5 Appendix

The appendix is organized as follows:

1. Subsection 5.1 gives simple steps to use our algorithm for any robot learning problem.
2. Subsection 5.2 presents the Wilcoxon p-values for Table 3.
3. Subsection 5.3 details the Model Predictive Control formulation we used in all our experiments.
4. Subsection 5.4 describes the DNN architectures, training hyper-parameters, and train/test datasets for all experiments. Specifically, Subsections 5.4.1 - 5.4.3 describe the Toy 2D-Planning experiments, RoboSuite Manipulation experiments, and X-plane experiments respectively.
5. Subsection 5.5 provides an ablation study showing how to tune the weight for the consistency loss κ that determines how adversarial rendered scenarios are.
6. Finally, Figs. 11, 12, and 13 depict adversarial examples for all experiments.
7. We have also submitted all the code and will be making it **open-source**.

5.1 Steps to use Task-Driven Training for Robust Robotic Learning

1. Train a VAE (or any generative model) on the training dataset.
2. Train a task model (i.e., perception and planning model) on the training data, $\mathscr{D}^{\text{train}}_{\text{orig}}$. Note that the perception network does not need to be a VAE Encoder. The task model can be any network, such as MLPs, CNNs, or Transformers.
3. Use Alg. 1 to generate a new adversarial synthetic dataset $\mathscr{D}^{\text{train}}_{\text{adv}}$.
4. Retrain the task model on the combined dataset $\mathscr{D}^{\text{train}}_{\text{orig}} \cup \mathscr{D}^{\text{train}}_{\text{adv}}$.

Scaling to the real-world: Scaling to the real world mainly depends on the output quality of the differentiable renderer (VAE in our experiments). Therefore, if the differentiable renderer can generate real-world images, our adversarial learning procedure will be able to find examples that lead to a higher model-based controller cost. Recently, there have been significant improvements in generative models (used as differentiable renders in our work), such as realistic VAEs [42], transformers (DALLE-2) [43], and diffusion models [44], which can be used as the differentiable renderer in our proposed algorithm. Therefore, due to the flexibility of our proposed algorithm, we can plug in any generative model to generate adversarial scenarios for more realistic and unstructured environments.

Self-driving cars can be a real-world use case of our method. LidarSim [45] and SurfelGan [46] are generative models to synthesize realistic self-driving scenarios, trained on a vast amount of self-driving scenarios (Waymo and Uber datasets). Using our proposed method, we can use these differentiable simulators as differentiable renderers to synthesize new adversarial scenarios.

5.2 Wilcoxon p-values

The difference between standard data augmentation and our task-driven approach is **statistically significant** for both control cost and waypoint error with a Wilcoxon p-value much below 0.01.

Domain	Dataset	Avg MPC Cost Difference Wilcoxon p-value	Avg Waypoint Error Difference Wilcoxon p-value
Toy	$\mathscr{D}^{\text{test}}_{\text{OoD}}$	$1e^{-4}$	$1e^{-4}$
	$\mathscr{D}^{\text{test}}_{\text{adv}}$	0	$1e^{-4}$
Manipulation	$\mathscr{D}^{\text{test}}_{\text{OoD}}$	$1e^{-4}$	$1e^{-4}$
	$\mathscr{D}^{\text{test}}_{\text{adv}}$	0	$1e^{-4}$
X-plane	$\mathscr{D}^{\text{test}}_{\text{night}}$	0	$1e^{-4}$
	$\mathscr{D}^{\text{test}}_{\text{adv}}$	0	$1e^{-4}$

Figure 6: **TASK-DRIVEN vs DATA AUGMENTATION schemes for MPC Costs and Waypoint Error Wilcoxon P-values**: We present the corresponding Wilcoxon p-values for Table 3.

5.3 MPC Cost Functions

Toy 2-D Planning and X-Plane: *Differentiable Model Predictive Control (MPC):* The planning module provides the MPC controller with a waypoint vector w_F for F future timesteps to track. The robot's state at discrete time t is denoted by $\zeta_t \in \mathbb{R}^{4\times 1}$, defined as $\zeta_t = \{l_x, l_y, v, \theta\}$ for both experiments. Here, l_x and l_y are the robot's x, y location in meters, v is the speed in meters/sec, and θ is the robot's heading angle in radians. The robot's control input at time t is defined as $u_t = \{a, \gamma\}$, where a is the robot's acceleration in m/s^2 and γ is the robot's steering input in radians/sec.

Our main goal is to formulate the task cost $J(w_F)$, which is a standard quadratic cost penalizing state deviations and control effort. Namely, $\mathbf{Q} \in \mathbb{R}^{4\times 4}$ is the state cost matrix and $\mathbf{R} \in \mathbb{R}^{2\times 2}$ is the control cost matrix. The matrices $\mathbf{A} \in \mathbb{R}^{4\times 4}$ and $\mathbf{B} \in \mathbb{R}^{4\times 2}$ define the robot's dynamics. For the toy 2-D planning experiment, we use standard double integrator dynamics while the X-Plane experiment uses linearized unicycle dynamics, which is common for experiments on the TaxiNet DARPA Assured Autonomy Dataset [38]. The robot has box control constraints $u_{min} \leq u_t \leq u_{max}$ bounding the minimum and maximum acceleration and steering angle. Likewise, box state constraints $\zeta_{min} \leq \zeta_t \leq \zeta_{max}$ define limits on the velocity and linear obstacle avoidance constraints. These constraints, used only in our first motion planning example, were calculated by drawing perpendicular hyperplanes from the obstacles. Given a set of F future waypoints $w_F = \{w_0, w_1, \dots, w_{F-1}\}$ for a planning horizon starting at $t = 0$, our MPC task cost and problem become:

$$\begin{aligned}
\min_{\zeta_{0:F}, u_{0:F-1}} \quad & J(w_F) = \sum_{t=0}^{F} (w_t^\top - \zeta_t^\top)\mathbf{Q}(w_t - \zeta_t) + \sum_{t=0}^{F-1} u_t^\top \mathbf{R} u_t \\
\text{such that} \quad & \zeta_{t+1} = \mathbf{A}\zeta_t + \mathbf{B}u_t \\
& u_{min} \leq u_t \leq u_{max} \\
& \zeta_{min} \leq \zeta_t \leq \zeta_{max}.
\end{aligned} \tag{2}$$

RoboSuite Manipulation: The MPC problem is discussed in Subsec. 5.4.2.

