Supplementary Material for Risk-averse Zero Order Trajectory Optimization

Anonymous Author(s)
Affiliation
Address
email

In this supplementary we provide additional details for our method. We also provide videos that showcase risk-averse behavior of RAZER at https://sites.google.com/view/razer-traj-opt.

Our research suffered from pandemic impacts on lab access which is detailed in Sec. E.

A Implementation Details

A.1 Model Learning

Parameters used for model learning in the *BridgeMaze* experiments.

Table S1: Model parameters for the *BridgeMaze* Environment.

Ensemble parameters

Name	Value
num_layers	6
size	400
activation	silu
ensemble_size (n)	5
output_activation	None
l1_reg	0
weight_initializer	truncated_normal
bias_initializer	0
use_spectral_normalization	False

Stochastic NN parameters

Name	Value
var_clipping_low	-10.0
var_clipping_high	4
state_dependent_var	True
regularize_automatic_var_scaling	False

Remaining parameters

Name	Value
lr	0.002
grad_norm	2.0
batch_size	512
weight_decay	$1e^{-5}$
use_input_normalization	True
use_output_normalization	False
epochs	25
predict_deltas	True
train_epochs_only_with_latest_data	False
iterations	0
optimizer	Adam
propagation_method	TS1
sampling_method	sample

We bound the predicted log variance by applying (as in [1, A.1])

```
logvar = max_logvar - softplus(max_logvar - logvar)

logvar = min_logvar + softplus(logvar - min_logvar)
```

to the output of the network that predicts the log variance, `logvar`. In principle, we could differentiate through this bound to automatically adjust the bounds `max_logvar` and `min_logvar`. However, we decided to not make these parameters learnable.

Parameters used for model learning in the *Noisy-HalfCheetah* environment (only differences to *BridgeMaze* environment).

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

Table S2: Model parameters (only differences wrt S1 are shown) for *Noisy-HalfCheetah* environment.

Ensemble parameters

Name	Value
num_layers	4
size	200

Stochastic NN parameters

Name	Value
var_clipping_low	-6.0
state_dependent_var	True

Remaining parameters

Name	Value
lr	0.0002
grad_norm	None
batch_size	256
weight_decay	$3e^{-5}$
epochs	50

For training the predictive model, we alternate between two phases: data collection and model fitting. In the *BridgeMaze* environment, we collect 5 rollouts of length 80 steps and append them to the previous rollouts. Afterwards, we fit the model for 25 epochs. For *Noisy-HalfCheetah*, we collect 1 rollout and fit for 50 epochs. For Noisy-FetchPickAndPlace and Solo8-LeanOverObject we replace the $\hat{f}$ in Fig. 2 with independent instances of noisy ground truth simulators.

A.2 Controller Parameters

Parameters used in the CEM controller. For an explanation of the different parameters, we refer the reader to[2].

Table S3: Controller parameters, BridgeMaze environment.

Action sampler parameters

Name	Value
alpha	0.1
colored_noise	true
elite_size	10
execute_best_elite	true
finetune_first_action	false
fraction_elites_reused	0.3
init_std	0.5
keep_previous_elites	true
noise_beta	2.0
opt_iterations	3
relative_init	true
shift_elites_over_time	true
use_mean_actions	true

Remaining parameters

Name	Value
cost_along_trajectory	sum
delta	0.0
factor_decrease_num	1
horizon	30
num_simulated_trajectories	128

Table S4: Controller parameters, Noisy-HalfCheetah environment (only difference wrt S3 are shown).

Action sampler parameters

Name	Value
noise_beta	0.25
opt_iterations	4

Remaining parameters

Name	Value
num_simulated_trajectories	120

A.3 Timings

While our code is not tuned for speed specifically, in table S6 we provide some timings for a single step in the environment (hyper-parameters are set as specified in Suppl. A.1 and Suppl. A.2, with num_simulated_trajectories $= 128$ and op_iterations $= 3$).

Table S5: Controller parameters, Solo8-LeanOverObject environment (only difference wrt S3 are shwon).

Action sampler parameters

Name	Value
init_std	0.3
noise_beta	3.0

Table S6: Timings per one environment step in ms. We measured the timings on a system with 1 GeForce GTX 1050 Ti, an Intel Core i7-6800K and 31GB of memory.

