Uncertainty-awareness in reinforcement learning for vehicle navigation

Peng Huang
Dept of Computer Science
Graduate School of Arts & Sciences, Boston University
Boston, MA
phuang@bu.edu

Parker Dunn
Dept of Electrical and Computer Engineering
College of Engineering, Boston University
Boston, MA
pgdunn@bu.edu

***Abstract*— Teaching a vehicle to avoid collisions requires it to undergo training that involves experiencing collisions. However, such training poses a risk of damage to the vehicle, particularly in high-speed collisions. This presents a significant challenge for deploying reinforcement learning in autonomous driving, as the training process itself can jeopardize the safety of pedestrians, other vehicles, and the ego-vehicle. As a result, ensuring the safety of all parties involved during the training process is a crucial consideration for the successful implementation of reinforcement learning in autonomous driving.**

Our proposed method harnesses the power of uncertainty estimation acquired from prior imitation learning to evaluate and mitigate risks during reinforcement learning training of autonomous driving vehicles. The first stage involves training an end-to-end neural network to drive using imitation learning. In the second stage, we leverage the pre-trained neural network to estimate collision risks through its uncertainty and subsequently integrate the speed-dependent penalty for the risk of collisions into the immediate reward function for reinforcement learning training.

During our experiments, we employed DDPG as the primary reinforcement learning method. The experimental results indicate that incorporating uncertainty evaluation derived from prior imitation learning as collision risk prediction has led to a reduction of 7.64% in the agent collision speed in the simulation environment.

I. Introduction

Autonomous driving is a rapidly developing field that aims to revolutionize transportation by using artificial intelligence and robotics to provide safer and more efficient transportation solutions. Reinforcement learning, a subfield of machine learning, has shown great potential in training autonomous driving agents to make decisions in complex and dynamic environments. However, the training process of such agents poses a significant challenge, as it involves experiencing collisions that can jeopardize the safety of pedestrians, other vehicles, and the ego-vehicle.

The conventional approach to training autonomous driving agents involves the use of reinforcement learning, where the agent learns by interacting with the environment and receiving rewards or penalties for its actions. This approach requires the agent to experience collisions to learn how to avoid them in the future, which poses a risk to both the agent and other parties involved. Therefore, ensuring the safety of all parties involved during the training process is a crucial consideration for the successful implementation of reinforcement learning in autonomous driving.

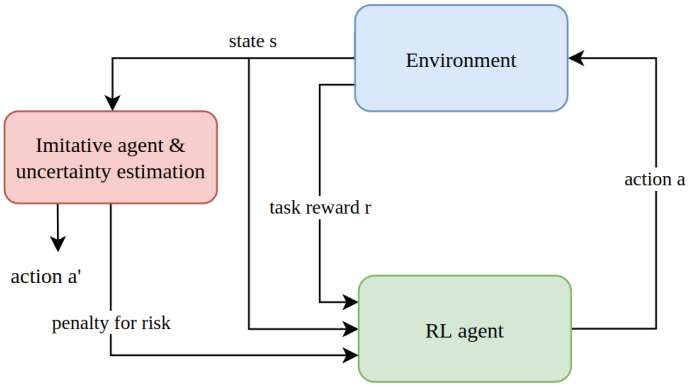


Fig. 1: This paper proposes a novel method that leverages uncertainty estimation acquired from prior imitation learning to evaluate and mitigate risks during reinforcement learning training of autonomous driving vehicles.

To address this challenge, this paper proposes a novel method that leverages uncertainty estimation acquired from prior imitation learning to evaluate and mitigate risks during reinforcement learning training of autonomous driving vehicles, as shown in Figure 1. The proposed method involves training an end-to-end neural network using imitation learning, which involves learning from demonstrations of expert drivers. The pre-trained network is then used to estimate collision risks through its uncertainty, which provides an indication of the level of unconfidence the network has in its predictions. The estimated collision risks are subsequently integrated into the immediate reward function for reinforcement learning training to penalize high-risk decisions.

Imitation learning is capable of training a neural network that can map perceptual inputs to control commands using demonstrations from human experts [1]. However, self-driving neural networks trained solely through imitation learning tend to struggle when presented with off-road conditions and may

fail to recover from offset data [2]. These networks also exhibit higher uncertainty levels in their outputs when confronted with unfamiliar inputs [3]. Evaluating uncertainties can help to identify potential issues and mitigate risks. In an autonomous driving scenario, such unfamiliar states are deemed risky, such as when the vehicle goes off-road. To optimize the policy, we incorporate reinforcement learning in our training process, with prior imitation learning serving as a risk indicator for potential collisions. To the best of our knowledge, our method represents the first successful attempt at merging reinforcement and imitation learning approaches in an uncertain setting.

In this paper, we employ Deep Deterministic Policy Gradient (DDPG) [4] as the primary reinforcement learning method and evaluate the effectiveness of our proposed method in reducing the speed of agent collisions. Our experimental results demonstrate a reduction of 7.64% in the agent collision speed by incorporating uncertainty evaluation derived from prior imitation learning as collision risk prediction. These results highlight the potential of our proposed method to enhance the safety and effectiveness of reinforcement learning in autonomous driving.

