Lecture Notes in Computer Science: Safe Exploration Techniques for Reinforcement Learning – An Overview

Martin Pecka[1⋆], and Tomas Svoboda[1,2⋆⋆]

[1] Center for Machine Perception, Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, martin.pecka@fel.cvut.cz, http://cmp.felk.cvut.cz/∼peckama2
[2] Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic

Abstract. We overview different approaches to safety in (semi)autonomous robotics. Particularly, we focus on how to achieve safe behavior of a robot if it is requested to perform exploration of unknown states. Presented methods are studied from the viewpoint of *reinforcement learning*, a partially-supervised machine learning method. To collect training data for this algorithm, the robot is required to freely explore the state space – which can lead to possibly dangerous situations. The role of *safe exploration* is to provide a framework allowing exploration while preserving safety. The examined methods range from simple algorithms to sophisticated methods based on previous experience or state prediction. Our overview also addresses the issues of how to define safety in the real-world applications (apparently absolute safety is unachievable in the continuous and random real world). In the conclusion we also suggest several ways that are worth researching more thoroughly.

Keywords: Safe exploration, policy search, reinforcement learning

1 Introduction

Reinforcement learning (RL) as a machine learning method has been thoroughly examined since 80's. In 1981, Sutton and Barto [3] inspired themselves in the reinforcement learning discoveries in behavioral psychology and devised the *Temporal Difference* machine learning algorithm that had to simulate psychological classical conditioning. In contrast with *supervised learning*, reinforcement learning does not need a teacher's classification for every sample presented. Instead, it just collects rewards (or punishment) on-the-go and optimizes for the expected

⋆ Supported by the Grant Agency of the Czech Technical University in Prague under Project SGS13/142/OHK3/2T/13
⋆⋆ Supported by the EC project FP7-ICT-609763 TRADR

long-term reward (whereas supervised learning optimizes for the immediate reward). The key advantage is that the design of the rewards is often much simpler and straight-forward than classifying all data samples.

Reinforcement learning proved to be extremely useful in the case of state-space exploration – the long-term reward corresponds to the value of each state [17]. From such values, we can compose a *policy* which tells the agent to always take the action leading to the state with the highest value. As an addition, state values are easily interpretable for humans.

Since the early years, a lot of advanced methods were devised in the area of reinforcement learning. To name one, *Q-learning* [25] is often used in connection with safe exploration. Instead of computing the values of states, it computes the values of state–action pairs, which has some simplifying consequences. For example, Q-learning doesn't need any *transition model* (i.e. dynamics model) of the examined system.

A completely different approach is *policy iteration*. This algorithm starts with a (more or less random) policy and tries to improve it step-by-step [16]. This case is very valuable if there already exists a good policy and we only want to improve it [11].

What do all of these methods have in common, is the need for rather large training data sets. For simulated environments it is usually not a problem. But with real robotic hardware, the collection of training samples is not only lengthy, but also dangerous (be it mechanical wear or other effects). Another common feature of RL algorithms is the need to enter *unknown* states, which is inherently unsafe.

As can be seen from the previous paragraph, *safety* is an important issue connected with reinforcement learning. However, the first articles focused on maintaining safety during exploration started to appear much later after the "discovery" of RL. Among the first, Heger [15] "borrowed" the concept of a *worst-case criterion* from control theory community. In 1994 he created a variant of Q-learning where maximization of long-term reward is replaced with maximization of minimum of the possible rewards. That basically means his algorithm prefers to never encounter a bad state (or, at least to choose the best of the bad states). This approach has one substantial drawback – the resulting policies are far from being optimal in the long-term–reward sense [10].

In this paper we show the various approaches to safe exploration that have emerged so far. We classify the methods by various criteria and suggest suitable use cases for them. To better illustrate some of the practical details, we use the UGV (Unmanned Ground Vehicle) robotic platform from EU FP7 project NIFTi [6] (see Figure 1) as a reference agent. It may happen that in these practical details we assume some advantages of UGVs over UAVs (Unmanned Aerial Vehicles), like the ability to stand still without much effort, but it is mostly easy to convert these assumptions to UAVs, too.

Further organization of this paper is the following: in Section 2 we discuss some basics of reinforcement learning (the reader may skip it if he is familiar with reinforcement learning); Section 3 is an overview of the safety definitions