Environment	Timing [ms]
BridgeMaze	0.25
Noisy-HalfCheetah	0.14

A.4 Uncertainty Separation

In our method, we separate the epistemic uncertainty, denoted as $\mathfrak{E}$ and aleatoric uncertainty, denoted as $\mathfrak{A}$, the details of which are explained in Sec. 3 with the resulting costs that arise. Since we are using a variant of the CEM algorithm that needs to sort the sampled action sequences $\boldsymbol{u}$ according to their cost, the cost of an action sequence is a single floating point number.

The stochastic NN ensemble that we are using samples trajectories from the predictive distribution ψ_τ for each action sequence $\boldsymbol{u}$. In addition, our variant (PETSUS), also propagates the mean prediction $\bar{x}_t$ for each ensemble member for an action sequence $\boldsymbol{u}$. The auto-regressive prediction follows a recursive relation:

$$[\bar{x}_{t+1}, \Sigma_{t+1}] = \boldsymbol{\vartheta}(\bar{x}_t, u_t)$$

We make use of this in order to estimate the epistemic uncertainty $\mathfrak{E}$. At each time point of the predicted sequence of observations, we take the empirical variance of the outputted Gaussian parameters $\boldsymbol{\vartheta}(\bar{x}_t, u_t)$, predicted from the previous mean prediction $\bar{x}_t$ and control u_t, across the ensembles for that time slice in the predicted trajectories. This is then summed up across horizon H to obtain the epistemic bonus for action sequence $\boldsymbol{u}$.

Fig. S1 shows that scaling $w_{\mathfrak{E}}$ results in better state-coverage. This is of particular interest if we want to learn models that are able to generalize to different task settings, e.g. when changing the cost function. While the naive PETS algorithm overfits the model to the task at hand, RAZER learns a truly task-agnostic model and is able to reap the benefits of model-based approaches to control.

For the aleatoric penalty we rely on the actual predictions of the covariance $\Sigma(x_t, u_t)$ and average them across the time slice, following with the sum across horizon H. Alternatively to this, we also use the entropy of the Gaussian as the $\mathfrak{A}$ uncertainty measurement. In Sec. A.5 we argue how these terms are interchangeable.

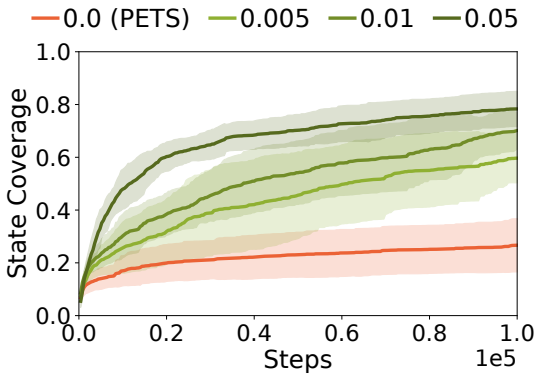

Figure S1: Exploration over time.

Note that, for the safety term ideally we want to use the full distribution ψ_τ and separation in aleatoric and epistemic uncertainty is neither required nor desirable.

A.5 Entropy vs. Variance as Uncertainty Measurement

We use entropy of Gaussian and variance interchangebly as uncertainty estimates. Indeed, since the Gaussian distribution is the maximum entropy distribution for certain variance σ^2, the entropy scales linearly with $\log \sigma^2$. We have found that utilizing the variance directly causes RAZER to be much more risk-averse, which can be explained by the variance not being suppressed by the $\log$ term in the entropy. Moreover, using the variance directly is much more interpretable and easier to tune because it's of the same scale as the observation space.

A.6 Observation Space vs. Cost Space Uncertainty

A natural question to ask when attempting to make efficient use of uncertainties in MPC is where to measure these uncertainties. As an alternative to observation space uncertainties, one could measure uncertainty in cost space. Here we argue why this is not a reasonable thing to do for each of the individual cost terms.

Epistemic Bonus Since we operate under the desiderata that the benefit of model-based methods is in task-agnosticism, we shouldn't measure epistemic uncertainty in the cost space, since this would decouple the task definition through the cost from the observation space and would lead to learning models that are not task-agnostic.

