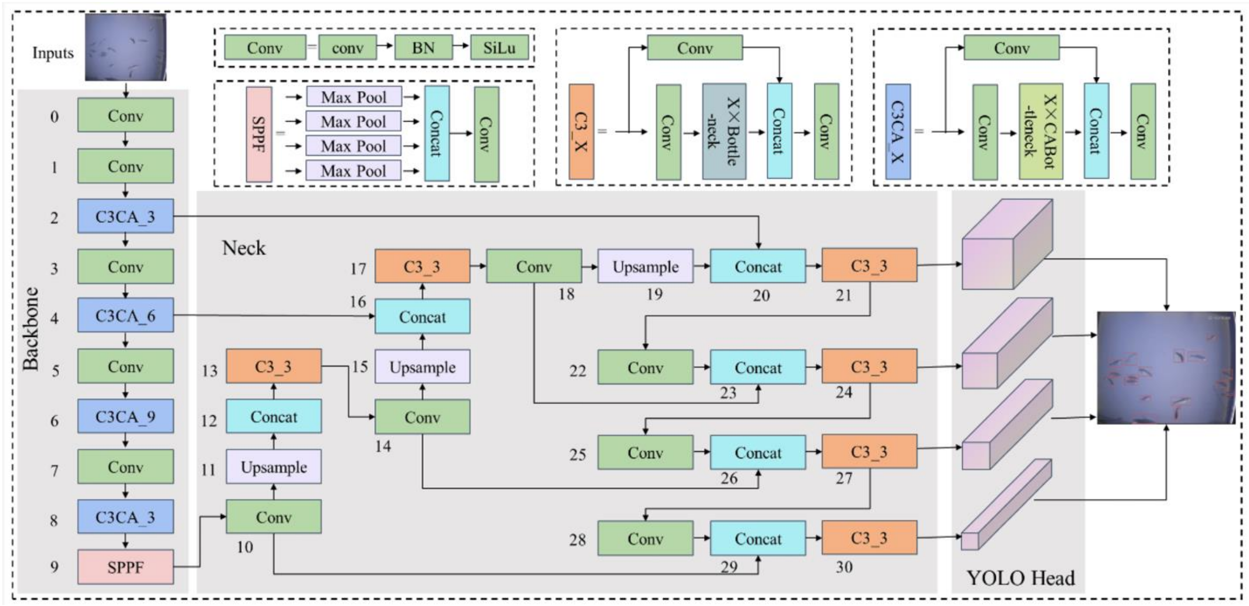


Figure 2 YOLOv5 Network Architecture

RL

Now that the detection system uses a state of the art architecture, using contemporary methods for vehicle control is needed. While there are many solutions to the autonomous driving problem, one way researchers are approaching the problem is through reinforcement learning. Traditional approaches include separate detection system and a pre-specified control algorithm. RL on the other hand is a framework for a decision making agent which learns to update its actions based on its experience, encoded in a reward function. Figure 3 provides a foundational understanding of how RL work and the interaction between the agent and the environment. As seen in Figure 3, the agent sends an action to the environment. This action updates the environment to change how it looks, i.e. turn the car left. Based on the outcome of this action, there is a calculated reward and a new state that is sent to the agent. Depending on the reward, the agent will try to improve the reward if negative, or continue to improve the reward if positive. Two important terms in RL are, exploitation vs exploration. In exploitation, the agent takes the highest immediate rewarding action given a state [5]. This can be a challenge during the implementation phase because the agent may continuously turn left if the reward is higher when performing this action. However, in exploration the agent chooses a random action with no consideration for the reward [5]. This exploration allows the agent to learn other actions that could give higher rewards. If the agent only performs exploitation but no exploration, it may never learn about these higher rewards because it is continuously performing the same action. Exploitation vs exploration is an issue that needed to be considered during the implementation phase.

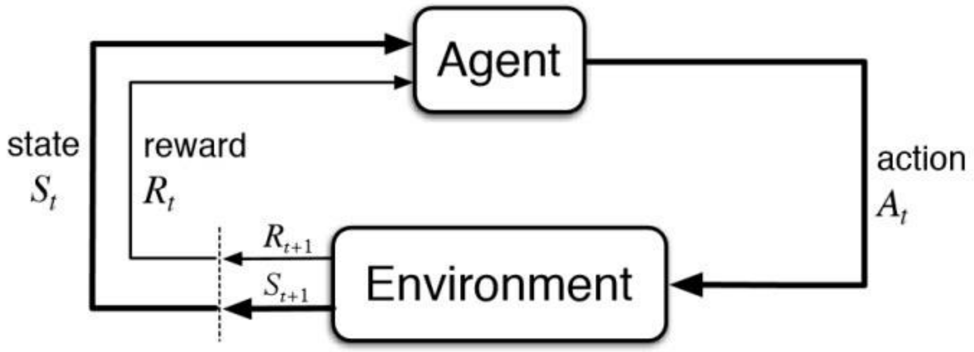


Figure 3 Basic RL Block Diagram

Under the branch of RL are many ways to implement a system. In the value-based function, the agent tries to quantify the value of every state-action pair and selects the highest score[6]. In the policy-based function, the agent defines a probability distribution over a set of possible actions and selects the highest score[7]. Figure 4 and Figure 5 display how the function maps look different. Another implementation known as Actor-Critic utilizes both value-based and policy-based. The actor controls the agent's actions making it policy-based and the critic measures how good the action taken was making it value-based. Within these functions are more detailed algorithms such as DDPG, PPO, A2C, DQN, and so on. Of these algorithms, Advantage Actor Critic (A2C) was the best algorithm for autonomous driving.

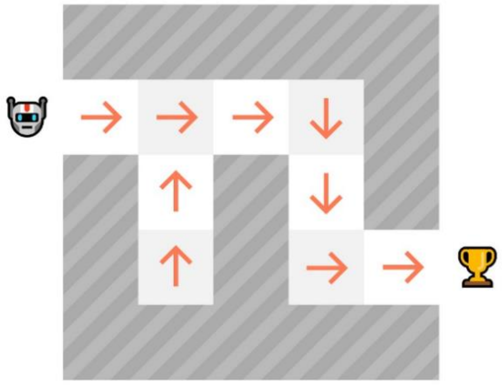


Figure 4 Policy-Based

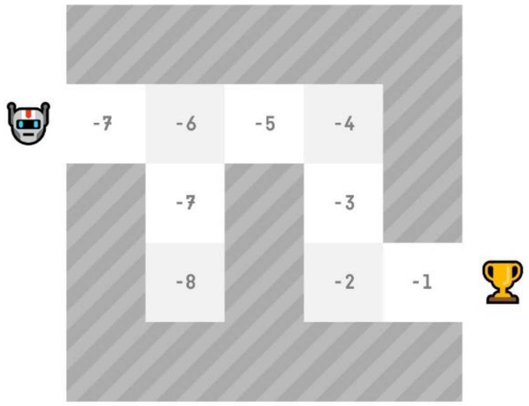


Figure 5 Value-Based

A2C was selected for the project because the architecture is similar to that of a Generative Adversarial Network (GAN) which is an established, well-known framework. The main difference between actor-critic and advantage actor-critic is the advantage equation. This equation can be seen in Figure 6. The r represents the immediate reward after taking the action. The gamma is the discount factor, which determines how much the agent should value the future reward ranging from 0 to 1, 0 being none and 1 being 100%. V(s') is the value function of the next state and V(s) is the value function of the current state. The benefit of the advantage equation is that the agent will reflect after every time step rather than at the end of every episode. This is important because the agent will quickly learn which actions give a higher reward and which ones do not. A2C is the best algorithm for autonomous driving as it will learn quicker and consider the value of all actions.