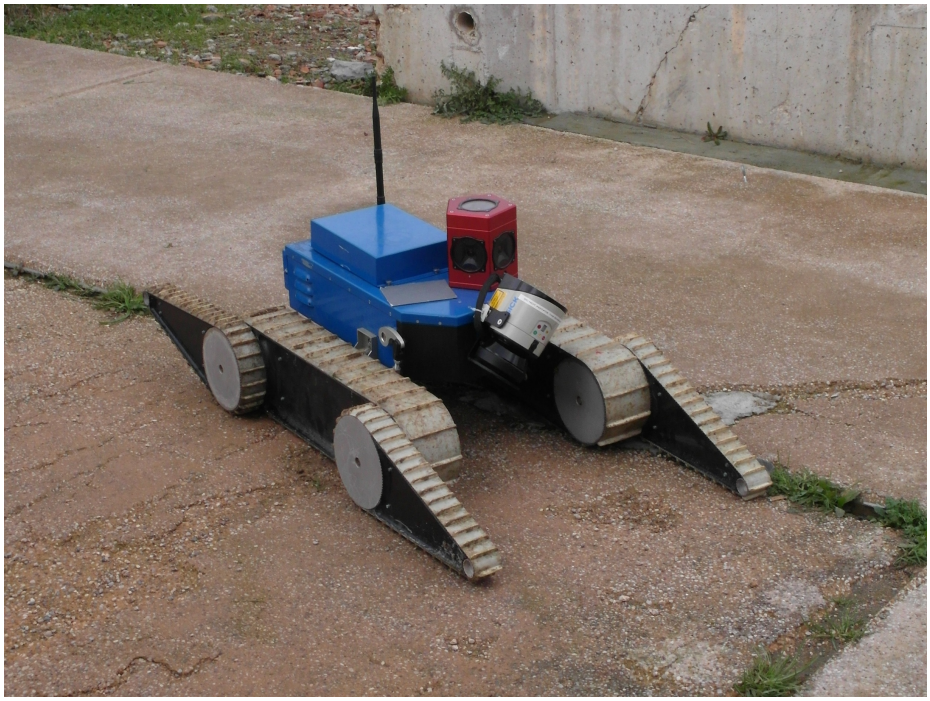

Fig. 1. NIFTi UGV robotic platform

used in literature; Section 4 is the main part concerning the various approaches to safe exploration, and in Section 5 we conclude the findings and we suggest some further areas of possible research.

2 Reinforcement learning basics

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) are the standard model for deliberating about reinforcement learning problems. They provide a lot of simplifications, but are sufficiently robust to describe a large set of real-world problems.

The simplest discrete stochastic MDP comprises of: [17]

- a finite set of states $\mathbf{S}$
- a finite set of actions $\mathbf{A}$
- a stochastic *transition model* $\mathbf{P}$: $\mathbf{P}_t(s, a, s') = Pr(s_{t+1} = s' \mid s_t = s,\ a_t = a)$ for each $s, s' \in \mathbf{S}$, $a\ \in \mathbf{A}$, where Pr stands for probability
- and the *immediate reward function* $\mathbf{R}$: $\mathbf{S} \times \mathbf{A} \to \mathbb{R}$ (or $\mathbf{R}$: $\mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$ if the reward depends on the stochastic action result)

To interpret this definition, we say that the at every time instant t the *agent* is in a state s, and by executing action a it gets to a new state s'. Furthermore, executing a particular action in a particular state may bring a *reward* to the agent (defined by $\mathbf{R}$).

The most important and interesting property of MDPs is the *Markov property*. If you have a look at the definition of the transition model, the next state only depends on the current state and the chosen action. Particularly, the next state is independent of all the previous states and actions but the current one. To give an example, the robot's battery level cannot be treated implicitly by counting the elapsed time, but rather it has to be modeled as a part of the robot's state.

Once the model is set up, everything is ready for utilizing an MDP. "The agent's job is to find a policy π mapping states to actions, that maximizes some long-run measure of reinforcement" [17]. The "long-run" may have different meanings, but there are two favorite optimality models: the first one is the *finite horizon* model, where the term $J = \sum_{t=0}^{h} r_t$ is maximized (h is a predefined time horizon and r_t is the reward obtained in time instant t while executing policy π). The dependency of r_t on the policy is no longer obvious from this notation, but this is the convention used in literature when it is clear which policy is used. This model represents the behavior of the robot which only depends on a predefined number of future states and actions.

The other optimality model is called *discounted infinite horizon*, which means we maximize the discounted sum $J = \sum_{t=0}^{\infty} \gamma^t r_t$ with $\gamma \in (0, 1)$ being the *discount factor*. The infinite horizon tries to find a policy that is the best one taking into account the whole future. Please note the hidden dependency on the policy π (and the starting state s_0) – it is the policy that decides on which action to take, which in turn specifies what will the reward be.

Other extensions of MDPs to continuous states, time or actions are beyond the scope of this overview. However, some of the referenced papers make use of these continuous extensions, which proved to be useful for practical applications.

2.2 Value iteration

Value iteration is one of the basic methods for finding the optimal policy. To describe this algorithm, it is first needed to define the essential notion of the *optimal value of a state*. In this whole subsection we suppose the discounted infinite horizon model, but analogous results can be shown for finite horizon, too. "The optimal value of a state is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy." [17] Given a policy π, the induced value function is therefore defined as

$$\mathbf{V}_\pi(s) = E\left[\sum_{t=0}^{\infty} r_k \gamma^k\right], \tag{1}$$

where E denotes the expected value and r_k are the rewards for executing policy π. Taking the best value function over all policies then yields the *optimal value*

function $\mathbf{V}^*$: [17]

$$\mathbf{V}^*(s) = \max_{\pi} \mathbf{V}_\pi(s)\,. \tag{2}$$

Inversely, if we have the value function given, we can derive a policy from that. It is a simple policy that always takes the action leading to the most profitable neighbor state (with the highest value).

One useful formulation of the properties of the optimal value function is the formulation using the recurrent *Bellman equations* which define a dynamic system that is stable for the optimal value function. We can say a state's optimal value is the best immediate reward plus its best neighbor's optimal value: [17]

$$\mathbf{V}^*(s) = \max_{a} \left(\mathbf{R}(s,a) + \gamma \sum_{s' \in \mathbf{S}} \mathbf{P}(s,a,s') \mathbf{V}^*(s') \right). \tag{3}$$

Analogously, we can find the optimal policy using the same Bellman equation:

$$\pi^*(s) = \operatorname*{argmax}_{a} \left(\mathbf{R}(s,a) + \gamma \sum_{s' \in \mathbf{S}} \mathbf{P}(s,a,s') \mathbf{V}^*(s') \right). \tag{4}$$

The Value iteration algorithm is based on trying to compute the solution of Equation 4 using iterative Bellman updates (refer to Algorithm 1). In the algorithm, we use a structure called $\mathbf{Q}$ to store the "value" of state-action pairs. In Value iteration it is just a structure to save intermediate results, but it is the core of the Q-learning algorithm (described in Section 2.3). The stopping criterion of the Value iteration algorithm is not obvious, but Williams and Baird [26] derived an easily applicable upper bound on the error of the computed value function.

