A Extra Algorithm Details

A.1 Proof of Uncertainty Set Regularized Robust Bellman Equation

The proof is as follows:

$$
\begin{aligned}
Q^{\pi}(s,a) &= r(s,a) + \gamma \min_{w \in \Omega_w} \int_{s'} P(s'|s,a;w) V^{\pi}(s') ds' \\
&= r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) V^{\pi}(s') ds' + \gamma \min_{w \in \Omega_w} \int_{s'} (w - \bar{w})^T \nabla_w P(s'|s,a;\bar{w}) V^{\pi}(s') ds' \\
&= r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) V^{\pi}(s') ds' - \gamma \left[\max_{\tilde{w}} \int_{s'} -\tilde{w}^T \nabla_w P(s'|s,a;\bar{w}) V^{\pi}(s') ds' - \delta_{\Omega_{\bar{w}}}(\tilde{w}) \right] \\
&= r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) V^{\pi}(s') ds' - \gamma \int_{s'} \delta^*_{\Omega_{\bar{w}}} \left[-\nabla_w P(s'|s,a;\bar{w}) V^{\pi}(s') \right] ds'
\end{aligned}
\tag{6}
$$

The second line utilizes the first-order Taylor Expansion at $\bar{w}$. The third line reformulates the minimization on w to maximization on $\tilde{w}$ and adds an indicator function as a hard constraint on $\tilde{w}$. The last line directly follows the definition of convex conjugate function.

A.2 Convergence of Robust Policy Evaluation

We now prove that the Bellman operator with an extra regularizer term as the *policy evaluation* stage can converge to the robust action value function given some specific conditions.

Since $V^{\pi}(s) = \int_a \pi(a|s) Q^{\pi}(s,a) da$, we define the equivalent operator with respect to Q^{π} to the one proposed in this paper as T such that

$$
\begin{aligned}
TQ^{\pi}(s,a) &= r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) \int_{a'} Q^{\pi}(s',a') da' ds' - \alpha \int_{s'} \|\nabla_w P(s'|s,a;\bar{w}) \int_{a'} Q^{\pi}(s',a') da'\|_2 ds' \\
&= \underbrace{r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) V^{\pi}(s') ds' - \alpha \int_{s'} \|\nabla_w P(s'|s,a;\bar{w}) V^{\pi}(s')\|_2 ds'}_{\text{The operator proposed in this paper.}}.
\end{aligned}
\tag{7}
$$

To ease the derivation, we will not expand the term V^{π} as the form of Q^{π} at the beginning in the following procedures.

$$
\begin{aligned}
||TQ_1^{\pi} - TQ_2^{\pi}||_{\infty} &= \max_{s,a} \Bigg| r(s,a) + \gamma \int_{s'} P(s'|s,a;\bar{w}) V_1^{\pi}(s') ds' - \alpha \int_{s'} \|\nabla_w P(s'|s,a;\bar{w}) V_1^{\pi}(s')\|_2 ds' \\
&\quad - r(s,a) - \gamma \int_{s'} P(s'|s,a;\bar{w}) V_2^{\pi}(s') ds' + \alpha \int_{s'} \|\nabla_w P(s'|s,a;\bar{w}) V_2^{\pi}(s')\|_2 ds' \Bigg| \\
&= \max_{s,a} \Bigg| \gamma \int_{s'} P(s'|s,a;\bar{w}) \Big[V_1^{\pi}(s') - V_2^{\pi}(s') \Big] ds' + \\
&\quad \alpha \int_{s'} \Big[\|\nabla_w P(s'|s,a;\bar{w}) V_2^{\pi}(s')\|_2 - \|\nabla_w P(s'|s,a;\bar{w}) V_1^{\pi}(s')\|_2 \Big] ds' \Bigg| \\
&\leq \underbrace{\gamma \max_{s,a} \left| \int_{s'} P(s'|s,a;\bar{w}) \Big[V_1^{\pi}(s') - V_2^{\pi}(s') \Big] ds' \right|}_{\text{Partition 1.}} + \\
&\quad \underbrace{\alpha \max_{s,a} \left| \int_{s'} \Big[\|\nabla_w P(s'|s,a;\bar{w}) V_2^{\pi}(s')\|_2 - \|\nabla_w P(s'|s,a;\bar{w}) V_1^{\pi}(s')\|_2 \Big] ds' \right|}_{\text{Partition 2.}}
\end{aligned}
$$