$$A(s,a) = r + \gamma V(s') - V(s)$$

Figure 6 Advantage Equation

CARLA

Implementing RL on a real vehicle is unsafe and requires very simple environments, while gradually increasing the difficulty of it. Because of this, the system was developed through simulation which allows for a safe and controlled environment. Autonomous driving researchers have developed systems in simulators such as Gazebo, Unity3D, and Car Learning to Act (CARLA). CARLA was chosen because it is an open-source platform that provides a customizable environment. The simulator is equipped with 12 urban towns, each with 15 different weather conditions. Each town simulates buildings, roads, pedestrians, and other types of obstacles and traffic information devices (e.g. stop signs, lane lines) at a high resolution with realistic physics and behaviors. An example of such an environment can be seen in Figure 7. Using such a high quality simulator creates a more practical and transferable system. The customizable environments also give the ability to test, train, and validate the system with different scenarios which will ensure that it is robust and safe.

To integrate the RL agent within CARLA, a python library known as OpenAI Gymnasium (GYM) was used. This defines a standard API for communication between the agent and the environment i.e. CARLA. By using GYM, the RL agent can be trained online because it provides an interface between the agent and CARLA.

Figure 7 Carla Environment
In this environment, the weather is wet and cloudy. There are many vehicles, trucks, motorcycles, and pedestrians. This showcases the many different customizable parameters within CARLA.

Implementation

Having such a large design, it was important to start small and continue to build on the implementation. Therefore, the following design includes the system solutions and the focus during the implementation phase:

- The YOLO detection system will efficiently detect vehicles and red, yellow, and green traffic signals.
- The RL agent will not collide with any object.
- The RL agent will not exceed 8 m/s.
- The RL agent will stay within the lane lines.
- The RL agent will not accelerate laterally.
- The RL agent will not steer sharply.
- The RL agent will not stand still.

After determining which machine learning algorithms were best suited for the system, the block diagram was created to include the system modules and how they will work together. The block diagram can be seen in Figure 8.

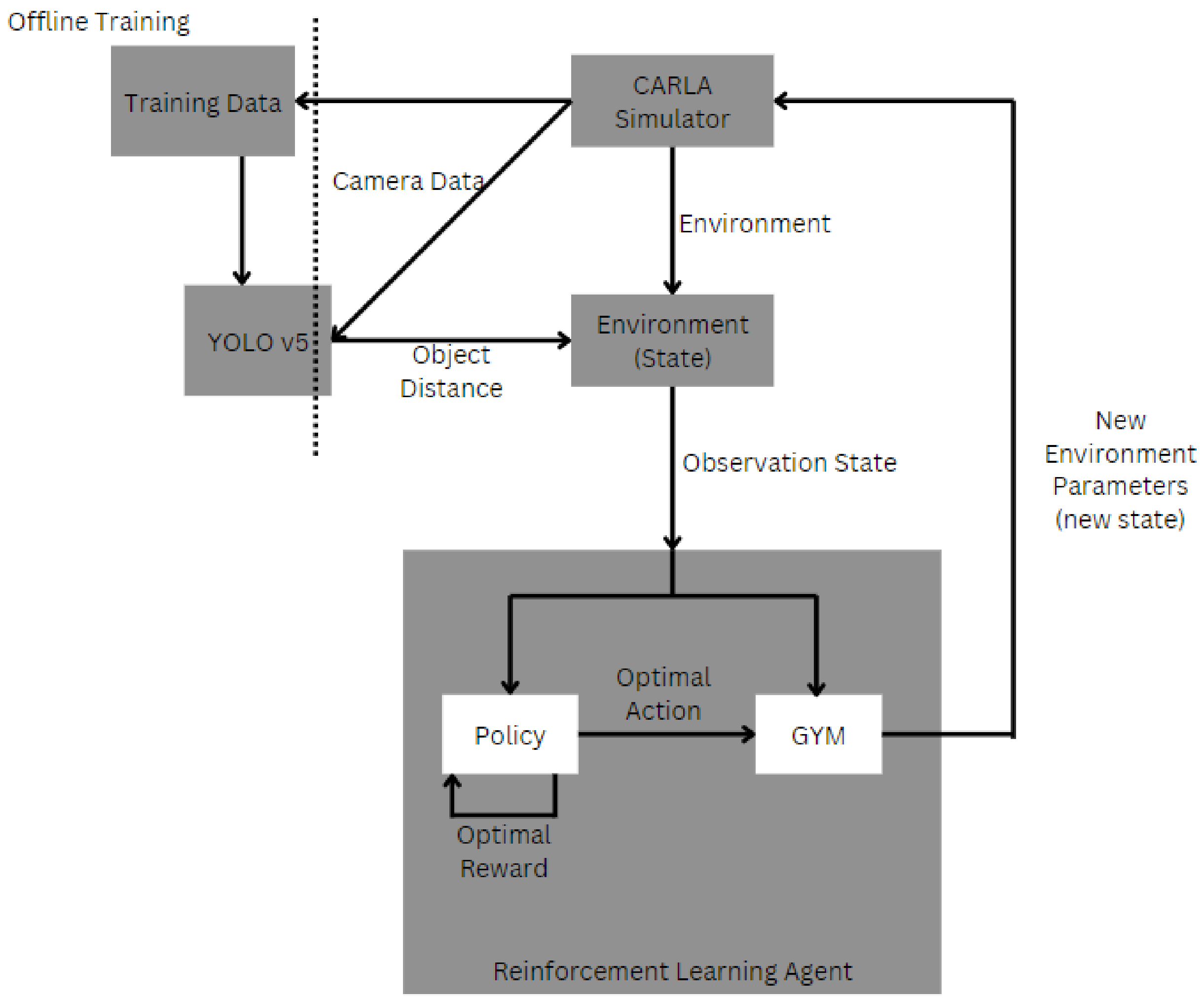


Figure 8 System Block Diagram
The YOLO v5 system was trained offline using images generated directly from CARLA. The hyperparameters produced after training are used to identify objects when training the RL agent. The observation space is made up of information from CARLA and image classification. This observation space is what the agent uses in the policy to choose the best action. This is sent through an API known as GYM which generates a new environment in CARLA.