5.4 Experiment Details

5.4.1 Illustrative Toy Example: 2D Obstacle Avoidance.

We now introduce a simple task where a planar 2-D robot has to navigate around obstacles to reach a goal location, as pictured in Fig. 7. This experiment demonstrates how adversarial training creates more challenging scenarios for the task model to plan in. Furthermore, we quantitatively show that a task model trained on adversarial scenarios $\mathscr{D}^{\text{train}}_{\text{adv}}$ learns to avoid collisions more successfully and with lower control cost compared to standard training procedures.

Visual Input: The input to the 2-D planar robot is a 100×100 grey-scale image s of the scene, including obstacles and the goal g, indicated by a star-shaped marker. Given the sensory input s, the task model $w_F = \phi_{\text{task}}(s, w_P)$ is a perception-planning module that outputs a sequence of future waypoints w_F for the robot to navigate. A waypoint w_F is simply a pose the robot must move to in the xy plane. The task model ϕ_{task} uses a history of past waypoints w_P to improve its predictions. The linear MPC task cost is defined in Eq. 2.

Training and Test Datasets: We synthetically created the training $\mathscr{D}^{\text{train}}_{\text{orig}}$ and the held-out test datasets $\mathscr{D}^{\text{test}}_{\text{orig}}$ and $\mathscr{D}^{\text{test}}_{\text{OoD}}$ for the experiments. As shown in Fig. 7, each scenario had randomly generated ellipsoid obstacles with random sizes and locations. The ground truth waypoints $y = w_F$ are generated using a standard Frenet Planner [34] that has full access to all obstacles and start/goal locations for the entire scene. The synthetic dataset contained $|\mathscr{D}^{\text{train}}_{\text{orig}}| = 7000$ training scenarios and $|\mathscr{D}^{\text{test}}_{\text{orig}}| = 2200$ test scenarios. A larger similar dataset had $|\mathscr{D}^{\text{train}}_{\text{add}}| = 10000$ training examples. Another dataset of $|\mathscr{D}^{\text{train}}_{\text{aug}}| = 10000$ training examples was generated from task-agnostic data augmentation. The synthetic adversarial dataset generated by our method had $|\mathscr{D}^{\text{train}}_{\text{adv}}| = 10000$ training scenarios and $|\mathscr{D}^{\text{test}}_{\text{adv}}| = 2200$ adversarial test scenarios. We used $K = 10$ gradient steps in Algorithm 1 to generate the adversarial scenarios. Finally, we also generated a challenging OoD test dataset, $\mathscr{D}^{\text{test}}_{\text{OoD}}$, where the ellipsoid obstacles were samples from a different distribution than the original

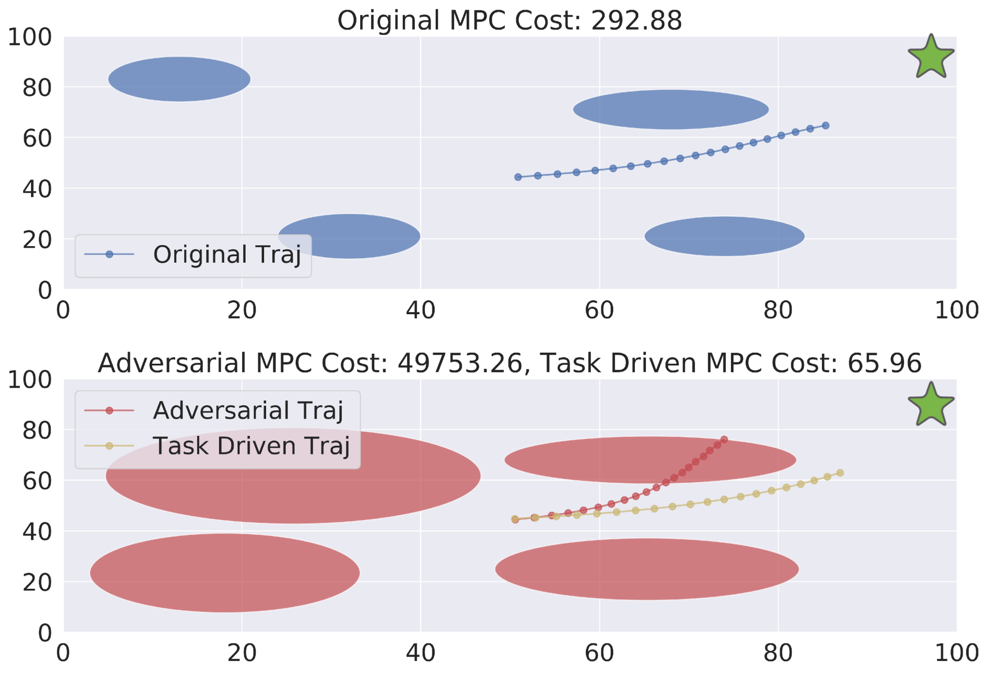

Figure 7: **Our method's Synthesized Adversarial Scenarios for Toy 2-D Planning:** The robot's goal is to navigate from a start location at $(0,0)$ to the goal at $(100,100)$ (green star). Original Scene (Top): The blue obstacles (4 ellipsoids) represent the original scene. Adversarial Scene (Bottom): The red obstacles in the bottom are adversarially created using our method for the original scenario above it. The waypoint trajectories (small sequential dots) are generated by the task model for the corresponding scene, and MPC task cost is given in the title. Clearly, the adversarial scenario generates much higher MPC costs by moving the obstacles in the robot's intended path. Crucially, the waypoint trajectory generated by our *re-trained* TASK DRIVEN scheme (yellow) learns to better avoid the obstacles with lower MPC cost. Fig. 11 shows additional examples.

dataset $\mathscr{D}_{\text{orig}}^{\text{train}}$. Specifically, ellipsoid obstacles were bigger and closer in the OoD test dataset, thus making it hard for the robot to plan.

