Reinforcement Learning Algorithms for the Risk Sensitive Criterion

Velpula Chaithanya (22940)
Stochastic Systems Lab, IISc Bengaluru
chaithanyav@iisc.ac.in

Guided by ***Prof. Shalabh Bhatnagar***

Abstract

Traditional Reinforcement Learning (RL) algorithms focus on maximizing expected cumulative rewards while ignoring risk, leading to policies that may fail catastrophically in real-world environments with uncertainty. We address this limitation through a **Risk-Sensitive Reinforcement Learning (RSRL)** [1] framework based on exponential utility optimization, which explicitly trades off reward maximization against risk minimization.

We develop risk-sensitive variants of Proximal Policy Optimization (PPO) [2] and Soft Actor-Critic (SAC) [3] that solve a multiplicative Bellman equation via stochastic approximation. Our approach incorporates risk sensitivity directly into the policy update mechanism through an exponential transformation of returns, enabling automatic adaptation to environmental uncertainties. The resulting algorithms maintain the sample efficiency of their risk-neutral counterparts while significantly improving robustness.

For evaluation, we implement our methods in the CARLA autonomous driving simulator, modifying the standard architecture to support risk-sensitive policy learning under varying weather conditions and urban layouts. As demonstrated in our experimental results (see Section 5), the risk-sensitive policies consistently outperform conventional approaches across multiple robustness metrics, showing particular strength in challenging scenarios with environmental perturbations. These findings highlight the importance of risk-aware optimization for safety-critical applications like autonomous driving. The implementation is available at: *code*

1 Introduction

Reinforcement Learning (RL) is a powerful framework for sequential decision-making in stochastic environments, where agents learn through trial-and-error to maximize long-term rewards. However, traditional RL adopts a *risk-neutral* stance, focusing solely on expected returns and neglecting outcome variability—an oversight that limits its effectiveness in real-world settings.

This risk-neutral focus proves inadequate in dynamic environments where:

- System perturbations and model mismatches are common
- Rare, high-impact events can cause catastrophic failures
- Small deviations from training conditions lead to performance drops

Such limitations are evident in Actor-Critic methods, where policies trained in idealized settings often underperform when deployed in non-stationary environments.

Risk-Sensitive Reinforcement Learning (RSRL) tackles these issues by incorporating uncertainty directly into the optimization objective. Unlike risk-neutral RL, RSRL accounts for both expected performance and its variability, leading to policies that are more robust to disturbances. This is achieved through:

- Exponential utility-based reward shaping
- Variance-aware objective functions
- Constrained optimization frameworks

Building on prior work [1], we propose new risk-sensitive Policy Gradient variants that retain computational efficiency while enhancing robustness—crucial for safety-critical domains like autonomous driving.

2 RL Preliminaries and Policy Gradient Methods

This section introduces the foundational concepts of Reinforcement Learning (RL)[4], including the Markov Decision Process (MDP) formulation, policy representations, and value functions. These preliminaries form the basis for modern RL algorithms, particularly **policy gradient methods** such as **Proximal Policy Optimization (PPO)** and **Soft Actor-Critic (SAC)**, which directly optimize the agent's behavior using gradient-based updates in continuous and high-dimensional action spaces.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a framework for sequential decision-making where an agent learns to act optimally by interacting with an environment. The mathematical foundation of RL is the **Markov Decision Process (MDP)**, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ is the set of all possible states,
- $\mathcal{A}$ is the set of all possible actions,
- $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,
- $P(s'|s, a)$ is the transition probability model,
- $\gamma \in [0, 1)$ is the discount factor.

At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi(a|s)$, receives a scalar reward $r_t = r(s_t, a_t)$, and transitions to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$. The reward signal is designed to encourage desirable behavior; for example, assigning low rewards for undesirable events (e.g., collisions) and high rewards for desirable events (e.g., lane following).

A **policy** $\pi(a|s)$ is a mapping from states to probability distributions over actions. The objective in RL is to find the **optimal policy** π^* that maximizes the expected **discounted return**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

To evaluate and improve policies, RL relies on two key value functions:

- **State-Value Function**:
$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \,\middle|\, s_t = s \right]$$
which represents the expected return when starting in state s and following policy π.
- **Action-Value Function**:
$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \,\middle|\, s_t = s, a_t = a \right]$$
which represents the expected return when starting in state s, taking action a, and thereafter following policy π.

The optimal value functions are defined as:

$$V^*(s) = \max_\pi V^\pi(s), \quad Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

Once $Q^*(s,a)$ is learned, the **optimal policy** can be derived as:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

In practice, the agent estimates these optimal value functions through continuous interaction with the environment. Upon convergence, the agent behaves optimally with respect to the learned policy.

2.2 Policy Gradient Methods

Policy Gradient (PG) [5] methods aim to directly learn a parameterized policy $\pi_\theta(a|s)$ that maximizes the expected cumulative reward. Unlike value-based methods, PG approaches are particularly suitable for continuous and high-dimensional action spaces.

The objective is:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$$

Using the Policy Gradient Theorem, the gradient of $J(\theta)$ can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{A}_t \right]$$

where $\hat{A}_t$ is the **advantage function**:

$$\hat{A}_t = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Baseline Subtraction and Variance Reduction

To reduce variance without introducing bias, a baseline function $b(s_t)$ is often subtracted:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot (R_t - b(s_t)) \right]$$

A common choice is $b(s_t) = V^\pi(s_t)$, which gives the advantage formulation.

Entropy Regularization

To encourage exploration, an entropy term is added to the objective:

$$J_{\text{entropy}}(\theta) = J(\theta) + \lambda \mathbb{E}_{\pi_\theta} \left[\mathcal{H}(\pi_\theta(\cdot|s_t)) \right]$$

where $\mathcal{H}(\pi) = -\sum_a \pi(a|s) \log \pi(a|s)$ and λ controls the exploration-exploitation balance.