Implementing YOLO: Details & Challenges

YOLO is a popular algorithm that many researchers have implemented. Knowing this, we searched for a code that uses YOLO within CARLA. After finding what seemed to be the perfect fit, the main challenge was fixing a corrupted output. Other small challenges included getting the packages and dependencies the code uses to work properly. After finally fixing the issues, the next step was to train the YOLO with CARLA data. With little time left, a pre-made dataset was used to train it. YOLO worked within CARLA after training with this dataset. However, the labels used in this dataset were not ideal for our implementation. This led to relabeling the images and collecting more to retrain YOLO for our purpose. The classes used in the relabeled data set were 'vehicle', 'traffic light red', ‘traffic light yellow’, and ‘traffic light green’. By focusing on these classes, the data collection process was easier and created an output that can be fed to and used by RL agent. The pseudocode for how the YOLO code detects the labels classes in the simulation can be found in Figure 9.

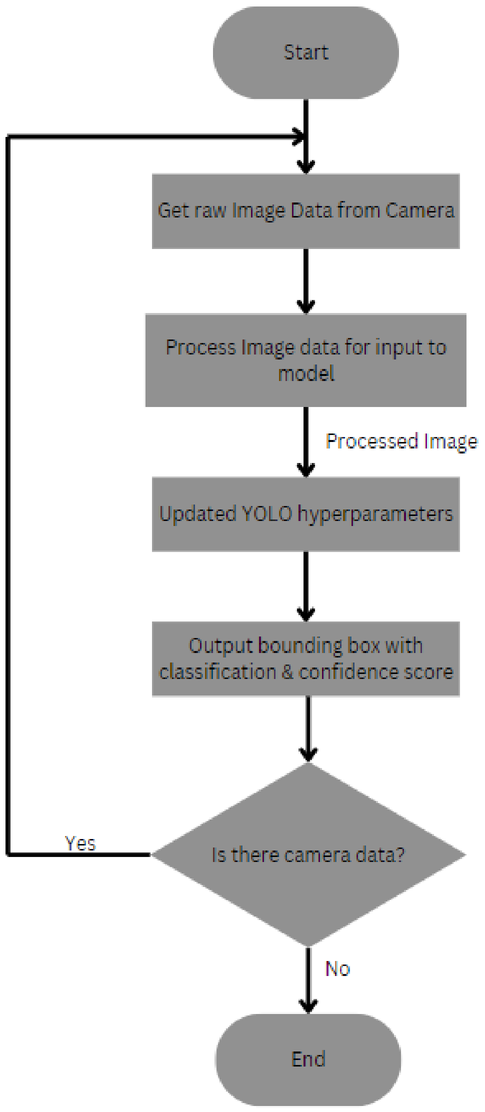

Start
Get raw Image Data from Camera
Process Image data for input to model
Processed Image
Updated YOLO hyperparameters
Output bounding box with classification & confidence score
Is there camera data?
Yes
No
End

Figure 9 YOLO Online Pseudocode
The model gets raw data from the camera in CARLA. This data is processed to prepare it for input to the model. The processed image is sent through the hyperparameters that were generated after training the model. From here the objects in the image are classified with a confidence score and a bounding box.

RL

Implementing an RL agent is a challenging task, even after learning the underlying concepts. To gain insight into the intricacies of RL code, our team initially examined simple, pre-made codes such as Lunar Lander [9]. These codes provided insight into how the critical components are written such the observation space, the state representation, the A2C algorithm, environment construction, and the various functions associated with the GYM library. As we progressed, we authored custom code for a specialized environment and algorithm. Once we had a solid grasp of the complexities involved in RL framework implementation, we transitioned to working with the more challenging environment: the CARLA simulator.

While we had a strong knowledge base, bridging the gap between GYM and CARLA remained uncharted territory. Consequently, we embarked on a quest for a codebase that seamlessly integrated the two platforms. After identifying a promising codebase, the next hurdle was making it functional. This endeavor involved installing numerous dependencies and packages, a process that took several weeks. However, our persistence paid off, and we successfully got the code running.

Once the code was running, we encountered a perplexing issue: the RL agent showed no signs of improvement. Delving deeper, we realized that the functional code served as the foundation for implementing RL with the CARLA simulator. While other codes accompanied by published research papers had implemented various algorithms, these codes were accompanied with its own hurdles that we did not have time for. Having a solid understanding of the core components from the pre-existing codes, we crafted a custom training script. Through this script, the agent embarked on its learning journey. Subsequently, fine-tuning hyperparameters became essential to enhance the agent’s performance. The pseudocode for the RL agent is depicted in Figure 10.

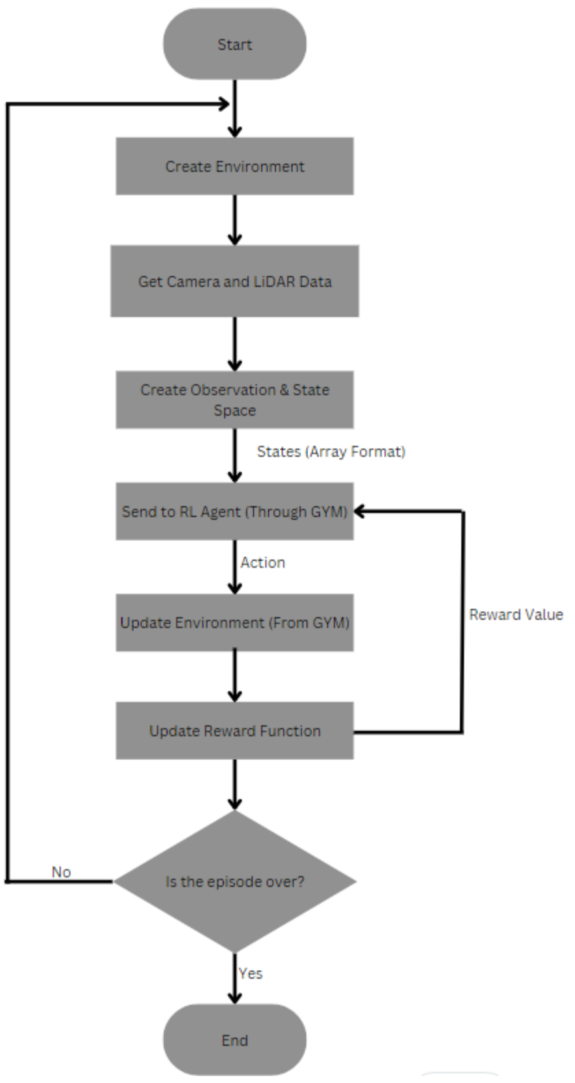

Start
Create Environment
Get Camera and LiDAR Data
Create Observation & State Space
States (Array Format)
Send to RL Agent (Through GYM)
Action
Update Environment (From GYM)
Reward Value
Update Reward Function
Is the episode over?
No
Yes
End

Figure 10 RL Pseudocode
The environment is first created/initialized. Then the sensor data is collected and used to create the observation space. This observation space is sent to the agent where it provides an action. The environment is updated and the reward is calculated. This reward value is sent to the agent to continuously improve the reward. This loop continues until the episode is terminated.