II. Related Work

A. Estimating Uncertainties in Neural Network Predictions

Neural networks typically comprise a large number of parameters and non-linear activation functions, making it difficult to compute the posterior distribution of network predictions, especially in the presence of multiple modes [5]. To approximate this distribution, researchers have developed various methods based on Bayesian inference and Monte-Carlo sampling. These methods involve modeling the uncertainty in the network's predictions by estimating the distribution over its parameters. Bayesian inference techniques such as Variational Inference and Markov Chain Monte Carlo (MCMC) are commonly used to approximate this distribution. On the other hand, Monte-Carlo sampling techniques such as Monte Carlo Dropout [6] and Monte Carlo Gradient Estimation [7] are used to estimate epistemic uncertainty in the network's predictions. These techniques enable the network to not only produce accurate predictions but also provide a measure of confidence or uncertainty in those predictions.

Bayesian approaches are a popular method for recovering probabilistic predictions from neural networks. These methods represent the neural network weights through parametric distributions, such as exponential-family distributions [8]–[11]. Consequently, the network's predictions can also be represented using these distributions and computed analytically using non-linear belief networks [9] or graphical models [12]. More recently, Wang et al. [11] proposed natural parameter networks, which model inputs, parameters, nodes, and targets using Gaussian distributions. These approaches can recover uncertainties in a principled way but tend to increase the number of trainable parameters in a super-linear fashion, which limits their impact in real-world applications. Additionally, specific optimization techniques [10] are required for these methods, which can be computationally expensive. Despite these limitations, Bayesian approaches remain a powerful tool for handling uncertainty in neural network predictions.

Dropout [5], [6] is a computationally inexpensive method for improving uncertainty estimates in neural networks. It is commonly used to reduce overfitting during training by randomly dropping units from the network [13]. During training, each unit with dropout is set to 0 with probability p and left at its original value with probability $1-p$. This prevents units from co-adapting too much and overfitting, as different units are sampled for each forward pass, effectively sampling a new, related network during each step of training. By applying dropout, a new randomized version of the network is constructed by sampling independent Bernoulli random variables to act as masks on each neuron. This approach improves uncertainty estimates in neural networks without requiring additional computational resources, making it a popular choice for handling uncertainty in practical applications [14].

B. Combination of IL and RL

Reinforcement learning (RL) and imitation learning (IL) are two popular paradigms in machine learning for enabling agents to learn and execute complex tasks. RL involves learning from interactions with the environment, where an agent receives rewards for taking certain actions and learns to maximize its long-term reward. IL, on the other hand, involves learning from demonstrations provided by an expert, where the agent learns to mimic the expert's behavior.

Several works have explored the integration of RL and IL to improve learning efficiency and performance. For instance, the Guided Cost Learning (GCL) algorithm proposed by Finn et al. [15], which combines IL and RL to learn a cost function that guides the RL agent toward the expert's behavior. GCL was shown to achieve better performance than pure IL and RL in several continuous control tasks.

Furthermore, other works have explored the use of IL to collect demonstration data to improve the performance of RL algorithms. For example, Hester et al. [16] proposed an approach DQfD that uses demonstration data to train the Deep Q-learning neural network. This approach was shown to improve the learning speed and stability in several Atari games. Similarly, Vecerik et al. [17] presented DDPGfD, an algorithm that modifies DDPG to take advantage of demonstrations in a similar way to DQfD. Another notable work was by Sun et al. [18] which proposed a method combining IL and RL through the idea of cost shaping with an expert oracle which can achieve superior performance compared to RL baselines and IL baselines even when the oracle is sub-optimal.

Among all the works combining IL and RL, the most similar work to ours is the framework proposed by Huang et al. [19]. In their method, the first step is using behavioral cloning and uncertainty estimation to derive the policy on the expert's demonstration data. Then in the reinforcement learning step, the imitative expert policy is utilized to guide the learning of the RL agent by regularizing the KL divergence between the RL agent's policy and the imitative expert policy. To maintain the RL agent's exploration capability, they also proposed to

use uncertainty estimation in deriving the expert policy with imitation learning, so that the agent can explore more when the uncertainty of the expert policy is high, meaning the action given by the imitative expert policy is not reliable.

Our approach differs from theirs in that we do not employ direct guidance from the imitative policy network to the RL policy network. This ensures that the RL policy network is not adversely influenced by potentially suboptimal policies from the imitative policy network. By avoiding this direct influence, we allow the RL policy network to learn and explore its own optimal policies through trial and error, rather than being limited to the demonstrated behaviors of the imitative policy network.

Furthermore, their approach suggests that high variance in the action predictions of the imitative policy indicates unreliability and prompts the RL agents to increase exploration. However, in the context of self-driving vehicles, we believe that high variance signifies that the perceptual input to the imitative policy has diverged from the typical data distribution. This divergence often implies potential risks such as collisions or driving off-road, and as a result, the vehicle must proceed with cautious exploration. Thus, in our approach, we prioritize safety by considering high variance as a signal for careful exploration rather than a trigger for increased exploration.

C. Deep Deterministic Policy Gradient

DDPG (Deep Deterministic Policy Gradient) [4] is a reinforcement learning algorithm that is proposed by Lillicrap et al. and designed to solve continuous control problems with high-dimensional state and action spaces. It combines ideas from both policy gradient methods and Q-learning [20] to learn a deterministic policy function that maps states to actions.

