# Mind the Uncertainty: Risk-Aware and Actively Exploring Model-Based Reinforcement Learning

**Marin Vlastelica***, **Sebastian Blases***, **Cristina Pinneri, Georg Martius**

Max Planck Institute for Intelligent Systems
Tübingen, Germany
{mvlastelica,sblaes}@tue.mpg.de

## Abstract

We introduce a simple but effective method for managing risk in model-based reinforcement learning with trajectory sampling that involves probabilistic safety constraints and balancing of optimism in the face of epistemic uncertainty and pessimism in the face of aleatoric uncertainty of an ensemble of stochastic neural networks. Various experiments indicate that the separation of uncertainties is essential to performing well with data-driven MPC approaches in uncertain and safety-critical control environments.

## Introduction

Data-driven approaches to sequential decision-making are becoming increasingly popular (Yang et al. 2019; Hussein et al. 2017; Polydoros and Nalpantidis 2017; Schrittwieser et al. 2020). They hold the promise of reducing the number of prior assumptions about the system that are imposed by traditional approaches that are based on nominal models.

Such approaches come in several different flavors (Kober, Bagnell, and Peters 2013). Model-free approaches attempt to extract closed-loop control policies directly from data, while model-based approaches rely on a learned model of the dynamics to either generate novel data to extract a policy or to be used in a model-predictive control fashion (MPC). This study belongs to the latter line of work.

Model-based methods have several advantages over pure model-free approaches. Firstly, humans tend to have a better intuition on how to incorporate prior knowledge into a model rather than into a policy or value function. Secondly, most model-free policies are bounded to a specific task, while models are task-agnostic and can be applied for optimizing arbitrary cost functions, given sufficient exploration.

Nevertheless, learning models for control come with certain caveats. Traditional MPC methods require the model and cost function to permit a closed-form solution which restricts the function class prohibitively. Alternatively, gradient-based iterative optimization can be employed, which allows for a larger class of functions but typically fails to yield satisfactory solutions for complicated function approximators such as deep neural network models. In addition, calculating first-order or even second-order information for trajectory optimization tends to be computationally costly, which makes it hard to meet the time constraints of real-world settings. This motivates the usage of zero-order, i.e gradient-free or sample-based methods, such as the Cross-entropy Method (CEM) that do not rely on gradient information but are efficiently parallelizable.

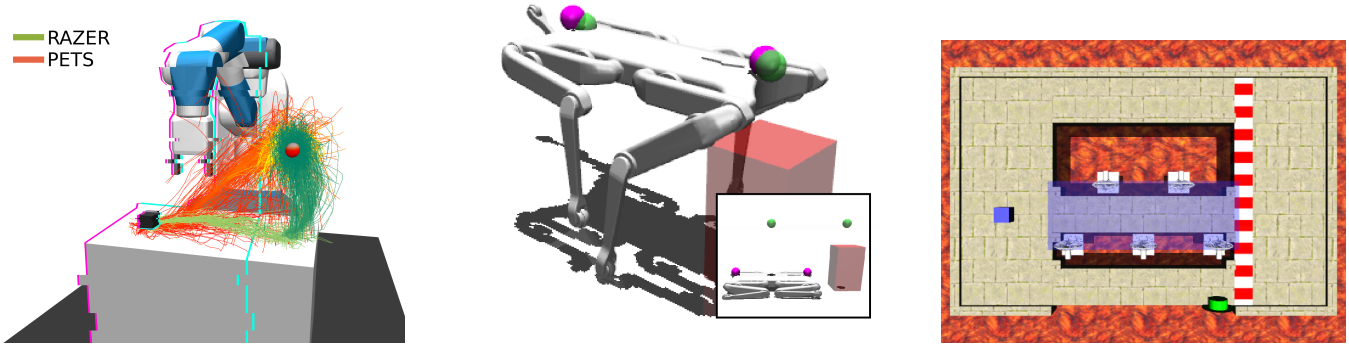
Many methods relying on a learned model and zero-order trajectory optimizers have been proposed (Chua et al. 2018; Wang and Ba 2020; Williams, Aldrich, and Theodorou 2015), but all share the same problem: compounding of errors through auto-regressive model prediction. This naturally brings us to the question of how can we effectively manage model errors and uncertainty to be more data-efficient and safe. Arguably, this is one of the main obstacles to applying data-driven model-based methods to the real world, e.g. to robotics settings.

In this work, we introduce a risk-averse zero-order trajectory optimization method (RAZER) for managing errors and uncertainty in zero-order MPC and test it on challenging scenarios (Fig. 1). We argue that it is essential to differentiate between the two types of uncertainty in the model-predictive setting: the aleatoric uncertainty arising from inherent noise in the system and epistemic uncertainty arising from the lack of knowledge (Hora 1996; Kiureghian and Ditlevsen 2009). We measure these uncertainties by making use of probabilistic ensembles with trajectory sampling (Chua et al. 2018) (PETS). Our contributions can be summarized as follows: (i) method for separation of uncertainties in probabilistic ensembles (termed PETSUS); (ii) efficient use of aleatoric and epistemic uncertainty in model-based zero-order trajectory optimizers; (iii) an simple but practical approach to probabilistic safety constraints in zero-order MPC.

## Related Work

**Uncertainty Estimation.** In the typical model-based reinforcement learning (MBRL) setting, the true transition dynamics function is modeled through an approximator. Impressive results have been achieved by both parametric models (Lenz, Knepper, and Saxena 2015; Fu, Levine, and Abbeel 2016; Gal, McAllister, and Rasmussen 2016; Hafner et al. 2019), such as neural networks, and nonparametric models (Kocijan et al. 2004; Nguyen-Tuong, Peters, and Seeger 2008; Grancharova, Kocijan, and Johansen 2008; Deisen-

*These authors contributed equally.