Since Partition 1 is a conventional Bellman operator, following the contraction mapping property we can directly write the inequality such that

$$\gamma \max_{s,a} \left| \int_{s'} P(s'|s,a;\bar{w}) \Big[V_1^\pi(s') - V_2^\pi(s') \Big] ds' \right| \leq \gamma ||V_1^\pi(s') - V_2^\pi(s')||_\infty. \tag{8}$$

Next, we will process Partition 2. The details are as follows:

$$\begin{aligned}
&\alpha \max_{s,a} \left| \int_{s'} \Big[\|\nabla_w P(s'|s,a;\bar{w}) V_2^\pi(s')\|_2 - \|\nabla_w P(s'|s,a;\bar{w}) V_1^\pi(s')\|_2 \Big] ds' \right| \\
&\leq \alpha \max_{s,a} \left| \int_{s'} \left\| \nabla_w P(s'|s,a;\bar{w}) \Big[V_2^\pi(s') - V_1^\pi(s') \Big] \right\|_2 ds' \right| \\
&\leq \alpha \max_{s,a} \left| \int_{s'} \|\nabla_w P(s'|s,a;\bar{w})\|_2 |V_2^\pi(s') - V_1^\pi(s')| ds' \right| \\
&\leq \alpha \max_{s,a} \left| \max_s |V_2^\pi(s') - V_1^\pi(s')| \int_{s'} \|\nabla_w P(s'|s,a;\bar{w})\|_2 ds' \right| \\
&\leq \underbrace{\alpha \max_{s,a} \left| \int_{s'} \|\nabla_w P(s'|s,a;\bar{w})\|_2 ds' \right|}_{:=\delta} \max_{s'} |V_1^\pi(s') - V_2^\pi(s')| \\
&= \delta ||V_1^\pi - V_2^\pi||_\infty.
\end{aligned} \tag{9}$$

Combining the results of Eq.8 and Eq.9 and expanding the term V^π, we can directly get that

$$\begin{aligned}
||TQ_1^\pi - TQ_2^\pi||_\infty &\leq (\gamma + \delta) ||V_1^\pi - V_2^\pi||_\infty \\
&\leq (\gamma + \delta) \max_{s'} |V_1^\pi(s') - V_2^\pi(s')| \\
&\leq (\gamma + \delta) \max_{s',a'} |Q_1^\pi(s',a') - Q_2^\pi(s',a')| \\
&\leq (\gamma + \delta) ||Q_1^\pi - Q_2^\pi||_\infty
\end{aligned}$$

To enable T to be a contraction mapping in order to converge to the exact robust Q-values, we have to let $0 \leq \gamma + \delta \leq 1$. In more details, the norm of the gradient of transition function with respect to the uncertainty set (i.e., $\|\nabla_w P(s'|s,a;\bar{w})\|_2$ inside δ) is critical to the convergence of robust Q-values under some robust policy. Given the above conditions, the robust value function will finally converge.

A.3 Algorithm on Incorporating Uncertainty Set Regularizer into Soft Actor Critick

The proposed Uncertainty Set Regularizer method is flexible to be plugged into any existing RL frameworks as introduced in Section 3.2. Here, we include a specific implementation on Soft Actor Critic algorithm, see Algorithm 1.

A.4 Algorithm on Generation of Adversarial Uncertainty Set

This is the pseudo-code to generate the adversarial uncertainty set introduced in the paper.

B Extra Experimental Setups

B.1 RWRL Benchmarks

In this paper, we conduct experiments on six tasks: *cartpole_balance*, *cartpole_swingup*, *walker_stand*, *walker_walk*, *quadruped_walk*, *quadruped_run*. All tasks involve a domain and a movement. For example, *cartpole_balance* task represents the balance movement on cartpole domain. In this paper, we consider 3 domains with 2 movements each.