The DDPG algorithm consists of two primary components: an actor network and a critic network. The actor network takes the current state as input and outputs an action that is taken by the agent. The critic network takes the current state and action as input and outputs an estimate of the Q-value of the state-action pair.

During training, the agent interacts with the environment and collects experience tuples of the form (state, action, reward, next state) in a replay buffer. The agent then samples a batch of experiences from the replay buffer and uses them to update the actor and critic networks.

The actor network is updated by computing the gradient of the expected return with respect to the policy parameters and using it to update the weights of the actor network. The critic network is updated by minimizing the mean-squared error between the predicted Q-value and the target Q-value, which is computed using the Bellman equation.

One key innovation of DDPG is the use of a target network to estimate the Q-value. The target network is a copy of the critic network that is updated less frequently, which helps to stabilize the learning process and prevent the overestimation of Q-values. Another innovation is the use of a replay buffer, which helps to decorrelate the training data and improve the stability of the learning process.

Overall, DDPG is an effective algorithm for continuous control problems because it can learn a deterministic policy function that maps states to actions, which is important in these types of problems. Its success has led to further research in the area, such as the development of Twin Delayed DDPG (TD3) [21] and Soft Actor-Critic (SAC) [22], which build on the DDPG algorithm to further improve its performance.

III. Methodology

A. Imitation Learning

This section will detail how an imitation learning agent was trained and evaluated with regard to its ability to measure uncertainty in the lateral control of a navigation agent.

Objective To support uncertainty awareness in early phases of RL training, a policy was trained to learn to map input images to steering angle predictions - a proxy for lateral control. A prototypical supervised imitation learning formulation is used for training. Since MC sampling measures model uncertainty during testing, experimentation, and optimization are done in the testing phase too. Various configurations for MC sampling were evaluated on identical data.

Data collection Image-steering pairs, $(I,\ a)$, pairs were collected in CARLA (version 9.14). RGB and segmented images were collected (dimensions 3x640x360 - C, W, H), but only the RGB images were used in practice. The data was collected from Town 01 - 05 in CARLA by selecting a random spawn point and random destination, then allowing an expert agent (provided by CARLA - `BehaviorAgent` https://carla.readthedocs.io/en/0.9.14/adv_agents/) to provide expert demonstrations. The steering angles were collected from the expert agent, which are always in the range $(-1, 1)$. The dataset used for the policies discussed in this paper included over 10,000 sample pairs. The distribution of the steering angles is detailed in Figure 2. This data set is dominated by actions where the steering angle is ≈ 0. To account for this imbalance, the primary variation of this data set used for imitation learning had a balancing policy.

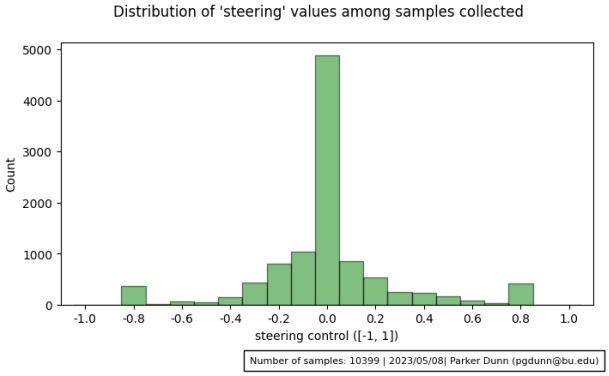


Fig. 2: The distribution of actions among the data set collected from the CARLA simulator.

The balancing policy involved establishing a threshold, $T_{straight}$, that established a classification for steering angles that should be considered "straight" (i.e., ≈ 0). Then, with probability $p_{straight}$, these samples were included in the data set. For all of the work shown in this paper, $T_{straight} \leftarrow \pm 0.05$ and $p_{straight} \leftarrow 0.4$. This means that $\approx 40\%$ of the middle bar in Figure (2) was included in the data set along with all other values.

Imitation learning From the (I,a) data, a policy was trained to learn to perform the mapping $f_\theta : I \longrightarrow a$. While a high-performing IL policy is desirable for this task, since higher accuracy in outputs should lead to better uncertainty estimates, it was not the focus of this approach. Therefore, a slight variation of the ResNet18 architecture was selected and used to train the IL policy. The ResNet structure is modified for the uncertainty estimation task by incorporating a dropout layer after each convolutional block. This is necessary to properly collect MC samples for estimating variance at test time. The problem is formulated as a regression task where the objective is to minimize the Huber loss, L_H.

$$argmin_\theta \sum_{i=1}^{N} L_H(a, f_\theta(I))$$

where L_H is

$$L_\delta(a, f(I)) = \left\{ \begin{array}{ll} \frac{1}{2}(a - f(I))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta * (|a - f(I)| - \frac{1}{2}\delta & o/w \end{array} \right\}$$

Estimating variance To estimate model uncertainty during evaluation, we use MC sampling. This is a straightforward procedure in which the forward pass for each input is repeated multiple times. The outputs for each forward pass are treated as realizations of a random variable, and the expected value and variance for this random variable are calculated as outputs. Therefore, we can return an expected value (or mean) as the prediction for the input and the variance as a representation of the uncertainty of the input. There are two parameters related to this process that were changed and evaluated. First, the probability of dropout during the forward pass, which controls the rate that values are randomly set to zero. This parameter impacts the amount of variance produced by a neural network [6]. Second, the number of forward passes completed (N), i.e., realizations of the random variable, is a parameter that impacts variance and the expected value of the output. In this work, three combinations of these values $p_{dropout}$ and N are considered: $(0.2, 10)$, $(0.3, 20)$, $(0.4, 20)$. See the algorithm below for the full outline of the MC sampling procedure.