(a) Noisy-FetchPickAndPlace (b) Solo8-LeanOverObject (c) BridgeMaze

Figure 1: Environments considered for uncertainty-aware planning.

roth, Fox, and Rasmussen 2013), such as Gaussian Processes (GP). The latter inspired seminal work on the incorporation of the dynamics model's uncertainty for long-term planning (Deisenroth, Fox, and Rasmussen 2013; Kamthe and Deisenroth 2018). However, their usability is limited to low-data, low-dimensional regimes with smooth dynamics (Rasmussen and Kuss 2003; Rasmussen and Williams 2006), which is not ideal for robotics applications. Alternative parametric approaches include ensembling of deep neural networks, used both in the MBRL community (Chua et al. 2018; Kurutach et al. 2018), and outside (Osband et al. 2016; Lakshminarayanan, Pritzel, and Blundell 2017). In particular, ensembles of *probabilistic* neural networks established state-of-the-art results in the MBRL community (Chua et al. 2018), but focus mainly on estimating the expected cost and disregard the underlying uncertainties. In comparison, we propose a treatment of the resulting uncertainties of the ensemble model.

**Zero-order MPC.** The learned model can be used for policy search like in PILCO (Deisenroth and Rasmussen 2011; Deisenroth, Fox, and Rasmussen 2013; Kamthe and Deisenroth 2018; Curi, Berkenkamp, and Krause 2020) or for online model-predictive control (MPC) (Morari and Lee 1999; Williams et al. 2017; Chua et al. 2018). In this work, we do planning in an MPC fashion and employ a zero-order method as a trajectory optimizer, since less sensitive to hyperparameter tuning and less likely to get stuck in local minima of complex objective functions. Specifically, we consider a sample-efficient implementation of the Cross-Entropy method (Rubinstein and Davidson 1999; Botev et al. 2013) introduced in (Pinneri et al. 2020).

**Safe MPC.** Separating the sources of uncertainty is of particular importance for AI applications directly affecting humans' safety, as self-driving cars, elderly care systems, or in general any application that involves a physical interaction between the AI system and humans. Disentangling epistemic from aleatoric uncertainty allows for separate optimization of the two, as they represent semantically different objectives: efficient exploration and risk-awareness. Extensive research on uncertainty decomposition has been done in the Bayesian setting and the context of safe policy search (Mihatsch and Neuneier 2002; Garcıa and Fernández 2015; Depeweg et al. 2017, 2018), MPC planning (Arruda et al. 2017; Lee et al. 2020; Abraham et al. 2020), and distributional RL (Clements et al. 2020; Zhang and Weng 2021). On the other side, a state-of-the-art baseline for ensemble learning like PETS (Chua et al. 2018), despite estimating both uncertainties, only optimizes for the *expected* cost during action evaluation. Our work aims at filling this gap by explicitly integrating the propagated uncertainty information in the zero-order MPC planner.

## Method

Our approach concerns itself with the efficient usage of uncertainties in zero-order trajectory optimization and is therefore generally applicable to such optimizers. We are interested in modeling noisy system dynamics $x_{t+1} = f(x_t, u_t, w(x_t, u_t))$ where $f$ is a nonlinear function, $x_t$ the observation vector, $u_t$ applied control input and $w(x_t, u_t)$ a noise term sampled from an arbitrary distribution.
Consequently, in the absence of prior knowledge about the function $f$, the system needs to be modeled by a complex function approximator such as a neural network. Furthermore, we are interested in managing uncertainties based on our fitted model, which is erroneous. To this end, we use stochastic ensembles of size $K$, where the output of each model $\boldsymbol{\vartheta}^k(x_t, u_t)$ are parameters of a normal distribution depending on input observation $x_t$ and control $u_t$. As a by-product, our auto-regressive model prediction based on controls $\boldsymbol{u}$ becomes a predictive distribution over trajectories $\tau$; $\psi^\tau(x_t, \boldsymbol{u}) := p(\tau | x_t, \boldsymbol{u}; \theta)$ where $\theta$ denotes the parameters of the ensemble. For convenience, from this point onward we will differentiate between multiple usages of $\psi^\tau$. We denote with $\psi^x_{\Delta t}$ the distribution $p(x_{t+\Delta t} | x_t, \boldsymbol{u}_{t:t+h}; \theta)$ over states at time step $t+\Delta t$ and $\psi^{\boldsymbol{\vartheta}}_{\Delta t}$ the distribution over the Gaussian parameter outputs $p(\boldsymbol{\vartheta}_{t+\Delta t} | x_t, \boldsymbol{u}_{t:t+h}; \theta)$ at time step $t+\Delta t$ of the planner.

### Planning and Control

To validate our hypothesis that accounting for uncertainty in the environment and model prediction is essential to develop risk-averse policies, we use the Cross-Entropy Method

(CEM) with improvements suggested in Pinneri et al. (2020). Accordingly, at each time step $t$ we sample a finite number of control sequences $\boldsymbol{u}$ for a finite horizon $H$ from an isotropic Gaussian prior distribution which we evaluate from the state $x_t$ using an auto-regressive forward-model and the cost function. The sampling distribution is refitted in multiple rounds based on good-performing (elite) trajectories. After this optimization step, the first action of the mean of the fitted Gaussian distribution is executed. Since this approach utilizes a predictive model for a finite horizon at each time step, it naturally falls into the category of Model Predictive Control (MPC) methods.

Although we use CEM, our approach of managing uncertainty can generically be applied to other zero-order trajectory optimizers such as MPPI (Williams et al. 2017), by a modification of the trajectory cost function.

## The Problem of Uncertainty Estimation

Since we have a stochastic model of the dynamics, at the model prediction time step $t$ we observe a distribution over potential outcomes. Indeed, since our model outputs are parameters of a Gaussian distribution, with auto-regressive predictions we end up with a distribution over possible Gaussians for a certain time step $t$.

Given a sampled action sequence $\boldsymbol{u}$ and the initial state $x_t$ we observe a distribution over trajectories $\psi_\tau$. To efficiently sample from the trajectory distribution $\psi_\tau$ we use the technique introduced by Chua et al. (2018) (PETS) which involves prediction particles that are sampled from the probabilistic models and randomly mixed between ensemble members at each prediction step. In this way, the sampled trajectories are used to perform a Monte Carlo estimate of the expected trajectory cost $\mathbb{E}_{\tau\sim\psi^\tau}[c(\tau)]$. However, this does not take the properties of $\psi^\tau$ into account, which might be a high-entropy distribution and may lead to very risky and unsafe behavior. In this work, we alleviate this by looking at the properties of $\psi^\tau$, i.e. different kinds of uncertainties arising from the predictive distribution.