That said, after a sufficient number of those simple iterations, we can compute the almost optimal value function. The number of iterations needed for Value iteration to converge may be impractically high, but it is shown that the optimal policy converges faster [4], thus making Value iteration practical.

2.3 Q-learning

Just a small change to the Value iteration algorithm results in Q-learning. The basic algorithm is the same as Value iteration, just the update step is done differently (refer to Algorithm 2). The consequence of this change is that no model of the system (transition function $\mathbf{P}$) is needed. It is sufficient to execute all actions in all states equally often, and Watkins [25] proved that if Q-learning were run for an infinite time, the computed $\mathbf{Q}$ would converge to the optimal $\mathbf{Q}^*$ (an analogue of $\mathbf{V}^*$).

2.4 Policy iteration

Policy iteration is a completely different approach to computing the optimal policy. Instead of deriving the policy from the Value or Q function, Policy iteration

Algorithm 1 The Value iteration algorithm [17]

Input: an MDP (states S, actions A, rewards R, transition model P)
Output: the optimal value function $\mathbf{V}^*$, resp. the optimal policy π^* derived from the value function

1. $\mathbf{V}(s) :=$ arbitrary function
2. $\pi :=$ the policy derived from $\mathbf{V}$
3. while π is not good enough do
4. for all $s \in \mathbf{S}$ do
5. for all $a \in \mathbf{A}$ do
   Update:
6. $\mathbf{Q}(s,a) := \mathbf{R}(s,a) + \gamma \sum_{s' \in \mathbf{S}} \mathbf{P}(s,a,s')\mathbf{V}(s')$
7. end for
8. $\mathbf{V}(s) := \max_a \mathbf{Q}(s,a)$
9. end for
10. $\pi :=$ the policy derived from $\mathbf{V}$
11. end while
12. $\mathbf{V}^* := \mathbf{V}$, $\pi^* := \pi$

Algorithm 2 The Q-learning algorithm (only the parts that differ from Value iteration when $\mathbf{V}$ is substituted with $\mathbf{Q}$) [17]

Input: an MDP (states S, actions A, rewards R, transition model may be unknown)
Output: the optimal state-value function $\mathbf{Q}^*$, resp. the optimal policy π^* derived from the state-value function

6. $\mathbf{Q}(s,a) := \mathbf{Q}(s,a) +$
   $\alpha \Big[\mathbf{R}(s,a) + \gamma \max_{a'} \mathbf{Q}(s',a') - \mathbf{Q}(s,a) \Big]$
8. line left out

works directly with policies. In the first step, a random policy is chosen. Then a loop consisting of policy evaluation and policy improvement repeats as long as the policy can be improved [17] (refer to Algorithm 3 for details). Since in every step the policy gets better, and there is a finite number of different policies, it is apparent that the algorithm converges [23].

Policy iteration can be initialized by a known, but suboptimal policy. Such policy can be obtained e.g. by a human operator driving the UGV. If the initial policy is good, Policy iteration has to search much smaller subspace and thus should converge more quickly than with a random initial policy [11].

Algorithm 3 The Policy iteration algorithm [17]

1. $\pi' =$ `arbitrary policy`
2. `repeat`
3. $\pi := \pi'$

   `Policy evaluation: (system of linear equations)`
4. $\mathbf{V}_\pi(s) = \mathbf{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathbf{S}} \mathbf{P}(s, \pi(s), s') \mathbf{V}_\pi(s')$

   `Policy improvement:`

   $\pi'(s) := \operatorname{argmax}_{a \in \mathbf{A}} \Big[$
5. $\quad \mathbf{R}(s, a) + \gamma \sum_{s' \in \mathbf{S}} \mathbf{P}(s, a, s') \mathbf{V}_\pi(s') \Big]$
6. `until` $\pi = \pi'$

3 Defining safety

To examine the problems of safe exploration, it is first needed to define what exactly is the *safety* we want to maintain. Unfortunately, there is no unified definition that would satisfy all use cases; thus, several different approaches are found in the literature. An intuitive (but vague) definition could be e.g.: "State-space exploration is considered *safe* if it doesn't lead the agent to unrecoverable and unwanted states." It is worth noticing here that *unwanted* doesn't necessarily mean low-reward. In the next subsections we present the main interpretations of this vague definition.

3.1 Safety through labeling

The largely most used definition of safety is labeling the states/actions with one of several labels indicating the level of safety in that state/action. What varies from author to author is the number and names of these labels.

To start with, Hans [14] has the most granular division of state/action space. His definitions are as follows (slightly reformulated):

- an (s, a, r, s') tuple (transition) is **fatal** if the reward r is less than a certain threshold (s is the original state, a is an action and s' is the state obtained after executing a in state s, yielding the reward r),
- an action a is **fatal** in state s if there is non-zero probability of leading to a fatal transition,
- state s is called **supercritical** if there exists no policy that would guarantee no fatal transition occurs when the agent starts in state s,
- action a is **supercritical** in state s if it can lead to a supercritical state,
- state s is called **critical** if there is a supercritical or fatal action in that state (and the state itself is not supercritical),
- action a is **critical** in state s if it leads to a critical state (and the action itself is neither supercritical nor fatal in s),
- state s is called **safe** if it is neither critical nor supercritical,
- action a is **safe** in state s if it is neither critical, nor supercritical, nor fatal in state s,
- and finally a policy is **safe** if for all critical states it leads to a safe state in a finite number of non-fatal transitions (and if it only executes safe actions in safe states).