Proximal Policy Optimization (PPO)

PPO [2] improves policy stability by using a clipped surrogate objective. Define the **probability ratio**:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

The clipped objective is:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

This prevents large updates by constraining the ratio $r_t(\theta)$ within a trust region $[1-\epsilon, 1+\epsilon]$, where ϵ is typically 0.1–0.3. The final PPO objective includes an entropy bonus:

$$L^{\text{PPO-full}}(\theta) = L^{\text{PPO}}(\theta) + \lambda \mathbb{E}_t \left[\mathcal{H}(\pi_\theta(\cdot|s_t)) \right]$$

Soft Actor-Critic (SAC)

SAC [3] is an off-policy algorithm that optimizes a stochastic policy in the **maximum entropy** framework. The objective is to maximize both reward and entropy:

$$J(\pi_\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[Q(s_t,a_t) - \alpha \log \pi_\theta(a_t|s_t)\right]$$

Here:

- $\alpha > 0$ is the temperature parameter, balancing reward and entropy,
- $\mathcal{D}$ is a replay buffer of sampled transitions.

The policy is updated by minimizing:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi_\theta}\left[\alpha \log \pi_\theta(a_t|s_t) - Q_\phi(s_t,a_t)\right]$$

The Q-function is updated by minimizing the soft Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(Q_\phi(s_t,a_t) - y_t\right)^2\right]$$

with target:

$$y_t = r_t + \gamma \mathbb{E}_{a_{t+1}\sim\pi_\theta}\left[Q_{\bar{\phi}}(s_{t+1},a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1})\right]$$

To stabilize learning, SAC uses:

- Two Q-networks and a target network,
- Polyak averaging for target updates,
- Automatic entropy tuning by minimizing:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_\theta}\left[-\alpha\left(\log \pi_\theta(a_t|s_t) + \mathcal{H}_{\text{target}}\right)\right]$$

Summary

Policy Gradient methods optimize policies through gradient ascent. PPO ensures reliable learning via clipped updates, while SAC encourages exploration and robustness with entropy regularization and off-policy learning.

Table 1: Comparison of PPO and SAC

Aspect	**PPO**	**SAC**
Policy Type	Stochastic	Stochastic
Learning Type	On-policy	Off-policy
Exploration Mechanism	Entropy bonus (optional)	Max entropy (mandatory)
Sample Efficiency	Lower	Higher
Stability	High	High
Action Space	Continuous/Discrete	Typically continuous

3 Risk-Sensitive Reinforcement Learning and Simulator

We provide a brief overview of Risk-Sensitive Reinforcement Learning (RSRL), an extension of conventional reinforcement learning that incorporates robustness and uncertainty-awareness into the learning objective. This is especially critical for real-world deployment scenarios, where unpredictable environmental perturbations and model uncertainties can significantly degrade the performance of risk-neutral policies.

To evaluate the effectiveness of RSRL methods, we employ the CARLA simulator, a high-fidelity environment for autonomous driving research. This section introduces the CARLA-based experimental setup, detailing the environment configuration, the design of state and action spaces, and the custom reward function tailored to reflect risk-sensitive criteria.

3.1 Risk-Sensitive Reinforcement Learning

Risk-Sensitive Reinforcement Learning (RSRL) [1] enhances traditional RL by accounting for the variability in returns, allowing policies to be optimized not just for reward maximization but also for robustness under uncertainty.

In a standard Markov Decision Process (MDP), the objective is to maximize expected cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{T} r(s_t, a_t)\right],$$

where $\tau = (s_0, a_0, \ldots, s_T)$ is a trajectory, $R(\tau)$ is the total reward, and ρ_θ is the trajectory distribution under policy π_θ.

RSRL introduces an exponential utility objective:

$$J_\beta(\theta) = \frac{1}{\beta}\log \mathbb{E}_{\tau\sim\rho_\theta}\left[e^{\beta R(\tau)}\right],$$

where $\beta \in \mathbb{R}$ is the **risk sensitivity parameter**:

- $\beta > 0$: risk-seeking (prefers high-variance rewards),
- $\beta < 0$: risk-averse (prefers low-variance, stable rewards).

Using Taylor expansion, this can be approximated as:

$$J_\beta(\theta) \approx \mathbb{E}[R(\tau)] + \frac{\beta}{2}\text{Var}[R(\tau)] + \mathcal{O}(\beta^2),$$

where $\text{Var}[R(\tau)] = \mathbb{E}[(R(\tau) - \mathbb{E}[R(\tau)])^2]$.

The second term, $\frac{\beta}{2}\text{Var}[R(\tau)]$, modulates the objective based on reward variance: If $\beta < 0$, this term penalizes high-variance trajectories, guiding the policy toward more consistent and predictable outcomes.

Environmental perturbations are modeled as:

$$P'(s'|s,a) = P(s'|s,a) + \Delta P(s'|s,a), \quad r'(s,a) = r(s,a) + \Delta r(s,a),$$

leading to changes in trajectory distribution and expected return. The effect on the expected reward is approximated as:

$$\Delta\mathbb{E}[R(\tau)] \propto \sqrt{\text{Var}[R(\tau)]} \cdot \|\Delta\|,$$

where $\|\Delta\|$ denotes the magnitude of environmental perturbations.

This shows that:

- **Higher variance** increases the sensitivity of the expected return to perturbations.
- **Risk-averse optimization** (i.e., $\beta < 0$) naturally reduces this sensitivity by discouraging high-variance trajectories.

Hence, minimizing $\text{Var}[R(\tau)]$ through a negative β enhances the **robustness** of the policy:

$$\boxed{\text{Robustness} \uparrow \quad \text{as} \quad \beta \downarrow < 0 \quad \text{and} \quad \text{Var}[R(\tau)] \downarrow}$$

This trade-off between reward and robustness makes RSRL particularly effective in uncertain, dynamic, or adversarial environments.