## Learned Dynamics Model

We learn a dynamics model $f_\theta$ that approximates the true system dynamics $x_{t+1} = f(x_t, u_t, w(x_t, u_t))$. As a model class, we use an ensemble of neural networks with stochastic outputs as in Chua et al. (2018). Each model $k$, parameterizes a multivariate Gaussian distribution with diagonal covariance, $f_\theta^k(x_t, u_t) = \mathcal{N}(x_{t+1}; x_t + \mu_\theta^k(x_t, u_t), \Sigma_\theta^k(x_t, u_t))$ where $\mu_\theta^k(\cdot,\cdot)$ and $\Sigma_\theta^k(\cdot,\cdot)$ are model functions outputting the respective parameters.

Iteratively, while interacting with the environment, we collect a dataset of transitions $\mathcal{D}$ and train each model $k$ in the ensemble by the following negative log-likelihood loss on the Gaussian outputs:

$$\mathcal{L}(\theta, k) = \mathbb{E}_{\mathcal{D}}\Big[-\log\mathcal{N}(x_{t+1}; x_t + \mu_\theta^k(x_t, u_t), \Sigma_\theta^k(x_t, u_t))\Big] \tag{1}$$

In addition, we use several regularization terms to make the model training more stable. We provide more details on this in Suppl. .

## Separation of Uncertainties

In the realm of parametric estimators, two uncertainties are of particular interest. *Aleatoric* uncertainty is the kind that is irreducible and results from inherent noise of the system, e.g. sensor noises in robots. On the other hand, we have *epistemic* uncertainty resulting from lack of data or knowledge which is reducible. This begs the question, how can we separate these uncertainties given an auto-regressive dynamics model $f_\theta$? The way that we efficiently sample from $\psi^\tau$ is by mixing sampled prediction particles, similarly as in PETS(Chua et al. 2018). This process is illustrated by the red lines in Fig. 2.

Simple model prediction disagreement is not a good measure for *aleatoric* uncertainty since it can be entangled with epistemic uncertainty. Given our assumptions about the system dynamics, we measure *aleatoric* uncertainty as the entropy of the predicted normal distributions of the ensemble models. More concretely, given a sampled particle state $\tilde{x}_t$, we define the estimated aleatoric uncertainty for ensemble model associated to particle $b$ at time step $t$ as:

$$\mathfrak{A}_b(x|\tilde{x}_t, u_t) = \mathcal{H}_{x\sim\psi^x_{\Delta t,b}}(x) \tag{2}$$

Where $\psi^x_{\Delta t,b}$ is the output distribution of ensemble model based on inputs $\tilde{x}_t$, $u_t$. Since in the end we are interested in the aleatoric uncertainty incurred from applying the action sequence $\boldsymbol{u}$ from initial state $x_t$, the quantity of interest for us is the expected aleatoric uncertainty for time slice $t$:

$$\mathfrak{A}(x|u_t) = \mathbb{E}_{\tilde{x}_b\sim\psi^x_{\Delta t}}\Big[\mathfrak{A}_b(x|\tilde{x}_b, u_t)\Big] \tag{3}$$

Intuitively, because we only have access to the ensemble for sampling, we take a time-slice in the sampled trajectories from $\psi^\tau$ and compute the output entropies. Moreover, since we assume a Gaussian 1-step predictive distribution this is an expectation over differential Gaussian entropy. An alternative way of computation which we also explore in this work is calculating the expected particle variance for time slice $t$ of the prediction horizon:

$$\mathrm{Var}^{\mathfrak{A}}_{t+1} = \frac{1}{B}\sum_{b=1}^{B}\Sigma^k_\theta(\tilde{x}_{t,b}, u_t) \tag{4}$$

For estimating the *epistemic* uncertainty, one would be tempted to look at the disagreement between ensemble models in parameter space $\mathrm{Var}[\theta]$, but this is not completely satisfying, since neural networks tend to be over-parametrized and variance within the ensemble still may exist albeit the optimum has been reached by all ensemble models. An alternative would be to calculate the Fisher information metric $\mathcal{I} \coloneqq \mathrm{Var}[\nabla_\theta \log\mathcal{L}(x_{t+1}|x_t, u_t)]$ where $\mathcal{L}$ denotes the likelihood function, but this tends to be expensive to compute.

Given the assumption of local Gaussianity, the true epistemic uncertainty for this case is the predictive entropy over the Gaussian parameters $\boldsymbol{\vartheta}$ at time step $t+h$.

$$\mathfrak{E}(x_t, \boldsymbol{u}_{t:t+h}) = \mathcal{H}_{\psi^{\boldsymbol{\vartheta}}_{\Delta t}}(\boldsymbol{\vartheta} \mid x_t, \boldsymbol{u}_{t:t+h}) \tag{5}$$

It is easy to verify that this quantity is 0 given perfect predictions of the model. Note that, because of auto-regressive predictions of a nonlinear model, this is a very difficult object to handle. Nevertheless, since our predictive distribution

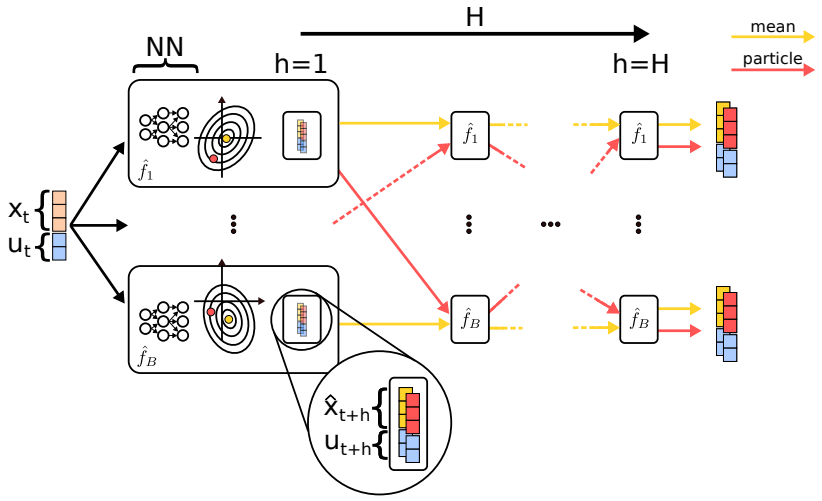



Figure 2: Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS)