Since we will compare other definitions the the Hans', it is needed to define one more category. A state s is called **fatal** if it is an undesired or unrecoverable state, e.g. if the robot is considered broken in that state. The fatal transition can then be redefined as a transition ending in a fatal state. Opposite to the precisely defined terms in Hans' definition, the meaning of words "undesired" and "unrecoverable" here is vague and strongly task-dependent.

Continuing on, Geibel [12] defines only two categories – *fatal* and *goal* states. "Fatal states are terminal states. This means, that the existence of the agent ends when it reaches a fatal state" [12]. This roughly corresponds to our defined set of **fatal** states. *Goal states* are the rest of final states that correspond to successful termination. Since Geibel only considers terminal states for safety, his *goal* states correspond to a subset of **safe** states. The other Hans' categories need not be represented, since they are meaningless for final states.

An extension of Geibel's *fatal* and *goal* states is a division presented by García [10]. His *error* and *non-error* states correspond to *fatal* and *goal* states, but García adds another division of the space – the *known* and *unknown* states, where *known* states are those already visited (and *known* have empty intersection with *error*). He then mentions a prerequisite on the MDP that if an action leads to a known *error*/*non-error* state, then its slight modification must also lead to an *error*/*non-error* state (a metric over the state space is required).

In Ertle's work [9], again the two basic regions are considered – they are called *desired* and *hazardous* (corresponding to **safe** and **fatal**). However, due to the used learning technique, one more region emerges – the *undesired* region. It contains the whole *hazardous* region and a "small span" comprising of *desired* states, and denotes the set of states where no training (safe) samples are available, because it would be dangerous to acquire those samples. In particular, he says that "The hazards must be 'encircled' by the indications of the undesired approaching so that it becomes clear which area [...] is undesired" [9].

A summary of the labeling-based definitions is shown in Figure 3. We examined the apparent imbalance between the number of categories Hans defines, and the other definitions, and that led us to the following observations.

The first observation is that creating labels for actions or transitions is unnecessary. If we need to talk about the "level of safety" of an action, we can use the worst label out of all possible results of that action (which retains compatibility with Hans' definitions). Moreover, as "it is impossible to completely avoid error states" [22], we can ignore the effects of the action which have only small probability (lower than a safety threshold) – we will call such effects the *negligible effects*.

A second remark is that the **fatal** and **supercritical** sets can be merged. In Hans' work we haven't found any situation where distinguishing between **supercritical** and **fatal** would bring any benefit. Specifically, in his work Hans states that: "Our objective is to never observe supercritical states" [14], which effectively involves avoiding fatal transitions, too. And since we avoid both supercritical and fatal, we can as well avoid their union.

Third, safety of a state does not necessarily depend on the reward for getting to that state. E.g. when the UGV performs a victim detection task, going away from the target area may be perfectly safe, but the reward for such action should be small or even negative.

Putting these observations together, we propose a novelty definition of safety for stochastic MDPs, which is a simplification of Hans' model and a generalization of the other models:

- A state is **unsafe** if it means the agent is damaged/destroyed/stuck... or it is highly probable that it will get to such state regardless of further actions taken.
- A state is **critical** if there is a not negligible action leading to an unsafe state from it.
- A state is **safe** if no available action leads to an unsafe state (however, there may be an action leading to a critical state).

To illustrate the definition on a real example, please refer to Figure 2. In 2(a), the UGV is in a **safe** state, because all actions it can take lead again to safe states (supposing that actions for movement do not move the robot for more than a few centimeters). On the other hand, the robot as depicted in 2(b) is in a **critical** state, because going forward would make the robot fall over and break. If the robot executed action "go forward" once more, it would come to an **unsafe** state. Right after executing the action it would still not be broken; however, it would start falling and that is **unsafe**, because it is not equipped to withstand such fall and therefore it is almost sure it will break when it meets the ground.

3.2 Safety through ergodicity

An MDP is called *ergodic* iff for every state there exists a policy that gets the agent to any other state [20]. In other words, every mistake can be remedied in

(a) A safe state.

(b) A critical state – if the robot went still forward, it would fall down and probably break.

Fig. 2. An illustration of safe and critical states.

