Guided Safe Shooting: model based reinforcement learning with safety constraints

Anonymous authors
Paper under double-blind review

Abstract

In the last decade, reinforcement learning has successfully solved complex control tasks and decision-making problems, such as the Go board game. Yet, there have been few success stories in deploying these algorithms to real-world scenarios. One of the reasons is the lack of guarantees when dealing with and avoiding unsafe states, a fundamental requirement in critical control engineering systems. In this paper, we introduce Guided Safe Shooting (GuSS), a model-based reinforcement learning approach that can learn to control systems with minimal violations of the safety constraints through a MAP-Elites based planner. Experiments show that the new planner helps the agent avoid unsafe situations while maximally exploring the state space, a necessary aspect when learning an accurate model of the system.

1 Introduction

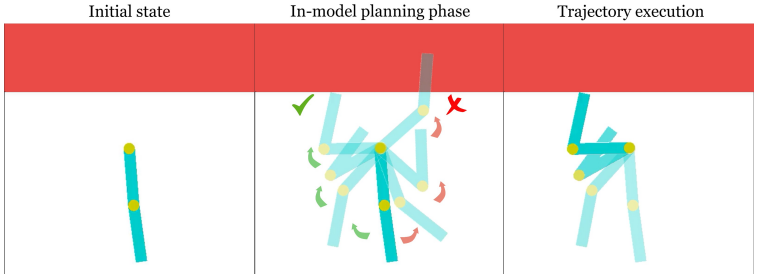

Figure 1: An illustrative example of planning with a model on the Acrobot environment. The agent controls the torque on the first joint with the goal of getting its end effector as high as possible, avoiding the unsafe zone (red area). Starting in the rest position (left) the agent uses its model to find the best plan (middle) that will maximize the reward while satisfying the safety constraint and execute it on the real system (right).

In recent years, deep Reinforcement Learning (RL) solved complex sequential decision-making problems in a variety of domains, such as controlling robots, video and board games Mnih et al. (2013); Andrychowicz et al. (2020); Silver et al. (2016). However, these achievements have largely been limited to simulated environments. The practical application of RL solutions to real-world systems remains a challenge. The fundamental principle of RL involves learning through trial and error to maximize a reward signal Sutton & Barto (2018). This approach requires unrestricted access to the system for exploration and action execution, which can lead to undesirable outcomes and safety risks. For instance, consider the task of optimizing the control strategy for a data center cooling problem Lazic et al. (2018). During the learning process, an RL algorithm may inadvertently cause high temperatures, thereby affecting and potentially damaging the system. Safety is also crucial in robotics, where unsafe actions can pose risks to both the robot and humans. Addressing this issue, known as *safe exploration*, is a central problem in AI safety Amodei et al. (2016).

With the long-term goal of deploying RL algorithms on real engineering systems, it is imperative to overcome those limitations. A straightforward way to address this issue is to develop algorithms that can provide

guarantees with respect to constraints, such as safety, to ensure the integrity of the system. Going towards this goal, in this paper we introduce GuSS, a safe Model Based Reinforcement Learning (MBRL) algorithm that learns a model of the system and uses it to plan for a safe course of actions through Model Predictive Control (MPC) Garcia et al. (1989). GuSS learns the model of the system in an iterated batch fashion Matsushima et al. (2021); Kégl et al. (2021), allowing for minimal real-system interactions. This is a desirable property for safe RL approaches, as fewer interactions with the real-system mean less chance of entering unsafe states, a condition difficult to attain with model-free safe RL methods Achiam et al. (2017); Ray et al. (2019); Tessler et al. (2018). Moreover, by learning a model of the system, this allows flexibility and safety guarantees as by using the model we can anticipate unsafe actions before they occur. Fig. 1 illustrates this concept, where the agent uses the model's dynamics to perform 'mental simulation' and select a plan that achieves its goal while avoiding unsafe zones. This contrasts with many of the methods in the literature that address the problem of finding a safe course of action through Lagrangian optimization or by penalizing the reward function Webster & Flach (2021); Ma et al. (2021); Cowen-Rivers et al. (2022). GuSS avoids unsafe situations by discarding trajectories that are deemed unsafe using the model predictions. Within this framework, we propose a novel planner based on the Quality-Diversity (QD) algorithm MAP-Elites (ME) Mouret & Clune (2015). The planner generates, evaluates, and selects the best action to apply on the system with respect to their safety and reward. Planning with a QD-based method provides the agent with the ability to explore a wide range of possible actions, which is crucial in safe RL where a trade-off between reward and safety must be found. This approach leads to a safer and more efficient search, covering a larger portion of the state space while discovering safer plans.

We evaluate *GuSS* on three different environments, demonstrating its ability to find strategies that achieve high rewards with minimal costs, even in scenarios where these metrics are conflicting, such as the Safe Acrobot environment. Moreover, we evaluate the exploration performance of our agent in an environment in which good exploration is fundamental to safely solve the problem, showing how, thanks to QD, GuSS can generate better and safer plans compared to commonly used approaches.

In summary, the contributions of the paper are twofold:

- We propose the use of QD methods like ME as planning techniques in MBRL approaches;
- We introduce GuSS, an algorithm that, by safely planning through ME, can efficiently learn to avoid unsafe states while quickly optimizing the reward.