3.2 CARLA (Car Learning to Act) Simulator

CARLA is an open-source, high-fidelity simulator developed for autonomous driving research by the Computer Vision Center (CVC) at the Universitat Autònoma de Barcelona. It provides photorealistic rendering, realistic physics, and a flexible API for vehicle and sensor control.

Key features of CARLA include:

- Realistic 3D urban environments and vehicle dynamics,
- A rich suite of configurable sensors (camera, LiDAR, GPS, IMU),
- Multi-agent simulation support,
- Full Python API and ROS compatibility.

4 Related Work

This section presents the simulation environment and learning framework used to train autonomous driving agents in CARLA. We describe how sensor inputs are captured and preprocessed, how spatial and dynamic features are fused into a unified state representation [6], and how policy and value functions are trained using deep reinforcement learning algorithms[7] such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

4.1 Image Capture and Preprocessing in CARLA

In the CARLA simulator, an RGB camera sensor captures forward-facing images at a resolution of $800 \times 600 \times 3$ pixels. The preprocessing steps include:

- Conversion of RGB to grayscale,
- Resizing to 128×128,
- Normalization to $[-1, 1]$ using:

$$\text{normalized\_pixel} = \frac{\text{pixel}}{255.0} - 0.5$$

This preprocessing reduces computational overhead and ensures consistent input scaling.

4.2 State Representation Using ResNet-50 and Dynamic Features

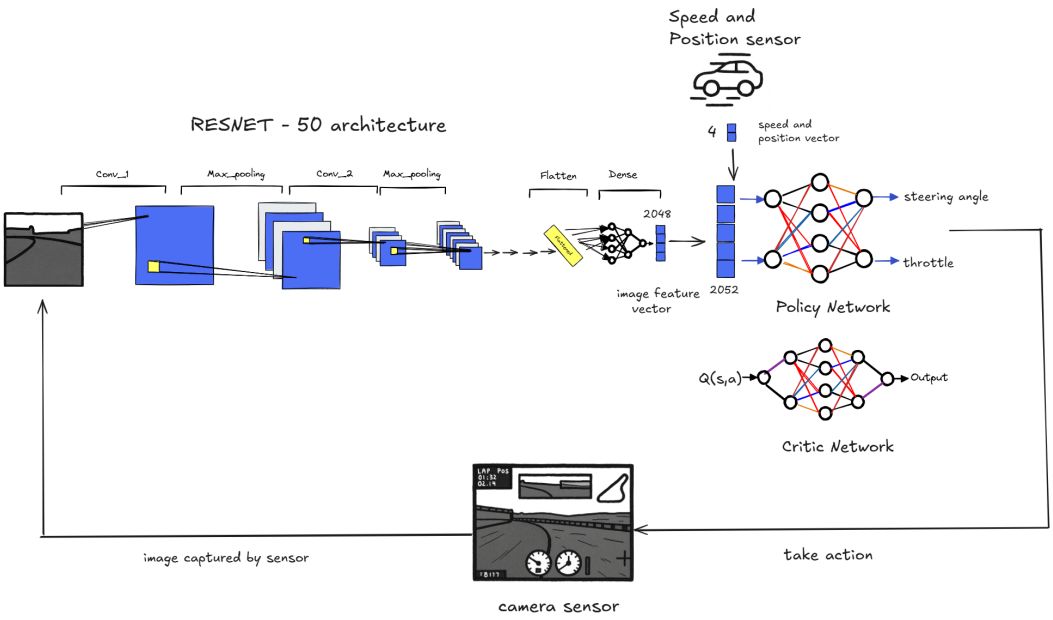

Figure 1: State representation and control pipeline: visual features extracted from a camera image using ResNet-50 are concatenated with dynamic features (speed, throttle, relative position) to form a 2052-dimensional state vector. This state is then processed by the policy and critic networks to generate control actions and value estimates.

The final state representation used for control comprises:

- **Visual Features:** Extracted from the camera image using a pre-trained ResNet-50 network [8], yielding a 2048-dimensional embedding.
- **Dynamic Features:** A 4-dimensional vector encoding speed, throttle, and relative positional information.

These are concatenated to form a 2052-dimensional state vector, which is fed into both the policy and critic networks.

4.3 Simulation Environment

At each time step, the agent receives a *state* from its sensors, selects an *action*, and receives a *reward*. The environment transitions to the next state based on the underlying simulation dynamics [6].

State Representation: The state s_t includes:

- Frontal camera image,
- Positional coordinates of the vehicle,
- Engine speed.

Action Space: The agent controls the vehicle using:

$$a_t = [\text{steering},\ \text{throttle}] \in [-1, 1] \times [0, 1]$$

- Steering $\in [-1, 1]$: full-left to full-right,
- Throttle $\in [0, 1]$: no acceleration to full throttle.

Reward Function:

The reward at time t is computed as:

$$R_t = \lambda_1 R_{\text{safety}} + \lambda_2 R_{\text{progress}} + \lambda_3 R_{\text{comfort}} + \lambda_4 R_{\text{rules}}$$

with weighting coefficients:

$$\lambda_1 = 1.5, \quad \lambda_2 = 1.0, \quad \lambda_3 = 0.8, \quad \lambda_4 = 1.2$$

Each component is defined as follows:

$$R_{\text{safety}} = \begin{cases} -300 \cdot (1 + \min(\frac{v}{50}, 1)) & \text{if collision occurs} \\ -\frac{5}{\text{TTC}} & \text{if TTC} < 2 \\ +5 & \text{otherwise} \end{cases}$$

$$R_{\text{progress}} = \begin{cases} 2.0 \cdot \exp(-\frac{d}{10}) & \text{if } d \leq 10 \\ -0.5 & \text{otherwise} \end{cases} + 5 \cdot \max\left(0, \frac{v}{40}\right)$$

$$R_{\text{comfort}} = -|\Delta\text{throttle}| - 0.1 \cdot |\text{steering}|$$

$$R_{\text{rules}} = -5 \cdot \mathbb{1}_{v>50} - 10 \cdot \mathbb{1}_{\text{lane violation}} - 7 \cdot \mathbb{1}_{\text{headway violation}}$$

where **TTC** is the time-to-collision computed as the nearest vehicle distance divided by speed (m/s), Δ**throttle** is the change in throttle between steps, and **steering** is the current steering control input. This reward function idea is adapted from [9].