Results

By intricately merging the RL agent within the CARLA environment, we've ensured the agent's ability to act and accrue rewards effectively, a testament to the system's coherent integration. The efficiency of our RL model hinges on detailed training, comprehensive evaluation, and the strategic tuning of hyperparameters. Throughout the iterative process of agent-environment interaction, a range of outcomes surfaced. These outcomes were documented and analyzed, with the results graphed as shown in Figure 12 and Figure 13.

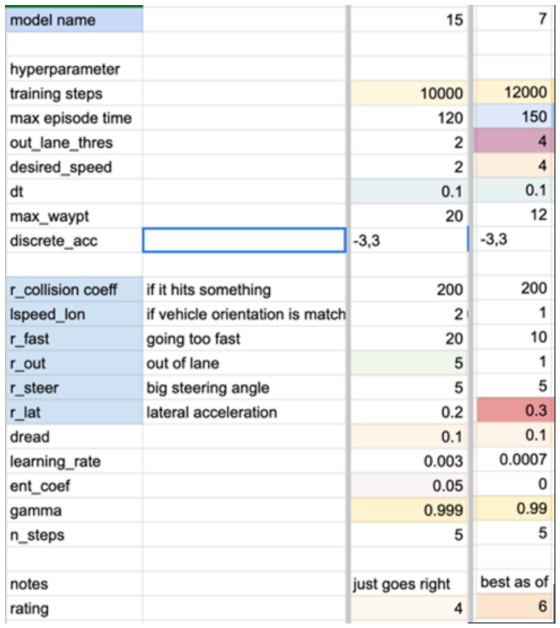
Among the 20 experimental runs, each with its different hyperparameter adjustments, we observed varying degrees of success. Analyzing the hyperparameters shown in Figure 11 from run 15 and run 7 allows us to draw conclusions about their influence on the agent's performance within the CARLA environment. Below are descriptions of each hyperparameter.

- **Training Steps:** Dictates the number of iterations the agent undergoes during the learning process.
- **Max Episode Time:** Sets the upper limit on the duration for each simulation episode, influencing the depth of experience the agent can accrue in a single run.
- **Out of Lane Threshold (out_lane_thres):** Controls the agent's tolerance for deviating from the lane, impacting lane-keeping precision.
- **Desired Speed:** Determines the target velocity the agent aims to maintain, crucial for aligning with speed limits and traffic flow.
- **Time Step (dt):** Establishes the granularity of time intervals for the agent's decision-making, affecting the resolution of actions.
- **Maximum Waypoint (max_waypt):** Sets the number of waypoints the agent considers for path planning, influencing navigation complexity.
- **Discrete Acceleration (discrete_acc):** Adjusts the increments of acceleration or deceleration the agent can apply, affecting driving smoothness.
- **Reward Parameters (r_collision, r_speed_lon, r_fast, r_out, r_steer, r_lat):** These critical parameters shape the agent's learning by providing incentives for specific behaviors and disincentives for others, such as collisions or excessive speed.
- **Dread:** A penalty applied for undesirable states, discouraging the agent from certain actions.
- **Learning Rate:** Affects how significantly the agent's neural network weights are updated during training.
- **Entropy Coefficient (ent_coef):** Balances the exploration-exploitation trade-off in the agent's policy development.
- **Discount Factor (gamma):** Reflects the importance the agent places on future rewards versus immediate gains.
- **Number of Steps (n_steps):** Determines how many future steps the agent takes into account when planning its actions.

When comparing the effects of hyperparameters on runs 7 and 15, we observe that the right tuning can profoundly affect the agent's learning trajectory and its operational competence. With more training steps, an agent may develop a more comprehensive understanding of the environment, as seen in the more successful run. Similarly, a lower out of lane threshold ensures strict adherence to lane discipline, a crucial aspect of autonomous driving. A well-chosen desired speed enables the agent to match the pace of simulated traffic, essential for realistic navigation. Time step intervals and the number of considered waypoints directly influence the agent's ability to make nuanced decisions and navigate complex paths, respectively.

Reward parameters are pivotal; they incentivize the agent towards favorable behaviors, such as maintaining safe speeds and staying within lanes, while penalizing negative actions like collisions. The adjustment of these parameters between runs highlights the intricate calibration required to fine-tune agent behavior. Learning rate and entropy coefficient adjustments impact how quickly the agent learns from its experiences and how much it explores new strategies, which could explain the varied success between runs. The discount factor and the number of steps encapsulate the agent's strategic foresight, with more emphasis on future rewards guiding it towards long-term success.

Overall, hyperparameter tuning is a delicate and decisive process in reinforcement learning that can make or break an autonomous agent's ability to navigate complex environments effectively. The interplay between these parameters shapes the very essence of the agent's driving strategy within the dynamic landscape of the CARLA simulator.