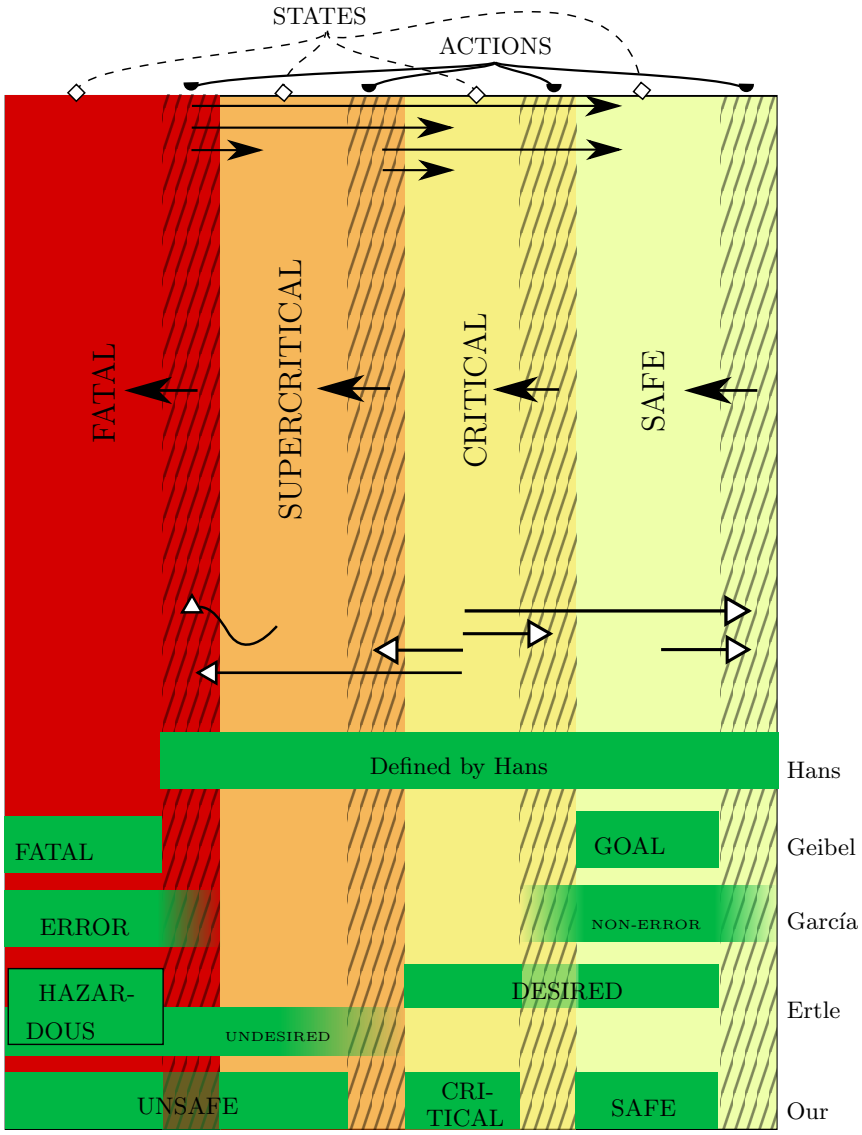

Fig. 3. *A summary of the definitions of safety.* The basic division is taken from Hans [14] and **fatal** states are added. States are drawn with solid background and white-headed arrows (⇾) denote the possible actions in the states. Actions are rendered with striped background and black-headed arrows (➔) end in states where it is possible to end up using the action.

such MDP. Moldovan [20] then defines *δ-safe policies* as policies guaranteeing that from any state the agent can get to the starting state with probability at least δ (using a *return policy*, which is different from the δ-safe one). Stated this way, the safety constraint may seem intractable, or at least impractical – it is even proved, that expressing the set of δ-safe policies is NP-hard [20]. An approximation of the constraint can be expressed in the terms of two other MDP problems which are easily solved [20]; that still leads to δ-safe policies, but the exploration performance may be suboptimal.

In our view, safety through ergodicity imposes too much constraints on the problems the agent can learn. It sometimes happens that a robot has to learn some task after which it is not able to return to the initial state (e.g. drive down a hill it cannot go upwards; a human operator then carries the robot back to the starting position). But the inability to "return home" in no means indicates the robot is in an unsafe state.

3.3 Safety through costs

Another definition of safety is to define a cost for taking an action/being in a state and minimize the worst-case cost of the generated policies (up to some failure probability). Such approach is presented in [15].

However, unless a threshold is set, this definition leads only to the *safest possible* policies, which are not necessarily *safe*. Expressing the safety using costs is natural for some RL tasks (e.g. when learning the function of a dynamic controller of an engine, the engine's temperature can be treated as a cost). Unfortunately, not all **unsafe** states can be described using such costs in general. In addition, specifying the right costs may be a difficult task.

3.4 Safety as variance of the expected return

An alternative to safety as minimization of a cost (either worst-case or expected) is minimizing both the cost and its variance. This approach is called *expected value-variance criterion* [15] and is used mainly in works prior 2000, e.g. [7]. A safe policy by this criterion can be viewed as a policy that minimizes the number of **critical** actions (because fatal transitions are expected to yield much larger costs than safe transitions, increasing the variance significantly).

As stated in [10], the worst-case approach is too restrictive and cautious. The other expected value-variance criteria suffer from the same disadvantages as safety through costs – mainly from the general difficulty to tune up the costs.

4 Safe exploration approaches

Finally, when the theoretical concepts have been shown and the various safety definitions have been presented, we can focus on the main part of this overview. Our categorization of safe exploration techniques is based on the work of García [10]. The basic division is as follows: approaches utilizing the expected return

or its variance (Sec. 4.1), labeling-based approaches (Sec. 4.2) and approaches benefiting from prior knowledge (Sec. 4.3).

4.1 Optimal control approaches

Techniques in this category utilize variations of the *expected value-variance* safety criterion. The most basic one is treating the rewards as costs (when a reward is denoted by r_t, the corresponding cost is denoted by c_t). Standard RL methods can then be used to solve the safe exploration task, as described e.g. in [7] for discounted infinite horizon.