Algorithm 1 Monte-Carlo sampling procedure for estimating variance

INPUT: I
$A \leftarrow \emptyset$
for all $i \in \{1, ..., N\}$ **do**
  $A \leftarrow A \cup f_{\theta,i}(I)$
end for
return Mean(A), Variance(A)

B. Reinforcement Learning

This section provides a detailed description of the experimental setup used to train the reinforcement learning (RL) agent using the uncertainty estimation obtained from prior imitation learning as the collision risk prediction.

MDP We adopt the standard Markov Decision Process (MDP) formalism [23] for this experiment. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$ that consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a reward function $R(s, a, s')$, a transition function $P(s'|s, a)$, and a discount factor γ. For our simulation experiment on CARLA, let $\mathcal{I}$ be the set of all possible images obtained from the front camera of the ego-vehicle, and let $\mathcal{V}$ be the set of all possible speeds. The state space S is then given by the Cartesian product of these two sets, i.e. $\mathcal{S} = \mathcal{I} \times \mathcal{V}$. Thus, a state $s \in \mathcal{S}$ can be represented as a tuple (I, v), where $I \in \mathcal{I}$ is an image of shape $C \times H \times W$ and $v \in \mathcal{V}$ is a speed value.

Let $\mathcal{A}_l$ be the set of all possible lateral control actions, and let $\mathcal{A}_L$ be the set of all possible longitudinal control actions. A control action can be viewed as a continuous value, such as a steering angle or throttle/brake value. The action space $\mathcal{A}$ is then given by the Cartesian product of these two sets: $\mathcal{A} = \mathcal{A}_l \times \mathcal{A}_L$. Thus, an action $a \in \mathcal{A}$ can be represented as a tuple (a_l, a_L), where $a_l \in \mathcal{A}_l$ is a lateral control action and $a_L \in \mathcal{A}_L$ is a longitudinal control action.

In each state $s \in \mathcal{S}$, the agent takes an action $a \in \mathcal{A}$. Upon taking this action, the agent receives a reward of $R(s, a)$ and reaches a new state s', determined from the probability distribution $P(s'|s, a)$.

Reward and speed-dependent penalty

Based on the previously defined MDP model, we can now formulate an immediate reward function that will naturally favor slow, cautious exploration in regions of high uncertainty. The reward function $R(s, a)$ that we use has the form:

$$R(s, a) = \lambda_{task} R_{task}(s) - \lambda_{coll} C_{coll}(s) - \lambda_{risk} C_{risk}(s)$$

In the formula, R_{task} is the collision-independent reward for task completion, which might include, for instance, flying to a desired position or in a desired direction. In our experiment, the task reward R_{task} can be gained if the agent arrived at the central waypoints of the road. The task reward depends on the number of waypoints reached, the more the greater.

The collision cost $C_{coll}(s)$ serves as the penalty for a collision, which can be formulated as the product of an indicator for collision $\mathbb{1}_{coll}$ and the square of speed v.

$$C_{coll}(s) = v^2 \mathbb{1}_{coll}(s)$$

The collision cost may penalize high-speed collisions more than relatively harmless low-speed collisions.

As mentioned previously, the risk cost $C_{risk}(s)$ is calculated based on the uncertainty estimation obtained from the variance of output samples generated using dropout to the pre-trained imitative policy network f_θ. More specifically, the uncertainty estimation is used to quantify the probability of collision in a given state s. The risk cost is then defined as a product of this probability and the square of speed v.

$$C_{risk}(s) = v^2 tanh(\sqrt{Var(f_\theta(s))})$$

When the agent is in a high-risk area, it is encouraged to slow down its speed and explore the environment cautiously. This is achieved by incorporating the risk cost into the agent's reward function, which also incentivizes it to take actions that minimize the probability of collision.

To balance the tradeoff between task completion and safety during the agent's decision-making process, the coefficients λ_{task}, λ_{coll}, and λ_{risk} are introduced. These coefficients determine the relative importance of task completion, collision avoidance, and risk management objectives in the agent's overall reward function. By adjusting these coefficients, the agent's behavior can be fine-tuned to prioritize safety over task completion or vice versa, depending on the specific application and requirements.