Risk-Sensitive Reward Adjustment:

enhance robustness and account for reward variance, we apply an exponential utility-based adjustment [1] to the cumulative reward:

$$\widetilde{R}_t = -\frac{1}{\beta} \log\left(\mathbb{E}\left[\exp(-\beta R_t)\right]\right)$$

This is approximated using a finite reward history of N samples:

$$\widetilde{R}_t \approx -\frac{1}{\beta} \log \left(\frac{1}{N} \sum_{i=1}^{N} \exp \left(-\beta \frac{R_i}{s} \right) \right)$$

where β is the risk sensitivity parameter, and s is a scaling factor (typically set to 5000).

Termination Conditions:

An episode terminates if any of the following conditions are met:

- A collision is detected,
- The vehicle deviates significantly from the target path ($d > 10$),
- The episode exceeds a predefined maximum number of steps (5000 steps).

4.4 Training Setup

We employ two popular deep reinforcement learning algorithm Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) to train autonomous driving policies in the CARLA simulator.

- **PPO** is an on-policy algorithm that stabilizes learning through clipped surrogate objectives and generalized advantage estimation.
- **SAC** is an off-policy algorithm that incorporates entropy maximization to encourage exploration and improve robustness.

Both algorithms are implemented using the Stable-Baselines3 library and trained for 7,000 episodes (with a cap at 10,000 for safety) using a custom CARLA environment. Policies are trained end-to-end, with perception inputs processed via a ResNet-based encoder as part of the forward pass.

Training uses a risk-sensitive reward function, incorporating a tunable parameter $\beta \in \{0, 5, 10\}$ to modulate the agent's sensitivity to risk during decision-making. For each algorithm, we train three models corresponding to the different β values, resulting in six distinct agent configurations.

Throughout training, episodic rewards, episode lengths, and termination reasons (e.g., timesteps reached, collision, or deviation) are logged using a custom callback. Models are checkpointed every 100 episodes, and final models are saved upon completion.

SAC training uses a learning rate of 5×10^{-5}, a batch size of 256, and a replay buffer of one million transitions. PPO uses a learning rate of 3×10^{-5}, a batch size of 64, and performs 10 epochs of policy updates per rollout.

All training was conducted using an NVIDIA RTX 3090 GPU. The training duration varied between 4 to 6 days, depending on the algorithm (PPO or SAC) and the risk sensitivity parameter β. For each modified environment, inference was performed over 500 episodes and required approximately 9 to 15 hours per configuration.

4.5 Evaluation Under Modified Environments

To evaluate generalization, PPO and SAC policies were trained in a fixed CARLA environment and tested under modified conditions. These included changes in weather—no rain (precipitation = 0), light rain (10), and heavy rain (100)—and map layouts (Town02 and Town03). Weather was controlled using the CARLA API by setting the precipitation parameter, where precipitation $\in [0, 100]$ determines rain intensity.

Inference was performed for 500 episodes for each combination of algorithm and risk parameter $\beta \in \{0, 5, 10\}$, resulting in six distinct configurations. This evaluation framework allows a comprehensive assessment of policy robustness to environmental changes and varying levels of risk sensitivity. Results are presented in Section **Results**.

4.6 Risk-Sensitive Learning Algorithms Pseudocode

The following algorithms summarize the training procedures for R-PPO and R-SAC.
The **red-colored text** in the algorithm highlights the modifications made to standard PPO and SAC to incorporate risk sensitivity.

Algorithm 1 Risk-sensitive Proximal Policy Optimization (R-PPO)

Input: stochastic policy $\pi(a|s;\theta)$, value function $V(s;\phi)$
Parameters: learning rate α, risk sensitivity β, clip threshold ϵ, epochs K

Initialize policy parameters θ_0, value parameters ϕ_0
for each iteration **do**
  Collect trajectories $\{\tau_i\}$ using current policy $\pi(\cdot|\cdot;\theta)$
  for each trajectory τ **do**
    for each timestep t in τ **do**
      Compute exponential return:

$$\hat{R}_t^\beta \leftarrow \frac{1}{\beta}\log\left(\sum_{k=t}^{T} e^{\beta r_k}\right)$$

      Compute risk-sensitive advantage: $\hat{A}_t^\beta \leftarrow \hat{R}_t^\beta - V(s_t;\phi)$
      Compute ratio: $r_t(\theta) \leftarrow \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}$
      PPO surrogate loss:

$$L_t^\beta(\theta) \leftarrow \min\left(r_t(\theta)\cdot\hat{A}_t^\beta,\ \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\cdot\hat{A}_t^\beta\right)$$

    end for
  end for
  Update policy: $\theta \leftarrow \theta + \alpha\nabla_\theta \sum_t L_t^\beta(\theta)$
  Update value function: $\phi \leftarrow \phi - \alpha\nabla_\phi \sum_t (V(s_t;\phi) - \hat{R}_t^\beta)^2$
end for=0

Algorithm 2 Risk-sensitive Soft Actor-Critic (R-SAC)

Input: policy $\pi(a|s;\theta)$, Q-networks Q_1, Q_2 with targets Q_1', Q_2'
Parameters: learning rates η, ζ, temperature α, risk parameter β