DNN architectures: We use a history of $P = 10$ past waypoints to predict the next $F = 20$ future waypoints. In this experiment, the VAE Encoder is made up of 3 CNN layers with filter sizes of $[(3,3),(4,4),(5,5)]$, strides of $[1,2,2]$, and padding of $[1,1,2]$ respectively. Similarly, the decoder is made up of 3 CNN layers with filter sizes of $[(6,6),(6,6),(5,5)]$, strides of $[2,2,1]$, and padding of $[2,2,2]$ respectively. Rather than hand-tuning a desired weight between reconstruction error and KL divergence with a prior in the VAE loss, we use a σ-VAE [14] which allows for this tuning to happen automatically. We used a VAE with latent size of 20. The VAE is trained with the ADAM optimizer with a learning rate of $1e-3$. The encoder is used for the perception module $\phi_{\text{perc.}}$ and the decoder is used for the differentiable rendering module ψ_{render}. Note that the encoder is re-trained after adversarial scenario generation.

Additionally, for the planning module, ϕ_{plan} is a NN consisting of three fully connected layers with ReLU activation functions with output sizes $[80,60,40]$ respectively. The planning module, ϕ_{plan}, is trained with the ADAM optimizer with a learning rate of $1e-3$. The robot's past waypoints w_P are of length $P = 10$ and the future waypoints w_F are of length $F = 20$. All the planning modules, ϕ_{plan}, were trained for 2000 epochs with early stopping. We experimented with multiple κ values (in loss Eq. 1) and finally used $\kappa = 30$ for the toy experiment. We found our results to be quite robust across a wide range of values κ.

Quantitative results: Table 8 shows the number of scenarios with collisions for all training schemes. Our TASK DRIVEN approach was able to perform as well as the DATA AUGMENTATION , DATA ADDED , and ORIGINAL benchmarks on the original test dataset. Additionally, the TASK DRIVEN approach significantly outperformed the DATA AUGMENTATION scheme by 49%, DATA ADDED scheme by 67% and ORIGINAL scheme by 69% on the adversarial test dataset $\mathscr{D}_{\text{adv}}^{\text{test}}$. Furthermore, our TASK DRIVEN approach outperformed the DATA AUGMENTATION scheme by 15.8%, DATA ADDED scheme by 30.8% and ORIGINAL scheme by 38.6% on the unseen OoD test dataset $\mathscr{D}_{\text{OoD}}^{\text{test}}$. Thus, our training scheme TASK-DRIVEN leads to significantly fewer collisions on challenging OoD scenarios.

Figure 8: **Our method improves collision avoidance on challenging motion planning environments**: We show the number of collisions with an obstacle for different training schemes on the original, OoD, and synthetic adversarial test environments. A single collision occurs when the MPC-enacted trajectory intersects an ellipsoidal obstacle. The columns represent how many scenarios of the 2200 test scenarios had even a single collision for held-out test datasets $\mathscr{D}_{\text{orig}}^{\text{test}}$, $\mathscr{D}_{\text{OoD}}^{\text{test}}$ and $\mathscr{D}_{\text{adv}}^{\text{test}}$ respectively. The best performance is bolded and rows correspond to benchmark schemes.

Algorithm	$\mathscr{D}_{\text{orig}}^{\text{test}}$	$\mathscr{D}_{\text{OoD}}^{\text{test}}$	$\mathscr{D}_{\text{adv}}^{\text{test}}$
ORIGINAL	389	585	1058
DATA-ADDED	**314**	552	1044
DATA AUGMENT	321	489	933
TASK-DRIVEN (OURS)	334	**422**	**623**

5.4.2 RoboSuite Manipulation

We now introduce the RoboSuite manipulation environment, where the Franka Emika Panda arm has to navigate around red box obstacles to reach a blue ball, as pictured in Fig. 9. This experiment demonstrates how adversarial training creates more challenging scenarios for the task model to plan in. Furthermore, we quantitatively show that a task model trained on adversarial scenarios $\mathscr{D}_{\text{adv}}^{\text{train}}$ learns to avoid collisions with red boxes more successfully and with lower control cost compared to standard training procedures. The main difference between this domain and the Toy 2-D Planning domain is that inputs scenes for the manipulation task are much more realistic.

Visual Input: The input to the arm is a $64 \times 64 \times 3$ RGB image s of the scene, including red box obstacles and the blue sphere. Given the sensory input s, the task model $w_F = \phi_{\text{task}}(s, w_P)$ is a perception-planning module that outputs a sequence of future waypoints w_F for the arm end-affector to navigate. A waypoint w_F is simply a pose the arm end-affector must move to in the xy plane. The task model ϕ_{task} uses a history of past waypoints w_P to improve its predictions. The linearized MPC task space cost is defined in Eq. 2. Specifically, when solving the MPC problem, we treat the middle of the arm-gripper as a point mass trying to reach the blue ball, while avoiding the red box obstacles. After doing this relaxation, we can use the same MPC formulation as shown in Eq. 2. The states ζ_t and the control inputs u_t will relate to the "virtual" point mass at the middle of the arm-gripper. Finally, after obtaining a sequence of MPC waypoints, we used Robosuite's built-in joint space controller, OSC_POSE, to track MPC's state trajectory, $\zeta_{0:F}^{\star}$. In all our experiments, the joint space controller was able to generate joint poses to reach the desired task space configuration.

Training and Test Datasets: We create the training $\mathscr{D}_{\text{orig}}^{\text{train}}$ and the held-out test datasets $\mathscr{D}_{\text{orig}}^{\text{test}}$ by randomly placing all the red boxes and the blue sphere on a table for the arm to navigate to. The ground truth waypoints $y = w_F$ are generated using a standard Frenet Planner [34] that has full access to all obstacles and start/goal locations for the entire scene. The synthetic dataset contained $|\mathscr{D}_{\text{orig}}^{\text{train}}| = 20000$ training scenarios and $|\mathscr{D}_{\text{orig}}^{\text{test}}| = 5000$ test scenarios. A larger similar dataset had $|\mathscr{D}_{\text{add}}^{\text{train}}| = 30000$ training examples. Another dataset of $|\mathscr{D}_{\text{aug}}^{\text{train}}| = 20000$ training examples was generated from task-agnostic data augmentation. The synthetic adversarial dataset generated by our method had $|\mathscr{D}_{\text{adv}}^{\text{train}}| = 10000$ training scenarios and $|\mathscr{D}_{\text{adv}}^{\text{test}}| = 5000$ adversarial test scenarios. We used $K = 5$ gradient steps in Algorithm 1 to generate the adversarial scenarios. Finally, we also generated a challenging OoD test dataset, $\mathscr{D}_{\text{OoD}}^{\text{test}}$, where the red box obstacles were samples from a different distribution than the original dataset $\mathscr{D}_{\text{orig}}^{\text{train}}$. Specifically, red box obstacles were bigger and closer in the OoD test dataset, thus making it hard for the arm to reach the blue sphere.