DDPG

The goal of the agent vehicle is to find the policy π mapping states to actions that maximize the expected discounted total reward over the agent's lifetime. This concept is formalized by the action value function:

$$Q^\pi = \mathbb{E}^\pi[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)]$$

where $\mathbb{E}^\pi$ is the expectation over the distribution of the admissible trajectories $(s_0, a_0, s_1, a_1, ...)$ obtained by executing the policy π starting from $s_0 = s$ and $a_0 = a$. Here we are interested in continuous control problems, i.e. the lateral control and longitudinal control are continuous positive real values, and take an actor-critic approach in which both components are represented using neural networks. These methods consist of maximizing a mean value

$$J(\theta) = \mathbb{E}_{s\sim\mu}[Q^{\pi(.|\theta)}(s, \pi(s|\theta)]$$

with respect to parameters θ that parameterise the policy π and where μ is an initial state distribution. To do so, a gradient approach is considered, and the parameters θ and updated as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

DDPG is an actor-critic algorithm that directly uses the gradient of the Q-function with respect to the action to train the policy π. DDPG maintains a parameterized policy network $\pi(.|\theta^\pi)$ (actor-function) and a parameterized action-value function network (critic function) $Q(.|\theta^Q)$. It produces new transitions $e = (s, a, r = R(s, a), s' \sim P(.|s, a))$ by acting according to $a = \pi(s|\theta^\pi) + \epsilon$ where $\epsilon \sim \mathcal{N}$ is a random process allowing action exploration. Those transitions are added to a replay buffer B. To update the action-value network, a one-step off-policy evaluation is used and consists of minimizing the following loss:

$$L_1(\theta^Q) = \mathbb{E}_{(s,a,r,s')\sim D}[R_1 - Q(s, a|\theta^Q)]^2$$

where D is a distribution over transitions $e = (s, a, r = R(s, a), s' \sim P(.|s, a))$ contained in a replay buffer and the one-step return R_1 is defined as $R_1 = r + \gamma Q'(s', \pi'(s')|\theta^{\pi'})$.

Here $Q'(.|\theta^{Q'})$ and $\pi'(.|\theta^{\pi'})$ are the associated target networks of $Q(.|\theta^Q)$ and $\pi(.|\theta^\pi)$ which stabilizes the learning (updated every N') steps to the values of their associated networks). To update the policy network, a gradient step is taken with respect to:

$$\nabla_{\theta^\pi} J(\theta^\pi) \approx \mathbb{E}_{(s,a)\sim D}[\nabla_a Q(s, a|\theta^Q)_{|a=\pi(s|\theta^Q)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)]$$

The pseudocode of DDPG is presented in the Algorithm 2.

Algorithm 2 DDPG

Input: initial policy parameters θ^π, Q-function parameters θ^Q, empty replay buffer B
Set target parameters equal to main parameters $\theta^{\pi'} \leftarrow \theta^\pi$, $\theta^{Q'} \leftarrow \theta^Q$
repeat
  Observe state s and select action $a = clip(\pi(s) + \epsilon, a_{low}, a_{high})$, where $\epsilon \sim \mathcal{N}$ Execute a in the environment
  Observe next state s', reward r and done signal d to indicate whether s' in terminal
  Store (s, a, r, s', d) in replay buffer B
  If s' is terminal, reset environment state
  if it's time to update **then**
    for however many updates **do**
      Randomly sample ab batch of transitions, $b = (s, a, r, s', d)$ from B
      Compute targets $R_1 = r + \gamma Q'(s', \pi'(s')|\theta^{\pi'})$.
      Update Q-function by one step of gradient descent using

$$\nabla_{\theta^Q} L_1(\theta^Q) = \nabla_{\theta^Q} \mathbb{E}_{(s,a,r,s')\sim b}[R_1 - Q(s, a|\theta^Q)]^2$$

      Update policy by one step of gradient ascent using

$$\nabla_{\theta^\pi} \mathbb{E}_{s\sim b} Q(s, \pi(s)|\theta^Q)$$

      Update target networks with

$$\theta^{\pi'} \leftarrow \rho\theta^{\pi'} + (1-\rho)\theta^\pi$$
$$\theta^{Q'} \leftarrow \rho\theta^{Q'} + (1-\rho)\theta^Q$$

    end for
  end if
until converge

IV. Results

A. Uncertainty from IL Policy

The goal of the IL policy is to provide uncertainty information to aid uncertainty-aware RL. The IL policy is trained to output lateral control, but the variance from MC sampling used to approximate uncertainty is more relevant to the policy's purpose. Since variance is measured from the lateral control outputs, the quality of the outputs is related to the uncertainty estimate as well as the variance calculated from these values. Therefore, we must consider the policy's ability to predict steering angle, then its ability to estimate variance.

Fig. 3: **Demonstration**: Examples of prediction and variance from an IL policy with $p_{drop} = 0.3$ and $N = 20$. These demonstrations show the true steering angle (blue) and predicted/mean steering angle output (red) along with the estimated uncertainty of the steering angle prediction. The *left* image demonstrates an accurate, low uncertainty output in a straightforward navigation scenario. The *right* image demonstrates a demonstration with another vehicle as an obstruction. In this image, we see the uncertainty increase to account for the potential uncertainty created by the other vehicle, while the true lateral control remains within the range of expected values.

Evaluating the IL policy's ability to predict steering angle and variance

The quality of the steering angle and variance outputs from the IL policy were primarily evaluated qualitatively. Ideally, it would have been possible to either compare the IL policy to other approaches for predicting steering angle to compare performance or perform online evaluation of the policy. Either would have given a frame of reference for performance. The only quantitative comparison that is available is from the original work in [5]. However, this offers limited comparison quality because their results are on a real-world data set with a configuration that uses more inputs - e.g., IMU, GPS, brake, throttle and speed.