2 Related Work

Some of the most common techniques for addressing safety in RL involve solving a Constrained Markov Decision Process (CMDP) Altman (1999) through model-free RL methods Achiam et al. (2017); Ray et al. (2019); Tessler et al. (2018); Hsu et al. (2021); Zhang et al. (2020). A popular approach to solve the CMDP is through Lagrangian-based methods, which transform the constrained optimization problem into an unconstrained form Ray et al. (2019). Another well-known method is Constrained Policy Optimization (CPO) Achiam et al. (2017), which adds constraints to the policy optimization process in a way similar to Trust Region Policy Optimization (TRPO) Schulman et al. (2015). A similar approach is taken by Projected Constrained Policy Optimization (PCPO) Yang et al. (2020a) and its extension Yang et al. (2020b). The algorithm works by first optimizing the policy with respect to the reward, then projecting it back onto the constraint set in an iterated two-step process. A different strategy involves storing all the "recovery" actions that the agent took to leave unsafe regions in a separate replay buffer Hsu et al. (2021). This buffer is then used whenever the agent enters an unsafe state by selecting the most similar transition in the safe replay buffer and performing the same action to escape the unsafe state.

Model-free RL methods need many interactions with the real-system in order to collect the necessary data for training. This can be a significant limitation in situations where safety is critical, as increasing the number of samples increases the probability of entering unsafe states. MBRL addresses this issue by learning a model of the system that can then be used to learn a safe policy. This allows for increased flexibility in dealing with unsafe situations, particularly when safety constraints change over time. There are several works that use MBRL to tackle safety. A common approach is to rely on Gaussian Processes (GPs) to model

the environment and use the dynamics model's uncertainty to guarantee safe exploration Berkenkamp et al. (2017); Cowen-Rivers et al. (2022). While GPs allow for good representation of the dynamics uncertainty, their usability is limited to low-data, low-dimensional regimes with smooth dynamics Kuss & Rasmussen (2003). A common alternative to GPs is the use of ensemble networks Webster & Flach (2021); Liu et al. (2020) which scale better. The learned dynamics model can then be used to learn a safe policy. In SAMBA Cowen-Rivers et al. (2022), the authors rely on a modified version of the soft-actor critic algorithm by including the safety constraint with Lagrangian multipliers. Thomas2021 use an automatic reward shaping approach, which eliminates the need for a separate cost function and falls back into a classical MDP formulation that is solved with SAC trained on the model.

A different approach adopted in the control community is to rely on MPC to select the safest trajectories with a learned or given model in closed-loop fashion Wen & Topcu (2018); Liu et al. (2020); Zanon & Gros (2020). For example, the work of Koller2018 uses the propagation of uncertainty to recursively guarantee the existence of a safe trajectory that satisfies the constraints of the system. The authors of Uncertainty Guided Cross-Entropy Methods (CEM) Webster & Flach (2021) extend PETS Chua et al. (2018) by modifying the objective function of the CEM-based planner to avoid unsafe areas. In this setting, an unsafe area is defined as the set of states for which the ensemble of models has the highest uncertainty. Vlastelica2021 relies on CEM to perform MPC zero-order trajectory optimization. The authors use a technique inspired by PETS Chua et al. (2018) by using predictions particles 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 to estimate the uncertainty of the dynamics.

3 Background

In this section, we introduce the concepts of safe RL and QD algorithms on which our method builds.

3.1 Reinforcement learning (RL) and safe RL

Episodic reinforcement learning problems are usually represented as a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p_{\text{real}}, R, T \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p_{\text{real}} : \mathcal{S} \times \mathcal{A} \rightsquigarrow \mathcal{S}$[1] is the transition dynamics, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $T \in \mathbb{N}^+$ is the length of the episode. The state vector $\boldsymbol{s}_t = (s_t^1, \ldots, s_t^{d_s})$ contains d_s numerical or categorical variables, measured on the system at time t. The action vector $\boldsymbol{a}_t$ contains d_a numerical or categorical action variables. Given a (deterministic or stochastic) control policy $\pi : \mathcal{S} \rightsquigarrow \mathcal{A}$, we can roll out the policy π and the system dynamics p_{real} to obtain a *trace* or *trajectory* $\mathcal{T} = \big((\boldsymbol{s}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{s}_T, \boldsymbol{a}_T)\big)$ by repeatedly applying $\boldsymbol{a}_t \leftsquigarrow \pi(\boldsymbol{s}_t)$ and observing $\boldsymbol{s}_{t+1} \leftsquigarrow p_{\text{real}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The performance of the policy is measured by the mean reward $\text{MR}(\mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} R(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The goal of RL is to find a policy which maximizes the expected mean reward:

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}_{\mathcal{T} \leftsquigarrow (\pi, p_{\text{real}})} \{\text{MR}(\mathcal{T})\} \tag{1}$$

To incorporate constraints (for example representing safety requirements), we define a cost function $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which, in our case, is a simple indicator for whether the system entered into an unsafe state ($C(\boldsymbol{s}, \boldsymbol{a}) = 1$ if the state $\boldsymbol{s}$ is unsafe and $C(\boldsymbol{s}, \boldsymbol{a}) = 0$ otherwise). With this definition, the mean cost $\text{MC}(\mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} C(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is an estimate of the probability of being in an unsafe state. The new goal is then to find a policy π in the policy set Π with high expected reward $\mathbb{E}_{\mathcal{T} \leftsquigarrow (\pi, p_{\text{real}})} \{\text{MR}(\mathcal{T})\}$ and low safety cost $\mathbb{E}_{\mathcal{T} \leftsquigarrow (\pi, p_{\text{real}})} \{\text{MC}(\mathcal{T})\}$. One way to solve this new problem is to rely on constrained Markov decision processes (CMDPs) Altman (1999) by adding constraints to the expectation La & Ghavamzadeh (2013) or to the variance of the return Chow et al. (2017). CMDP is based on MDP, in which the additional constraint set $C = \{(C_i, l_i)\}_{i=1}^{m}$ is considered, where each C_i is a cost value functions, and l_i the associated safety constraint bound. The feasible policy set Π_c that can satisfy the safety constraint bound is as follows:

$$\Pi_c = \cap_{i=1}^{m} \{\pi \in \Pi \text{ and } C_i(\pi) \leq l_i\} \tag{2}$$

[1] We use $\rightsquigarrow$ and $\leftsquigarrow$ to denote both probabilistic and deterministic mapping.