DNN Architectures: We use a history of $P = 5$ past waypoints to predict the next $F = 5$ future waypoints. In this experiment, we used a Soft-Intro VAE [13] in order to deal with realistic images. We used a Soft-Intro VAE with a latent size of 500. The Soft-Intro VAE is trained with the ADAM

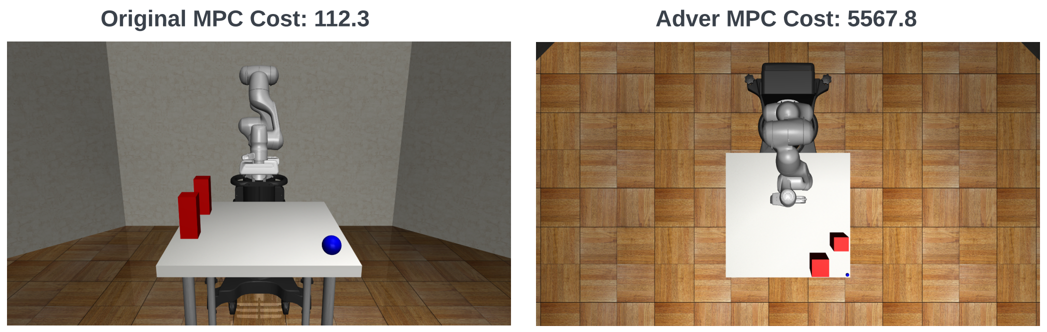

Figure 9: **Our method's Synthesized Adversarial Scenarios for Robosuite Manipulation:** The arm's goal is to navigate from a start location to pick up the blue ball. Original Scene (Left): The two red box obstacles and the blue sphere were randomly placed in the the original scene. Adversarial Scene (Right): The red box obstacles are adversarially replaced by our method starting from the original scenario (left). The adversarial scenario generates much higher MPC costs by moving the obstacles in the arm's intended path and placing them such that arm will have to do big swerves to reach the blue ball.

optimizer with a learning rate of $1e-3$. The encoder is used for the perception module $\phi_{\text{perc.}}$ and the decoder is used for the differentiable rendering module ψ_{render}. Note that the encoder is re-trained after adversarial scenario generation.

Additionally, for the planning module, ϕ_{plan} is a NN consisting of three fully connected layers with ReLU activation functions with output sizes $[1024, 512, 20]$ respectively. The planning module, ϕ_{plan}, is trained with the ADAM optimizer with a learning rate of $1e-3$. The robot's past waypoints w_P are of length $P = 5$ and the future waypoints w_F are of length $F = 5$. All the planning modules, ϕ_{plan}, were trained for 2000 epochs with early stopping. We experimented with multiple κ values (in loss Eq. 1) and finally used $\kappa = 100$. We found our results to be quite robust across a wide range of values κ.

5.4.3 X-Plane: Autonomous Vision-Based Airplane Taxiing

As shown in Fig. 10, the airplane has a wing-mounted camera and passes the images through a perception model to estimate its distance and heading angle relative to the center-line of a runway. Then, it uses differentiable MPC to navigate to the runway center-line with minimal control cost. We use a public dataset from the photo-realistic X-Plane simulator [40] consisting of images from the plane's wing-mounted camera in diverse weather conditions and at various poses on the runway. This standard benchmark dataset has been used in recent works on robust and verified perception [37–39].

Visual Input: The input to our "robot" is a 128×128 RGB image s of the runway taken from the right wing. The task model $w_F = \phi_{\text{task}}(s, w_P)$ is a perception-planning module that outputs the desired pose of the airplane w_F, given this sensory input s and a history of waypoints w_P. Specifically, the task model outputs a single waypoint w_F, which is a tuple of the distance to the centerline of runway c and the heading angle θ, denoted by $w_F = (c_i, \theta_i)_{i=0}^{F}$ for $F = 1$. $w_P = (c_i, \theta_i)_{i=0}^{P}$ is a history of the $P = 10$ past waypoints which are used to improve the planning model's predictions. MPC with linearized dynamics (Eq. 2) is used to plan a sequence of controls to navigate to the center-line.

DNN Architectures: In this experiment, we used a Soft-Intro VAE [13] in order to deal with realistic images. We used a Soft-Intro VAE with a latent size of 500. The Soft-Intro VAE is trained with the ADAM optimizer with a learning rate of $1e-3$. The encoder is used for the perception module $\phi_{\text{perc.}}$ and the decoder is used for the differentiable rendering module ψ_{render}. Note that the encoder is re-trained after adversarial scenario generation.

Additionally, for the planning module, ϕ_{plan} is a NN consisting of three fully connected layers with ReLU activation functions with output sizes $[256, 128, 2]$ respectively. The planning module, ϕ_{plan}, is trained with the ADAM optimizer with learning rate of $1e-3$. The robot's past waypoints w_P are of length $P = 4$ and the future waypoints w_F are of length $F = 1$. All the planning modules, ϕ_{plan},