The RL objective function

$$J = E\left[\sum_{t=0}^{\infty} \gamma^t c_t\right] \tag{5}$$

is called the *risk-neutral objective*. To make this objective *risk-sensitive*, we specify a *risk factor* α and rewrite the objective as: [15]

$$\begin{aligned} J = & \quad \tfrac{1}{\alpha} \log E\left[\exp\left(\alpha\gamma^t \textstyle\sum_{t=0}^{\infty} c_t\right)\right] \\ & \simeq E\left[\textstyle\sum_{t=0}^{\infty} \gamma^t c_t\right] + \tfrac{\alpha}{2} Var\left[\textstyle\sum_{t=0}^{\infty} \gamma^t c_t\right], \end{aligned} \tag{6}$$

which is also called the *expected value-variance criterion*. This approach is a part of theory using *exponential utility functions*, which is popular in optimal control [19]. To complete this section, the worst-case objective function (also called the *minimax objective*) is defined as

$$J = \sup\left[\sum_{t=0}^{\infty} \gamma^t c_t\right]. \tag{7}$$

As can be seen, the objective functions containing expectations cannot in fact assure that no unsafe state will be encountered. On the other hand, the minimax objective provides absolute certainty of the safety. However, it may happen that some of the unsafe states can only be reached with a negligible probability. In such cases, the *α-value* criterion defined by [15] can be used – it only takes into account rewards that can be reached with probability greater than α. In the work of Mihatsch [19], a scheme is presented that allows to "interpolate" between risk-neutral and worst-case behavior by changing a single parameter.

Delage's work [8] takes into account the uncertainty of parameters of the MDP. It is often the case that the parameters of the MDP are only estimated from a limited number of samples, causing the parameter uncertainty. He then proposes a possibility that the agent may "invest" some cost to lower the uncertainty in the parameters (by receiving some observations from other sources than exploration). A completely new research area then appears – to decide whether it is more valuable to pay the cost for observations, or to perform exploration by itself.

An approximation scheme for dealing with transition matrix uncertainty is presented in [21]. It considers a robust MDP problem and provides a worst-case, but also robust policy (with respect to the transition matrix uncertainty).

A theory generalizing these approaches can be found in [24]. The theory states that the optimal control decision is based on three terms – the *deterministic*, *cautionary* and *probing* terms.

> The deterministic term assumes the model is perfect and attempts to control for the best performance. Clearly, this may lead to disaster if the model is inaccurate. Adding a cautionary term yields a controller that considers the uncertainty in the model and chooses a control for the best expected performance. Finally, if the system learns while it is operating, there may be some benefit to choosing controls that are suboptimal and/or risky in order to obtain better data for the model and ultimately achieve better long-term performance. The addition of the probing term does this and gives a controller that yields the best long-term performance.[24]

To conclude this section, we think that these methods are not well suited for safe exploration – the expected value-variance and similar criteria provide no warranties on the actual safety. On the other hand, the worst-case approaches seem to be too strict.

4.2 Labeling-based approaches

The approaches utilizing some kind of state/action labeling (refer to Section 3.1 for the various labeling types) usually make use of two basic components – a *risk function* and a *backup policy*. The task of the *safety function* is to estimate the safety of a state or action. In the simplest case, the safety function can just provide the labeling of the given action; or it can return a likelihood that the action is safe; and in the best case, it would answer with a likelihood to be safe plus a variance (certainty) of its answer. The *backup policy* is a policy that is able to lead the agent out of the critical states back to the safe area. It is not obvious how to get such a policy, but the authors show some ways how to get one.

In the work of Hans [14], the most granular labeling is used, where fatal transitions are said to be the transitions with reward less than a given threshold. The safety function is learned during the exploration by collecting the so-called *min-reward samples* – this is the minimum reward ever obtained for executing a particular action in a particular state. The backup policy is then told to either exist naturally (e.g. a known safe, but suboptimal controller), or it can also be learned. To learn the backup policy, an RL task with altered Bellman equations is used:

$$\mathbf{Q}^*_{min}(s,a) = \max_{s'} \min \left[R(s,a,s'), \max_{a'} \mathbf{Q}^*_{min}(s',a') \right].$$

A policy derived from the computed $\mathbf{Q}^*_{min}$ function is then taken as the backup policy (as it maximizes the minimum reward obtained, and the fatal transitions

are defined by low reward). He defines a policy to be *safe*, if it executes only safe actions in safe states and produces non-fatal transitions in critical states. To learn such safe policy, he then suggests a level-based exploration scheme (although he gives no proofs why it should be better than any other exploration scheme). This scheme is based on the idea that it is better to be always near the known safe space when exploring. All unknown actions from one "level" are explored, and their resulting states are queued to the next "level". For exploration of unknown actions he proposes that the action should be considered critical until proved otherwise, so the exploration scheme uses the backup policy after every unknown action execution. A disadvantage of this approach is that the agent needs some kind of "path planning" to be able to get to the queued states and continue exploration from them.

García's PI-SRL algorithm [10] is a way to safeguard the classical policy iteration algorithm. Since the labels *error*/*non-error* are only for final states, the risk function here is extended by a so called *Case-based memory*, which is in short a constant-sized memory for storing the historical (s, a, $\mathbf{V}(s)$) samples and is able to find nearest neighbors for a given query (using e.g. the Euclidean distance). In addition to the *error* and *non-error* states, he adds the definition of *known* and *unknown* states, where *known* states are those that have a neighbor in the case-based memory closer than a threshold. A safe policy is then said to be a policy that always leads to *known non-error* final states. To find such policy, the policy iteration is initialized with the safe backup policy and exploration is done via adding a small amount of Gaussian noise to the actions. This approach is suitable for continuous state- and action-spaces.