The new objective to the CMDP can be seen as finding the best policy π in the space of considered safe policies Π_c:

$$\pi^* = \arg\max_{\pi \in \Pi_c} \{\mathrm{MR}(\mathcal{T})\} \tag{3}$$

The hard constraint imposed with the safety constraint bound l_i can be in some applications difficult to tune. When safety is critical (integrity of the system) the safety constraint has to be strict ($l_i = 0$) to ensure no violation at all as it can break the system. For many others applications such critical safety constraint can be relaxed. For example, in a robot navigation task, safety can be defined as the number of collisions. While ideally we would like to have zero collisions, in practice we can accept some violations. One requirement for this setting is to ensure that all environment are ergodic MDPs Hutter (2002) which guarantee that any state is reachable from any other state by following a suitable policy. In this setup, safety violations can be accepted as at any time-step t it is possible to recover a safe policy.

Algorithm 1: Iterated Batch Model Based RL

INPUT: real-system p_{real}, number of episodes N, number of plans for planning step n, planning horizon length h, episode length T, initial random policy π^0
RESULT: learned model $p(\cdot)$ and planner

$\mathcal{T}r = \varnothing$ // Initialize empty trace collection
$\boldsymbol{s}_0 \sim p_{\text{real}}$ // Sample initial state from real system
for *t in [0, ..., T]* **do**
  $\boldsymbol{a}_t \sim \pi^0(\boldsymbol{s}_t)$ // Generate random action
  $\boldsymbol{s}_{t+1} \sim p_{\text{real}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ // Apply action on real system
  $\mathcal{T}r = \mathcal{T}r \bigcup (\boldsymbol{s}_t, \boldsymbol{a}_t)$ // Store transition in trace collection
for $\tau \in [1, ..., N]$ **do**
  $\hat{p}^{(\tau)} \leftarrow \mathrm{TRAIN}(\hat{p}^{(\tau-1)}, \mathcal{T}r)$ // Train model on collected traces
  for *t in [0, ..., T]* **do**
    $\boldsymbol{a}_t^{(\tau)} \sim \mathrm{PLAN}(\hat{p}^{(\tau)}, \boldsymbol{s}_t^{(\tau)}, h, n)$ // Use planner to generate next action
    $\boldsymbol{s}_{t+1}^{(\tau)} \sim p_{\text{real}}(\boldsymbol{s}_t^{(\tau)}, \boldsymbol{a}_t^{(\tau)})$ // Apply action on real system
    $\mathcal{T}r = \mathcal{T}r \bigcup (\boldsymbol{s}_t^{(\tau)}, \boldsymbol{a}_t^{(\tau)})$ // Store transition in trace collection

3.2 Model-based reinforcement learning with MPC

In this work, we address safety using an MBRL approach Moerland et al. (2021) based on MPC. This approach approximates the problem in Eq.(3) by repeatedly solving a simplified version of the problem initialized at the currently measured state $\boldsymbol{s}_t$ over a shorter horizon h in a receding horizon fashion. The MPC scheme relies on a sufficiently descriptive transition dynamics of the system to optimize performance and ensure constraint satisfaction. In this setting, the transition dynamics p_{real} is estimated using the data collected when interacting with the real system. The objective is to learn a model $\hat{p}(\boldsymbol{s}_t, \boldsymbol{a}_t) \rightsquigarrow \boldsymbol{s}_{t+1}$ to predict $\boldsymbol{s}_{t+1}$ given $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ and use it to plan in order to optimize a given performance metric. In this work, we consider *iterated batch* MBRL (also known as *growing batch* Lange et al. (2012) or *semi-batch* Singh et al. (1995)). In this setting (Alg. 1), the algorithm starts with an initial random policy $\pi^{(0)}$. Then, in an iteration over $\tau = 1, \ldots, N$, it updates the model $\hat{p}^{(\tau)}$ in a two-step process of (i) performing MPC on the real system p_{real} for a whole episode to obtain the trace $\mathcal{T}^{(\tau)} = \left((\boldsymbol{s}_1^{(\tau)}, \boldsymbol{a}_1^{(\tau)}), \ldots, (\boldsymbol{s}_T^{(\tau)}, \boldsymbol{a}_T^{(\tau)})\right)$, (ii) training the model $\hat{p}^{(\tau)}$ on the growing transition data $\mathcal{T}r = \bigcup_{\tau'=1}^{\tau} \mathcal{T}^{(\tau')}$ collected up to iteration τ [2]. The process is repeated until a given number of evaluations N or until a performance is reached. While the system model is a core element of MPC, learning a good controller (i.e policy) also has a major influence on the resulting closed-loop performance. One way to see the MPC controller is as a search problem where at each time-step t we produce a set of possible policies Π_m and evaluate them according to our performance criteria (in our case reward and safety). This is possible because it is cheap to evaluate each policy with R and C and rank them according to

[2] In order to keep notation light, we will omit the superscript $^{(\tau)}$ when it is clear to which episode the tuple $(\boldsymbol{s}_1^{(\tau)}, \boldsymbol{a}_1^{(\tau)})$ belongs.