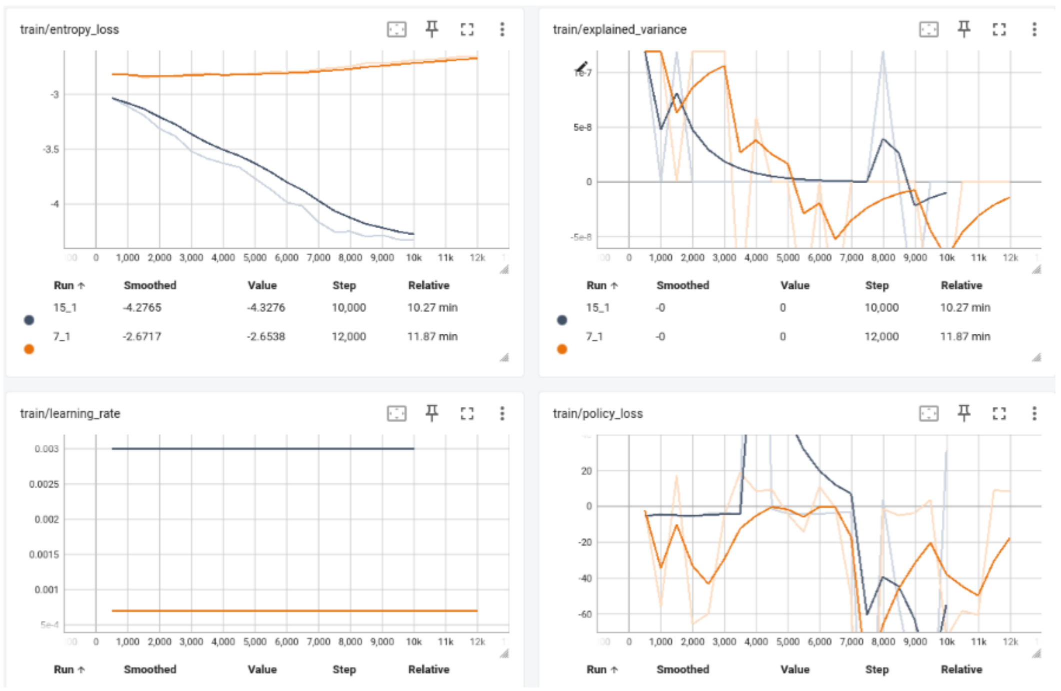


model name		15	7
hyperparameter			
training steps		10000	12000
max episode time		120	150
out_lane_thres		2	4
desired_speed		2	4
dt		0.1	0.1
max_waypt		20	12
discrete_acc		-3,3	-3,3
r_collision coeff	if it hits something	200	200
lspeed_lon	if vehicle orientation is match	2	1
r_fast	going too fast	20	10
r_out	out of lane	5	1
r_steer	big steering angle	5	5
r_lat	lateral acceleration	0.2	0.3
dread		0.1	0.1
learning_rate		0.003	0.0007
ent_coef		0.05	0
gamma		0.999	0.99
n_steps		5	5
notes		just goes right	best as of
rating		4	6

Figure 11 Hyperparameters

Figure 12 showcases key metrics from two distinct runs of our reinforcement learning model, illustrating the contrasting outcomes based on the tuning of hyperparameters. Run 7, represented in orange, indicates the 'good' run, while run 15, in black, denotes the 'bad' run. Below, we discuss the performance metrics used in these results.

- **Entropy Loss:** Entropy loss measures the unpredictability of the agent's actions. In run 7, we observe a lower and stable entropy loss, implying a more predictable set of actions as training progresses, which is a hallmark of a well-converging policy. In contrast, run 15 shows higher entropy loss, suggesting the agent's policy remained uncertain and less optimized.
- **Explained Variance:** The explained variance graph shows how well the model's predictions are correlating with the actual rewards. A higher explained variance in run 7 indicates that the agent's predictions are in close alignment with the outcomes, which is preferable. The lower or negative values in run 15 suggest the model's predictions were often inaccurate, which can lead to ineffective learning.
- **Learning Rate:** The learning rate, which remained constant across both runs, influences the magnitude of updates to the agent's neural network weights. Despite being consistent, the learning rate interacts with other hyperparameters, affecting each run's outcome.
- **Policy Loss:** The policy loss graph reflects the agent's progress in learning the optimal policy. In run 7, we see a general downward trend, which points to the agent learning a more effective policy over time. The fluctuations in run 15, however, indicate instability in learning, which can be attributed to the agent's inability to consistently predict rewards from its actions.



Run ↑	Smoothed	Value	Step	Relative
● 15_1	-4.2765	-4.3276	10,000	10.27 min
● 7_1	-2.6717	-2.6538	12,000	11.87 min

Run ↑	Smoothed	Value	Step	Relative
● 15_1	-0	0	10,000	10.27 min
● 7_1	-0	0	12,000	11.87 min

Run ↑	Smoothed	Value	Step	Relative

Run ↑	Smoothed	Value	Step	Relative

Figure 12

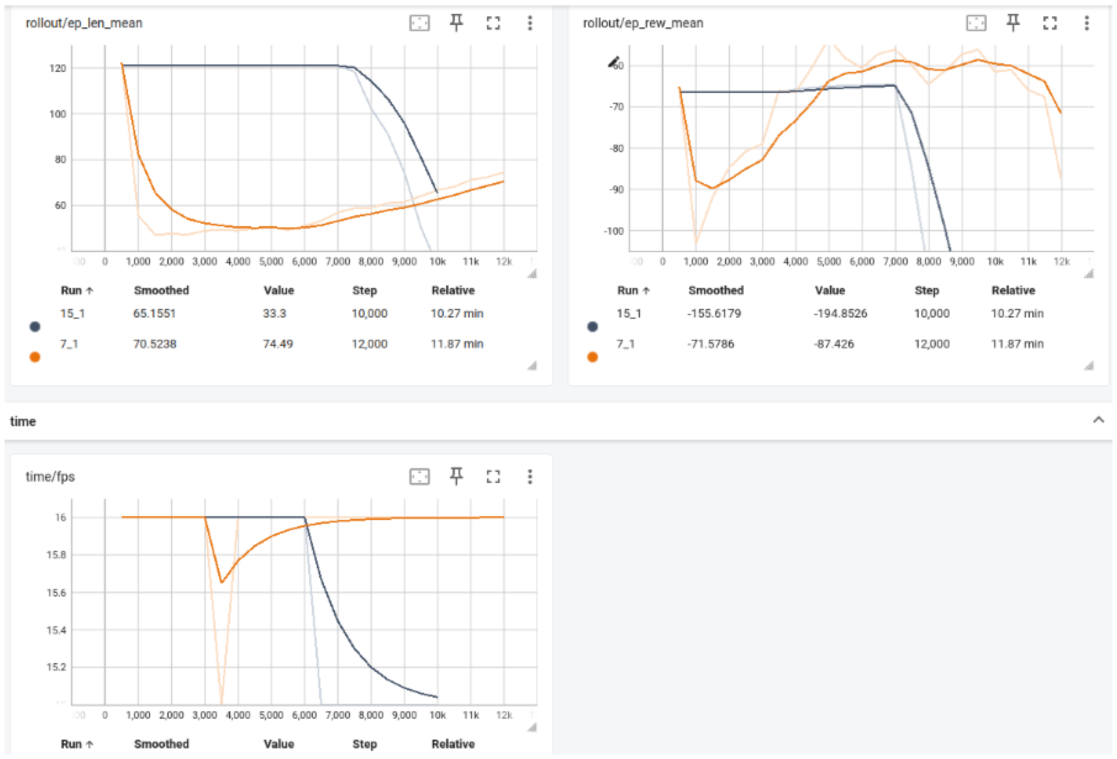
Run 7 is the good run represented as orange. Run 15 is the bad run represented as black.

Figure 13 provides us with further insights into the performance of the reinforcement learning model across two separate runs. Below, we discuss the performance metrics used in these results.

- **Episode Length Mean:** The top-left graph shows the mean episode length over time. For run 7, we see a gradual increase and then a stabilization, which suggests the agent is learning to sustain longer episodes, indicative of better performance in the simulated environment. Conversely, run 15 displays a sharp decline in episode length, signaling that the agent may be failing to complete its objectives as efficiently over time.
- **Episode Reward Mean:** The top-right graph plots the mean reward per episode. Run 7's consistently higher reward mean indicates that the agent is successfully achieving its objectives and maximizing rewards throughout the episodes. The erratic and generally lower values in run 15 suggest the agent is struggling to perform tasks that would yield higher rewards, denoting a less successful learning outcome.
- **Frames Per Second (time/fps):** The bottom graph depicts the simulation's frames per second over time. For run 7, the frame rate remains relatively stable, indicating that the computational load is well-managed throughout the training process. Run 15, however, shows volatility in the frame rate, which could be due to various factors including the agent's decisions leading to more computationally intensive scenarios or less optimal interaction with the simulation environment.