Initialize policy parameters θ, Q-network parameters ϕ_1, ϕ_2, target networks ϕ_1', ϕ_2'
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
  Collect experience (s, a, r, s') using $\pi(a|s;\theta)$, add to $\mathcal{D}$
  Sample mini-batch $\{(s_i, a_i, r_i, s_i')\}$ from $\mathcal{D}$
  for each sample **do**
    Compute exponential risk-sensitive target:

$$y_i^\beta = \frac{1}{\beta}\log\left(e^{\beta r_i} + \gamma\mathbb{E}_{a'\sim\pi}\left[e^{\beta(Q'_{\min}(s_i',a') - \alpha\log\pi(a'|s_i'))}\right]\right)$$

    Update critics:

$$\phi_j \leftarrow \phi_j - \eta\nabla_{\phi_j}\left(Q_{\phi_j}(s_i,a_i) - y_i^\beta\right)^2,\ j=1,2$$

    Update policy:

$$\theta \leftarrow \theta + \zeta\nabla_\theta\left(\alpha\log\pi(a_i|s_i;\theta) - Q_{\min}(s_i,a_i)\right)$$

  end for
  Update target networks: $\phi_j' \leftarrow \tau\phi_j + (1-\tau)\phi_j'$
end for=0

5 Results

We report and analyze the performance of CARLA agents trained using Risk-sensitive PPO (R-PPO) and Risk-sensitive SAC (R-SAC) for different risk aversion levels $\beta \in \{0, 5, 10\}$. The results are organized into three parts: (i) training behavior across algorithms and β values, (ii) generalization performance under varying weather conditions and across different maps (Town02 and Town03), and (iii) metric-based evaluation using mean reward, Conditional Value at Risk (CVaR), and entropic risk.

5.1 Training and Inference Curves for R-PPO

Figures 2, 3, and 4 show the training and inference curves for R-PPO with $\beta = 0$, $\beta = 5$, and $\beta = 10$, respectively. Each model was trained for 7000 episodes, followed by inference over 500 episodes. Inference performance is appended to the training curves to enable seamless comparison. Increasing β results in smoother training and more stable returns across environments.

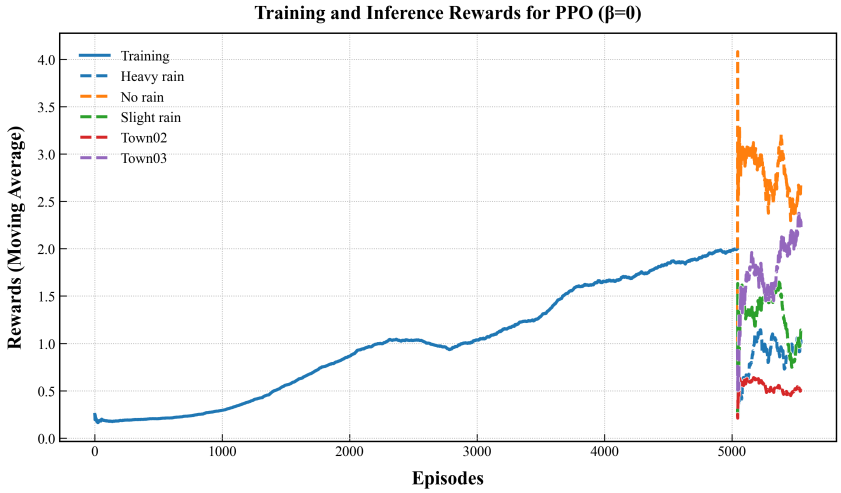

Figure 2: Training reward trajectory for Risk-PPO with $\beta = 0$.

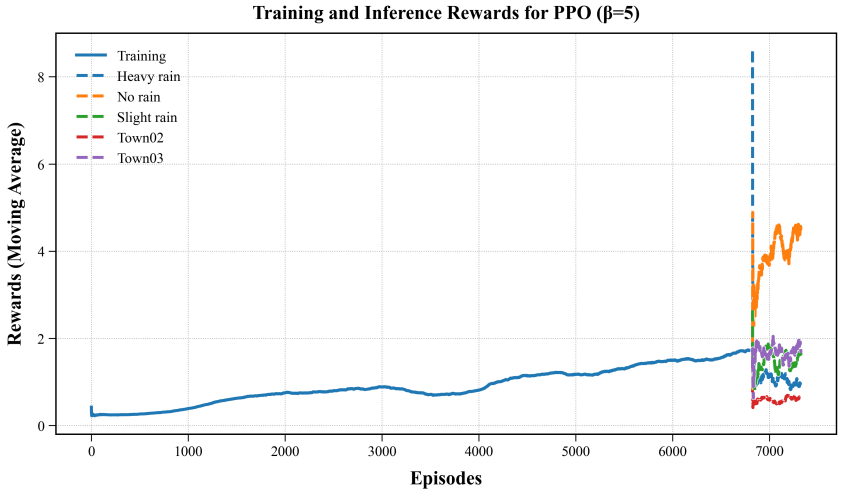

Figure 3: Training reward trajectory for Risk-PPO with $\beta = 5$.

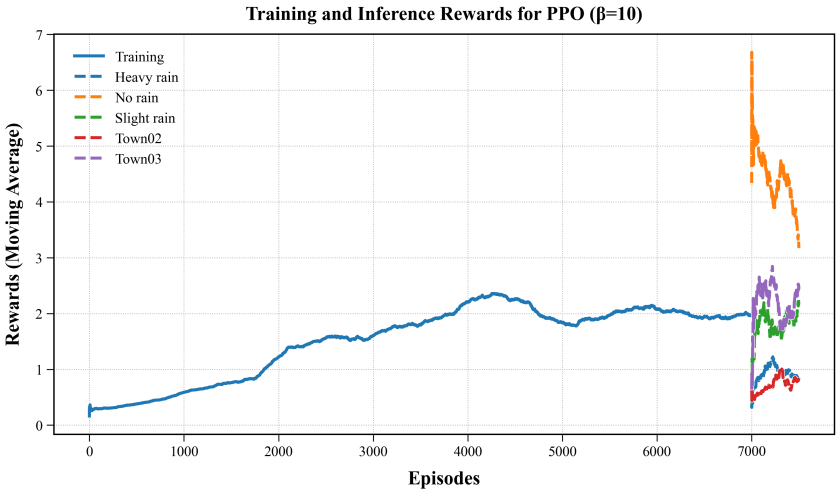

Figure 4: Training reward trajectory for Risk-PPO with $\beta = 10$.