Algorithm 2: Safe QD planning

INPUT: model $\hat{p}$, current real-system state $\boldsymbol{s}_t$, planning horizon h, evaluated action sequences M, discretized behavior space $\bar{\mathcal{B}}$, number of initial policies n, number of policies per iteration n_{new}, policy parameter space Φ

RESULT: action to perform $\boldsymbol{a}_t$

$\mathcal{A}_{\text{ME}} = \varnothing$ // Initialize empty collection of policies

$\Gamma^{(0)} \leftarrow \text{SAMPLE}(\Phi, n)$ // Sample initial n policies from Φ

for $\phi_i \in \Gamma^{(0)}$ **do**

  $R_i, C_i, \tilde{\mathcal{T}}_i = \text{ROLLOUT}(\hat{p}, \boldsymbol{s}_t, \phi_i, h)$ // Evaluate policy on the model

  $\bar{b}_i = f(\phi_i, \tilde{\mathcal{T}}_i)$ // Calculate policy behavior descriptors

  $\mathcal{A}_{\text{ME}} \leftarrow \text{STORE}(\mathcal{A}_{\text{ME}}, \phi_i, \bar{b}_i, R_i, C_i, \bar{\mathcal{B}})$ // Update collection

while M *not depleted* **do**

  $\Gamma^{(g)} \leftarrow \text{SELECT}(\mathcal{A}_{\text{ME}}, n_{new}, \Phi)$ // Select n_{new} policies

  for $\phi_i \in \Gamma^{(g)}$ **do**

    $\phi' = \phi_i + \epsilon, \quad \text{with} \quad \epsilon \sim N(0, 0.05)$ // Add noise to policy parameters

    $R', C', \tilde{\mathcal{T}} = \text{ROLLOUT}(\hat{p}, s_t, \phi', h)$ // Evaluate policy on the model

    $\bar{b}' = f(\phi', \tilde{\mathcal{T}})$ // Calculate policy behavior descriptors

    $\mathcal{A}_{\text{ME}} \leftarrow \text{STORE}(\mathcal{A}_{\text{ME}}, \phi', \bar{b}', R', C', \bar{\mathcal{B}})$ // Update collection

$\Gamma_{\text{lc}} \leftarrow \mathcal{A}_{\text{ME}}[\min \text{C}]$ // Get policies with lowest cost

$\phi_{\text{best}} \leftarrow \Gamma_{\text{lc}}[\max \text{R}]$ // Get policy with highest reward

$a_t \leftsquigarrow \phi_{\text{best}}(s_t)$ // Get next action

our criterion. With a good planner that spans the search space we can expect to have $\Pi_c \subseteq \Pi_m$. The optimal policy $\pi^* \in \Pi_m$ is the one such that there is no other policy π' with a lower $\text{MC}(\mathcal{T})$ and higher $\text{MR}(\mathcal{T})$ Ray et al. (2019).

3.3 Quality diversity and MAP-Elites

QD methods belong to the family of Evolution Algorithms (EAs) and are designed to achieve two goals simultaneously: generating policies that exhibit diverse behaviors and achieving high performance Pugh et al. (2016); Cully & Demiris (2017). Each policy is parametrized by $\phi_i \in \Phi$ and is executed on the system, resulting in a trajectory $\mathcal{T}_i$. The trajectory is then mapped to a *behavior descriptor* $b_i \in \mathcal{B}$ through an associated behavior function: $f(\mathcal{T}_i) = b_i \in \mathcal{B}$. The space $\mathcal{B}$ is a hand-designed space in which the behavior of each policy is represented. By maximizing the distance of the policies in this space, QD methods can generate a collection of highly diverse policies that is then returned as output of the algorithm. We select the ME algorithm Mouret & Clune (2015) from the range of QD methods Lehman & Stanley (2011); Mouret & Clune (2015); Paolo et al. (2021) due to its simplicity and effectiveness. ME operates by discretizing the behavior space $\mathcal{B}$ into a grid $\bar{\mathcal{B}}$ and searching for the best policies whose discretized behaviors fill up the cells of the grid.

4 Guided safe shooting

This section explains in detail how GuSS works by first describing the training of the learned model and then how the safe-planning process is carried by the QD planner. The code of all the algorithms and experiments is available at `<URL hidden for review>`.

4.1 The learned system model

The goal of model learning in MBRL is, in each iteration τ, to learn $\hat{p}^{(\tau)} : (\boldsymbol{s}_t, \boldsymbol{a}_t) \rightsquigarrow \boldsymbol{s}_{t+1}$ from the collected traces $\mathcal{T}r = \bigcup_{\tau'=1}^{\tau} \mathcal{T}^{(\tau)}$ (Alg. 1). As model class we use autoregressive (DARMDN) and non-autoregressive mixture density nets (DMDN) Bishop (1994), that have recently been used in multiple works Kégl et al. (2021); Chua et al. (2018); Wang et al. (2019). Both are neural nets, outputting parameters of Gaussian

Algorithm 3: STORE function of safe planner

INPUT: collection of policies $\mathcal{A}_{\text{ME}}$, policy parameters ϕ, discretized policy behavior descriptor $\bar{b}$, policy reward R, policy cost C, discretized behavior space $\bar{\mathcal{B}}$
RESULT: updated policy collection $\mathcal{A}_{\text{ME}}$
if $\bar{\mathcal{B}}[\bar{b}] = \varnothing$ **then**
  `/* If no policy with similar behavior descriptor has been found */`
  $\mathcal{A}_{\text{ME}} = \mathcal{A}_{\text{ME}} \bigcup (\phi, \bar{b}, C, R)$; `// Add policy to collection`
else
  $(\phi', \bar{b}', C', R') \leftarrow \mathcal{A}_{\text{ME}}[\bar{b}]$; `// Get policy with similar behavior from` $\mathcal{A}_{\text{ME}}$
  if $(C < C')$ *or* $(C = C' \ \& \ R > R')$ **then**
    $\mathcal{A}_{\text{ME}} = \big(\mathcal{A}_{\text{ME}} - (\phi', \bar{b}', C', R')\big) \bigcup (\phi, \bar{b}, C, R)$; `// Replace` ϕ' `with` ϕ `in` $\mathcal{A}_{\text{ME}}$