$p(x \mid x_t, u_t; \boldsymbol{\vartheta})$ is parametrized by model outputs, we may utilize disagreement in $\boldsymbol{\vartheta}_t$ to approximate $\mathfrak{E}$. To get correct estimations, we need to propagate mean predictions $\bar{x}$ in addition to the particles as illustrated as the yellow lines in Fig. 2. We quantify epistemic uncertainty as ensemble disagreement at time step $t$:

$$\mathrm{Var}^{\mathfrak{E}}(x_{t+1}) = \mathrm{Var}^{e}[\mu_\theta^k(\bar{x}_t, u_t)] + \mathrm{Var}^{e}[\Sigma_\theta^k(\bar{x}_t, u_t)] \quad (6)$$

where $\mathrm{Var}^e$ is the empirical variance over the $k = 1 \dots K$ ensembles.

## Probabilistic Safety Constraints

When applying data-driven control algorithms to real systems, safety is of utmost importance. In the realm of zero-order optimization, safety constraints can be easily introduced by putting an infinite cost on constraint-violating trajectories. Nevertheless, we are dealing with erroneous stochastic nonlinear models which lead to nontrivial predictive distributions of future states, based on the control sequence $\boldsymbol{u}$. For this reason, we want to control the risk of violating the safety constraints that we, as practitioners, are willing to tolerate. If we denote the observation space as $\mathbb{X}$, given a violation set $\mathbb{C} \subset \mathbb{X}$, we define the probability of the control sequence $\boldsymbol{u}$ to enter the violation set at time $t + \Delta t$ as $p(x \in \mathbb{C} \mid x_t, \boldsymbol{u}) = \int_{x \in \mathbb{C}} \psi^x_{\Delta t}(x \mid x_t, \boldsymbol{u})$. In practice, it is hard to compute this integral efficiently, since our distribution $\psi^x_{\Delta t}$ is nontrivial as a result of nonlinear propagation of uncertainty. Furthermore, the violation set $\mathbb{C}$ might not have the structure necessary to allow an efficient solution to the integral, in which case one needs to resort to Monte Carlo estimation.

To simplify computation and gain speed, we consider box violation sets resulting in each dimension of $x$ being constrained to be outside of $[a, b] \in \{a, b \,|\, a, b \in \mathbb{R}^2,\ a < b\}$. By performing moment matching by a Gaussian in each time-slice $\psi^x_{\Delta t}$, the probability of ending up in state $x$ at time step $t + \Delta t$ is given by integrating $\mathcal{N}(x; \mu_{t+\Delta t}, \Sigma_{t+\Delta t})$, where $\mu$ and $\Sigma$ are estimated by Monte Carlo sampling. If we further assume a diagonal covariance $\Sigma$, this integral can be deconstructed into $d$ univariate Gaussian integrals, which can be computed fast and in closed form (error function). Hence, the probability of a constraint violation happening at time step $t$ is defined by:

$$p(x \in \mathbb{C} \mid x_t, \boldsymbol{u}) = \prod_{i=0}^{d} \int_{x \in \mathbb{C}} \mathcal{N}(x^i; \mu^i_{t+\Delta t}, \sigma^i_{t+\Delta t}) \quad (7)$$

## Implementing Risk-Averse ZERo-Order Trajectory Optimization (RAZER)

We assume the task definition is provided by the cost $c(x_t, \boldsymbol{u})$. For trajectory optimization, we start from a state $x_t$ and predict with an action sequence $\boldsymbol{u}$ the future development of the trajectory $\tau$. Along this trajectory, we want to compute a single cost term which is conveniently defined as the expected cost of all particles $\tilde{x}$ summed over the planning horizon $H$:

$$c(x_t, \boldsymbol{u}) = \sum_{\Delta t=1}^{H} \frac{1}{B} \sum_{b=1}^{B} c(\tilde{x}^b_{t+\Delta t}, u_{t+\Delta t}). \quad (8)$$

The optimizer, in our case CEM, will optimize the action sequence $\boldsymbol{u}$ to minimize the cost in a probabilistic sense, i.e. $p(\boldsymbol{u} \mid x) \propto \exp(-\beta\, c(x, \boldsymbol{u}))$ where $\beta$ reflects the strength of the optimizer (the higher the more likely it finds the global optimum). To make the planner uncertainty-aware, we need to make sure it avoids unpredictable parts of the state space by making them less likely. Using the aleatoric uncertainty provided by PETSUS Eq. 4, we define the aleatoric penalty as

$$c_{\mathfrak{A}}(x_t, \boldsymbol{u}) = w_{\mathfrak{A}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{A}}_{t+\Delta t}}, \quad (9)$$

where $w_{\mathfrak{A}} > 0$ is a weighting constant. The larger the aleatoric uncertainty, the higher the cost.

To guide the exploration to states where the model has epistemic uncertainty Eq. 6 (due to lack of data), we use an

epistemic bonus:

$$c_{\mathfrak{E}}(x_t, \boldsymbol{u}) = -w_{\mathfrak{E}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{E}}_{t+\Delta t}}, \tag{10}$$

where $w_{\mathfrak{E}} > 0$ is a weighting constant. To be able to operate on a real system, the most important part is to adhere to safety constraints. As formulated in Eq. 7, the predicted safety violations need to be uncertainty aware, independent of the source of uncertainty. We integrate this into the planning method by adding:

$$c_{\mathfrak{S}}(x_t, \boldsymbol{u}) = w_{\mathfrak{S}} \cdot \sum_{\Delta t=1}^{H} [\![ p(\hat{x}_{t+\Delta t} \in \mathbb{C}) > \delta ]\!] \tag{11}$$

where $[\![\cdot]\!]$ is Iverson bracket and $w_{\mathfrak{S}}$ is either a large penalty $c_{\max}$ or 0 to disable safety. An alternative for implementing safety constraints into CEM is by changing the ranking function (Wen and Topcu 2018). The overall algorithm used in a model-predictive control fashion is outlined in Suppl. .