The 3 domains are cartpole, walker and quadruped respectively:

Algorithm 1 Uncertainty Set Regularized Robust Soft Actor Critic

Input: initial State s, action value $Q(s,a;\theta)$'s parameters θ_1, θ_2, policy $\pi(a|s;\phi)$'s parameters ϕ, replay buffer $\mathcal{D}=\emptyset$, transition nominal parameters $\bar{w}$, value target update rate ρ
Set target value parameters $\theta_{tar,1} \leftarrow \theta_1$, $\theta_{tar,2} \leftarrow \theta_2$
repeat
  Execute $a \sim \pi(a|s;\phi)$ in the environment
  Observe reward r and next State s'
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{s,a,r,s'\}$
  $s \leftarrow s'$
  for each gradient step **do**
    Randomly sample a batch transitions, $\mathcal{B}=\{s,a,r,s'\}$ from $\mathcal{D}$
    Construct adversarial adversarial uncertainty set $\Omega_{\tilde{w}}$ as introduced in Section 3.3 (for adversarial uncertainty set only)
    Compute robust value target $y(s,a,r,s',\bar{w})$ by calculating the RHS of Equation 4
    Update action State value parameters θ_i for $i \in \{1,2\}$ by minimizing mean squared loss to the target:
      $\nabla_{\theta_i} \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s')\in\mathcal{B}} (Q(s,a;\theta_i) - y(s,a,r,s',\bar{w}))^2 \quad$ for $i=1,2$
    Update policy parameters ϕ by policy gradient:
      $\nabla_{\phi} \frac{1}{|\mathcal{B}|} \sum_{(s)\in\mathcal{B}} (\min_{i=1,2} Q(s,\tilde{a};\theta_i) - \alpha \log \pi(\tilde{a}|s;\phi))$
    where $\tilde{a}$ is sampled from $\pi(a|s;\phi)$ and differentiable w.r.t. ϕ
    Update target value parameters:
      $\theta_{tar,i} \leftarrow (1-\rho)\theta_{tar,i} + \rho\theta_i \quad$ for $i=1,2$
  end for
until convergence

Algorithm 2 Generation of Adversarial Uncertainty Set

Input: current state s, current action a, value function $V(s;\theta)$, next state distribution $P(\cdot|s,a;\bar{w}) = \mathcal{N}(\mu(s,a;\bar{w}), \Sigma(s,a;\bar{w}))$
do
  Sample $\sigma \sim \mathcal{N}(0,I)$ and calculate next state $s' = \mu(s,a;\bar{w}) + \Sigma(s,a;\bar{w})\sigma$ (the reparameterization trick to sample $s' \sim P(\cdot|s,a;\bar{w})$);
  Forward pass to calculate the next state value $V(s';\theta)$;
  Backward pass to compute the derivative $g(\bar{w}) = \nabla_w V(s';\theta) = \frac{\partial V(s';\theta)}{\partial s'} \frac{\partial \mu(s,a;\bar{w}) + \Sigma(s,a;\bar{w})\sigma}{\partial w}$;
  Normalize the derivative by $d(\bar{w}) = g(\bar{w}) / [\sum_i^W g(\bar{w})_i^2]^{0.5}$;
  Generate the adversarial uncertainty set $\Omega_w = \{\bar{w} + \alpha\tilde{w} : \|\tilde{w}/d(\bar{w})\|_2 \leq 1\}$.
done

- **Cartpole** has an unactuated pole based on a cart. One can apply a one-direction force to balance the pole. For *cartpole_balance* task, the pole starts near the upright while in *cartpole_swingup* task the pole starts pointing down.
- **Walker** is a planar walker to be controlled in 6 dimensions. The *walker_stand* task requires an upright torso and some minimal torso height. The *walker_walk* task encourages a forward velocity.
- **Quadruped** is a generic quadruped with a more complex state and action space than cartpole and walker. The *quadruped_walk* and *quadruped_run* tasks encourage a different level of forward speed.

For a detailed description of these tasks, please refer DM_CONTROL [45]. For each task, we follow the RWRL's setup by selecting 4 environmental variables and perturbing them during the testing phase. The perturbed variables and their value ranges can be found in Table 2. We also report the nominal values of these perturbed variables to indicate the differences between training and testing environments. Notably, all tasks are run for a maximum of 1000 steps and the max return is 1000.

Table 2: Tasks in RWRL benchmark and the perturbed variables.