The results from runs 7 and 15 starkly differ. Run 7 shows a well-tuned agent, achieving a steady policy learning and high reward consistency, excelling in the CARLA environment. In contrast, run 15 displays erratic learning and lower performance, indicated by volatile policy loss and rewards, leading to suboptimal navigation and shorter episodes. When viewed collectively, these outcomes not only underscore the effective hyperparameter tuning inherent to run 7, but also the importance of continuous fine-tuning to enhance the reinforcement learning model’s performance. The disparities between the runs highlight the nuanced relationship between hyperparameter settings and agent behavior, informing the crucial balance of exploration, exploitation, and adaptability required for sophisticated autonomous navigation.

Our RL benchmarking, tailored for our unique autonomous vehicle model in CARLA, diverges from conventional standards. We have created an internal benchmark based on comparative analysis of varied runs, fine-tuning hyperparameters to evaluate different learning outcomes. This introspective approach ensures progress is measured by the agent's adaptability and learning within our system, aligning with the project's specific objectives and complexities.



rollout/ep_len_mean

	Run ↑	Smoothed	Value	Step	Relative
●	15_1	65.1551	33.3	10,000	10.27 min
●	7_1	70.5238	74.49	12,000	11.87 min

rollout/ep_rew_mean

	Run ↑	Smoothed	Value	Step	Relative
●	15_1	-155.6179	-194.8526	10,000	10.27 min
●	7_1	-71.5786	-87.426	12,000	11.87 min

time/fps

Run ↑	Smoothed	Value	Step	Relative

Figure 13

run 7 is the good run represented as orange. run 15 is the bad run represented as black.

YOLO's object detection within the CARLA simulation showcases its adept capability to identify and classify critical elements such as vehicles and traffic lights with notable accuracy as shown in Figure 14. YOLO excels in classifying objects with impressive accuracy, enclosing them within bounding boxes highlighted with confidence scores. The unwavering consistency in identifying traffic lights—regardless of their color state—highlights the system's reliability, crucial for its application in the real world. The system's precision in discerning vehicles, denoted by blue boxes with confidence scores as high as 0.81, illustrates its proficiency in detecting correct objects with a high degree of certainty. Additionally, traffic lights are not only detected but are also accurately categorized based on their operational status, with scores like 0.72 for red lights, emphasizing the system's capability to perceive and interpret critical traffic signals.

This skill in detecting and differentiating diverse elements within the complex simulation environment showcases YOLO's significant role in autonomous vehicle systems. These systems depend on precise, real-time information to make decisions that ensure safety and efficiency. YOLO's robust performance under various simulated conditions affirms its readiness for real-world challenges, positioning it as an indispensable component for object recognition in autonomous driving applications. The demonstration of YOLO's effectiveness in a controlled simulation, as shown in Figure 14, also suggests its potential for deployment in unpredictable real-world settings, where precise and real-time detection is paramount.

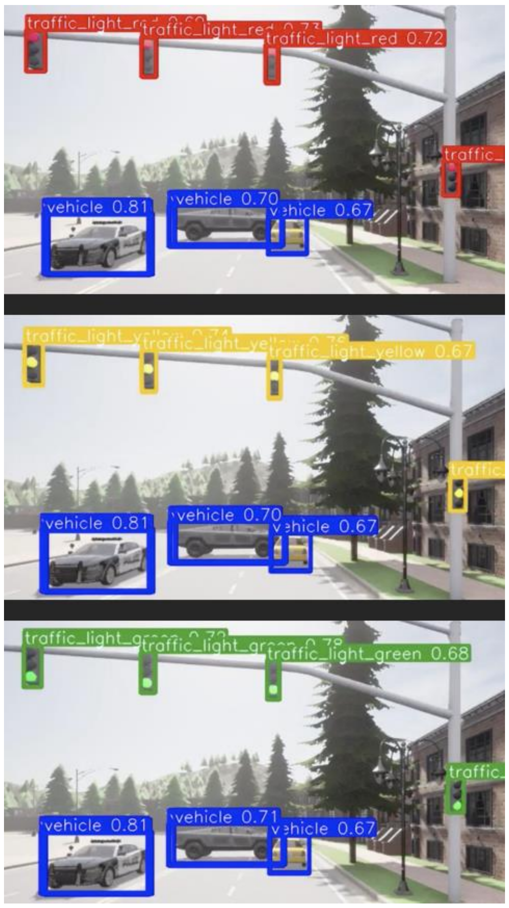
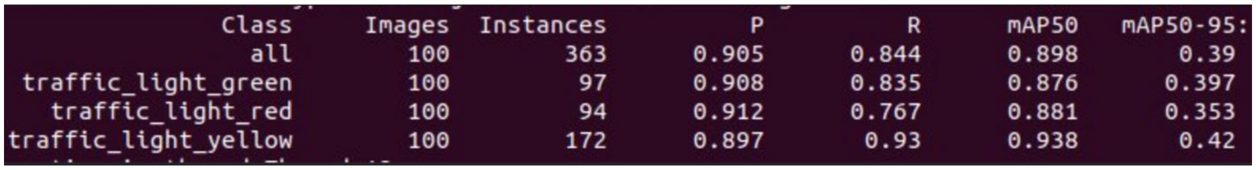


Figure 14 YOLO's Output in Carla

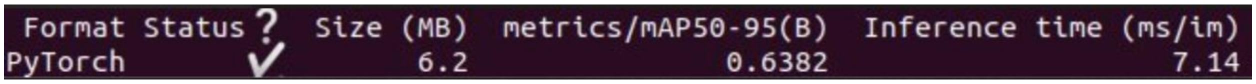
When assessing the efficacy of object detection models, it is imperative to understand the metrics used to evaluate their performance. Mean Average Precision (mAP) is a crucial metric in this domain. Specifically, the mAP50-95 refers to the mean average precision calculated at different Intersection over Union (IoU) thresholds, ranging from 0.5 (more lenient) to 0.95 (more stringent). IoU is a measure of overlap between the predicted bounding box and the ground truth, with a higher IoU indicating greater accuracy in the prediction. Precision, in this context, measures the proportion of positive identifications that were actually correct, meaning it assesses the model's ability to return more relevant results rather than irrelevant ones. Recall, on the other hand, measures the proportion of actual positives that were correctly identified, gauging the model's capability to find all relevant instances in the dataset.