Another approach is presented in the work of Geibel [12], where the risk and objective functions are treated separately. So the risk function only classifies the states (again only final states) as either *fatal* or *goal*, and the risk of a policy (risk function) is then computed as the expected risk following the policy (where *fatal* states have risk 1 and *goal* states have risk 0). The task is then said to be to maximize the objective function (e.g. discounted infinite horizon) w.r.t. the condition that the risk of the considered policies is less than a safety threshold. The optimization itself is done using modified Q-learning, and the optimized objective function is a linear combination of the original objective function and the risk function. By changing the weights in the linear combination the algorithm can be controlled to behave more safely or in a more risk-neutral way.

A generalization of Geibel's idea to take the risk and reward functions separately can be found in the work of Kim [18]. In this work, the constrained RL task is treated as a *Constrained MDP* and the algorithm *CBEETLE* for solving the Constrained MDPs is shown. The advantage of this work is that it allows for several independent risk (cost) functions and doesn't need to convert them to the same scale.

A similar approach of using constrained MDP to solve the problem can be found in the work of Moldovan [20]. He does, however, use the ergodicity condition to tell safe and unsafe states apart (that is, safe are only those states from which the agent can get back to the initial state). Moreover, this approach is only

shown to work for toy examples like the grid world with only several thousands of discrete states, which may not be sufficient for real robotics tasks.

The idea of having several risk functions is further developed by Ertle [9]. The agent is told to have several behaviors and a separate safety function is learned for each behavior. This approach allows for modularity and sharing of the learned safety functions among different types of agents. More details on this work will be provided in the next section, because it belongs to learning with teachers.

An approach slightly different from the previously mentioned in this section is using the methods of reachability analysis to solve safe exploration. Gillula in his work [13] defines a set of *keep-out states* (corresponding to **unsafe** in our labeling) and then a set called $Pre(\tau)$ is defined as a set of all states from which it is possible to get to a *keep-out* state in less than τ steps. Reachability analysis is used to compute the $Pre(\tau)$ set. *Safe states* are then all states not in $Pre(\tau)$ for a desired τ. This approach, however, doesn't utilize reinforcement learning, it computes the optimal policy using standard supervised learning methods with one additional constraint – that the system must use safe actions near the $Pre(\tau)$ set. On the other hand, the system is free to use whatever action desired when it is not near $Pre(\tau)$.

As was presented in this section, the labeling-based approaches provide a number of different ways to reach safety in exploration. They are, however, limited in several ways – some of them make use of the (usually hard-to-obtain) transition matrix, the others may need to visit the unsafe states in order to learn how to avoid them, or need the state-space to be metric.

4.3 Approaches benefiting from prior knowledge

The last large group of safe exploration techniques are the ones benefiting from various kinds of prior knowledge (other than the parameters of the MDP). We consider this group the most promising for safe exploration, because "it is impossible to avoid undesirable situations in high-risk environments without a certain amount of prior knowledge about the task"[10].

The first option how to incorporate prior knowledge into exploration is to initialize the search using the prior knowledge. In fact, several works already mentioned in previous sections use prior knowledge – namely the approaches with a backup policy (Hans [14], García [10]). Also, García suggests that the initial estimate of the value function can be done by providing prior knowledge, which results in much faster convergence (since the agent does no more have to explore really random actions, the estimate of the value function already "leads it" the right way) [10].

Another option how to incorporate prior knowledge is by using *Learning from Demonstration* (LfD) methods. Due to the limited space, we will not give the basics of LfD – a good overview of the state-of-the-art methods is for example in [2]. For our overview, it is sufficient to state that LfD methods can derive a policy from a set of demonstrations provided by a teacher. What is important, is that the teacher does not necessarily have to have the same geometrical and

physical properties as the trainee (although it helps the process if possible). It is therefore possible to use LfD to teach a 5-joint arm to play tennis, while using 3-joint human arm as the source of demonstrations (but the learned policy may be suboptimal; RL should then be used to optimize the policy).

In *Apprenticeship Learning* [1], the reward function is learned using LfD. The human pilot flies a helicopter at his best, and both system dynamics and the reward function are learned from the demonstrations. It is however apparent that the performance of the agent is no longer objectively optimal, but that it depends on the abilities of the human pilot.

Another way of incorporating prior knowledge into the learning process is to manually select which demonstrations will be provided, as in the work of Ertle [9]. In the work it is suggested that more teacher demonstrations should come from the areas near the unsafe set, in order to teach the agent precisely where the border between safe and unsafe is located.

The last technique described in our overview is interleaving autonomous exploration with teacher demonstrations. As in the previous case, some teacher demonstrations are provided in advance, and then the exploration part starts utilizing the teacher-provided information. After some time, or in states very different from all other known states, the agent requests the teacher to provide more examples [2,5]. The idea behind this algorithm is that it is impossible to think out in advance what all demonstrations will the agent need in order to learn the optimal policy.

Finishing this section, the algorithms utilizing prior knowledge seem to be the most promising out of all the presented approaches. They provide both a speedup of the learning process (by discarding the low-reward areas) and a reasonable way to specify the safety conditions (via LfD or interleaving).

5 Conclusion

In our work we have given a short introduction on the basics of *Markov Decision Processes* as well as the basic *Reinforcement Learning* methods like *Value Iteration*, *Q-learning* and *Policy Iteration*. In Section 3 we have summarized many recent approaches on how to define *safety* in the framework of optimal control and reinforcement learning. We have also proposed a novelty definition of safety, which divides the state space to *safe*, *critical* and *unsafe* states. We have shown that all other labeling-based safety definitions are covered by our new definition.