Task Name	Observation Dimension	Action Dimension	Perturbed Variables	Perturbed Range	Nominal Value
cartpole_balance *cartpole_swingup*	5	1	pole_length pole_mass joint_damping slider_damping	[0.3, 3.0] [0.1, 10.0] [2e-6, 2e-1] [5e-4, 3.0]	1.0 0.1 2e-6 5e-4
walker_stand *walker_walk*	24	6	tigh_length torso_length joint_damping contact_friction	[0.1, 0.7] [0.1, 0.7] [0.1, 10.0] [0.01, 2.0]	0.225 0.3 0.1 0.7
quadruped_walk *quadruped_run*	78	12	shin_length torso_density joint_damping contact_friction	[0.25, 2.0] [500, 10000] [10, 150] [0.1, 4.5]	0.25 1000 30 1.5

B.2 Practical Implementations of *Robust-AUC*

To calculate *Robust-AUC* in RWRL experiments, each agent is trained with 5 random seeds. During the testing phase, for each environmental variable P, we uniformly sample 20 perturbed values v in the range of $[v_{min}, v_{max}]$. For each value v, the environment variable P is first modified to value v and the agent is tested for 100 episodes (20 episodes per seed). We then select the 10%-quantile [1] as the return r at value v. By doing so we roughly have an approximated curve $C(v, r)$ and can calculate *Robust-AUC* defined previously. We also report the area between 5%-quantile and 15%-quantile as the statistical uncertainty of the reported *Robust-AUC*.

B.3 Model Structure

The model structure for all experimental baselines is based on the Yarats and Kostrikov [46]'s implementation of Soft Actor Critic (SAC) [27] algorithm. The actor network is a 3-layer feed-forward network with 1024 hidden units and outputs the Gaussian distribution of action. The critic network adopts the double-Q structure [47] and also has 3 hidden layers with 1024 hidden units on each layer, but only outputs a real number as the action State value.

B.4 Hyperparameters

To compare all algorithms fairly, we set all hyperparameters equally except the robust method and its coefficient. All algorithms are trained with Adam optimizer [28]. The full hyperparameters are shown in Table 3. For regularizer coefficients of all robust update methods, please see Table 4. We can see that this parameter is pretty robust and do not need very handy work to finetune. All experiments are carried out on NVIDIA GeForce RTX 2080 Ti and Pytorch 1.10.1.

B.5 Extra Setups of Sim-to-real Task

We use the Unitree A1 robot [34] and the Bullet simulator [35] as the platform for sim-to-real transfer. The Unitree A1 is a quadruped robot with 12 motors (3 motors per leg). The Bullet simulator is a popular simulation tool specially designed for robotics.

It is well known that there is a non-negligible difference between simulators and real robots due to:(1) the simulator possesses a simplified dynamics model and suffers from accumulated error [33] and (2) there are significant differences between simulators and real hardware in robot's parameters, such as a quadruped example in Table 5. Therefore, training policies in simulation and applying them to real robots (sim-to-real) is a challenging task for robotics.

[1] 10%-quantile as a worst-case performance evaluates the robustness of RL algorithms more reasonably than common metrics.

Table 3: Hyperparameters of Robust RL algorithms.

HYPERPARAMETERS	VALUE	DESCRIPTION
BATCH SIZE	1024	THE NUMBER OF TRANSITIONS FOR EACH UPDATE
DISCOUNT FACTOR γ	0.99	THE IMPORTANCE OF FUTURE REWARDS
REPLAY BUFFER SIZE	1E6	THE MAXIMUM NUMBER OF TRANSITIONS STORED IN MEMORY
EPISODE LENGTH	1E3	THE MAXIMUM TIME STEPS PER EPISODE
MAX TRAINING STEP	1E6	THE NUMBER OF TRAINING STEPS
RANDOM STEPS	5000	THE NUMBER OF RANDOMLY ACTING STEPS AT THE BEGINNING
ACTOR LEARNING RATE	1E-4	THE LEARNING RATE FOR ACTOR NETWORK
ACTOR UPDATE FREQUENCY	1	THE FREQUENCY FOR UPDATING ACTOR NETWORK
ACTOR LOG STD BOUNDS	[-5, 2]	THE OUTPUT BOUND OF LOG STANDARD DEVIATION
CRITIC LEARNING RATE	1E-4	THE LEARNING RATE FOR CRITIC NETWORK
CRITIC TARGET UPDATE FREQUENCY	2	THE FREQUENCY FOR UPDATING CRITIC TARGET NETWORK
CRITIC TARGET UPDATE COEFFICIENT	0.005	THE UPDATE COEFFICIENT OF CRITIC TARGET NETWORK FOR SOFT LEARNING
INIT TEMPERATURE	0.1	INITIAL TEMPERATURE OF ACTOR'S OUTPUT FOR EXPLORATION
TEMPERATURE LEARNING RATE	1E-4	THE LEARNING RATE FOR UPDATING THE POLICY ENTROPY
SAMPLE SIZE	1	THE SAMPLE SIZE TO APPROXIMATE THE ROBUST REGULARIZOR

Table 4: Regularization coefficients of Robust RL algorithms.