5.2 Training and Inference Curves for R-SAC

Figures 5, 6, and 7 show the training and inference curves for R-SAC with $\beta = 0$, $\beta = 5$, and $\beta = 10$, respectively. Each model was trained for around 4000 episodes, followed by inference over 500 episodes. Inference performance is appended to the training curves to enable seamless comparison. Increasing β results in enhanced training and more stable returns across environments.

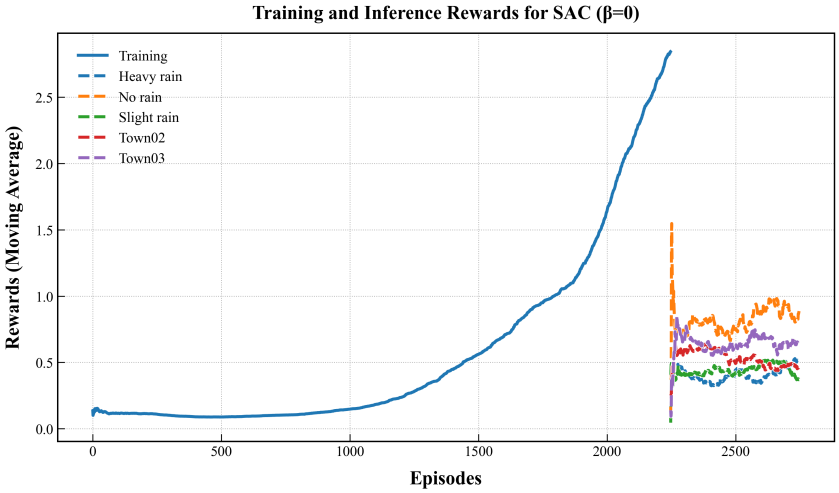

Figure 5: Training reward trajectory for Risk-SAC with $\beta = 0$.

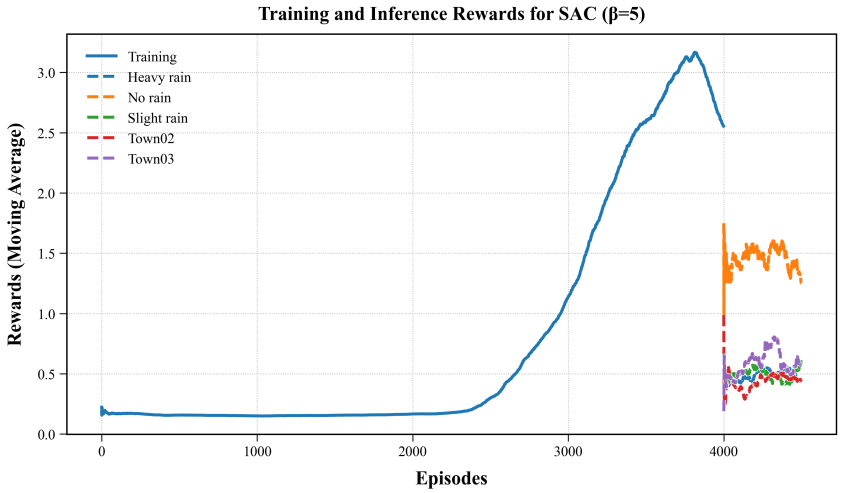

Figure 6: Training reward trajectory for Risk-SAC with $\beta = 5$.

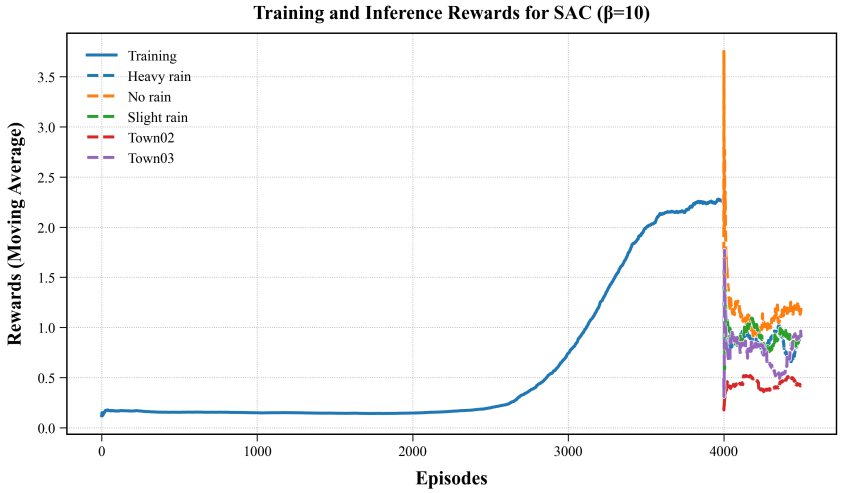

Figure 7: Training reward trajectory for Risk-SAC with $\beta = 10$.

5.3 Risk-Aware Performance Metrics

We compute the following metrics from episode rewards for each combination of algorithm, risk parameter β, and five modified test environments:

- **Mean Reward:** $\mathbb{E}[R] = \frac{1}{N}\sum_{i=1}^{N} R_i$, representing average performance.
- **CVaR (at $\alpha = 0.95$):** Average of the lowest 5% of rewards, reflecting worst-case risk.
- **Entropic Risk:** $\rho_\beta(R) = -\frac{1}{\beta}\log \mathbb{E}[\exp(-\beta R)]$, capturing risk-sensitive variability with parameter β controlling risk aversion.

These metrics collectively characterize expected returns, robustness to rare failures, and sensitivity to risk across different environments and training configurations.