In our comparative study, we find that the YOLOv5 model we trained achieves a mAP50-95 score of 0.39, demonstrating its competence in object detection tasks within the CARLA simulation environment. This score is depicted in Figure 15. Conversely, the YOLOv8 dataset, as shown in Figure 16, achieves a higher mAP50-95 score of 0.6382, indicating a more precise detection capability across a broader range of IoU thresholds, from generous to precise overlaps. It is important to note that the volume of data used for training can significantly impact these scores. Our model, trained on 2,376 images, naturally presents different performance metrics than the YOLOv8 model, which was trained on an extensive dataset of 380,000 images. This vast difference in the quantity of training data can contribute to the disparity observed in the mAP50-95 scores, where a larger dataset can lead to a model that generalizes better to new data, hence potentially higher mAP scores.



```
             Class     Images  Instances          P          R      mAP50   mAP50-95:
               all        100        363      0.905      0.844      0.898       0.39
 traffic_light_green      100         97      0.908      0.835      0.876      0.397
   traffic_light_red      100         94      0.912      0.767      0.881      0.353
traffic_light_yellow      100        172      0.897       0.93      0.938       0.42
```

Figure 15 YOLOv5 Dataset



```
 Format Status ?  Size (MB)  metrics/mAP50-95(B)  Inference time (ms/im)
PyTorch       ✓        6.2               0.6382                    7.14
```

Figure 16 YOLOv8 Dataset

While the two models do not work together, the results achieved for each model are better than expected. With more time, the next steps discussed below would have been the next focus.

Future Directions

Integration of YOLO and RL

Future directions for this area of research would include (1) integrating YOLO and RL in the same system and (2) investigating the use of multi-modal large language models (MM-LLMs) in the self-driving problem.

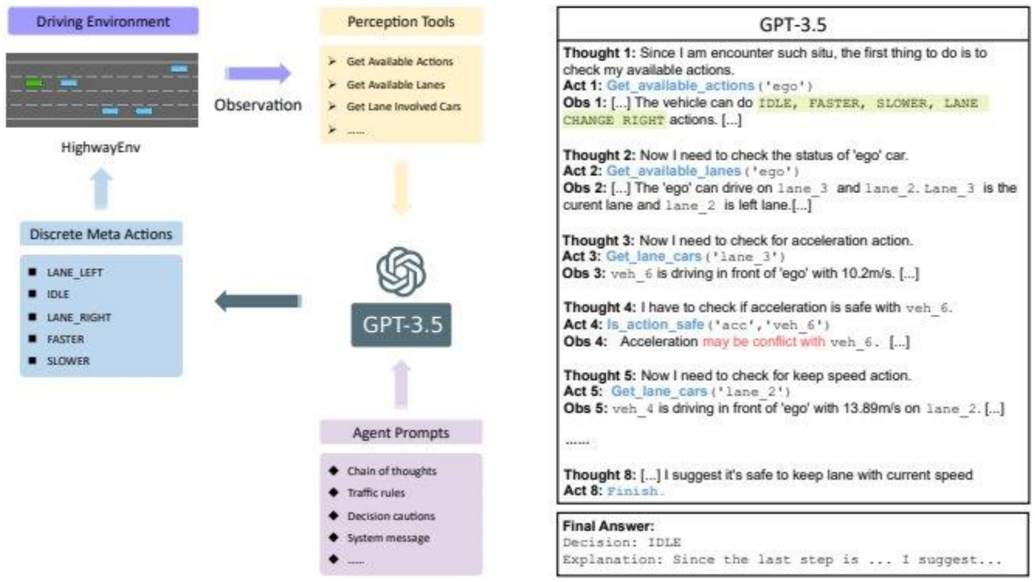
To enrich the data given to the RL agent, the output of YOLO could be fed into the agent as part of its observation space. By granting the model access to higher-level observations—such as identifying pedestrians or stop signs—the RL agent gains the ability to make more nuanced decisions. For instance, incorporating information about the proximity of the ego vehicle to pedestrians or stop signs into the reward function could lead to improved decision-making

MM-LLMs

Another interesting and emerging area in the self-driving problem is the integration of multi-modal large language models (MM-LLMs) [10]. LLMs are proving to be very useful in several areas of machine intelligence and could be useful in AV research as well. Their seeming ability to understand and explain could help alleviate several problems encountered in RL.

For instance, it is often difficult to understand *why* an RL agent is doing what it is doing. Because of this, attempting to mend bad behavior can prove difficult. MM-LLMs, however, offer an opportunity for the model to explain its reasoning. In Figure 17, the authors take a relatively simple Gym environment, HighwayEnv, and connect it to GPT-3.5 [10]. With the wrappers they have implemented, the agent is able to explain its process in a conversational way. This type of connectivity could allow for better understanding of RL agents' behaviors, and allow practitioners a clearer window into how to best tune a model for the task at hand.

Increasing LLMs' reasoning ability is a very active area of research and applying that increasing reasoning ability to the self-driving problem could prove fruitful. This is especially true when it comes to "edge" or "corner cases." On the real road, there are an infinite variety of scenarios an agent could face. In training, an agent will never come across all of them and will never be able to find all observation-action pairs. However, if the agent can reason, it could approach these scenarios with better "one-shot" success.



GPT-3.5

Thought 1: Since I am encounter such situ, the first thing to do is to check my available actions.
Act 1: Get_available_actions `('ego')`
Obs 1: [...] The vehicle can do `IDLE, FASTER, SLOWER, LANE CHANGE RIGHT` actions. [...]

Thought 2: Now I need to check the status of 'ego' car.
Act 2: Get_available_lanes `('ego')`
Obs 2: [...] The 'ego' can drive on `lane_3` and `lane_2`. `Lane_3` is the curent lane and `lane_2` is left lane.[...]

Thought 3: Now I need to check for acceleration action.
Act 3: Get_lane_cars `('lane_3')`
Obs 3: `veh_6` is driving in front of 'ego' with 10.2m/s. [...]

Thought 4: I have to check if acceleration is safe with `veh_6`.
Act 4: Is_action_safe `('acc','veh_6')`
Obs 4: Acceleration may be conflict with `veh_6`. [...]

Thought 5: Now I need to check for keep speed action.
Act 5: Get_lane_cars `('lane_2')`
Obs 5: `veh_4` is driving in front of 'ego' with 13.89m/s on `lane_2`. [...]

......

Thought 8: [...] I suggest it's safe to keep lane with current speed
Act 8: `Finish.`

Final Answer:
```
Decision: IDLE
Explanation: Since the last step is ... I suggest...
```

Figure 17
MM-LLMs, with their seeming ability to reason and explain, are a promising avenue for the self-driving problem.