Task Name	**Algorithms**					
	None-Reg	*L1-Reg*	*L2-Reg*	*L1-USR*	*L2-USR*	*Adv-USR*
cartpole_balance	-	1e-5	1e-4	5e-5	1e-4	1e-5
cartpole_swingup	-	1e-5	1e-4	1e-4	1e-4	1e-4
walker_stand	-	1e-4	1e-4	5e-5	1e-4	1e-4
walker_walk	-	1e-4	1e-4	1e-4	1e-4	5e-4
quadruped_walk	-	1e-5	1e-4	1e-4	1e-4	5e-4
quadruped_run	-	1e-4	1e-4	5e-5	1e-4	7e-5

Specifically, we perform 2 sim-to-real tasks: standing and locomotion following previous work [36]. The detailed description of the experiment is as follows:

Observation The observation contains the following features of 3 steps: motor angles (12 dim), root orientation (4 dim, roll, pitch, roll velocity, pitch velocity), and previous actions (12 dim). So the observation space is 84 dimensions.

Action All 12 motors can be controlled in the position control mode, which is further converted to torque with an internal PD controller. The action space for each leg is defined as $[p - o, p + o]$. The specific values for different parts (hip, upper leg, knee) in the standing task are $p = [0.00, 1.6, -1.8], o = [0.8, 2.6, 0.8]$. The values in the locomotion task are $p = [0.05, 0.7, -1.4], o = [0.2, 0.4, 0.4]$.

Reward For the standing task, the reward consists of 3 parts: $r(s,a) = 0.2 * r_{\text{HEIGHT}} + 0.6 * r_{\text{POSE}} + 0.2 * r_{\text{VEL}}$. $r_{\text{HEIGHT}} = 1 - |z - 0.2587|/0.2587$ is rewarded for approaching the standing position on the z-axis. $r_{\text{POSE}} = \exp\{-0.6 * \sum |m_{target} - m|\}$ is for correct motor positions. r_{VEL} is punished for positive velocity (standing should be still in the end). For the locomotion task, the reward function is inspired by Smith et al. [48]. $r(s,a) = r_v(s,a) - 0.1 * v_{\text{YAW}}^2$. v_{YAW} is angular yaw velocity and $r_v(s,a) = 1$ for $v_x \in [0.5, 1.0]$, $= 0$ for $v_x \geq 2.0$ and $= 1 - |v_x - 0.5|$ otherwise, is rewarded for velocity in x-axis.

Simulation Training We first train SAC agents with and without *Adv-USR* in simulations. Each step simulates 0.033 seconds so that the control frequency is 33 Hz. Agents are trained for $1e^6$ steps in the standing task and $2e^6$ steps in the locomotion task. The model structures and hyperparameters are the same as in RWRL experiments and can be referred to Appendix B.3 and B.4. The regularizer coefficients for *Adv-USR* are 1e-3 and 1e-4 for two tasks.

Real Robot Evaluation After training, we directly deploy the learned policies on real robots. Since all sensors are internal on robots, there are no external sensors required. The control frequency is set as 33 Hz. We run each policy 50 episodes with 1000 steps and report the 10%-quantile of the final return and $5\% - 15\%$-quantile as the error bar in Figure 3.

Table 5: Unitree A1's Parameters in Simulation and Real Robot.