The performance metrics for each algorithm and risk parameter β under the five modified environments are summarized in Figures 8, 9, and 10. These plots illustrate how risk sensitivity and environmental changes impact the agent's expected returns, worst-case outcomes, and risk-aware performance.

The entropic risk metric is undefined for $\beta = 0$; hence, it is computed and plotted only for $\beta = 5$ and $\beta = 10$ (see Figure 10).

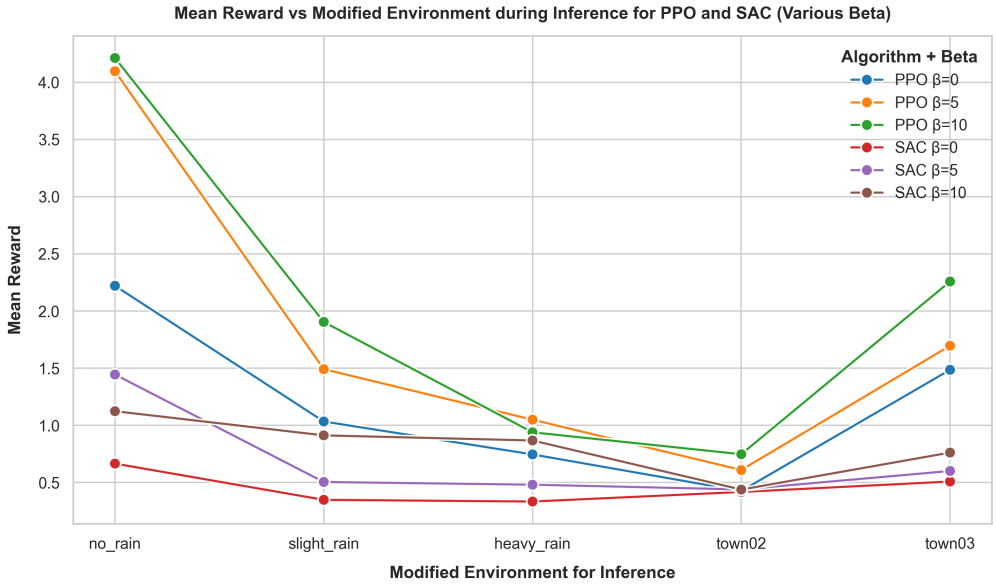

Figure 8: Mean Reward vs Modified Environment for different algorithms and β's.

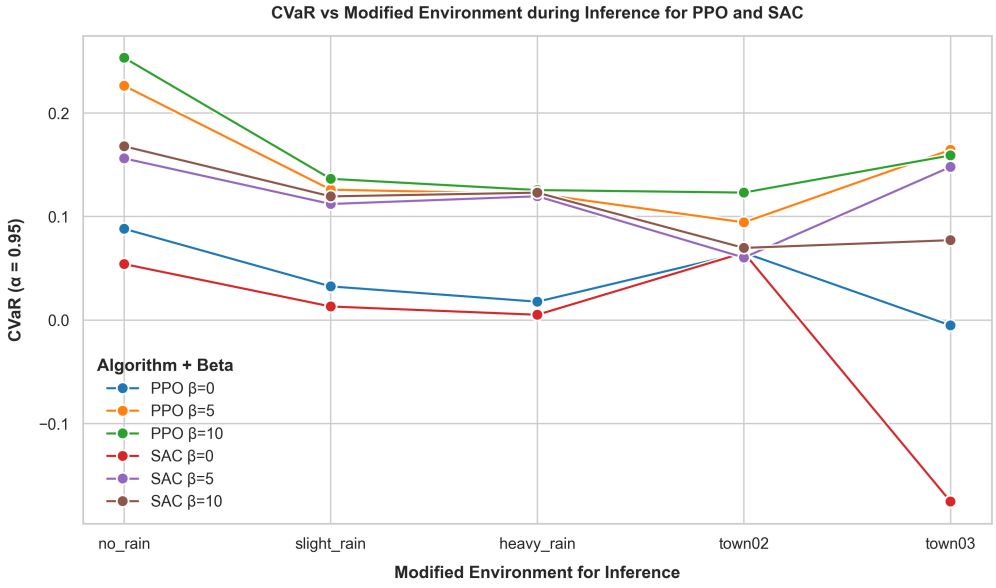

Figure 9: CVaR ($\alpha = 0.95$) vs Modified Environment for differnt algorithms and β's.

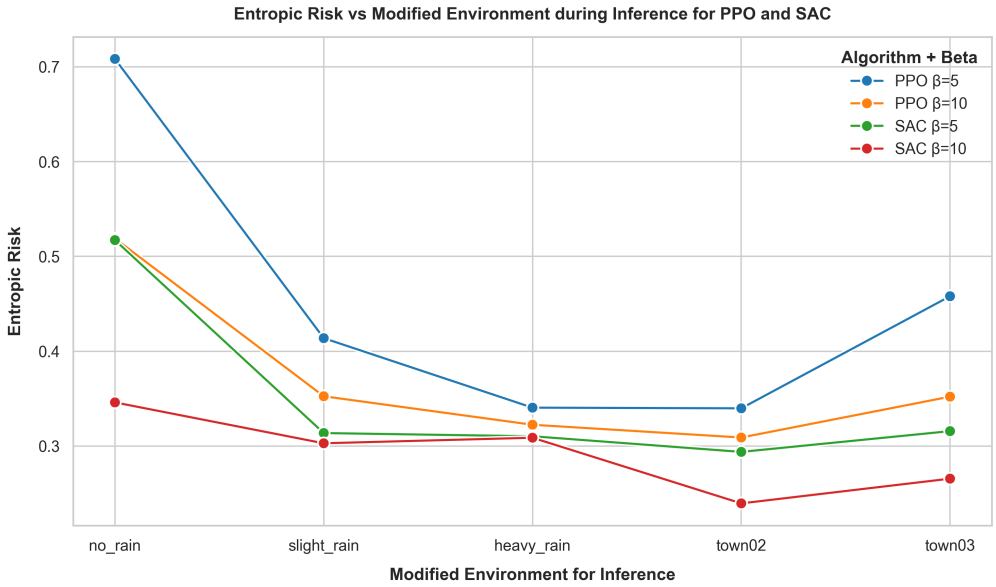

Figure 10: Entropic risk vs Modified Environment for different algorithms and β's.