distributions, conditioned on the previous state and action. In DARMDN, we learn d^{s} autoregressive deep neural nets $p_\ell(s^\ell_{t+1} | s^1_{t+1}, \ldots, s^{\ell-1}_{t+1}, \boldsymbol{s}_t, \boldsymbol{a}_t)$, $\ell = 1, \ldots, d^{\text{s}}$, outputting a scalar mean and standard deviation for each dimension of the state vector. DMDN learns a single spherical d^{s}-dimensional Gaussian, outputting a mean vector and a standard deviation vector. Both models are trained to maximize the likelihood on the training data $\bigcup_{\tau'=1}^{\tau} \mathcal{T}^{(\tau)}$. We choose DARMDN for smaller dimensional systems and DMDN for SafeCar-Goal. The models are trained by optimizing the negative log likelihood loss:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}' \sim \mathcal{T}} \left\{ -log \mathcal{N}(\boldsymbol{s}'; \boldsymbol{s} + \mu_\theta(\boldsymbol{s}, \boldsymbol{a}), \sum(\boldsymbol{s}, \boldsymbol{a})) \right\}, \tag{4}$$

where $(\boldsymbol{s}, \boldsymbol{a}) = (\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $\boldsymbol{s}' = \boldsymbol{s}_{t+1}$. All hyperparameters were tuned on static data generated from a random policy and kept unchanged for all episodes.

Algorithm 4: SELECT function of safe planner

INPUT: collection of policies $\mathcal{A}_{\text{ME}}$, number of selected policies n_{new}, policy parameter space Φ
RESULT: set of selected policies Γ
$\Gamma = \mathcal{A}_{\text{ME}}[n_{new}, C = 0]$ `// Select` n_{new} `policies with` $C = 0$ `from collection`
if $|\Gamma| < n_{new}$ **then**
  $\Gamma = \Gamma \bigcup \text{SAMPLE}(\Phi, |\Gamma| - n_{new})$ `// Sample missing policies from` Φ

4.2 Model-based safe quality-diversity

One of the main contributions of this paper is the application of QD in a learned model, enabling the use of this powerful technique in scenarios where system access is costly in terms of resources and safety. In our approach, the QD planner (Alg. 2) is invoked by GuSS at each time step t to generate the action $\boldsymbol{a}_t$ for the system. The planner begins by initializing a pool of n planning policies, denoted as $\Gamma^{(0)} = [\phi_i]_{i=1}^n$. In our case, these policies are represented by neural networks with random weights. These policies are evaluated on the learned model $\hat{p}$ for a duration of h timesteps, starting from the current state of the real system $\boldsymbol{s}_t$. The evaluation process produces a reward R_i, a cost C_i, and a trace of simulated states $\tilde{\mathcal{T}} i = ((\boldsymbol{s}_t, \boldsymbol{a}_0), \ldots, (\boldsymbol{s}t + h, \boldsymbol{a}_h))$ for each policy ϕ_i. To map the trace to the policy's discretized behavior descriptor, we employ a hand-designed, environment-specific behavior function $f(\cdot)$, resulting in $\bar{b}_i \in \mathcal{B}$. Finally, the evaluated policies are stored in the collection $\mathcal{A}_{\text{ME}}$ through the STORE function (Alg. 3). This is a fundamental part of the planner as it is here that the policies' safety comes into play in the evaluation. When a policy ϕ_i with a unique discrete behavior descriptor is encountered, the tuple $(\phi_i, \bar{b}_i, C_i, R_i)$ is directly added to the collection. However, if another policy ϕ_j with $\bar{b}_j = \bar{b}_i$ already exists in the collection, only the better of the two policies is retained. Note that in this context the "better policy" refers to the one with the *lowest cost*. In the case of the two policies having the same cost, the one with the highest reward is stored. This strategy allows the generation of a low-cost and high-reward collection of policies that can then be used to generate the next action at the end of the planning episode.

The planner then starts an iteration of multiple evolutionary generations until a total of M planning policies have been evaluated. In each generation g, a pool $\Gamma^{(g)}$ of n_{new} policies with cost $C = 0$ are uniformly sampled

from $\mathcal{A}_{\text{ME}}$ through the SELECT function (Alg. 4). If not enough policies with zero cost are present in the collection, additional policies with random weights are included in $\Gamma^{(g)}$ until its size matches n_{new}. This facilitates increased exploration, aiding in the discovery of safer planning policies. The parameters of the policies in $\Gamma^{(g)}$ are then perturbed by adding Gaussian noise $\epsilon \sim N(0, 0.05)$ to generate new policy parameters, denoted as $\phi' = \phi_i + \epsilon$. The new policies are evaluated on the model and stored in the collection using the STORE function. Once a total of M policies have been generated, a pool Γ_{lc} consisting of policies with the lowest cost is selected from $\mathcal{A}_{\text{ME}}$. Among these policies, the one with the highest reward is chosen as the final policy to generate the next action for application on the real system, i.e., $\boldsymbol{a}_t = \phi_{\text{best}}(\boldsymbol{s}_t)$.

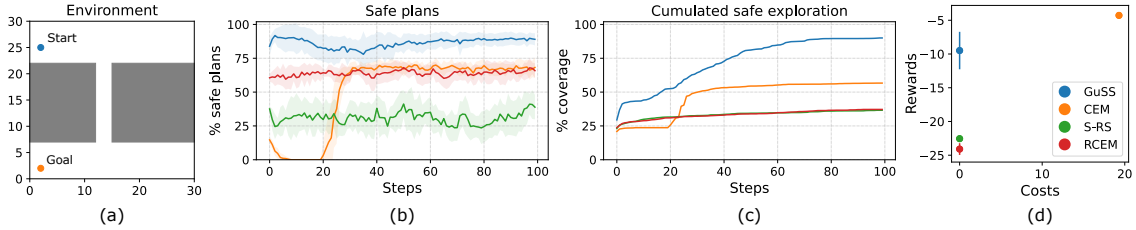

Figure 2: (a) Toy environment. The agent has to navigate from Start to Goal without traversing the unsafe areas in gray. (b) Percentage of safe plans at each step. (c) Total amount of the safe space explored through safe plans. (d) Average performance of the algorithms. The results show the mean over 10 seeds, shaded areas represent one standard deviation.