## Experiments

We study our uncertainty-aware planner in 4 continuous state and action space environments and compare to naively optimizing the particle-based estimate of the expected cost similarly to Chua et al. (2018). We start by giving a description of the environments.

**BridgeMaze** This toy environment (see Fig. 1c) was specifically designed to study the different aspects of uncertainty independently. The agent (blue cube) starts on the left platform and has to reach the goal platform on the right. To reach the goal platform, the agent has to move over one of three bridges without falling into the lava. The upper bridge is safeguarded by walls; hence, it is the safest path to the goal but also the longest. The lower bridge has no walls and therefore is more dangerous for an unskilled agent to cross but the path is shorter. The middle bridge is the shortest path to the goal. However, randomly appearing strong winds perpendicular to the bridge might cause the agent to fall off the bridge with some probability, making this bridge dangerous.

**Noisy-HalfCheetah** This environment is based on *HalfCheetah-v3* from the OpenAI Gym toolkit. We introduce aleatoric uncertainty to the system by adding Gaussian noise $\xi \sim \mathcal{N}(\mu, \sigma^2)$ to the actions when the forward velocity is above 6. The action noise translates into a non-Gaussian and potentially very complicated state space noise distribution that makes the control problem very challenging.

**Noisy-FetchPickAndPlace** Based on the *FetchPickAndPlace-v1* gym environment. Additive action noise is applied to the gripper so that its grip on the box might become tighter or looser. The noise is applied for $x$-positions $< 0.8$ which is illustrated in Fig. 1a by a blue line causing the agent to drop the box with high probability if it tries to lift the box too early.

**Solo8-LeanOverObject** In this robotic environment, the task of a quadrupedal robot (Grimminger et al. 2020) is to stand up and lean forward to reach a target position (purple markers need to reach green dots in Fig. 1b) without hitting an object visualized by the red cube representing the unsafe zone. The robot starts in a laying position as shown in the inset of Fig. 1b. As in the *Noisy-HalfCheetah* environment, Gaussian action noise is applied to mimic real-world perturbances.

### Algorithmic Choices and Training Details

For model-predictive planning we use the CEM implementation from Pinneri et al. (2020). Further details about hyperparameters can be found in Suppl. . For planning, we use the same architecture for the ensemble of probabilistic models, both in RAZER and in PETS. The only difference is that in RAZER we also forward propagate the mean state predictions in addition to the sampled state predictions. Further details can be found in Suppl. .

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.

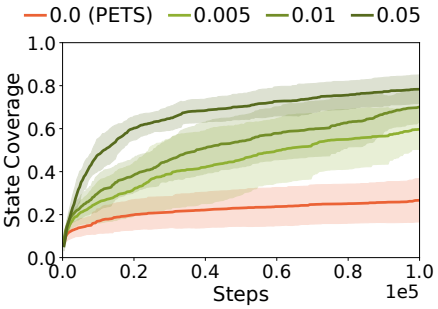
Next, we will present RAZER's exploration and safety behavior in the *BridgeMaze* environment. Afterwards, we are going to discuss planning with external safety constraints in the *Solo8-LeanOverObject* environment. We complete this section with results on *Noisy-HalfCheetah* and *Noisy-FetchPickAndPlace*.

### Active Learning for Model Improvement

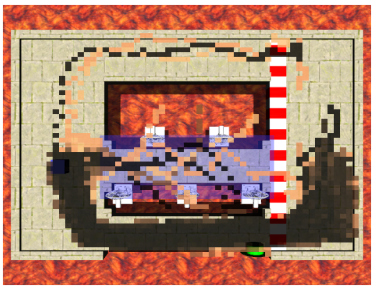
If model uncertainties are used for risk-averse planning, they are only meaningful if the model has the right training data. Only from good data can the parameters of the approximate noise model be learned correctly. In case of too little data, the agent might avoid parts of the state space due to an overestimation of the model uncertainties. On the other hand, the agent might enter unsafe regions for which the uncertainties are underestimated. By adding the epistemic bonus to our domain-specific cost, the planner can actively seek states with high epistemic uncertainty, i.e. for which no or only little training data exists.

Figure 4a shows this active data gathering process for the *BridgeMaze* environment. PETS finds one particular solution to the problem of reaching the goal platform. It chooses the path over the safer, lower bridge rather than the dangerous middle path and the longer path via the upper bridge (Fig. 3b). Once, one solution is found, the model overfits to it without exploring any other parts of the state space. This is also reflected in the plateauing of the red curve in Fig. 3a.

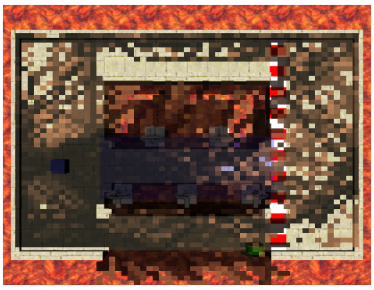
In comparison, RAZER actively explores larger and larger parts of the state space with an increasing weight of the epistemic bonus (Fig. 3a). RAZER not only finds the easy solution found by PETS but also extensively explores other parts of the state space (Fig. 3c). To not get stuck at the middle bridge during exploration due to the inherent noise, it is important to separate between epistemic and aleatoric uncertainties. Only the former should be used for exploration. With enough data, our model can correctly capture the uncertainties of these states resulting in the epistemic uncertainty approaching zero.




(a) State space exploration over time depending on epistemic bonus ($w_{\mathfrak{E}}$).



(b) State space coverage with $w_{\mathfrak{E}} = 0$.



(c) State space coverage with $w_{\mathfrak{E}} = 0.05$.