Qualitative. The best approach with the current status of work is to consider the validation and testing visualizations. These visualizations can be found in the code repository - see "carla_steering_imitation_learning/imitation_learning/logs/" for directories with "training_visualizations" as well as "carla_steering_imitation_learning/imitation_learning/results" for visualizations from evaluation.

Quantitative. In terms of steering angle prediction, performance can only be assessed in terms of loss on validation and test data. The trained policy used for evaluations (identified by directories and files with the name "0011_2023-05-10" achieved an MSE loss of ≈ 0.06 (or RMSE of 0.245) on validation data - all policies with similar performance had an MSE in this same range or worse, and an MSE loss of 0.1596 (or RMSE of 0.399) on the evaluation data. In comparison, the policy in [5], [24] achieved an RMSE of 0.09 on real data with a problem formulation that was likely less flattering since the errors are always on values in the range (-1, 1) for the experiments in our work. It's not a comparison between like experiments yet the policy developed here does not perform well in a more favorable situation.

Using a formulation of negative-log likelihood (NLL) for this type of problem [3], [6], a similar (although not ideal) comparison can be done with uncertainties. NLL is calculated as indicated below to identify accuracy estimation for the uncertainty outputs, where σ is uncertainty and y_{gt} and y_{pred} are the steering angle prediction.

$$NLL = -\frac{1}{2} * log(\sigma) + \frac{1}{2\sigma} * (y_{gt} - y_{pred})^2$$

This formulation of NLL has a trade-off between prediction accuracy and uncertainty. As uncertainty increases, error in the predictions is less impactful. In the ideal situation of low variance and low error, NLL is a small value, even negative for very low uncertainty. For high uncertainty and poor prediction, NLL is large.

In this work, the NLL during the evaluation was consistently greater than 100, while the results in [5] are negative. Although it is not a direct comparison between experiments, the difference in NLL would indicate that the IL policy here does not produce helpful uncertainty estimates or accuracy values. Poor performance indicates low variance when predictions are poor and/or high variance when predictions are accurate from the IL policy.

Optimal configuration for IL policy

In this section, a single policy and data set are used to evaluate the configuration of the policy during evaluation. Here, the objective is accurate and helpful variance/uncertainty estimates that will be helpful for uncertainty-aware RL. Table I contains results from testing with three different configurations.

TABLE I: Comparison of evaluation configurations for MC sampling and their performance in evaluation.

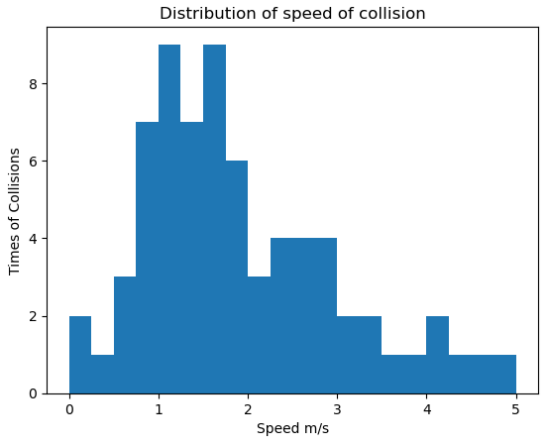
Configuration		**Results**		
$p_{dropout}$	N	NLL (↓)	MSE (↓)	L1 Loss (↓)
0.2	10	7085.3	**0.160**	**0.307**
0.3	20	516.83	0.235	0.373
0.4	20	**190.6**	0.327	0.440

These results indicate better steering angle outputs from low dropout rate configurations. Results also indicate more accuracy in the variance from the higher dropout rate. Since the accuracy was high with the low (N = 10) forward passes, it does not appear that additional forward passes significantly improved accuracy.

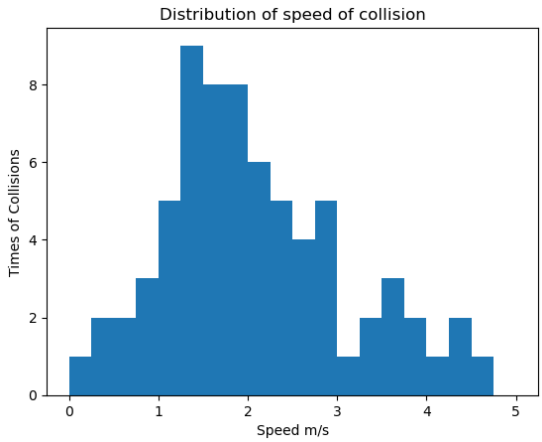
Since the objective is uncertainty estimation, NLL is the primary metric to consider in this case. The results indicate

that higher dropout and samples are beneficial. However, qualitative analysis of results indicates that this high dropout rate is leading to high variance that makes accurate and inaccurate predictions appear to have low likelihood. As a result, the uncertainty estimate, in this case, does not appear to be well correlated with anything meaningful about the model's true uncertainty about the state. Since the results are limited, further exploration and better steering angle prediction are needed to draw helpful conclusions about what configuration should be used to estimate uncertainty. However, we successfully developed a policy that outputs an estimate of uncertainty for lateral control when driving.