Parameters	**Simulation**	**Real Robot**
Mass (kg)	12	[10, 14]
Center of Mass (cm)	0	[-0.2, 0.2]
Motor Strength ($\times$ default value)	1.0	[0.8, 1.2]
Motor Friction (Nms/rad)	1.0	[0.8, 1.2]
Sensor Latency (ms)	0	[0, 40]
Initial position (m)	(0, 0, 0.25)	([-1, 1], [-1, 1], [0.2, 0.3])

C Extra Experimental Results

C.1 Constant Perturbation on System Parameters

Extra experimental results for task *cartpole_balance*, *walker_walk* and *quadruped_run* can be found in Table 6. We can observe similar results as in the main paper that both *L2-USR* and *L1-USR* can outperform the default version under some certain perturbations (e.g. *L1-USR* in *cartpole_balance* for pole_mass, *L2-USR* in *walker_walk* for thigh_length), while *Adv-USR* achieves the best average rank among all perturbed scenarios, showing the best zero-shot generalization performance in continuous control tasks. Notably, *L2-Reg* in *walker_walk* and *L1-Reg* in *quadruped_run* also achieve a competitive robust performance compared with *Adv-USR*. A possible reason is that, for environments with high-dimensional state and action spaces, some of them are redundant and direct regularization on value function's parameters is effective to perform dimensionality reduction and thus learns a generalized policy.

Table 6: *Robust-AUC* of all algorithms and their uncertainties on RWRL benchmark.

Task Name	**Variables**	**Algorithms**					
		None-Reg	*L1-Reg*	*L2-Reg*	*L1-USR*	*L2-USR*	*Adv-USR*
cartpole_balance	pole_length	981.45 (5.92)	**989.85 (3.74)**	989.33 (8.32)	798.07 (22.93)	944.89 (23.33)	959.66 (26.58)
	pole_mass	623.88 (28.64)	605.35 (55.60)	607.79 (23.18)	**632.54 (14.74)**	588.13 (38.33)	627.00 (22.90)
	joint_damping	970.83 (21.89)	978.97 (9.95)	982.71 (15.24)	**985.57 (10.01)**	978.62 (17.52)	982.43 (130.03)
	slider_damping	999.44 (0.26)	999.30 (0.43)	999.34 (0.57)	999.45 (0.31)	999.49 (0.48)	**999.55 (0.32)**
	average rank	4.00	4.00	3.25	2.75	4.50	**2.50**
walker_walk	thigh_length	315.64 (37.24)	237.90 (25.04)	345.12 (40.30)	316.61 (37.86)	**350.01 (34.37)**	318.88 (53.73)
	torso_length	498.01 (54.04)	300.39 (114.06)	533.96 (47.73)	**550.44 (50.83)**	543.39 (42.52)	543.91 (54.36)
	joint_damping	364.70 (50.33)	283.19 (30.18)	**420.23 (51.84)**	357.39 (56.04)	356.22 (49.74)	368.35 (64.76)
	contact_friction	885.01 (27.47)	714.94 (27.15)	**907.13 (18.94)**	897.65 (23.49)	900.58 (21.46)	902.03 (24.68)
	average	4.50	6.00	**2.00**	3.25	3.00	2.25
quadruped_run	shin_length	204.14 (91.36)	**280.11 (61.49)**	168.95 (38.19)	246.43 (117.07)	214.18 (56.06)	250.07 (79.37)
	torso_density	321.24 (76.70)	**417.68 (88.55)**	252.37 (88.41)	319.43 (90.79)	225.32 (80.49)	383.14 (67.34)
	joint_damping	367.05 (139.61)	641.08 (19.12)	687.42 (12.85)	324.38 (14.73)	**692.02 (6.98)**	664.25 (19.35)
	contact_friction	**654.43 (57.94)**	614.21 (76.60)	473.58 (61.72)	632.64 (95.18)	624.32 (124.39)	537.19 (76.22)
	average rank	3.75	**3.00**	5.00	**3.00**	3.25	**3.00**

C.2 Constant Perturbation on Multiple System Parameters

In real-world scenarios, there would be uncertainties in all system parameters. We provide the following additional experimental results to show the robustness when all parameters are perturbed simultaneously. The specific environmental setup is that all 4 parameters are perturbed simultaneously during testing. The perturbation intensity grows from 0 to 1. 0 resembles training environments without perturbations and 1 represents the allowed maximum perturbed values in Table 2. We adopt the same metric *Robust-AUC* and report it in the following table. All methods become less robust due to the increasing difficulty of perturbations, but *Adv-USR* still outperforms others.

Table 7: *Robust-AUC* of all algorithms and their uncertainties on RWRL benchmark.