Tabular results for mean reward, CVaR (0.95), and entropic risk across algorithms, β, and conditions.

Table 2: Performance Metrics for PPO and SAC under Different Modified Environments

Algorithm	β	**Disturbance**	**Mean Reward**	**CVaR (0.95)**	**Entropic Risk**
PPO	0	no_rain	2.220	0.088	*undefined*
		slight_rain	1.030	0.033	
		heavy_rain	0.750	0.018	
		town02	0.430	0.065	
		town03	1.490	−0.005	
	5	no_rain	4.100	0.226	0.708
		slight_rain	1.490	0.126	0.414
		heavy_rain	**1.050**	0.122	0.341
		town02	0.610	0.094	0.340
		town03	1.700	**0.164**	0.458
	10	no_rain	**4.210**	**0.253**	0.519
		slight_rain	**1.900**	**0.137**	0.353
		heavy_rain	0.940	**0.126**	0.323
		town02	**0.750**	**0.123**	0.309
		town03	**2.260**	0.159	0.352
SAC	0	no_rain	0.670	0.054	*undefined*
		slight_rain	0.350	0.013	
		heavy_rain	0.330	0.005	
		town02	0.420	0.065	
		town03	0.510	−0.175	
	5	no_rain	1.440	0.156	0.517
		slight_rain	0.500	0.112	0.314
		heavy_rain	0.480	0.120	0.310
		town02	0.440	0.060	0.294
		town03	0.600	0.148	0.316
	10	no_rain	1.120	0.168	**0.346**
		slight_rain	0.910	0.120	**0.303**
		heavy_rain	0.870	0.123	**0.309**
		town02	0.440	0.070	**0.240**
		town03	0.760	0.077	**0.266**

5.4 Insights from Results

The experimental results comprising PPO (Proximal Policy Optimization) and SAC (Soft Actor Critic) in the CARLA simulator reveal key differences in training dynamics and inference performance.

Training Phase Comparison: SAC vs PPO

- **SAC** benefits from its *off-policy* formulation, which leverages a replay buffer to reuse past experiences, significantly improving sample efficiency. Additionally, its entropy regularization encourages exploration, enabling faster convergence and the early discovery of effective policies.
- **PPO**, in contrast, is an *on-policy* method that discards data after each update, leading to lower sample efficiency and requiring more environment interactions. However, its *clipped surrogate objective* ensures stable training by preventing large policy updates, making it more robust but slower to learn.

Inference Phase Comparison: PPO vs SAC

- **PPO** outperformed **SAC** during inference by producing smoother and more stable control. Although both learn stochastic policies, PPO typically uses the mean action during deployment, ensuring consistent behavior. In contrast, SAC often retains stochastic sampling, which can introduce variability in throttle and steering commands.
- PPO's clipped surrogate objective yields conservative and robust policy updates, which translate to precise control during high-accuracy tasks like lane keeping in CARLA. SAC's entropy-driven exploration, while helpful in training, may lead to noisy or unstable decisions at test time.

Conclusion

The evaluation highlights that **PPO with $\beta = 10$** consistently outperformed other configurations, showing strong stability and adherence to driving rules under uncertainty. In contrast, **SAC with $\beta = 0$** delivered the weakest performance, indicating poor adaptability and control.

These results suggest that:

- Policies exposed to high reward variance are more sensitive to environmental perturbations.
- Risk-aware learning (with $\beta < 0$) effectively dampens this variance, encouraging more consistent and reliable behavior.

This supports the view that increasing the level of risk aversion (larger absolute β) leads to greater policy robustness. In dynamic or uncertain driving scenarios, such robustness becomes essential:

$$\boxed{\text{As } |\beta| \uparrow, \quad \text{Reward Variance} \downarrow, \quad \text{Robustness} \uparrow}$$

Thus, RSRL strikes a meaningful balance between reward optimization and stability, proving valuable for safety-critical domains like autonomous driving.

6 Future Work

Future work will focus on improving risk-sensitive RL in realistic settings:

- Test inference on modified environments with pedestrians and vehicle traffic for better policy adaptation.
- Modify reward functions to handle traffic signals, pedestrians, and complex road disturbances.
- Explore other RL algorithms (DDPG, TRPO) and different risk parameters (β) for robustness.
- Extend experiments from CARLA to drone environments like Microsoft AirSim.

References

[1] Erfaun Noorani, Christos Mavridis, and John Baras. *Risk-Sensitive Reinforcement Learning with Exponential Criteria*. 2024. arXiv: 2212.09010 [eess.SY]. URL: https://arxiv.org/abs/2212.09010.

[2] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[3] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[5] Richard S. Sutton et al. *Policy Gradient Methods for Reinforcement Learning with Function Approximation*. Tech. rep. 180 Park Avenue, Florham Park, NJ 07932: AT&T Labs - Research, 2000.

[6] B. Peng et al. "End-to-End Autonomous Driving Through Dueling Double Deep Q-Network". In: *Automotive Innovation* 4.4 (2021), pp. 328–337. DOI: 10.1007/s42154-021-00151-3.

[7] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG]. URL: https://arxiv.org/abs/1312.5602.

[8] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[9] Ahmed Abouelazm, Jonas Michel, and J. Marius Zoellner. *A Review of Reward Functions for Reinforcement Learning in the context of Autonomous Driving*. 2024. arXiv: 2405.01440 [cs.RO]. URL: https://arxiv.org/abs/2405.01440.