5 Experiments

5.1 Exploration

In this section, we test our safe planner on a toy environment (Fig. 2.(a)) to highlight the importance of safe exploration in safety-critical tasks and analyze the performance of our algorithm. The environment is designed to require a significant amount of safe exploration to reach the goal. The agent must move from Start to the Goal by observing its current position, $\boldsymbol{s} = (x, y)$, and performing actions, $\boldsymbol{a} = (\delta_x, \delta_y) \in \{-1, 0, 1\}$, which control the (x, y) movement at the next time step. At each time step, the reward is given by the negative distance between the agent and the goal, $r(\boldsymbol{s}, \boldsymbol{a}) = -\|\boldsymbol{s} - \boldsymbol{s}_{goal}\|^2$, while the cost is equal to $c(\boldsymbol{s}, \boldsymbol{a}) = 1$ if the agent is in the unsafe areas and to $c(\boldsymbol{s}, \boldsymbol{a}) = 0$ otherwise. Each algorithm has an evaluation budget of $N = 500$ plans with a planning horizon $H = 50$. The behavior space used by GuSS is the plans' final (x, y) position. We compare our method against three other planners: CEM Chua et al. (2018), RCEM Liu et al. (2020) and Safe Random Shooting (S-RS). The first one does not take into account the safety constraints while the other two do, with S-RS being a random-shooting planner that rejects all plans with safety violations.

To decouple the performance of the planners from the performance of the model, we perform no model learning and instead use a perfect model to evaluate the plans. The evaluation is performed for a single episode of length $T = 100$ over 10 random seeds. The exploration is evaluated as the percentage of safe plans generated at each step (Fig. 2.(b)) and the amount of safe space covered by the generated safe plans (Fig. 2.(c)). This last metric is calculated by dividing the space into a 50×50 grid and counting the percentage of cells in the safe space visited by *safe plans*. We can see that GuSS performs better than the other approaches in both metrics ($p < 1.8e-4$), generating a high percentage of safe plans while exploring a large portion of the safe state-space. This advantage is also reflected in the high reward with zero cost that GuSS can obtain (Fig. 2.(d)). At the same time, the two other safe planners (RCEM and S-RS) explore a very low percentage of the safe space, never discovering the narrow path that leads to the goal. The only other planner obtaining high rewards is the unsafe CEM. This is achieved by directly traversing the unsafe areas, as can be seen by the percentage of safe plans dropping to 0% in Fig. 2.(b) before going up again once the agent leaves the unsafe zone.

Table 1: Summary of the different methods on the three different environments. All the metrics are average over all epochs and seeds and ↓ and ↑ mean lower and higher the better, respectively. The best **safe methods** with respect to each metric are highlighted in bold. All ± values are 90% Gaussian confidence interval.

Method	MAR ↑	MRCP×10^3 ↓	p(unsafe)[%] ↓	p(unsafe)$_{trans}$[%] ↓
	Safe Pendulum $r_{thr} = -2.5$			
S-RS	**-2.09 ± 0.09**	2.07 ± 1.21	**0.31 ± 0.55**	1.11 ± 1.12
GuSS	-2.2 ± 0.16	**1.7 ± 0.88**	0.68 ± 0.59	**0.68 ± 0.65**
Safe-MBPO	-3.38 ± 0.38	7.1 ± 0.61	0.61 ± 0.79	1.56 ± 0.83
RCEM	-6.12 ± 0.10	- ± -	0.73 ± 0.63	0.94 ± 0.52
RS	-2.7 ± 0.14	2.43 ± 1.03	1.49 ± 0.96	1.26 ± 0.49
CEM	-2.99 ± 0.15	1.57 ± 0.36	1.57 ± 0.80	1.67 ± 0.43
CPO	-6.31 ± 0.03	22 ± 0.0	1.72 ± 0.80	1.61 ± 0.72
PPO lag	-5.47 ± 0.04	138 ± 34	3.36 ± 1.25	2.63 ± 0.85
	Safe Acrobot $r_{thr} = 1.6$			
S-RS	1.4 ± 0.05	1.6 ± 0.21	**1.23 ± 1.24**	**0.85 ± 1.09**
GuSS	1.64 ± 0.01	**1.36 ± 0.25**	1.56 ± 1.45	3.24 ± 2.51
RCEM	**1.67 ± 0.01**	1.6 ± 0.37	2.21 ± 1.53	2.03 ± 1.73
RS	2.06 ± 0.01	1.33 ± 0.25	20.94 ± 8.86	4.08 ± 3.61
CEM	2.09 ± 0.02	1.40 ± 0.43	20.07 ± 9.11	3.00 ± 2.90
CPO	0.87 ± 0.01	87 ± 59	5.09 ± 1.66	3.83 ± 2.20
PPO lag	0.93 ± 0.01	26 ± 4	3.82 ± 1.87	4.37 ± 2.19
	SafeCar-Goal $r_{thr} = 10$			
S-RS	-0.29 ± 0.22	- ± -	0.69 ± 1.04	0.49 ± 0.83
GuSS	**11.96 ± 0.71**	**48.17 ± 13.69**	2. ± 1.96	2.17 ± 2.44
Safe-MBPO	-17.92 ± 1.13	- ± -	1.54 ± 3.50	2.93 ± 3.29
RCEM	-1.38 ± 0.15	- ± -	**0.40 ± 0.52**	0.60 ± 0.58
RS	-0.43 ± 0.18	- ± -	0.63 ± 1.09	0.51 ± 0.65
CEM	3.87 ± 0.72	60 ± 6.41	1.49 ± 1.89	1.11 ± 1.49
CPO	3.95 ± 0.09	246 ± 136.4	0.45 ± 0.81	0.37 ± 0.52
PPO lag	5.36 ± 0.1	256 ± 87	0.56 ± 1.05	**0.35 ± 0.43**