Task Name	Algorithms					
	None-Reg	*L1-Reg*	*L2-Reg*	*L1-USR*	*L2-USR*	*Adv-USR*
cartpole_swingup	867.42 (0.27)	856.87 (0.52)	866.99 (0.23)	867.61 (0.44)	867.45 (0.26)	**881.36 (0.21)**
walker_stand	254.04 (32.91)	235.36 (25.29)	254.64 (35.01)	262.57 (25.02)	263.35 (24.85)	**266.97 (7.75)**
quadruped_walk	522.98 (34.93)	524.24 (85.03)	525.51 (24.58)	525.14 (85.68)	506.17 (54.92)	**534.61 (18.25)**

C.3 Noisy Perturbation on System Parameters

One may also be interested in the noisy perturbation setup where system parameters keep changing at every time step. This setup extends the Robust RL framework where the perturbation is fixed throughout the whole episode. The specific experimental setup noisy perturbation is as follows: the environmental parameter starts from the nominal value and follows a zero-mean Gaussian random walk at each time step. The nominal value and the standard deviation of the Gaussian random walk are recorded in Table 8. The experimental result on *quadruped_walk* is shown in Figure 4. In this experiment, *L1-Reg* achieves the best robustness, while our method *Adv-USR* achieves Top-2 performance in 3 out of 4 perturbations. While *L1-Reg* performs less effectively in the case of fixed perturbation, it implies that different regularizers do have different impacts on these two types of perturbations. For noisy perturbation, environmental parameters walk randomly around the nominal value and reach the extreme value less often, which requires a less conservative robust RL algorithm. Our algorithm *Adv-USR*, originally designed for fixed perturbation problem, achieves good but not the best performance, which leads to an interesting future research direction on the trade-off between robustness and conservativeness.

Table 8: The perturbed variables for the noise perturbation experiment.

Task Name	Perturbed Variables	Start Value	Step Standard Deviation	Value Range
quadruped_walk	shin_length	0.25	0.1	[0.25, 2.0]
	torso_density	1000	500	[500, 10000]
	joint_damping	30	10	[10, 150]
	contact_friction	1.5	0.5	[0.1, 4.5]

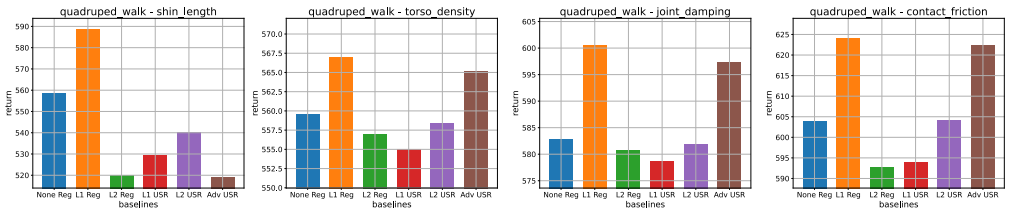

Figure 4: The parameter-return bar graph of all algorithms. All bars represent 10%-quantile value of episodic return under noisy environmental parameters.

C.4 Computational Cost of All Algorithms

We report the average computation time (in milliseconds) for a single value update of all algorithms in Table 9. We notice that the computation of all algorithms increases as the environment's complexity grows, and *L1-Reg*, *L1-Reg*, *Adv-USR*'s complexities are acceptable compared with other baselines ($\times\ 1 \sim 1.25$ time cost). The computation only becomes a problem when applying *USR* methods to dynamics with millions of parameters (common in model-based RL [44]). To tackle this issue, we can identify important parameters to reduce computation costs, as stated in Section 7.

Table 9: The computational cost (in milliseconds) for each value update of Robust RL algorithms.

Task Name	Algorithms					
	None-Reg	*L1-Reg*	*L2-Reg*	*L1-USR*	*L2-USR*	*Adv-USR*
cartpole_balance *cartpole_swingup*	14.72 ± 1.57	16.48 ± 1.68	17.05 ± 1.63	17.62 ± 1.49	22.48 ± 3.21	22.48 ± 3.21
walker_stand *walker_walk*	15.71 ± 1.23	18.89 ± 1.58	17.52 ± 1.92	18.06 ± 1.80	18.39 ± 1.90	23.16 ± 1.72
quadruped_walk *quadruped_run*	15.93 ± 1.68	19.47 ± 1.47	19.56 ± 1.67	20.79 ± 4.20	19.13 ± 1.98	25.23 ± 2.14

Theoretically, the additional computational cost largely depends on the norm term $\|\nabla_w P(s'|s,a;\bar{w})V^\pi(s')\|_2$ in Equation 5, time complexity is $O(W)$ (W is the number of parameters).