Standards

The Institute for Electrical and Electronics Engineers' (IEEE) Standards Association has several standards related to automated driving systems that either have been established or are in review to be incorporated. IEEE Std 2846-2022 is an active standard depicted in the article, *IEEE Standard for Assumptions in Safety-Related Models for Automated Driving Systems* [12]. This standard accounts for the futility of over-accounting for safety, where a balance between safe and efficient driving is necessary. It is represented through clauses that create a safety model to both ensure safety based on valid assumptions and tested attributes. In addition to this standard there are two Project Authorization Requests (PAR), P2846a and P3321, related to expanding the base assumptions to include some of the more complex scenarios that may arise. Our project kept safety in the forefront by using a simulator. Reinforcement learning inherently has risk as it is learning to properly imitate a responsible driver, where the simulator seems like a necessary step for the early stages of training. Future stages of reinforcement learning can be formed with a real car in a controlled environment. YOLO also has risk in terms of mislabeling or missing objects in an image. We account for that risk through the recall metric. If we were given the opportunity to implement the combination of both sections of our code, we would account for these shortcomings through continuously monitoring the RGB camera as an action is made.

Ethical Considerations

As a team, we used our understanding of ethical concepts to inform our decisions throughout each stage of this project. *The Code of Ethics for Engineers* [8] from The National Society of Professional Engineers (NSPE) uses the six Fundamental Canons as a framework to discuss best practices for the industry. In terms of societal and environmental considerations, the Professional Obligations instructs to "strive to serve the public interest" [8]. This is further broken down to enhance the "wellbeing of their community" and encourage "sustainable development in order to protect the environment" [8].

"Hold paramount the safety, health, and welfare of the public" [8], became important early in the design stages of our project as we researched the industry's use of reinforcement learning. Our initial design involved implementing our code on a prototype autonomous vehicle, the Jackal. After our research, we found that implementing reinforcement learning on a physical vehicle is not a viable option. The industry's best practice involves going through multiple stages of rigorous simulation training prior to any version of physical implementation.

The second, third and fourth cannon are not relevant to our project. "Perform services only in areas of their competence" [8], "Issue public statements only in an objective and truthful manner" [8], and "Act for each employer or client as faithful agents or trustees" [8] are more valuable in real world applications of *the Code of Ethics*. We chose our project with the strengths and weaknesses of our team in mind. There were aspects to our project that we were necessarily unfamiliar with, which we took as opportunities to learn and work with source codes and different packages. This project did not involve public statements, outside of the final presentation, which renders this point moot. Communications with the professor have been interacting with the client in a way that would not be considered public but was still attempted objectively and truthfully. The fourth canon largely focuses on conflicts of interest and wrongful financial incentives which we did not use in our project.

"Avoid deceptive acts" [8], has been important as we considered the validity of our approaches. The simulator we used, CARLA, has an inbuilt self-driving feature that we have considered using to various extents in our project. The original project design emphasized object detection and reacting to those objects, notably ignoring lane following, localization, mapping, and navigation. An early design involved using the inbuilt navigation for lane following and focusing the reinforcement learning code on responding to detections of traffic lights, stop signs, and vehicles. After some team discussions on the legitimacy of using the inbuilt self-driving feature, we decided it would be disingenuous to the spirit of the project to work with elements that would be incapable of being used in the real world. We prioritized implementing a reinforcement learning code that had elements of lane following and navigation satisfied.

"Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession" [8], has been fundamental to our team's interactions. Due to the size of our team being larger, cooperation and teamwork have been invaluable to the project's success. We have worked to reach a consensus throughout our design iterations and ensured every member was familiar with each element of the project. To ensure our project has been done lawfully, we have carefully read and followed the licensure for each source code that we have used. The codes we have used all use the MIT license which allows us to work without the software without restriction other than that there is no warranty associated with the codes. We have included the MIT license in our GitHub for future versioning and redistributions of the code.

References

[1] SAE International, “Taxonomy and Definitions for Terms Repated to Driving Automated Systems for On-Road Motor Vehicle.” SAE Mobilus. https://www.sae.org/standards/content/j3016_202104/ (accessed March 2024).

[2] B. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. Al Sallab, S. Yogamani, P. Perez, “Deep Reinforcement Leanring for Autonomous Driving: A Survey.” arXiv. https://arxiv.org/abs/2002.00444 (accessed November 2023).

[3] B. K., “Object Detection Algorithms and Libraries.” Neptune.ai. https://neptune.ai/blog/object-detection-algorithms-and-libraries (accessed March 2024).

[4] N/A, “Seeing the road ahead.” Waymo. https://waymo.com/intl/es/about/#blog (accessed January 2024).

[5] M. Gupta, “Reinforcement Learning for Beginners.” Medium. https://medium.com/data-science-in-your-pocket/reinforcement-learning-for-beginners-cd8a6e131d04 (accessed November 2023).

[6] J. Rodriguez, “Reinforcement learning Soup: MDPs, Policy vs. Value Learning, Q-Leanring and Deep-Q-Networks.” Medium. https://jrodthoughts.medium.com/reinforcement-learning-soup-mdps-policy-vs-value-learning-q-learning-and-deep-q-networks-4ac137acd07 (accessed January 2024).

[7] N/A. “Deep RL Course Documentation: Two main approaches for solving RL problems.” Huggingface.co. https://huggingface.co/learn/deep-rl-course/en/unit1/two-methods (March 2024).

[8] NSPE Executive Committee, “Code of Ethics for Engineers.” NSPE.org. https://www.nspe.org/sites/default/files/resources/pdfs/Ethics/CodeofEthics/NSPECodeofEthicsforEngineers.pdf (accessed October 2023).

[9] N/A, “An API standard for reinforcement learning with a diverse collection of reference environments.” Gymnasium.farama.org. https://gymnasium.farama.org/ (accessed November 2023).

[10] D. Fu, X. Li, L. Wen, M. Dou, P. Cai, B. Shi, Y. Qiao, “Drive Like a Human: Rethinking Autonomous Driving with Large Language Models.” arXiv. https://arxiv.org/abs/2307.07162 (accessed January 2024).

[11] X. Zhao, “Deep learning based visual perception and decision-making technology for autonomous vehicles.” Researchgate.net. https://www.researchgate.net/publication/377938813_Deep_learning_based_visual_perception_and_decision-making_technology_for_autonomous_vehicles (accessed March 2024).

[12] S. Kim, “IEEE Standards for Assumptions in Safety-Related Models for Automated Driving Systems” IEEE SA https://standards.ieee.org/ieee/2846/10831/

[13] S. Kim. “Standard for Assumptions in Safety-Related Models for Automated Driving Systems Amendment: Additional Scenarios and Road Users” https://standards.ieee.org/ieee/2846a/11079/

[14] S. Kim. “Recommended Practice for the Application of Assumptions on Reasonably Foreseeable Behavior of Other Road Users” https://standards.ieee.org/ieee/3321/11080/

[15] A. Dosovitskiy. “CARLA: An Open Urban Driving Simulator” https://carla.org/

Appendix A

https://github.com/legendairytri/YOLO-RL-CARLA