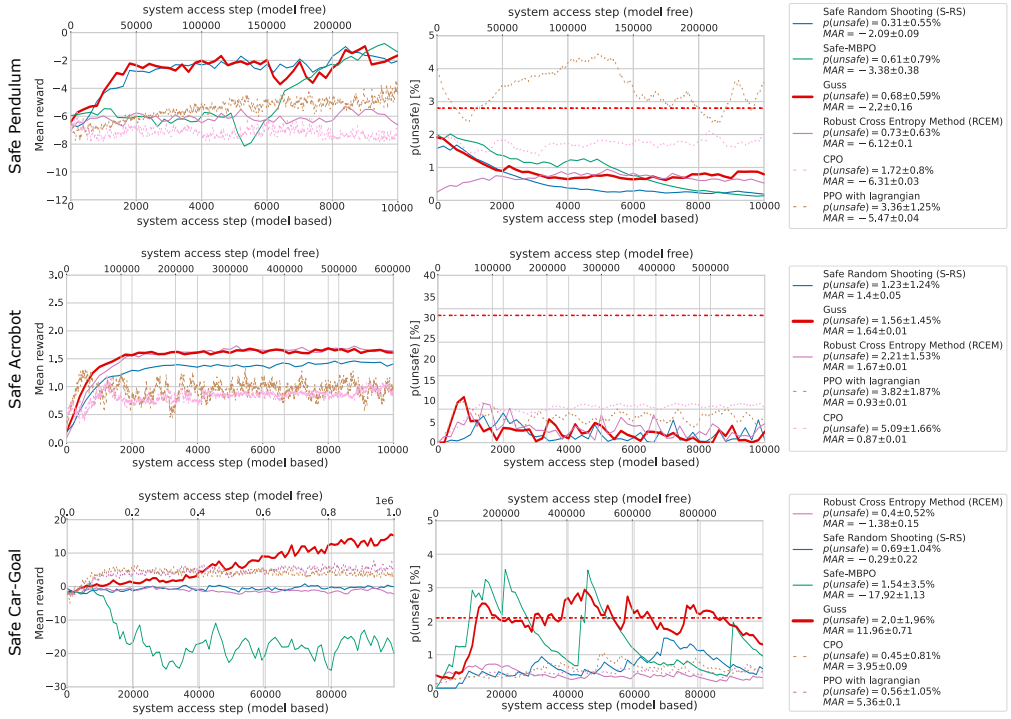

Figure 3: Mean reward and probability percentage of unsafe for the three test environments. Dashed curves indicate Model-free baselines and plain one Model-based approaches. The red dashed line indicates the random unsafe probability. All curves represent the mean over 6 random seed.

5.2 Environments

We evaluated the performance of GuSS on three different OpenAI gym environments with safety constraints: pendulum swing-up, Acrobot with discrete actions and SafeCar-Goal from the safety-gym environment Ray et al. (2019). In designing the environments, we followed previous work Cowen-Rivers et al. (2022); Ray et al. (2019) with the exception of SafeCar-Goal, for which we used the original version by Ray2019 with the position of the unsafe areas randomly resampled at the beginning of each episode. Moreover, we use lidar observations rather than the robot position. Each environment can be seen as an ergodic MDP so at each safe time-step it is possible to recover a safe policy.

5.3 Results

We compared GuSS against various baselines to determine how much different the performances of safe MPC methods are in comparison to unsafe ones. We compared against **RS** Nagabandi et al. (2018) , and **CEM** Chua et al. (2018). We also choose two recent safe MBRL approaches: **Safe-MBPO** Thomas et al. (2021), and **RCEM** Liu et al. (2020), together with the simple S-RS planner. Additionally, to demonstrate the sample efficiency of model-based approaches, we compared against two safe model-free baselines: **CPO** Achiam et al. (2017), **PPO Lagrangian**; all of which come from the Safety-Gym benchmark Ray et al. (2019).

The algorithms were compared according to four metrics: Mean Asymptotic Reward (**MAR**), Mean Reward Convergence Pace (**MRCP**), Probability percentage of unsafety (**p(unsafe)**[%]) and transient probability percentage of unsafety (**p(unsafe)**$[\%]_{trans}$). The details on how these metrics are calculated are defined in the Appendix. The MAR scores and the p(unsafe)[%] for the pendulum system, Acrobot system and SafeCar-Goal environment are shown in Fig. 3 and in Table 1. The results on the three environments show that the increased exploration provided by GuSS allows it to solve the task without incurring in high costs. On the pendulum system, while Safe-MBPO ($p < 5e-7$) reaches the highest reward, it needs many more

episodes than GuSS. At the same time, GuSS and S-RS reach similar high MAR ($p = 7.43e-2$) and low cost ($p = 1.64e-1$). The other approaches (RCEM, CPO and PPO langrangian) have instead much lower rewards than GuSS ($p < 5.5e-17$), and fail at solving this simple environment. As expected, on the Acrobot system, safe methods cannot reach the highest MAR scores possible due to the safety constraints blocking the high-rewarding states. GuSS outperforms the simple S-RS method in terms of MAR ($p < 3e-6$) but has a slightly lower MAR than RCEM ($p < 1e-4$). At the same time, the three algorithms have similar low cost ($p > 0.5$). On this environment as well, the two model-free approaches reach lower MAR scores than the model-based ones ($p < 7e-14$) with higher costs ($p < 8e-3$) and a much higher number of system access steps. Note that Safe-MBPO has not been tested on this environment as Safe-MBPO's SAC only works on continuous action spaces. The advantage of using a QD-based planner compared to simpler ones as S-RS and RCEM is clear from the results on the hardest of the three environment: SafeCar-Goal. GuSS is the only safe-MBRL method to solve the environment and even outperform model-free and unconstrained approaches in terms of MAR ($p < 1.75e-8$), with Safe-MBPO fails completely in reaching any of the goals. At the same time, all the tested algorithms have shown no statistically significant differences from the point of view of the cost.