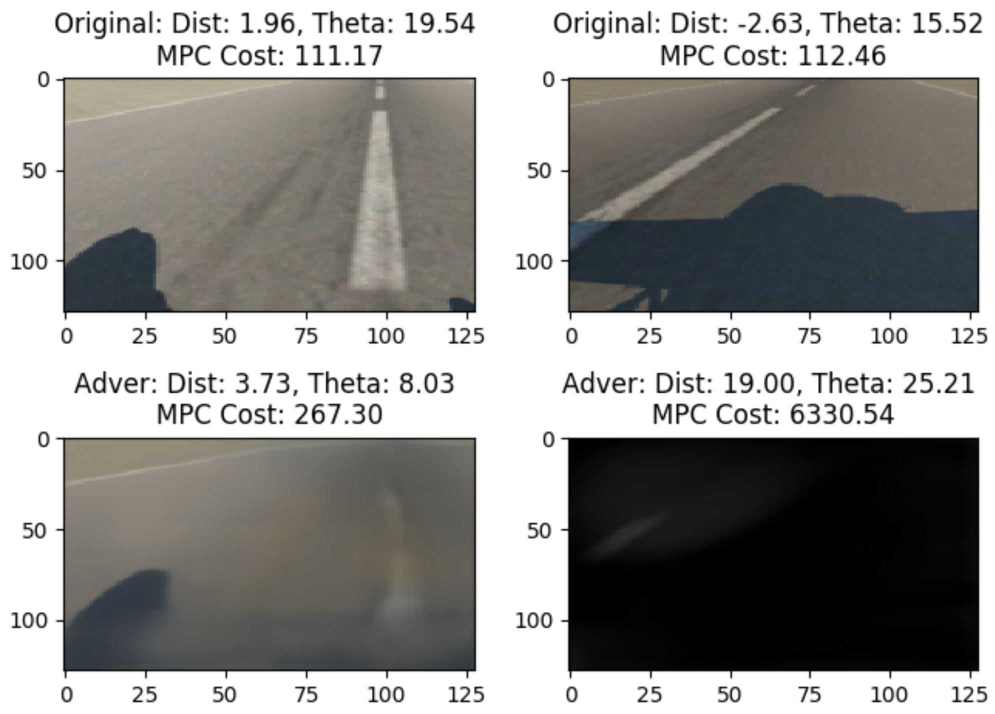
Original: Dist: 1.96, Theta: 19.54
MPC Cost: 111.17
Original: Dist: -2.63, Theta: 15.52
MPC Cost: 112.46
Adver: Dist: 3.73, Theta: 8.03
MPC Cost: 267.30
Adver: Dist: 19.00, Theta: 25.21
MPC Cost: 6330.54

Figure 10: **X-Plane Synthetic Adversarial Scenarios:** The top row contains images from the original dataset with the corresponding ground-truth label waypoint (distance to center-line and heading angle), as well as MPC task cost if this waypoint was correctly followed. The bottom row shows the corresponding adversarial scenario generated by our method. As expected, both the adversarial scenarios lead to higher MPC costs. On the left, the adversary learns to blur the runway center-line to emulate a foggy condition, while the right image emulates an OoD night scenario where only the center-line (top left) is faintly visible. Fig. 13 shows additional examples.

were trained for 2000 epochs with early stopping. We experimented with multiple κ values (in loss eq. 1) and finally used $\kappa = 100$ for the X-Plane experiment.

Images from Diverse Weather Conditions: We will represent the whole X-plane dataset as $\Gamma = \{\mathscr{D}_{\text{afternoon}}, \mathscr{D}_{\text{morning}}, \mathscr{D}_{\text{overcast}}, \mathscr{D}_{\text{night}}\}$, where each dataset corresponds to the subscripted weather condition. Each individual weather dataset in Γ has $44,000$ training scenarios and $11,000$ testing scenarios. Ground truth waypoints are obtained from the X-Plane simulator.

Our goal is to establish that we can re-train a perception model that robustly generalizes (with good control performance) to OoD weather conditions. To do so, we start with a perception model that is *initially* only trained on afternoon conditions and evaluate its performance on held-out test images across OoD weather conditions. To do so, we will compare 4 different task models ϕ_{task}. The ORIGINAL model is only trained on $\mathscr{D}^{\text{train}}_{\text{afternoon}}$, while the DATA ADDED model is trained on $\mathscr{D}^{\text{train}}_{\text{add}} = \mathscr{D}^{\text{train}}_{\text{afternoon}} \cup \mathscr{D}^{\text{train}}_{\text{morning}}$ since the morning and afternoon are visually similar.

Today's standard benchmark of task-agnostic data augmentation is trained on $\mathscr{D}^{\text{train}}_{\text{aug}} = \mathscr{D}^{\text{train}}_{\text{afternoon}} \cup \mathscr{D}^{\text{train}}_{\text{aug}}$, where we perform image augmentations (random crops, rotations, and hue alterations) on the original afternoon training data using the open-source Albumentations library [3]. Finally, our TASK DRIVEN model is trained on $\mathscr{D}^{\text{train}}_{\text{afternoon}} \cup \mathscr{D}^{\text{train}}_{\text{adv}}$, where $\mathscr{D}^{\text{train}}_{\text{adv}}$ are adversarial scenarios created using our method applied to the original afternoon training data. We trained the adversary for $K = 15$ steps in Algorithm 1. We quantitatively compare the performance of all benchmark task models on mean task cost J and waypoint MSE on held-out test datasets for each condition, such as afternoon, morning, night (OoD), and synthetic test images ($\mathscr{D}^{\text{test}}_{\text{adv}}$).

5.5 Consistency Loss Weight Ablation Study

This section will show how we can control how adversarial we want the synthetic dataset to be. As a reminder, the loss we used to train the adversary, Eq. 1, contained two parts: the MPC cost of the synthetic adversarial scenario and the consistency loss. The consistency loss calculates the distance between the original scene and the adversarial synthetic scene in the latent space of the renderer. We weigh the consistency loss by a scalar value, κ, which lets users decide how adversarial they want adversarial scenarios to be. From the loss function Eq. 1, it is easy to observe that higher values of κ will make the value of the consistency loss higher than that of the MPC cost. Therefore, the synthetic adversarial scenarios will be less adversarial. Similarly, for lower values of κ, the adversarial scenarios will be more adversarial and thus have higher MPC costs. Fig 14 shows this relation for the Toy 2-D Planning scenario, where we show the MPC Cost (y-axis) for 500 adversarial scenarios for different κ values (x-axis). Fig. 14 clearly shows that as we increase the κ values, the MPC costs of the synthesized adversarial scenarios decrease and are thus less adversarial. We also show that these results are statistically significant in Table 15, with Wilcoxon p-values being less than 0.05 for all cases. We empirically confirm the expected behavior for consistency loss weight κ.