B. Experimental results of uncertainty-aware RL



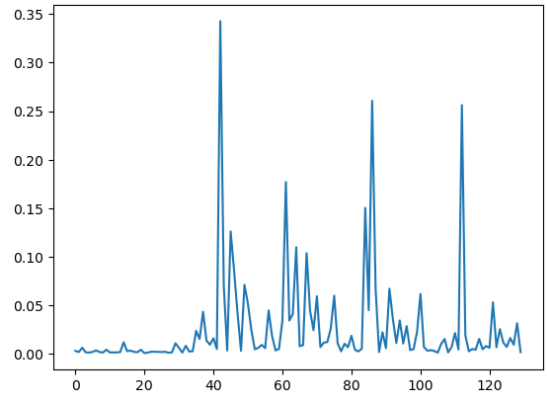
(a) Distribution of collision speed for our method accounting for uncertainty. The average collision speed is 1.932 m/s.



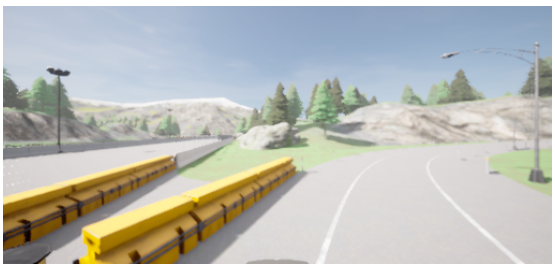
(b) Distribution of collision speed for baseline without accounting for uncertainty. The average collision speed is 2.092 m/s.

Fig. 4: Comparison of the distribution of collision speed. Compared to the baseline, which doesn't account for uncertainty and has an average collision speed of 2.092 m/s, our method has an average collision speed of 1.932 m/s.

Figure 4 illustrates the distribution of the agent's speed during the training process, comparing our method with the baseline method. The purpose of the training is to enable the agent to drive the same distance without collision. The speed distribution indicates that our method results in a decrease in the number of high-speed collisions (3 m/s - 5 m/s) and an increase in the number of low-speed collisions (0 m/s - 2 m/s) compared to the baseline. Additionally, when considering uncertainty as the risk prediction, the average speed of impact decreases from 2.092 m/s to 1.932 m/s, resulting in a reduction of the speed by 7.64%.



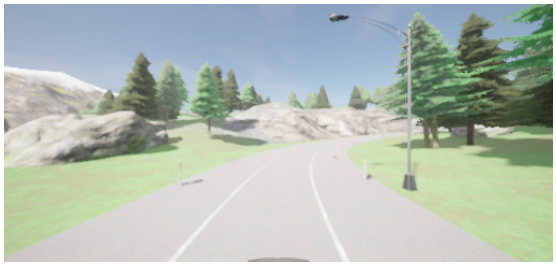
(a) The variance graph as the episode proceeds. The x-axis is the frames within the episode.



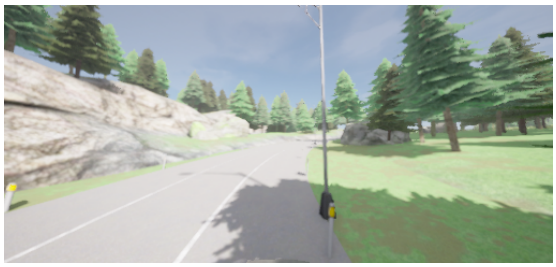
(b) At about the frame 40, the vehicle starts to deviates from its normal path towards the obstacle.

Fig. 5: In an episode, the RL agent deviates from its normal path towards the obstacle from about frame 40. The variance graph shows the increase in the collision risk faced by the agent.

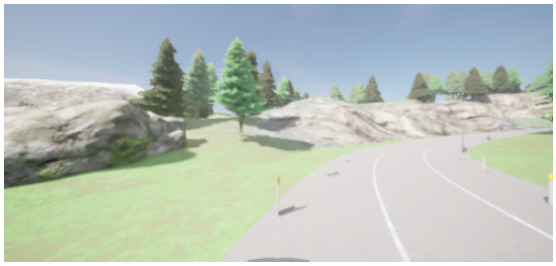
Figure 5 illustrates the obstacle scenario of an agent in a given episode, showing a deviation from its normal path towards an obstacle from approximately frame 40 onwards. The variance graph demonstrates an increase in the collision risk faced by the agent. This example provides evidence that it is possible to utilize uncertainty estimation, which has been learned from prior imitation learning, to predict the likelihood and risk of collision. The use of uncertainty estimation in this way can help improve the safety of an agent by enabling it to make more informed decisions when navigating its environment. Specifically, by identifying situations where the agent's state is likely to result in a collision, it can take corrective action to avoid the obstacle and minimize the risk of a collision.