Aleatoric Penalty This is perhaps the most questionable case for using observation space uncertainty instead of cost space uncertainty. Nevertheless, we assume that high-aleatoric uncertainty translates to control difficulty, and we want to avoid parts of the observation space that are difficult to control. Moreover, the uncertainty measurements become completely invalidated in the case of a task switch, which plays against the task-agnosticism desiderata.

Safety Penalty Safety is something that is enforced by infusing the algorithm with prior knowledge through a set of constraints which mostly manifest themselves as subsets of the observation space $\mathcal{X}$ or action space $\mathcal{U}$.

B Algorithm

In Algo. 1 we provide an overview of the CEM algorithm that we utilize for implementing RAZER. Concretely, we use an improved sample efficient version of CEM as proposed by Pinneri et al. [2] that involves shift-initialization of the distribution mean, sampling time-correlated noise and further improvements.

Algorithm 1: RAZER: Risk-aware and safe CEM-MPC

Parameters:
  N: number of samples; B: Number of particles, H: planning horizon; $w_{\mathfrak{A}}$, $w_{\mathfrak{E}}$, $w_{\mathfrak{S}}$ CEM-iterations
for *t = 1 to T* // loop over episode length
do
  for *i = 1 to CEM-iterations* **do**
    $(\text{samples}_p)_{p=1}^{P} \leftarrow N$ samples from CEM(μ_t^i, Σ_t^i), with P particles per sample
    c, $c_{\mathfrak{A}}$, $c_{\mathfrak{E}}$, $c_{\mathfrak{S}} \leftarrow$ compute cost functions over particles
    $c_{\text{tot}} = c + c_{\mathfrak{A}} + c_{\mathfrak{E}} + c_{\mathfrak{S}}$ // compute total cost
    elite-set$_t \leftarrow$ best K samples according to total cost
    μ_t^{i+1}, $\Sigma_t^{i+1} \leftarrow$ fit Gaussian distribution to elite-set$_t$
  execute first action of best elite sequence
  shift-initialize μ_{t+1}^1

C Environments

All environments are based on the MuJoCo physics engine [3]. The **Noisy-Halfcheetah** and **Noisy-FetchPickAndPlace** environments are based on *HalfCheetah-v3* and *FetchPickAndPlace-v1*, respectively.

BridgeMaze We designed the *BridgeMaze* environment to show the different aspects of uncertainty, namely the epistemic and aleatoric uncertainty, in isolation. The agent is a simple cube with only a free joint attached to it. The state-space $x = [x_0, x_1, x_2, a, b, c, d, v_{x_0}, v_{x_1}, v_{x_2}]$ is 10-dimensional, consisting of 3 positional (x_0 to x_2), 4 rotational (a to d) and 3 velocity-based (v_{x_0} to v_{x_2}), agent-centric coordinates. The action-space $u = [\tau_{x_0}, \tau_{x_1}]$ is 2-dimensional. The torque τ applied to the agent in x_0- and x_1-direction.

The task in the environment is to reach a goal platform at $x_0^\star \geq 12$ by crossing one of three bridges that go over deadly lava.

The domain reward is defined as

$$r_t(x_t, u_t, x_{t+1}) = \begin{cases} |(x_0)_t - x_0^\star| - |(x_0)_{t+1} - x_0^\star| & \text{, if } (x_1)_{t+1} \geq -1.5 \\ 0 & \text{, if } (x_0)_{t+1} \geq x_0^\star \text{ and } (x_1)_{t+1} \geq -1.5 \\ -1 & \text{, otherwise} \end{cases} \tag{S1}$$

where $x^\star$ is the goal state. We define the cost for planning as $c_t(x_t, u_t, x_{t+1}) = -r_t(s_t, u_t, s_{t+1})$.

We designed the environments such that the agent is able to accelerate fast and also comes to a full stop relatively fast if no torque is applied. This makes the control problem and the task of learning the model relatively easy.

Noise is added in form of an external force in x_1-direction injected through the `xfrc_applied` attribute of the model. The sign of the force, as well as the force amplitude, sampled from $f_{\text{ext}} \in \mathcal{U}(0, f_{\text{ext}}^{\max})$, are randomly changing every 5 simulation steps. The external force is added only if $-8 \leq x_0 \leq 8$ and $-3.6 \leq x_1 \leq 3.6$. Otherwise the external force is zero.