Figure 3: Active learning setting: The epistemic bonus allows RAZER to seek states for which no or only little training data exists (a,c). Means and standard deviations for (a) were computed over 5 runs. PETS overfits to a particular solution (b). In (b) and (c), the brightness of the dots is proportional to the time when they were first encountered.

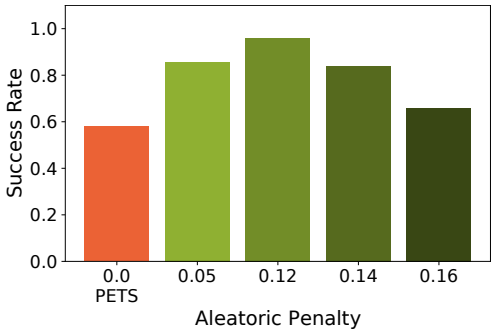
## Risk-Averse Planning

Once a good model is learned, it can be used for safe planning. What differentiates RAZER from PETS is that it makes explicit use of uncertainty estimates while in the latter uncertainties only enter planning by taking the mean over the particle costs and not differentiating between different sources of uncertainty.

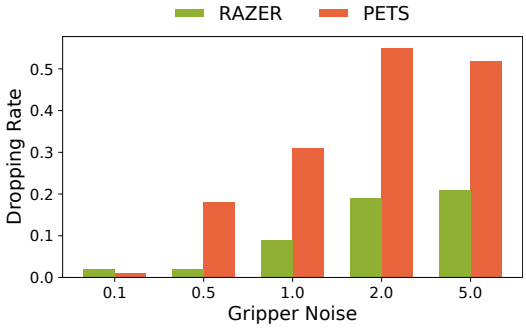
**BridgeMaze.** Figure 4a shows the success rate of PETS and RAZER in the *BridgeMaze*. In both cases, we use the same model that was trained from data collected during a training run with $w_{\mathfrak{E}} = 0.05$. Hence, the model saw enough training data from all parts of the state space. The noise in the environment is tuned such that there is a chance to cross the bridge without falling. While in Fig. 3b PETS avoided this path because of an overestimation of the state's value due to a lack of training data and sometimes sees a chance to cross the bridge. However, these attempts are very likely to fail because of stronger winds that occur randomly, resulting in a success rate of only $58\%$. RAZER does not rely on sampling for the aleatoric part and can thus avoid risk. With a higher penalty constant the success rate increases up to $96\%$ but only as long as the agent is willing to take a risk at all. For large values of $w_{\mathfrak{A}}$ the agent becomes so conservative that it only moves slowly (decreasing reward in Fig. 4a).

**Noisy-HalfCheetah.** How does RAZER perform on the *Noisy-HalfCheetah* environment when models are learned from scratch? Without aleatoric penalty, the planner is optimistic. Risky situations are only detected if a failing particle is sampled. Thus, the noise is mostly neglected and the robot increases its velocity, gets destabilized, and ends up slower than with the aleatoric penalty (Fig. 5a).

**Noisy-FetchPickAndPlace.** In this environment, a 7-DoF robot arm should bring the box to a target position – starting and target positions are at the opposite sides of the table. The shortest path is to lift the box and move in a straight line to the target. However, with noise applied to the gripper action, there is a certain probability to drop the box along the way. When penalizing aleatoric uncertainty, this is avoided and also fewer trajectory samples are "wasted" in high-entropic regions, as presented in Fig. 1a. Figure 4b shows the number

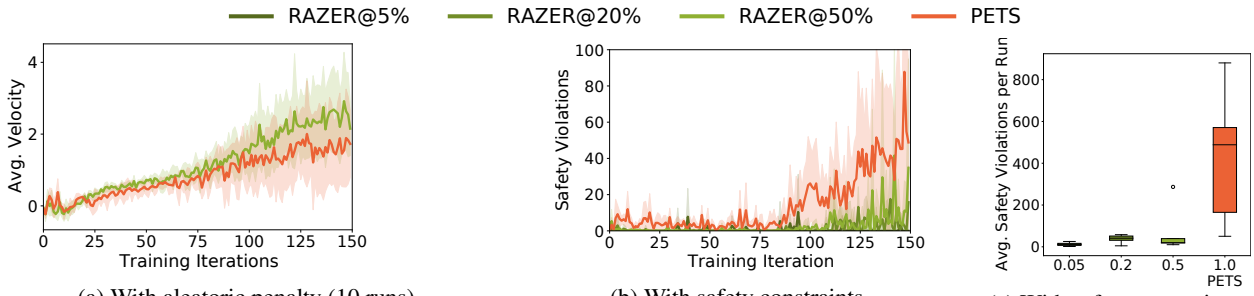



(a) *BridgeMaze* success depending on $w_{\mathfrak{A}}$ for 50 runs.




(b) Dropping rate in *Noisy-FetchPickAndPlace* for 100 runs.

Figure 4: Risk-averse planning in the face of aleatoric uncertainty yields higher success rates in noisy environments. For (b) we use ground truth models and a fixed aleatoric penalty weight $w_{\mathfrak{A}}$.




(a) With aleatoric penalty (10 runs). (b) With safety constraints. (c) With safety constraints.

Figure 5: *Noisy-HalfCheetah* environment (task lengths 300 steps) with learning models from scratch. At 150 iterations we have seen only 45k points. (a) Performance under noisy actions. By applying the aleatoric penalty, RAZER can navigate the uncertainties better – leading to higher returns faster. (b) Safety violations above a certain body height (simulating a low ceiling) for different values of $\delta$. With increasing $\delta$, RAZER is seldomly violating constraints in stark contrast to PETS. In (c) the number of violations is averaged over the last 50 iterations (summed over 10 rollouts).

of times the box is dropped on the table depending on the aleatoric penalty. RAZER adopts a cautious behavior, preferring to slide the box on the table and lifting it only in the area without action noise, achieving a dropping rate lower than 20%, even when considerable noise is applied.

### Planning with External Safety Constraints

**Noisy-HalfCheetah:.** We consider a safety constraint on the height of the body above ground simulating a narrow passage. Figure 5b shows the number of safety violations. Note that PETS has the same penalty cost for hard violations.