5.6 Comparison with CURL

As an added baseline, we compare our method with CURL [33]. CURL extracts high-level features from raw pixels using contrastive learning and performs waypoint estimation on top of the extracted features (similar to our architecture). Contrastive representations are learned by specifying an anchor observation created using standard augmentation techniques. Specifically, we used random brightness, contrast, blur and crop to create the anchor observations. Then, CURL maximizes/minimizes the agreement between positive/negative pairs in contrastive learning through Noise Contrastive Estimation. CURL explicitly leads to significantly better feature extraction than standard data augmentation and thus speeds up training, converging 2x-3x faster than standard training procedures in our experiments. We used default training parameters for CURL. The learning rate was $3e^{-4}$, and τ (moving average weight) was $5e^{-4}$.

5.7 Adversarial Labels

We now discuss how to generate labels for adversarial images in Algorithm 1. We can use either the same labels or generate new labels depending on how much the adversarial and original scenes differ in terms of task completion and semantics. Here are a few examples.

Case 1: Adversarial Scene Differs Significantly So the Label Changes.

For the Toy 2D Navigation and Robosuite experiments (Left and middle for Fig. 4), the adversary significantly changes the history of waypoints. Thus, we use an "oracle" Frenet Planner [34] to re-plan a collision-avoiding trajectory for the adversarial scenarios and use the new waypoints as ground-truth labels for training.

Case 2: The image is slightly distorted, but the desired waypoint does not change. In X-Plane, the image is only slightly perturbed to emulate new weather or background scenarios due to the consistency loss term. However, the ground-truth location of the runway centerline remains the same since the real pose of the aircraft has not changed. Thus, our method uses the same label for the original and adversarial images. Essentially, our method generates adversarial images that teach the perception model that many possible visual distortions (due to weather/background variation) should map to the same true runway location. Teaching the perception model about these possible visual distortions during training improves its generalization when similar distortions occur in OoD scenarios.

Crucially, we note that using the same labels as the original dataset for image based-tasks is a standard practice in classical adversarial training papers [47–49]. This is because these methods consider small, bounded perturbations on images with the same semantic meaning (same class).

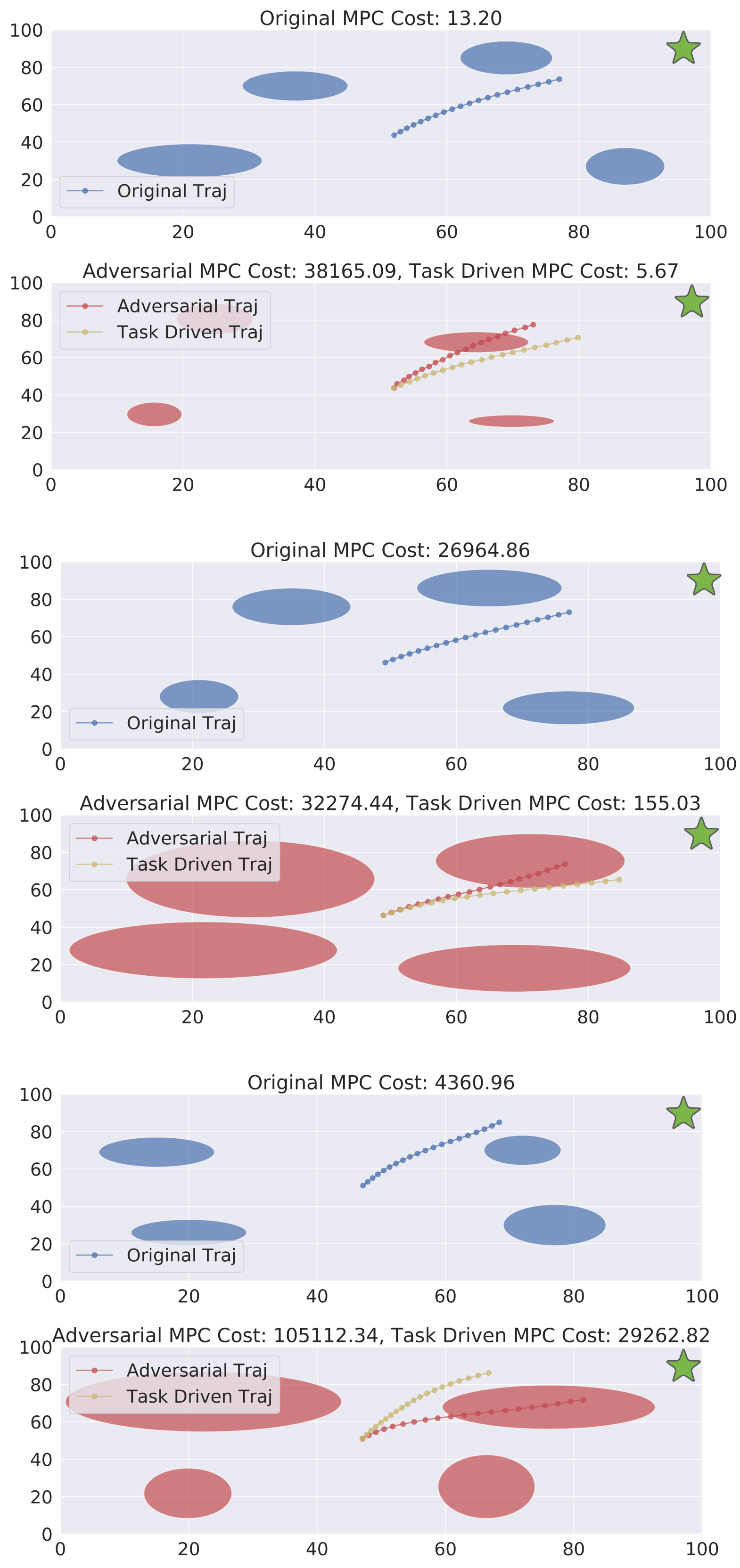

Figure 11: **Additional Synthesized Adversarial Scenarios for the Toy 2-D Planning Experiment**