6 Conclusion and Future work

In this study, we proposed GuSS, the first model based planning method using QD methods for safe reinforcement learning. QD algorithms are methods explicitly designed to provide good exploration. We leverage this property to design a safe planner for GuSS. We demonstrated the necessity of safe exploration in safety-critical settings by comparing our planner to three other (safe and unsafe) planners in a simple toy environment. Only the QD-based safe planner consistently solved the toy environment, achieving high rewards and no cost thanks to maximally exploring the safe space. We further demonstrated on three benchmark environments with safety constraints how GuSS compares favorably with state-of-the-art model-free and model-based safe algorithms in terms of the trade off between performance and safety, while requiring minimal computational complexity. Especially on SafeCar-Goal, GuSS is the only method that manage to solve the environment.

In conclusion, Guided Safe Shooting (GuSS), demonstrates promising results in balancing performance and cost in safety-critical reinforcement learning environments. However, the performance of GuSS is dependent on the accuracy of the model used. If the model is wrong, it could easily lead the agent to unsafe states. In future work, we will work on incorporating model uncertainty with QD to inform the agent about the risk of its actions to reduce unsafe behavior during the model learning phase. Additionally, the need to hand-design the behavior space in QD-based algorithms limits the range of applicability. While some works have been proposed to address this issue Cully (2019); Paolo et al. (2020), how to autonomously build such space still remains an open question. Having an algorithm that could learn both a model of the system and a good representation for the behavior space of its planner would likely greatly improve the performance and efficiency of such methods.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. PMLR, 2017.

Eitan Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30, 2017.

Christopher M. Bishop. Mixture density networks. Technical report, 1994.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pp. 4754–4765. Curran Associates, Inc., 2018.

Alexander I Cowen-Rivers, Daniel Palenicek, Vincent Moens, Mohammed Amin Abdullah, Aivar Sootla, Jun Wang, and Haitham Bou-Ammar. Samba: Safe model-based & active reinforcement learning. *Machine Learning*, pp. 1–31, 2022.

Antoine Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89, 2019.

Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

Hao-Lun Hsu, Qiuhua Huang, and Sehoon Ha. Improving safety in deep reinforcement learning using unsupervised action planning. *arXiv preprint arXiv:2109.14325*, 2021.

Marcus Hutter. Self-optimizing and pareto-optimal policies in general environments based on bayes-mixtures. In *Computational Learning Theory: 15th Annual Conference on Computational Learning Theory, COLT 2002 Sydney, Australia, July 8–10, 2002 Proceedings*, pp. 364–379. Springer, 2002.

Balázs Kégl, Gabriel Hurtado, and Albert Thomas. Model-based micro-data reinforcement learning: what are the crucial model properties and which model to choose? In *9th International Conference on Learning Representations, ICLR 2021*, 2021. URL https://openreview.net/forum?id=p5uylG94S68.

Malte Kuss and Carl Rasmussen. Gaussian processes in reinforcement learning. *Advances in neural information processing systems*, 16, 2003.

Prashanth La and Mohammad Ghavamzadeh. Actor-critic algorithms for risk-sensitive mdps. *Advances in neural information processing systems*, 26, 2013.

Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, 2012.

Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 3814–3823. Curran Associates, Inc., 2018.

Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, 2011.

Zuxin Liu, Hongyi Zhou, Baiming Chen, Sicheng Zhong, Martial Hebert, and Ding Zhao. Constrained model-based reinforcement learning with robust cross-entropy method. *arXiv preprint arXiv:2010.07968*, 2020.

Yecheng Jason Ma, Andrew Shen, Osbert Bastani, and Dinesh Jayaraman. Conservative and adaptive penalty for model-based safe reinforcement learning. *arXiv preprint arXiv:2112.07701*, 2021.

Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. Deployment-efficient reinforcement learning via model-based offline optimization. In *International Conference on Learning Representations*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2021.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018*, pp. 7559–7566. IEEE, 2018.

Giuseppe Paolo, Alban Laflaquiere, Alexandre Coninx, and Stephane Doncieux. Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2379–2385. IEEE, 2020.

Giuseppe Paolo, Alexandre Coninx, Stéphane Doncieux, and Alban Laflaquière. Sparse reward exploration via novelty search and emitters. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 154–162, 2021.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 2015. PMLR.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Satinder Singh, Tommi Jaakkola, and Michael Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. Touretzky, and T. Leen (eds.), *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1995.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future. *Advances in Neural Information Processing Systems*, 34, 2021.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Stefan Radic Webster and Peter Flach. Risk sensitive model-based reinforcement learning using uncertainty guided planning. *arXiv preprint arXiv:2111.04972*, 2021.

Min Wen and Ufuk Topcu. Constrained cross-entropy method for safe reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-based constrained policy optimization. *arXiv preprint arXiv:2010.03152*, 2020a.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Accelerating safe reinforcement learning with constraint-mismatched policies. *arXiv preprint arXiv:2006.11645*, 2020b.

Mario Zanon and Sébastien Gros. Safe reinforcement learning using robust mpc. *IEEE Transactions on Automatic Control*, 66(8):3638–3652, 2020.

Yiming Zhang, Quan Vuong, and Keith Ross. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 33:15338–15349, 2020.