**Solo8-LeanOverObject:.** In this experiment, the robot has to move to two target points with its front and rear of the trunk while avoiding entering a specified rectangular area (fragile object). The front feet are fixed. To track the points, the robot has to lean forward, such that it can lose balance due to noisy actions. In contrast to PETS, RAZER successfully manages to satisfy the safety constraints almost always as shown in Fig. 6. However, satisfying the safety constraint comes with the cost of reduced tracking accuracy.

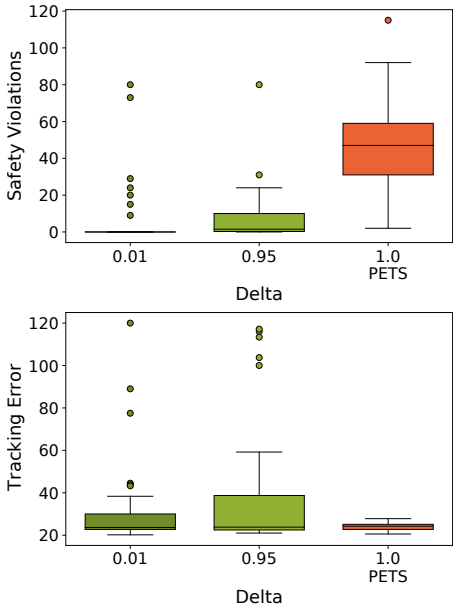



Figure 6: Safe planning vs. task-oriented planning in the *Solo8-LeanOverObject* environment with noisy actions. Left: number of safety violations for different values of $\delta$ (Eq. 11). Right: enforcing safety constraints causes slight reduction in tracking accuracy due to the fixed planning budget and the competing objectives of task and safety costs.

## Conclusion

In this work, we have provided a methodology to separate uncertainties in stochastic ensemble models (PETSUS) which can be used as a tool to build risk-averse model-based planners that are also data-efficient and enforce safety through probabilistic safety constraints (RAZER). This type of risk-averseness can be achieved by a simple modification of the cost function in form of uncertainty penalties in zero-order trajectory optimizers.

Furthermore, the separation of uncertainties allows us to do proper exploration via epistemic bonus which benefits generalization of the model and therefore makes it applicable to more settings. As future work, it would be of interest to see this approach applied to a proper transfer learning setting from simulations to real systems, where risk-averseness combined with exploratory behavior is crucial for efficient learning and safe operation.

## References

Abraham, I.; Handa, A.; Ratliff, N.; Lowrey, K.; Murphey, T. D.; and Fox, D. 2020. Model-Based Generalization Under Parameter Uncertainty Using Path Integral Control. *IEEE Robotics and Automation Letters*, 5(2): 2864 – 2871.

Arruda, E.; Mathew, M. J.; Kopicki, M.; Mistry, M.; Azad, M.; and Wyatt, J. 2017. Uncertainty averse pushing with model predictive path integral control. *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 497–502.

Botev, Z.; Kroese, D.; Rubinstein, R.; and L'Ecuyer, P. 2013.

*Chapter 3. The Cross-Entropy Method for Optimization*, volume 31, 35–59. Elsevier.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. 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.

Clements, W.; Robaglia, B.-M.; Delft, B. V.; Slaoui, R. B.; and Toth, S. 2020. Estimating Risk and Uncertainty in Deep Reinforcement Learning. *Workshop on Uncertainty & Robustness in Deep Learning Workshop at the International Conference on Machine Learning (ICML UDL)*.

Curi, S.; Berkenkamp, F.; and Krause, A. 2020. Efficient Model-Based Reinforcement Learning through Optimistic Policy Search and Planning. In *Proc. Neural Information Processing Systems (NeurIPS)*.

Deisenroth, M. P.; Fox, D.; and Rasmussen, C. E. 2013. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2): 408–423.

Deisenroth, M. P.; and Rasmussen, C. E. 2011. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning*, ICML.

Depeweg, S.; Hernandez-Lobato, J.-M.; Doshi-Velez, F.; and Udluft, S. 2018. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, 1184–1193. PMLR.

Depeweg, S.; Hernández-Lobato, J. M.; Doshi-Velez, F.; and Udluft, S. 2017. Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks. In *International Conference on Learning Representations (ICLR)*.

Fu, J.; Levine, S.; and Abbeel, P. 2016. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4019–4026. IEEE.

Gal, Y.; McAllister, R.; and Rasmussen, C. E. 2016. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, 25.

Garcıa, J.; and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1): 1437–1480.

Grancharova, A.; Kocijan, J.; and Johansen, T. A. 2008. Explicit stochastic predictive control of combustion plants based on Gaussian process models. *Automatica*, 44(6): 1621–1631.

Grimminger, F.; Meduri, A.; Khadiv, M.; Viereck, J.; Wüthrich, M.; Naveau, M.; Berenz, V.; Heim, S.; Widmaier, F.; Flayols, T.; et al. 2020. An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics and Automation Letters*, 5(2): 3650–3657.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2555–2565. PMLR.

Hora, S. C. 1996. Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management. *Reliability Engineering & System Safety*, 54(2): 217–223.

Hussein, A.; Gaber, M. M.; Elyan, E.; and Jayne, C. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2): 1–35.

Kamthe, S.; and Deisenroth, M. 2018. Data-efficient reinforcement learning with probabilistic model predictive control. In *International Conference on Artificial Intelligence and Statistics*, 1701–1710. PMLR.

Kiureghian, A. D.; and Ditlevsen, O. 2009. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2): 105–112.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274.

Kocijan, J.; Murray-Smith, R.; Rasmussen, C. E.; and Girard, A. 2004. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, 2214–2219. IEEE.

Kurutach, T.; Clavera, I.; Duan, Y.; Tamar, A.; and Abbeel, P. 2018. Model-Ensemble Trust-Region Policy Optimization. In *International Conference on Learning Representations*.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 6405–6416. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Lee, K.; An, G. N.; Zakharov, V.; and Theodorou, E. A. 2020. Perceptual Attention-based Predictive Control. In Kaelbling, L. P.; Kragic, D.; and Sugiura, K., eds., *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, 220–232. PMLR.