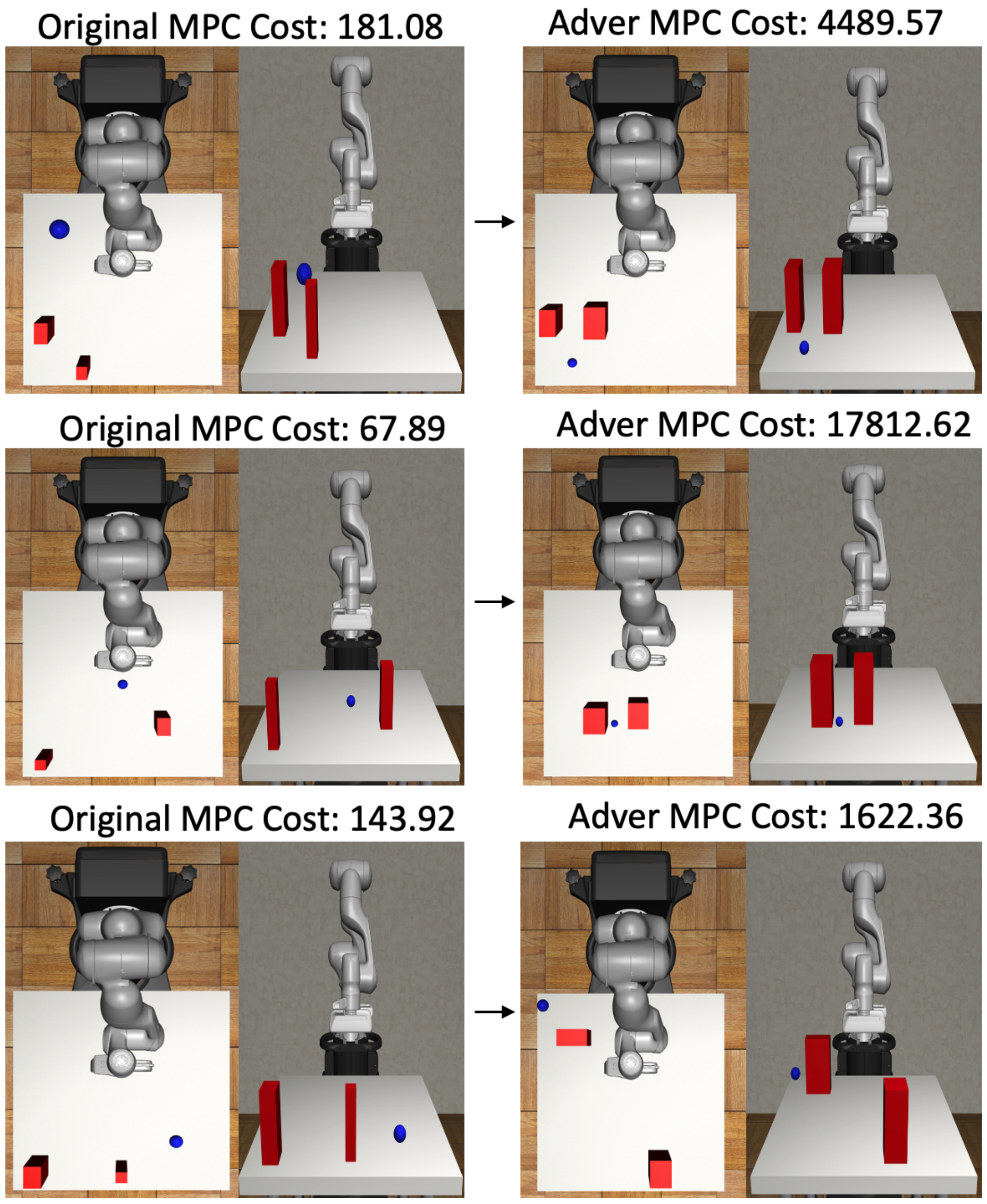

Figure 12: **Additional Synthesized Adversarial Scenarios for the Robosuite Manipulation Experiment**