Noisy-HalfCheetah We utilize a modified HalfCheetah environment where we apply a normally distributed noise term $\xi \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ to the simulator state in the case when the velocity of the cheetah is greater than 6. More concretely, let s_t denote the simulator state at time step t, then the modified state is calculated as follows:

$$s_t' = s_t + \xi_t \tag{S2}$$

In our case, Σ is a diagonal covariance matrix with the diagonal terms equal to 0.2. In addition, for the safety experiments with the Noisy-HalfCheetah we create a virtual ceiling at height $h = 0.3$. In the case that the body height crosses this threshold, the agent incurs a large penalty. When the safety-constraint is violated, we don't end the episode.

Noisy-FetchPickAndPlace We modified the *FetchPickAndPlace-v1* environment to show the effect of the aleatoric penalty on the CEM action plan. Given the difficulty of the task, we performed the experiments without the learned model, using instead an ensemble of noisy ground truth dynamics. In this way, we could more easily understand the role of the aleatoric uncertainty during planning.

The noise term $\xi \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is applied to the action controlling the gripper state: a positive additive noise forces the robot to open the grip with a force proportional to the noise magnitude. This noise is applied to all the ground truth models of the ensemble, and to the environment as well.

In particular, the box position is centered at y-coordinate -1.5 while the target is at $y = 2.0$. The gripper state is noisy until $y = 1.67$, right before the target.

The dropping rate is computed considering the height variation of the box (z-coordinate). If the downward velocity is greater than a fixed threshold, the box is considered dropped. The threshold velocity also includes cases in which the box is dropped and possibly re-grabbed, as this is still part of the risky behaviors we want to avoid. The plotted dropping is the minimum over different aleatoric penalties.

Solo8-LeanOverObject The state space of the this environment is 47-dimensional. It contains the absolute position, rotation, velocity and angular velocity of the robot as well as the positions and velocities of all the joints. In addition, the state contains the positions of the end-effectors and of the sites at the front and back of the robot. The actions space is 8-dimensional and controls the relative position of the joints. We fixed the two front legs of the robot with a soft-constraint to the ground to prevent the robot from uncontrollable jumping. We apply Gaussian noise to the action with a mean of 0 and a diagonal covariance matrix with the diagonal elements all being 0.3. The noise is uniformly applied over the entire state-action-space.

The experiments for the *Solo8-LeanOverObject* environment use the ground truth model during planning. The same noise were applied in the 'mental' as well as the 'real' environment.

C.1 Computing State-Space Coverage

For computing the state coverage in Fig. 3a we divided the continues state-space in 50 equally spaced bins in the range $-20 \leq x_0 \leq 20$ and $-10 \leq x_1 \leq 15$. The state space-coverage is the fractions between states visited at least once and the total number of states.

D Application to Transfer Learning

In this work we have demonstrated that an approach such as PETS[1] to data-driven MPC that relies on zero-order trajectory optimization of the expected cost is not enough to manage uncertain environments and safety constraints. These problems need to be addressed when dealing with sim-to-real. The separation of uncertainties allows us to effectively manage epistemic uncertainty in the real system, which is important for improving the model once distribution shift to the real system happens. This can be done in a way of combining the epistemic bonus and probabilistic safety constraints, such that the policy explores parts of the state space where there is knowledge to be obtained while avoiding high-cost regions as a consequence of the incurred safety and aleatoric penalties.

In comparison to standard approaches for sim-to-real which involve domain randomization at training time, this approach incurs lower computational overhead and relies on learning on the real system.

E Pandemic Impact

In our department the lab access during the last 1.5 years was heavily restricted, such that we could not perform the planned experiments with the real Solo8 quadruped. Instead, we decided to focus on simulations of the real robot and consider what is important for transfer to the real world: uncertainty estimation and safe model-based reinforcement learning.

In Sec. D we provide motivation for why uncertainty separation is in particular important for the sim-to-real setting. In this work, we have provided fundamental methodology for separating uncertainties and using them for more robust control and exploration. Therefore, this work is a stepping stone towards application to the real robots which we plan on exploring once the lab restrictions due to the pandemic are lifted.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31 of *NeurIPS*. Curran Associates, Inc., 2018.

[2] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient cross-entropy method for real-time planning. In *Conference on Robot Learning 2020*, 2020. URL https://corlconf.github.io/corl2020/paper_217/.

[3] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.