Lenz, I.; Knepper, R. A.; and Saxena, A. 2015. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy.

Mihatsch, O.; and Neuneier, R. 2002. Risk-sensitive reinforcement learning. *Machine learning*, 49(2): 267–290.

Morari, M.; and Lee, J. H. 1999. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5): 667–682.

Nguyen-Tuong, D.; Peters, J.; and Seeger, M. 2008. Local gaussian process regression for real time online model learning and control. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, 1193–1200.

Osband, I.; Blundell, C.; Pritzel, A.; and Roy, B. V. 2016. Deep Exploration via Bootstrapped DQN. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 4033–4041. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510838819.

Pinneri, C.; Sawant, S.; Blaes, S.; Achterhold, J.; Stueckler, J.; Rolinek, M.; and Martius, G. 2020. Sample-efficient Cross-Entropy Method for Real-time Planning. In *Conference on Robot Learning 2020*.

Polydoros, A. S.; and Nalpantidis, L. 2017. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2): 153–173.

Rasmussen, C.; and Kuss, M. 2003. Gaussian Processes in Reinforcement Learning. In *NIPS*.

Rasmussen, C.; and Williams, C. 2006. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press.

Rubinstein, R.; and Davidson, W. 1999. The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Methodology and Computing in Applied Probability*.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

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

Wang, T.; and Ba, J. 2020. Exploring Model-based Planning with Policy Networks. In *International Conference on Learning Representations (ICLR)*.

Wen, M.; and Topcu, U. 2018. Constrained Cross-Entropy Method for Safe Reinforcement Learning. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Williams, G.; Aldrich, A.; and Theodorou, E. 2015. Model Predictive Path Integral Control using Covariance Variable Importance Sampling. *ArXiv*, abs/1509.01149.

Williams, G.; Wagener, N.; Goldfain, B.; Drews, P.; Rehg, J. M.; Boots, B.; and Theodorou, E. A. 2017. Information theoretic MPC for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1714–1721. IEEE.

Yang, Y.; Caluwaerts, K.; Iscen, A.; Zhang, T.; Tan, J.; and Sindhwani, V. 2019. Data Efficient Reinforcement Learning for Legged Robots. *Conference on Robot Learning (CoRL)*.

Zhang, J.; and Weng, P. 2021. Safe Distributional Reinforcement Learning. *ArXiv*, abs/2102.13446.

# Supplementary Material for Mind the Uncertainty: Risk-Aware and Actively Exploring Model-Based Reinforcement Learning

In this supplementary we provide additional details for our method.
Our research suffered from pandemic impacts on lab access which is detailed in Sec. 12.

## Implementation Details

### Model Learning

Parameters used for model learning in the *BridgeMaze* experiments.
We bound the predicted log variance by applying (as in (Chua et al. 2018, 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).

### Controller Parameters

Parameters used in the CEM controller. For an explanation of the different parameters, we refer the reader to(Pinneri et al. 2020).

### Timings

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

### 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. 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 $\psi_\tau$ 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

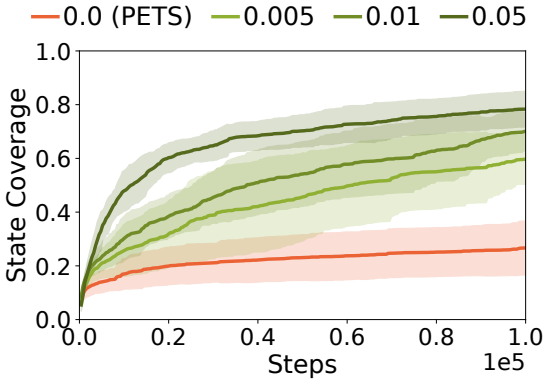



Figure S7: Exploration over time.

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. S7 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. we argue how these terms are interchangeable.
Note that, for the safety term ideally we want to use the full distribution $\psi_\tau$ and separation in aleatoric and epistemic uncertainty is neither required nor desirable.

### 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 $\sigma^2$, 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.

### 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,

Table S1: Model parameters

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 |

Table S2: Model parameters

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 |

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 |

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 $\mathcal{U}$.

## 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. (2020) that involves shift-initialization of the distribution mean, sampling time-correlated noise and further improvements.

## Environments

All environments are based on the MuJoCo physics engine (Todorov, Erez, and Tassa 2012). 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

Table S4: Controller parameters, Noisy-HalfCheetah environment (only difference to BridgeMaze environment).

Action sampler parameters

| Name | Value |
|---|---|
| noise_beta | 0.25 |
| opt_iterations | 4 |

Remaining parameters

| Name | Value |
|---|---|
| num_simulated_trajectories | 120 |

Table S5: Controller parameters, Solo8-LeanOverObject environment (only difference to BridgeMaze environment).

Action sampler parameters

| Name | Value |
|---|---|
| init_std | 0.3 |
| noise_beta | 3.0 |

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 $\tau$ 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 in (S12).
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:

$$s_t' = s_t + \xi_t \tag{S13}$$

In our case, $\Sigma$ 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.

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 |

**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 $\text{CEM}(\mu_t^i, \Sigma_t^i)$, 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
    $\mu_t^{i+1}$, $\Sigma_t^{i+1} \leftarrow$ fit Gaussian distribution to elite-set$_t$
  execute first action of best elite sequence
  shift-initialize $\mu_{t+1}^1$

**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.

**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.

$$r_t(x_t, u_t, x_{t+1}) = \begin{cases} |(x_0)_t - x_0^\star| - |(x_0)_{t+1} - x_0^\star| & \text{, if } (x_1)_{t+1} \geq -1.5 \\ 0 & \text{, if } (x_0)_{t+1} \geq x_0^\star \text{ and } (x_1)_{t+1} \geq -1.5 \\ -1 & \text{, otherwise} \end{cases} \tag{S12}$$

### 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.

## Application to Transfer Learning

In this work we have demonstrated that an approach such as PETS(Chua et al. 2018) 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.

## 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. 12 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.