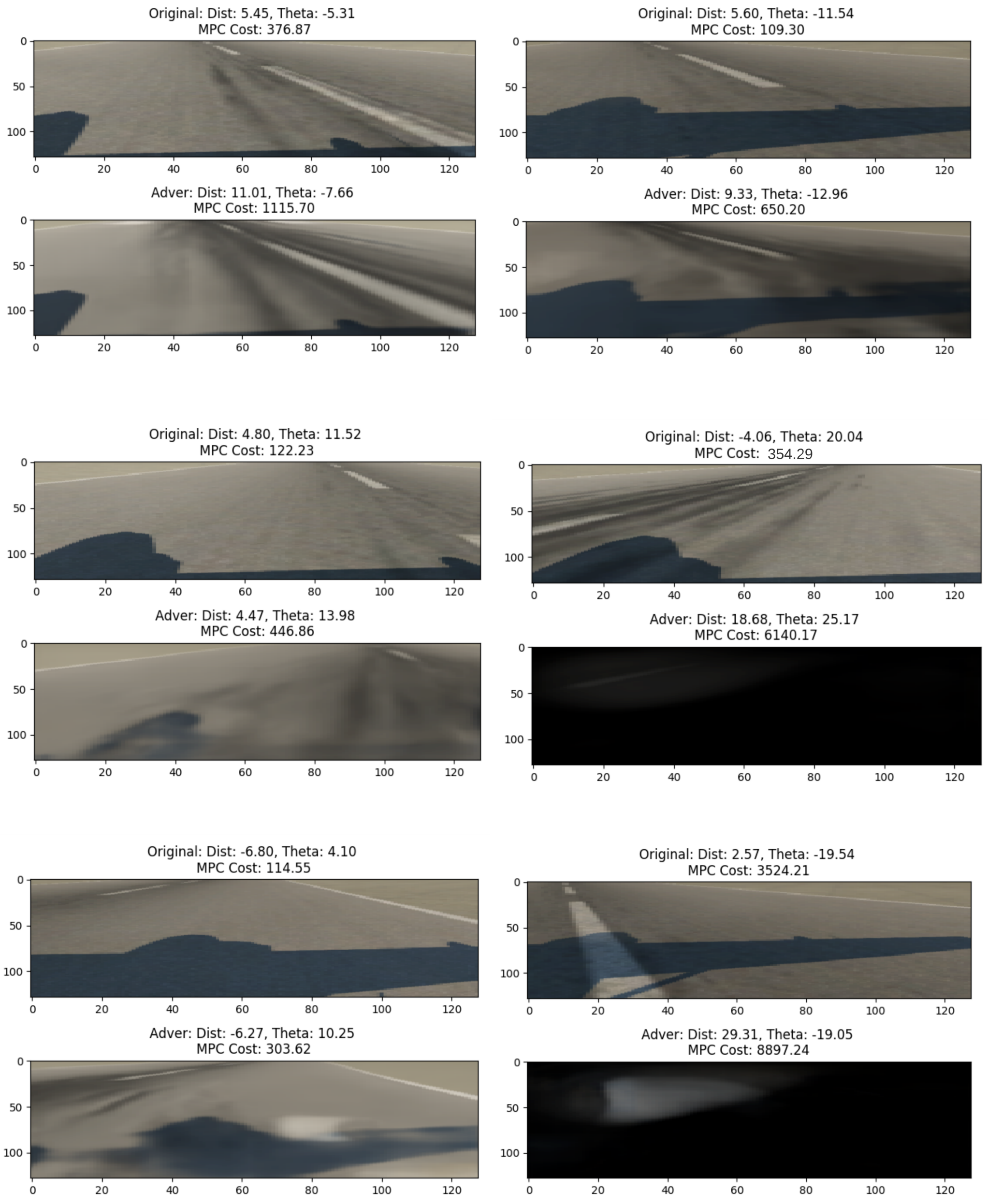
Original: Dist: 5.45, Theta: -5.31
MPC Cost: 376.87
Original: Dist: 5.60, Theta: -11.54
MPC Cost: 109.30
Adver: Dist: 11.01, Theta: -7.66
MPC Cost: 1115.70
Adver: Dist: 9.33, Theta: -12.96
MPC Cost: 650.20
Original: Dist: 4.80, Theta: 11.52
MPC Cost: 122.23
Original: Dist: -4.06, Theta: 20.04
MPC Cost: 354.29
Adver: Dist: 4.47, Theta: 13.98
MPC Cost: 446.86
Adver: Dist: 18.68, Theta: 25.17
MPC Cost: 6140.17
Original: Dist: -6.80, Theta: 4.10
MPC Cost: 114.55
Original: Dist: 2.57, Theta: -19.54
MPC Cost: 3524.21
Adver: Dist: -6.27, Theta: 10.25
MPC Cost: 303.62
Adver: Dist: 29.31, Theta: -19.05
MPC Cost: 8897.24

Figure 13: **Additional Synthesized Adversarial Scenarios for the X-plane Experiment**

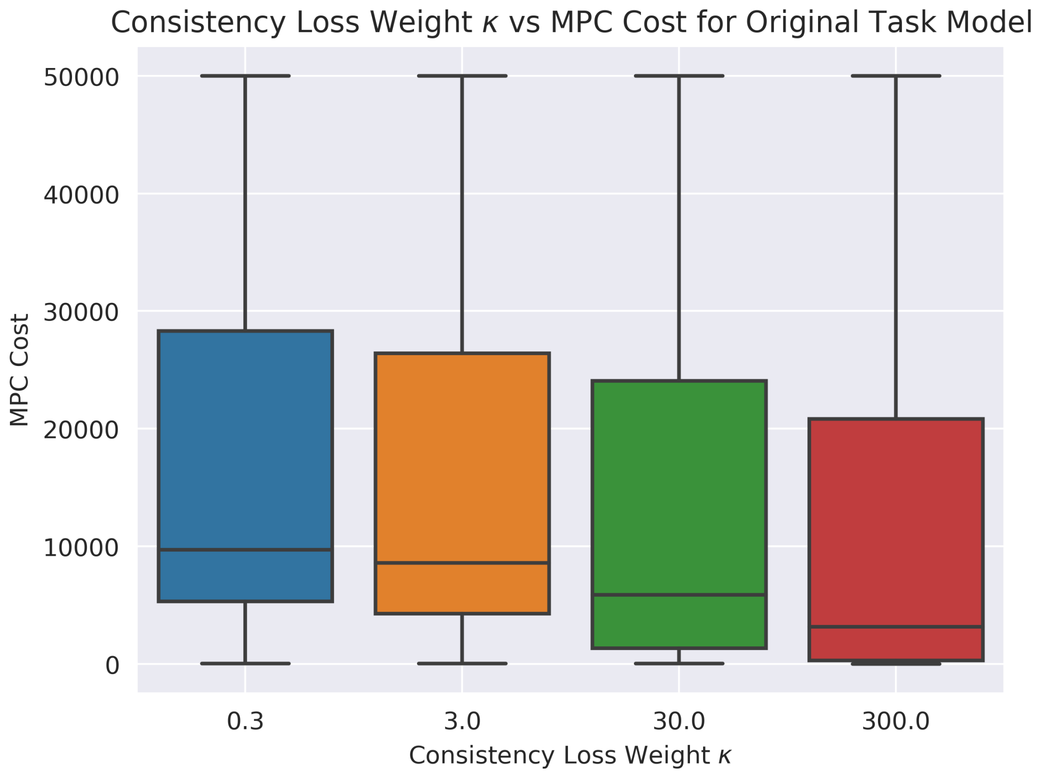

Figure 14: **Consistency Loss Weight Ablation Study for Toy 2D Planning:** The figure shows the relation of κ values to the MPC Cost of the corresponding synthesized adversarial scenarios for the Toy 2-D Planning experiment. On the x-axis, we have the κ value used in the adversary loss, Eq. 1, and on the y-axis, we have the corresponding MPC Cost. The plot shows that the user can easily control how adversarial the synthesized scenarios are by tuning the κ value. This plot also demonstrates that our loss function works as desired, as we can see the scenarios with higher κ values have lower MPC costs than scenarios with lower κ values.

Kappa Values	$\kappa = 3$	$\kappa = 30$	$\kappa = 300$
$\kappa = 0.3$	0.042	0.046	0.015
$\kappa = 3$	NA	0.048	0.039
$\kappa = 30$	NA	NA	0.026

Figure 15: **Consistency Loss Weight vs MPC Cost Wilcoxon P-values**: We present the corresponding Wilcoxon p-values for κ ablation study in Fig. 14.