D Extra Related Work

Action Perturbation. Early works in Robust RL concentrated on action space perturbations. Pinto et al. [42] first proposed an adversarial agent perturbing the action of the principle agent, training both alternately in a mini-max style. Tessler et al. [49] later performed action perturbations with probability α to simulate abrupt interruptions in the real world. Afterwards, Kamalaruban et al. [50] analyzed this mini-max problem from a game-theoretic perspective and claimed that an adversary with mixed strategy converges to a mixed Nash Equilibrium. Similarly, Vinitsky et al. [51] involved multiple adversarial agents to augment the robustness, which can also be explained in the view of a mixed strategy.

State Perturbation. State perturbation can lead to the change of state from s to s_p, and thus might worsen an agent's policy $\pi(a|s)$ [52]. Zhang et al. [53], Oikarinen et al. [54] both assume an L_p-norm uncertainty set on the state space (inspired by the idea of adversarial attacks widely used in computer vision [55]) and propose an auxiliary loss to encourage learning to resist such attacks. It is worth noting that state perturbation is a special case of transition perturbation, which can be covered by the framework proposed in this paper, as further explained in Appendix E.

Reward Perturbation. The robustness-regularization duality has been widely studied, especially when considering reward perturbations [21, 22, 23]. One reason is that the policy regularizer is closely related to a perturbation on the reward function without the need for a rectangular uncertainty assumption. However, it restricts the scope of these works as reward perturbation, since it can be shown to be a particular case of transition perturbation by augmenting the reward value in the state [22]. Besides, the majority of works focus on the analysis of regularization to robustness, which can only analyze the effect of existing regularizers instead of deriving novel regularizers for robustness as in the work we present here.

Bayesian RL. One commonality between Bayesian RL and Robust RL is that they both store uncertainties over the environmental parameter (posterior distribution $q(w)$ in Bayesian RL and uncertainty set Ω_w in Robust RL). Uncertainties learned in Bayesian RL can benefit Robust RL in two ways: (1) Robust RL can define an uncertainty set $\Omega_w = \{w : q(w) > \alpha\}$ to learn a robust policy that can tolerate model errors, which is attractive for offline RL and model-based RL; (2) A soft robust objective with respect to the distribution $q(w)$ can ease the conservative behaviours caused by the worst-case scenario [18].

Adaptive RL Adaptive RL (often referred as Meta RL [56]) is another popular technique to deal with the perturbations in environments parallel to Robust RL introduced in this paper. The main difference between Robust RL and Adaptive RL is whether policy parameters are allowed to change when environmental parameters vary. Robust RL is a zero-shot learning technique aiming to learn one single robust policy that can be applied to various perturbed environments. Adaptive RL is a few-shot learning technique aiming to quickly change policy to adapt to the changing environments.

These two techniques can be combined to increase the robustness in real-world robots. One can first use Robust RL to learn a base policy as the warm start and fine-tune the policy on certain perturbed environments with Adaptive RL.

E Comparison with State Perturbation

State perturbation describes the uncertainties in the output space of the dynamics model, which is a special case of transition perturbation. We illustrate how to transform the State perturbation into transition perturbation in the following case.

Considering a L_2 uncertainty set on the output of the dynamics model, $\Omega_{s_p} = \{s_p | s' \sim P(\cdot|s, a; \bar{w}), \|s' - s_p\| \leq 1\}$. We can rewrite the next State distribution $s' \sim P(\cdot|s, a; \bar{w})$ as $s' = f(s'|s, a; \bar{w}) + \eta$, where η is a random noise $\eta \sim \mathcal{N}(0, 1)$. Then the perturbed State can be written as $s_p = f(s'|s, a; \bar{w}) + \eta + \beta, \|\beta\| \leq 1$. Viewing β as one additional parameter in the dynamics model, the uncertainty set on β is actually $\Omega_\beta = \{\|\beta\| \leq 1\}$. Based on this uncertainty set on β, one can further design corresponding regularizers on value function to increase the robustness as discussed in the paper.

If the uncertainty set on state space is unknown, denoted as Ω_β, it is still feasible to include β as an additional parameter in the dynamics model. As a result, *Adv-USR* could still be used to handle this unknown uncertainty set on β.