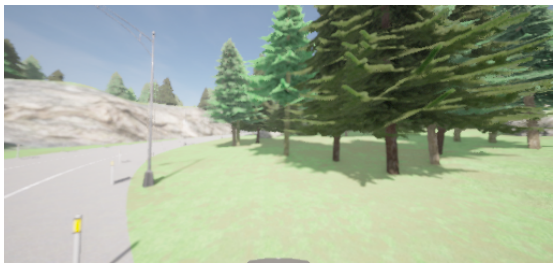
(a) Variance 0.027



(b) Variance 0.048



(c) variance 0.034



(d) variance 0.215

Fig. 6: The above figures present variance values obtained from uncertainty estimation in different states. Specifically, figure 6a represents a normal state with low risk, while figures 6b, 6c, and 6d correspond to states with a higher risk of collision. However, it is concerning that the variance values in figures 6b and 6c, are relatively low and seem to significantly underestimate the actual risk of collision in these states.

While our approach demonstrates promise, the experimental results also highlight its limitations and the potential for further improvement. Despite a 7.64% reduction in crash speed, the improvement is not as significant as initially anticipated. Figure 5 further reveals the limitations of utilizing uncertainty estimation to predict collision probability. For instance, starting from around the 40th frame onward, the agent faces a high risk of collision, yet the variance-based uncertainty estimation sometimes yields a lower value, resulting in an underestimation of the risk. This argument is also supported by Figure 6, where the uncertainty estimation still gives a low variance value in the high-risk state. Moreover, the fact that the variance fluctuates more frequently within a shorter period of time suggests that the accuracy of collision risk prediction via uncertainty estimation is not sufficient, leading to more deviation in the agent's behavior. As such, further refinement and improvement of the uncertainty estimation technique are necessary to enhance its efficacy in ensuring safe and reliable decision-making for agents.

The limitation of our method raises questions about the accuracy and effectiveness of the uncertainty estimation method used, as well as the potential implications for safety and risk management. Further analysis is needed to determine the factors that may be contributing to the underestimation of risk in these states and to identify alternative or supplemental methods for assessing and mitigating risk. Additionally, it may be necessary to consider other sources of information or data, such as sensor data or expert input, to improve the accuracy and reliability of the risk assessment. Overall, this highlights the importance of careful and rigorous risk management in situations where safety is a critical concern.

V. Conclusion

In this paper, we proposed a novel method that utilizes uncertainty estimation derived from prior imitation learning to evaluate and mitigate risks during reinforcement learning training of autonomous driving vehicles. Our experimental results demonstrate that our proposed method can reduce the speed of agent collisions, highlighting the potential of our approach to enhance the safety and effectiveness of reinforcement learning in autonomous driving. However, we also identified limitations in our method, indicating the need for further refinement and improvement in the uncertainty estimation technique to ensure the accuracy and reliability of the risk assessment.

To further improve the effectiveness and accuracy of our proposed method, future work could focus on developing more sophisticated uncertainty estimation techniques that can better capture the level of risk in uncertain states. Another future research could be done on incorporating additional sources of information or data, such as sensor data or expert input, to improve the accuracy and reliability of the risk assessment. It's also promising to conduct real-world experiments to validate the effectiveness and safety of our proposed method in a real-world autonomous driving scenario.

VI. Acknowledgement

We would like to express our sincere gratitude to Professor Eshed Ohn-Bar for imparting his invaluable expertise in robotic navigation and providing instructions for this project throughout the course of this semester. Additionally, we extend our heartfelt appreciation to our dedicated TAs, Jimuyang and Zhongkai, for their invaluable assistance with our project and hard work on grading.

VII. Code

You can access the code and data for this project from Google Drive.

References

[1] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4693–4700, IEEE, 2018.

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

[3] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[5] A. Loquercio, M. Segu, and D. Scaramuzza, "A general framework for uncertainty estimation in deep learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3153–3160, 2020.

[6] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, PMLR, 2016.

[7] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5183–5244, 2020.

[8] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*, pp. 1613–1622, PMLR, 2015.

[9] B. J. Frey and G. E. Hinton, "Variational learning in nonlinear gaussian belief networks," *Neural Computation*, vol. 11, no. 1, pp. 193–213, 1999.

[10] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International conference on machine learning*, pp. 1861–1869, PMLR, 2015.

[11] H. Wang, X. Shi, and D.-Y. Yeung, "Natural-parameter networks: A class of probabilistic neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[12] Q. Su, X. Liao, C. Chen, and L. Carin, "Nonlinear statistical learning with truncated gaussian graphical models," in *International Conference on Machine Learning*, pp. 1948–1957, PMLR, 2016.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[15] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International conference on machine learning*, pp. 49–58, PMLR, 2016.

[16] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[17] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[18] W. Sun, J. A. Bagnell, and B. Boots, "Truncated horizon policy search: Combining reinforcement learning & imitation learning," *arXiv preprint arXiv:1805.11240*, 2018.

[19] Z. Huang, J. Wu, and C. Lv, "Efficient deep reinforcement learning with imitative expert priors for autonomous driving," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[21] S. Dankwa and W. Zheng, "Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent," in *Proceedings of the 3rd international conference on vision, image and signal processing*, pp. 1–5, 2019.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, 2018.

VIII. Appendix

A. Formula for Calculations

1) PyTorch Implementation of Variance:

$$\sigma^2 = \frac{1}{N - \delta N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2 \implies$$

$$\sigma^2 = \frac{1}{N - 1} \sum_{i=0}^{N-1} (x_i - \bar{x})^2$$

In this equation, δN is the "correction" value. The default correction was used, which is $\delta N = 1$, so the equation simplifies to the second representation.