In Section 4 many different safe exploration methods are categorized into three basic groups – algorithms from optimal control theory, reinforcement learning algorithms based on state labeling, and algorithms utilizing extra prior knowledge. We have shortly summarized the advantages and disadvantages of the particular approaches. We have also stated that at least for difficult real-world problems, safe exploration without prior knowledge is practically impossible, and prior knowledge almost always helps to achieve faster convergence. Another observation has been that some of the safe exploration algorithms need to visit

unsafe states to correctly classify them later, which might discard them from some usage scenarios where the unsafe states are really fatal.

It seems to us that the field of safe exploration in reinforcement learning has been very fragmented and lacks an all-embracing theory. However, the question is, if it is even possible to find such theory – the main problem may be the fragmentation and differences of various RL methods themselves. At least, the safe exploration community would benefit from a unification of the terminology (and our proposal of the novelty safety labeling would like to help that).

Other ways of possible future research are for example the following. New ways of incorporating prior knowledge into methods not utilizing it yet could bring interesting speed-up of those algorithms. There is also a bottleneck in the estimation of the results of unknown actions – some advanced function approximation methods should be explored (we aim to investigate Gaussian Processes this way). There are not enough experiments from difficult continuous real-world environments, which would show for example how large problems can be solved using safe exploration. The interleaved learning needs some guidelines on how to cluster the queries for the teacher to some larger "packs" and "ask" them together, possibly increasing the fully autonomous operating time. Last, but not least, the possibility to share some learned safety functions among different kinds of robots seems to be an unexplored area with many practical applications (maybe robot-to-robot LfD could be used).

6 Acknowledgments

This work has been supported by the Grant Agency of the Czech Technical University in Prague under Project SGS13/142/OHK3/2T/13, and by EC project FP7-ICT-609763 TRADR. We would like to thank Karel Zimmermann and Michal Reinstein (Center for Machine Perception, CTU in Prague) for their valuable insights into reinforcement learning methods.

References

1. Abbeel, P., Coates, A., Quigley, M., Ng, A.Y.: An application of reinforcement learning to aerobatic helicopter flight. In: Proceedings of the 2006 Conference on Advances in Neural Information Processing Systems 19. vol. 19, p. 1 (2007)
2. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems 57(5), 469–483 (May 2009)
3. Barto, A.G., Sutton, R.S., Brouwer, P.S.: Associative search network: A reinforcement learning associative memory. Biological cybernetics (1981)
4. Bertsekas, D.P.: Dynamic programming: deterministic and stochastic models. Prentice-Hall (1987)
5. Chernova, S., Veloso, M.: Confidence-based policy learning from demonstration using Gaussian mixture models. In: AAMAS '07 Proceedings. p. 1. ACM Press (2007)
6. Consortium, N.: NIFTi robotic UGV platform (2010)

7. Coraluppi, S.P., Marcus, S.I.: Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. Automatica (1999)
8. Delage, E., Mannor, S.: Percentile optimization in uncertain Markov decision processes with application to efficient exploration. In: Proceedings of the 24th international conference on Machine learning - ICML '07. pp. 225–232. ACM Press, New York, New York, USA (2007)
9. Ertle, P., Tokic, M., Cubek, R., Voos, H., Soffker, D.: Towards learning of safety knowledge from human demonstrations. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5394–5399. IEEE (Oct 2012)
10. Garcia, J., Fernández, F.: Safe exploration of state and action spaces in reinforcement learning. Journal of Artificial Intelligence Research 45, 515–564 (2012)
11. Garcia Polo, F.J., Rebollo, F.F.: Safe reinforcement learning in high-risk tasks through policy improvement. In: 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL). pp. 76–83. IEEE (Apr 2011)
12. Geibel, P.: Reinforcement learning with bounded risk. ICML pp. 162–169 (2001)
13. Gillula, J.H., Tomlin, C.J.: Guaranteed safe online learning of a bounded system. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 2979–2984. IEEE (Sep 2011)
14. Hans, A., Schneegaß, D., Schäfer, A., Udluft, S.: Safe exploration for reinforcement learning. In: Proceedings of European Symposium on Artificial Neural Networks. pp. 23–25. No. April (2008)
15. Heger, M.: Consideration of risk in reinforcement learning. In: 11th International Machine Learning Conference (1994)
16. Howard, R.A.: Dynamic Programming and Markov Processes. Technology Press of Massachusetts Institute of Technology (1960)
17. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
18. Kim, D., Kim, K.E., Poupart, P.: Cost-Sensitive Exploration in Bayesian Reinforcement Learning. In: Proceedings of Neural Information Processing Systems (NIPS) (2012)
19. Mihatsch, O., Neuneier, R.: Risk-sensitive reinforcement learning. Machine learning 49(2-3), 267–290 (2002)
20. Moldovan, T.M., Abbeel, P.: Safe Exploration in Markov Decision Processes. In: Proceedings of the 29th International Conference on Machine Learning (May 2012)
21. Nilim, A., El Ghaoui, L.: Robust Control of Markov Decision Processes with Uncertain Transition Matrices. Operations Research 53(5), 780–798 (Oct 2005)
22. Peter Geibel, Wysotzki, F.: Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. Journal Of Artificial Intelligence Research 24, 81–108 (Sep 2011)
23. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1994)
24. Schneider, J.G.: Exploiting model uncertainty estimates for safe dynamic control learning. Neural Information Processing Systems 9, 1047–1053 (1996)
25. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8(3-4), 279–292 (May 1992)
26. Williams, R.J., Baird, L.C.: Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep., Northeastern University,College of Computer Science (1993)